Coloring Reconfiguration Problems and Their Generalizations

by
Hiroki Osawa

Submitted to
Department of System Information Sciences
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy (Information Sciences)

Graduate School of Information Sciences
Tohoku University, Japan

2020

Acknowledgements

First of all, I would like to express my great gratitude for my supervisor Professor Xiao Zhou. His deep insight gave me large amount of advices and suggestions. He also offered a comfortable space for my life and research activity. Without his continual help and encouragement, this thesis would not be completed.

I am really grateful to my thesis advisor, Associate Professor Akira Suzuki. He also helped me to write my thesis in comprehensible style, and gave me numerous number of advices to refine my writing style. Without his supports, this thesis would not be completed.

I would also like to thank Associate Professor Takehiro Ito for his many suggestions and guidance. All of his useful suggestions and advices are invaluable for my everyday research.

Furthermore, I would like to express my special thanks to the other members of my graduate committee, Professor Ayumi Shinohara and Professor Hiroki Shizuya, for their insightful suggestions and comments.

Finally, I would like to acknowledge with grateful the everyday cooperation rendered by the member of Algorithm Theory Laboratory for many things.

Abstract

Recently, the framework of reconfiguration problem is studied intensively in the field of theoretical computer science. The framework of reconfiguration deals with the problem where we wish to find a step-by-step transformation between initial and target configurations while preserving a constraint of some combinatorial search problem, and each step must respect a fixed reconfiguration rule. In this thesis we study a generalization of an well-studied reconfiguration problem k-COLORING RECONFIGURATION. In k-COLORING RECONFIGURATION, we are given two feasible k-colorings of a graph G, and asked to determine whether one coloring can be transformed into the other by recoloring one vertex at a time, while always maintaining a feasible k-coloring. In this thesis we generalize the reconfiguration rule of k-coloring reconfiguration by restricting recolorable pair of color, in the form of a (directed/undirected) graph whose vertex set is the color set $\{1, 2, \ldots, k\}$, and give a precise analysis of the complexity status of the generalized problem with respect to the graph class of the graph whose vertices are colors.

Contents

List of Figures

Chapter 1

Introduction

1.1 Reconfiguration

In the field of theoretical computer science, (*combinatorial*) *reconfiguration* is a framework which models the "dynamic" situation where we wish to find step-by-step transformation between two feasible states in which all intermediate states are also feasible and each step respects a fixed reconfiguration rule. A classical example of reconfiguration problem is *sliding block puzzle*, such as the *15-puzzle* [22]: the feasible states are the arrangements of distinct 15 rectangle blocks on a 4×4 rectangle grid, and the reconfiguration rule is to slide a block to an empty square on the grid (See Fig 1.1).

(Combinatorial) reconfiguration problem can be viewed as a problem asking reachability of two vertices of *reconfiguration graph*, whose vertices are feasible states and edges represent a fixed reconfiguration rule. In the example of 15-puzzle, the solution graph has all arrangements of blocks as the vertex set, and two vertices (arrangements) are joined by edge if one can be obtained by sliding a block of the another arrangement. The vertex set of reconfiguration graph is often defined as a solution space of well-studied search problem, such as INDEPENDENT SET [20], BOOLEAN SATISFIABILITY [25, 21], SHORTEST PATH [6, 7], MATCHING [17].

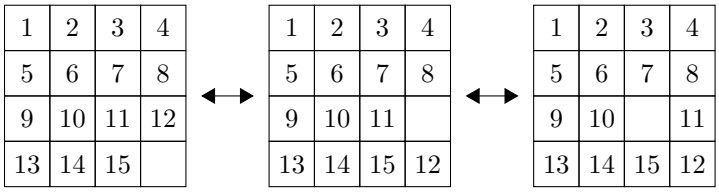


Figure 1.1: Arrangements and reconfiguration rule of 15-puzzle.

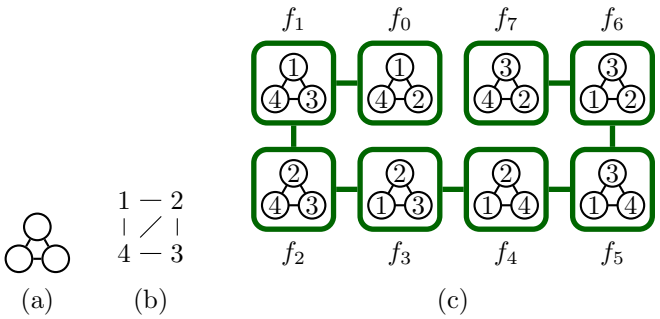


Figure 1.2: (a) An input graph G, (b) a recolorablity graph R with four colors 1, 2, 3 and 4, and (c) an $(f_0 \to f_7)$-reconfiguration sequence.

In this thesis we investigate COLORING RECONFIGURATION problem [8, 12], which is one of the most well-studied reconfiguration problem. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. A mapping $f : V(G) \to C$ from V to a color set $C = \{1, 2, \ldots, k\}$ of size k is called *k-coloring* of G if $f(v) \neq f(w)$ holds for any edge $vw \in E(G)$. Two colorings f, f' of a graph G are called *adjacent* if they are differ on exactly one vertex, i.e., $|\{f(v) \neq f'(v) \mid v \in V(G)\}| = 1$. For a graph G and its two k-colorings f_0, f_t, COLORING RECONFIGURATION asks whether there exists a sequence $\langle f_0, f_1, \ldots, f_r \rangle$ of k-colorings where f_i, f_{i+1} are adjacent, and $f_r = f_t$. Such a sequence is called *reconfiguration sequence*.

Figure 1.2(c) shows an reconfiguration sequence $\langle f_0, f_1, \ldots, f_7 \rangle$ of 4-colorings of a graph G, which is a complete graph K_3 illustrated in Figure 1.2(a).

COLORING RECONFIGURATION problem can be viewed as a reachability problem

on a *reconfiguration graph*, whose vertices are colorings, and whose adjacency relation is also an adjacency relation of colorings.

1.2 Known and related results

Coloring reconfiguration has been studied from various viewpoints [3, 4, 5, 8, 9, 10, 12, 14, 26, 15, 18, 19, 27, 28]. In particular, a sharp analysis has been obtained from the viewpoint of the number k of colors: Bonsma and Cereceda [8] showed that coloring reconfiguration is PSPACE-complete for any fixed number k of colors at least four. On the other hand, Cereceda et al. [12] proved that coloring reconfiguration can be solved in polynomial time if the number k of colors is at most three. This computing time was later improved to linear time by Johnson et al. [19]. Besides the number of colors, the complexity status of coloring reconfiguration has been clarified based on several "standard" measures. From the viewpoint of graph classes, coloring reconfiguration problem is known to be PSPACE-complete even for planar bipartite graph [8]. For bounded treewidth graph [3], chordal graph [5] and line graphs of trees [18], there are sufficient conditions that reconfiguration graph is connected. From the viewpoint of generalization of coloring, list coloring reconfiguration [14, 15, 18], H-coloring reconfiguration [11, 27] and their generalization CSP reconfiguration [9] are studied intensively.

As with other reconfiguration problems, bounded length variant is also studied for coloring reconfiguration: we are asked to find a transformation sequence of the length at most ℓ. Johnson et al [19] showed that the shortest length a reconfiguration sequence of 3-coloring reconfiguration can be computed in linear time if it exists. On the other hand, Bonsma et al. [9] showed that k-coloring

RECONFIGURATION is W[1]-hard for any fixed number k of colors at least four.

1.3 Our problem

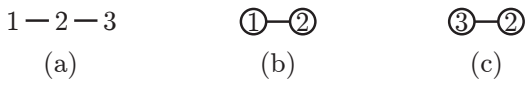
In this paper we introduce the concept of "recolorability" and generalize COLORING RECONFIGURATION problem. For an integer $k \geq 1$, let C be the *color set* of k colors $1, 2, \ldots, k$. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. Recall that a k-coloring of G is a mapping $f : V(G) \to C$ such that $f(v) \neq f(w)$ holds for any edge $vw \in E(G)$. The *recolorability* on C is given in terms of an undirected graph R, called the *recolorability graph* on C, such that $V(R) = C$; each edge $ij \in E(R)$ represents a "recolorable" pair of colors $i, j \in V(R) = C$. Then, two k-colorings f and f' of G are *adjacent* (*under* R) if the following two conditions hold:

(a) $\big|\{v \in V(G) : f(v) \neq f'(v)\}\big| = 1$, that is, f' can be obtained from f by *recoloring* a single vertex $v \in V(G)$; and

(b) if $f(v) \neq f'(v)$ for a vertex $v \in V(G)$, then $f(v)f'(v) \in E(R)$, that is, the colors $f(v)$ and $f'(v)$ form a recolorable pair.

For each $i \in \{1, 2, \ldots, 7\}$, two 4-colorings f_{i-1} and f_i in Figure 1.2(c) are adjacent under the recolorability graph R in Figure 1.2(b). As defined above, the known adjacency relation for COLORING RECONFIGURATION requires only Condition (a) above, that is, we can recolor a vertex from any color to any color directly. Observe that this corresponds to the case where R is a complete graph of size k, and hence our adjacency relation generalizes the known one.

Given a graph G, two k-colorings f_0 and f_r of G, and a recolorability graph R on C, the COLORING RECONFIGURATION problem UNDER R-RECOLORABILITY is the decision problem of determining whether there exists a sequence $\langle f_0, f_1, \ldots, f_\ell \rangle$ of k-colorings of G such that $f_\ell = f_r$ and f_{i-1} and f_i are adjacent under R for



(a) (b) (c)

Figure 1.3: (a) Recolorability graph R with three colors 1, 2 and 3, and (b) and (c) 3-colorings f_0 and f_r of a graph consisting of a single edge, respectively.

all $i \in \{1, 2, \ldots, \ell\}$; such a desired sequence is called an $(f_0 \to f_r)$-*reconfiguration sequence*, and its *length* (i.e., the number of recoloring steps) is defined as ℓ. For example, the sequence $\langle f_0, f_1, \ldots, f_7 \rangle$ in Figure 1.2(c) is an $(f_0 \to f_7)$-reconfiguration sequence whose length is 7.

We emphasize that the concept of recolorability constraints changes the reachability of k-colorings drastically. For example, the $(f_0 \to f_7)$-reconfiguration sequence in Figure 1.2(c) is a shortest one between f_0 and f_7 under the recolorability graph R in Figure 1.2(b). However, in COLORING RECONFIGURATION (in other words, if R would be K_4 and would have the edge joining colors 1 and 3), we can recolor the vertex from 1 to 3 directly.

As another example, the instance illustrated in Figure 1.3 is a no-instance for our problem, but is clearly a yes-instance for COLORING RECONFIGURATION with $k = 3$.

1.4 Our contribution

In this thesis we investigate COLORING RECONFIGURATION from the viewpoints of recolorability graph, irreversibility, and graph class.

1.4.1 Recolorability graph

We first classify the complexity status of COLORING RECONFIGURATION UNDER R-RECOLORABILITY with respect to the structure of the recolorability R. Our results

are summarized in Table 1.1. Our result of complexity status covers the cases

Table 1.1: Computational complexity of COLORING RECONFIGURATION UNDER R-RECOLORABILITY

	maximum number of cycles contained in one connected component	
maximum degree $\Delta(R)$ of R	at most one	at least two
$\Delta(R) \leq 2$	Linear	(Such a graph does not exist)
$\Delta(R) = 3$	Some results (See Section 3.4, 3.5)	PSPACE-c
$\Delta(R) \geq 3$	PSPACE-c	PSPACE-c

where the recolorability graph is a complete graph, therefore our results generalize the known results of computational complexity of COLORING RECONFIGURATION problem with respect to the number of colors [8, 12].

For the case where the recolorability graph R is of maximum degree at most two, we also show that we can compute the shortest length of the reconfiguration sequence in linear time. This is also a generalization of known result such that the reconfiguration of the shortest length can be computed in linear time for 3-COLORING RECONFIGURATIONreconfiguration [19].

1.4.2 Irreversible rules

In this thesis we also generalize the recolorability graphs to directed graphs, and introduce irreversible transformation rules to COLORING RECONFIGURATION. In our setting of directed recolorability graph, a vertex can be recolored from a color c to another color c' only if there is a arc from c to c' on the directed recolorablity graph. Such an irreversible transformation rules have not been considered for COLORING RECONFIGURATION problem, and such rules can capture the behavior of softwares

and machines in the real world. We show the results of tractability and intractability: the problem is NP-complete if the recolorability graph is polytree, on the other hand, the problem is polynomial-time solvable if the recolorability graph is rooted tree.

1.4.3 Edge-coloring reconfiguration

We also study the complexity status of EDGE-COLORING RECONFIGURATION and its generalization LIST EDGE-COLORING RECONFIGURATION. For about the computational complexity of LIST EDGE-COLORING RECONFIGURATION problem, partial answer is given by Ito et al. [18]: LIST EDGE-COLORING RECONFIGURATION is PSPACE-complete for any fixed number of colors at least six, while it is polynomial-time solvable if the number of colors is at most three. In this thesis we show that LIST EDGE-COLORING RECONFIGURATION is PSPACE-complete for any fixed number of colors at least four, and EDGE-COLORING RECONFIGURATION is PSPACE-complete for any fixed number of colors at least five. These results also apply to the graphs of bounded bandwidth.

1.5 Organization of this thesis

In Chapter 2, we give basic terminologies and notations of graph theory and theoretical computer science, and notions used for our problems. In Chapter 3 we prove the computational hardness of COLORING RECONFIGURATION UNDER R-RECOLORABILITY for some sorts of recolorability graph R. On the other hand, in Chapter 4 we give polynomial-time algorithms of COLORING RECONFIGURATION UNDER R-RECOLORABILITY for some sorts of recolorability graph R. In Chapter 5 we study a further generalization of COLORING RECONFIGURATION UNDER R-RECOLORABILITY, where the recolorability graph is undirected and transformation

is generally irreversible. In Chapter 6 we also investigate COLORING RECONFIGURATION problem from the viewpoint of graph classes. We show PSPACE-hardness of EDGE-COLORING RECONFIGURATION and its generalization LIST EDGE-COLORING RECONFIGURATION. Finally, in Chapter 7 we conclude our thesis.

Chapter 2

Preliminaries

In this chapter, we present basic terminologies and notations widely used in this thesis. We sometimes need additional definitions for description, which are given when necessary.

2.1 Basic terminologies of graph theory

2.1.1 Graph and subgraphs

An (undirected) graph G is an ordered pair (V, E) of a set V of *vertices* and a set $E \subseteq \{\{v, w\} \mid v, w \in V\}$ of *edges*, which are unordered pairs (sets of size two) of vertices. In this thesis we only deal with finite graph, whose vertex set is finite. We sometimes denote by $V(G)$ and $E(G)$ the vertex set V and edge set E of a graph $G = (V, E)$. We often denote an edge $\{v, w\} \in E$ by vw. If $vw \in E$ then we call v, w are *adjacent* or *joined*, and v, w is *incident* to vw. Also, if $v, w \in V$ are adjacent then w is called a *neighbor* of v. The *neighborhood* $N(G, v)$ is a set of neighbors of v in G, and the *closed neighborhood* $N[G, v]$ is a set $N(G, v) \cup \{v\}$.

A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$; and we sometimes denote it by $G' \subseteq G$. Conversely, G is called a *supergraph* of G' if G' is a subgraph of G. A subgraph $G' = (V', E')$ of $G = (V, E)$

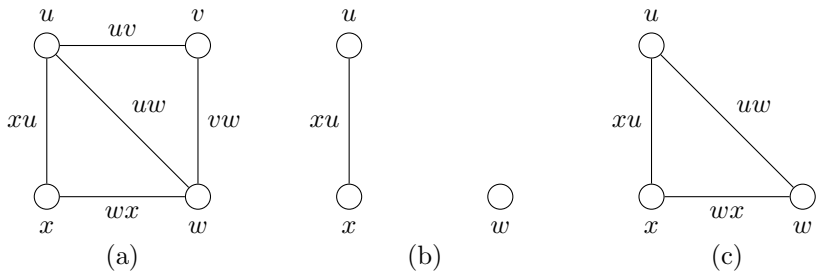


Figure 2.1: (a) A graph G with four vertices and five edges, (b) a (not induced) subgraph of G, and (c) an induced subgraph $G[\{u, w, x\}]$

.

is *induced by* V', or simply called *induced subgraph* if $vw \in E \Rightarrow vw \in E'$ for any two vertices $v, w \in V'$. We denote by $G[V']$ a subgraph of G induced by $V' \subseteq V(G)$. Figure 2.1 shows an graph (Figure 2.1(a)) and its subgraph and induced subgraph (Figure 2.1(b)(c)).

2.1.2 Walk, path, cycle, clique and connectedness

For a graph G, a sequence $\langle v_0, v_1, \ldots, v_\ell \rangle$ is called *walk* between v_0 and v_ℓ if $v_i v_{i+1} \in E(G)$ for each $i \in \{0, \ldots, \ell - 1\}$. The *length* of a walk $\langle v_0, v_1, \ldots, v_\ell \rangle$ is defined as ℓ. A walk without any repetition of vertices is called a *path*. A path $\langle v_0, v_1, \ldots, v_\ell \rangle$ is also called *cycle* if $v_\ell v_0$ is an edge. The terms "path" and "cycle" are also used in terms of the subgraphs. For a path (cycle) $\langle v_0, v_1, \ldots, v_\ell \rangle$ of G, a subgraph $G' = (\{v_0, v_1, \ldots, v_\ell\}, \{v_0 v_1, v_1 v_2, \ldots, v_{\ell-1} v_\ell\})$ of G is also called a *path (cycle)*. A subset $V' \subseteq V(G)$ of the vertex set of a graph G is called *clique* if any two vertices in V' are joined by an edge in $G[V']$. A graph G is *connected* if there exists a path between any two vertices of G. A connected maximal induced subgraph of G is called a *connected component* of G.

2.1.3 Digraphs

A *directed graph* or a *digraph* is an ordered pair $\overrightarrow{G} = (V, A)$ of a set V of vertices and a set $A \subseteq \{(v, w) \mid v, w \in V\}$ of *arcs*, which are ordered pairs of vertices. For a digraph (V, A), its *underlying graph* is an undirected graph (V, E) where $E = \{\{v, w\} \mid (v, w) \in A\}$. We sometimes denote by $V(\overrightarrow{G})$ and $A(\overrightarrow{G})$ the vertex set V and arc set A of a directed graph $\overrightarrow{G}$. For an undirected graph G, we denote by $\overrightarrow{G}$ a digraph whose underlying graph is G, and also denote by $A(\overrightarrow{G})$ the arc set of $\overrightarrow{G}$. We denote by vw an edge joining two vertices v and w in an undirected graph, while by (v, w) an arc from v to w in a digraph. A directed graph is *simple* if there is no *loop*, that is an arc (v, v) for some vertex v. In this paper we only deal with simple digraphs. A directed graph is *oriented* if it does not has both of arcs (v, w) and (w, v) for any two vertices v, w. In this paper, we say that a digraph $\overrightarrow{G}$ is *connected* if $\overrightarrow{G}$ is weakly connected, that is, the underlying graph G is connected. A vertex v in a digraph $\overrightarrow{G}$ is called a *source* vertex if the in-degree of v is zero, while it is called a *sink* vertex if the out-degree of v is zero. A sequence $\langle v_0, v_1, v_2, \ldots, v_l \rangle$ of vertices $v_0, v_1, \ldots, v_l$ and arcs is called a *forward walk from* v_0 *on* $\overrightarrow{G}$ if an arc from v_{i-1} to v_i exists for all $i \in \{1, 2, \ldots, l\}$; while it is called a *backward walk to* v_0 *on* $\overrightarrow{G}$ if an arc from v_i to v_{i-1} exists for all $i \in \{1, 2, \ldots, l\}$. The *length* of forward (backward) walk $\langle v_0, v_1, v_2, \ldots, v_\ell \rangle$ is ℓ. A forward walk is called a *directed path* if it has no repetition of a vertex in its sequence.

2.1.4 Graph classes of undirected graph

A graph is called a *path* (*cycle*) if itself is a path (cycle). A graph containing no cycle is called a *tree* if connected, otherwise it is called a *forest*. Especially, a tree of maximum degree three is called a *binary tree*. A graph G is called a *complete* if

$V(G)$ is a clique of G. A graph is called *unicyclic* if it contains exactly one cycle. A graph is called *planar* if it can be drawn in the plane in such a way, its vertices are point on the plane, and its edges are curves between two points, and there is no crossing of any two edges.

A *line graph* $L(G)$ of a graph G is defined as follows:

$$V(L(G)) = E(G)$$
$$E(L(G)) = \{ee' \mid e, e \in E(G), e \cap e' \neq \emptyset\}.$$

Line graph is also a class of graphs where all graphs are line graph of some graph.

2.1.5 Graph classes of digraph

An oriented digraph $\overrightarrow{G}$ is called *polytree* if its underlying graph is a tree. A polytree is also called *rooted tree* if it has a *root vertex* from which there are directed paths to any vertex in $V(\overrightarrow{G})$.

2.2 Terms of computational complexity

2.2.1 Problems

A *problem* is a language (set of strings) $\mathcal{P} \subseteq \Sigma^*$ where $\Sigma = \{0, 1\}$. Each string $x \in \Sigma^*$ is called an *instance*. An instance x is called *yes-instance* of a problem $\mathcal{P}$ if $x \in \mathcal{P}$, otherwise it is called *no-instance* of the problem $\mathcal{P}$.

2.2.2 P and Polynomial-time reduction

P is the class of all problems which can be solved in polynomial-time. For two problems $\mathcal{P}, \mathcal{P}'$, a *reduction* from $\mathcal{P}$ to $\mathcal{P}'$ is an algorithm which maps any instance $x \in \Sigma^*$ to an instance $y \in \Sigma^*$ such that $x \in \mathcal{P}$ if and only if $y \in \mathcal{P}'$. If a reduction runs in polynomial time, it is called a *polynomial-time reduction*.

2.2.3 NP and NP-hardness

NP is the class of all problems which can be solved in nondeterministic polynomial-time. A problem $\mathcal{P}$ is called *NP-hard* if for any problem $\mathcal{P}'$ in NP, there exists a polynomial-time reduction from $\mathcal{P}'$ to $\mathcal{P}$. A problem which is NP-hard and also in NP is called *NP-complete*.

2.2.4 PSPACE and PSPACE-hardness

PSPACE is the class of all problems which can be solved in nondeterministic polynomial-time. A problem $\mathcal{P}$ is called *PSPACE-hard* if for any problem $\mathcal{P}'$ in PSPACE, there exists a polynomial-time reduction from $\mathcal{P}'$ to $\mathcal{P}$. A problem which is PSPACE-hard and also in PSPACE is called *PSPACE-complete*.

2.3 Terms used in this thesis

2.3.1 Coloring

Let $C = \{1, 2, \ldots, k\}$ be a *color set* of size k *k-coloring* of the graph G is a mapping $f : V(G) \to C$ such that for any edge $vw \in E(G)$, $f(v) \neq f(w)$ holds. For a graph G and nonnegative integer $k \geq 1$, COLORING problem asks whether there exists a k-coloring of G.

2.3.2 Adjacency and recolorability

Two k-colorings f, f' of a graph G are called *adjacent* if the following condition holds:

(a) $|\{v \in V(G) \mid f(v) \neq f'(v)\}| = 1$.

For a color set $C = \{1, 2, \ldots, k\}$, a *recolorability graph* R is an undirected graph whose vertex set is the color set C. For a recolorability graph R, two k-colorings f, f'

are called *adjacent under* R if the above condition (a) and the following condition hold:

(b) For any vertex $v \in V(G)$, if $f(v) \neq f'(v)$ then $f(v)f'(v) \in E(R)$.

2.3.3 Reconfiguration graph and reconfiguration sequence

For a graph G and a recolorability graph R, *reconfiguration graph* $\mathcal{C}_R(G)$ is a graph whose vertex set is a set of all k-colorings of G, and whose two vertices are joined by an edge if they are adjacent under R.

For two colorings f, f' of G, a $(f \to f')$*-reconfiguration sequence under* R is a path between f and f' on $\mathcal{C}_R(G)$.

If R is a complete graph whose vertex set is $\{1, 2, \ldots, k\}$, we sometimes denote $C_R(G)$ by $C_k(G)$.

2.3.4 Frozen

Let G be a graph and R be a recolorability graph which has k colors. For a k-coloring f of G, a vertex $v \in V(G)$ is called *frozen on* f if for any $(f \to f')$-reconfiguration sequence $f(v) = f'(v)$. If G and R is trivial in the context, we denote the set of vertices frozen on f by $\mathsf{Frozen}(f)$.

2.3.5 Coloring reconfiguration

For a graph G and a recolorability graph R which has k colors, and two k-colorings f_0, f_t, COLORING RECONFIGURATION UNDER R-RECOLORABILITY asks whether there exists an $(f_0 \to f_r)$-reconfiguration sequence. We call f_0 and f_r *initial* and *target* colorings respectively. We redefine the k-COLORING RECONFIGURATION as a special case of COLORING RECONFIGURATION UNDER R-RECOLORABILITY where R is a complete graph of size k.

Chapter 3

Computational hardness

In this chapter, we clarify the computational hardness of the problem from the viewpoint of recolorability graphs R. Through this section we assume R is connected graph. Because if R is disconnected, any two vertices v, w whose initial colors a, b are in mutually disconnected two components of R cannot have the same color, therefore we can separate the instance into subinstances such that each of whose graph is a subgraph induced by the vertices whose initial colors are in the same connected component of R.

In Section 3.1, we first introduce the list variant of the problem. Interestingly, the list variant is equivalent with the non-list one in our reconfiguration problem, and hence it suffices to construct reductions to the list variant. In Section 3.2, we then show that the list variant (and hence the non-list one) is PSPACE-complete for an arbitrary recolorability graph with maximum degree at least four. In Section 3.3, we show that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is PSPACE-complete for an arbitrary recolorability graph R which has more than one cycle.

3.1 List recolorability

In the *list* variant, each vertex v of a graph G is associated with a subgraph $R_L(v)$ of the common recolorablity graph R; we call $R_L(v)$ the *list recolorability of* v, and sometime call the list assignment (mapping) R_L the *list* R*-recolorability*. Note that $R_L(v)$ is not necessarily a spanning subgraph of R. Let $k = |V(R)|$. Then, a k-coloring f of G is called a *list coloring* of G if $f(v) \in V(R_L(v))$ for all vertices v in G. Observe that for any supergraph R' of R, any list R-recolorability is also list R'-recolorability. We say that two list colorings f and f' are *adjacent under* R_L if they differ in exactly one vertex v such that $f(v)f'(v) \in E(R_L(v))$. Analogous to the R-reconfiguration graph, we define the R_L*-reconfiguration graph* on G, denoted by $\mathcal{C}_{R_L}(G)$, as the undirected graph whose nodes correspond to list colorings of G, and two nodes in $\mathcal{C}_{R_L}(G)$ are joined by an edge if their corresponding list colorings are adjacent under R_L.

Let G be an input graph with a list R-recolorability R_L. Then, for two list colorings f_0 and f_r of G, the COLORING RECONFIGURATION problem UNDER LIST R-RECOLORABILITY (the *list variant*, for short) is the decision problem of determining whether $\mathcal{C}_{R_L}(G)$ contains an $(f_0 \to f_r)$-reconfiguration sequence. Observe that COLORING RECONFIGURATION UNDER R-RECOLORABILITY can be seen as the list variant such that $R_L(v) = R$ holds for every vertex $v \in V(G)$. Furthermore, note that $\mathcal{C}_{R_L}(G)$ forms a subgraph of $\mathcal{C}_R(G)$.

Interestingly, the list variant for our reconfiguration problem is equivalent to the non-list one, as in the following theorem.

Theorem 1. COLORING RECONFIGURATION UNDER LIST R-RECOLORABILITY *can be reduced to* COLORING RECONFIGURATION UNDER R-RECOLORABILITY *in time*

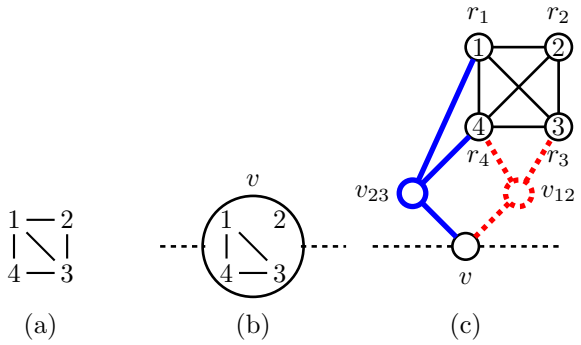


Figure 3.1: (a) Recolorability graph R such that $R_L(v) \subseteq R$ for any vertex $v \in V(G)$, (b) a vertex $v \in V(G)$ with the list recolorability $R_L(v)$, and (c) the vertex v in G', where the (red) thick dotted part corresponds to forbidding the pair of colors 1 and 2 in $E(R) \setminus E(R_L(v))$ and the (blue) thick part corresponds to forbidding the pair of colors 2 and 3 in $E(R) \setminus E(R_L(v))$.

polynomial in $|V(G)|$ and $|V(R)|$, where G is an input graph of the list variant.

Proof. Let G be an input graph for the list variant with a list R-recolorability R_L, and suppose that we are given two list colorings f_0 and f_r of G. Then, we construct a corresponding instance of COLORING RECONFIGURATION UNDER R-RECOLORABILITY; we denote by G' the corresponding graph, and by f'_0 and f'_r the corresponding initial and target k-colorings of G', respectively, where $k = |V(R)|$.

Indeed, we will give a gadget which forbids recoloring a vertex $v \in V(G)$ directly from a color i to another color j for each pair $ij \in E(R) \setminus E(R_L(v))$. Note that, for each color i in $V(R) \setminus V(R_L(v))$, we can add the vertex i to $R_L(v)$ as an isolated vertex (by adding the forbidding gadgets between i and all colors j such that $ij \in E(R)$). Then, since f_0 and f_r are list colorings of G, both $f_0(v) \neq i$ and $f_r(v) \neq i$ hold and hence v is never recolored to the isolated color i.

To construct such a forbidding gadget, we will use a (newly added) clique of size $k = |V(R)|$ such that all vertices are colored with distinct colors. Notice that no vertex in the clique can be recolored to any color, that is, they are frozen vertices

on the k-coloring. We use this property, and construct the corresponding instance, as follows.

Construction.

We first add to G a new clique K_k of k vertices $r_1, r_2, \ldots, r_k$. Then, for each vertex $v \in V(G)$, consider any pair of colors i and j such that $ij \in E(R) \setminus E(R_L(v))$. We add a new vertex v_{ij} to G, and join it with v. In addition, we join v_{ij} with all vertices in $V(K_k) \setminus \{r_i, r_j\}$. (See Figure 3.1(a)–(c) as an example of the application of this procedure.) Let G' be the resulting graph after applying the procedure above to all vertices $v \in V(G)$ and all pairs $ij \in E(R) \setminus E(R_L(v))$. For notational convenience, we denote by V_F the set of all vertices v_{ij} in G' that are newly added for each vertex $v \in V(G)$ and $ij \in E(R) \setminus E(R_L(v))$. We note that $V(G')$ is partitioned into $V(G), V(K_k)$, and V_F. Furthermore, each vertex $v_{ij} \in V_F$ satisfies $N(G', v_{ij}) \cap V(G) = \{v\}$. We always denote by v this unique vertex in $N(G', v_{ij}) \cap V(G)$ for each vertex $v_{ij} \in V_F$. Then, the corresponding k-colorings f'_0 and f'_r of G' are defined as follows: for each $l \in \{0, r\}$ and a vertex $w \in V(G')$,

$$f'_l(w) = \begin{cases} f_l(w) & \text{if } w \in V(G); \\ i & \text{if } w = r_i \in V(K_k); \\ j & \text{if } w = v_{ij} \in V_F \text{ and } f_l(v) = i; \text{ and} \\ i & \text{otherwise, that is, } w = v_{ij} \in V_F \text{ and } f_l(v) \neq i. \end{cases}$$

Then, all vertices $r_1, r_2, \ldots, r_k$ are frozen on both f'_0 and f'_r (indeed, under any recolorability graph). This completes the construction of the corresponding instance. Clearly, this construction can be done in time polynomial in $|V(G)|$ and $k = |V(R)|$.

Correctness.

We now show that $\mathcal{C}_{R_L}(G)$ contains an $(f_0 \to f_r)$-reconfiguration sequence if and only if $\mathcal{C}_R(G')$ contains an $(f'_0 \to f'_r)$-reconfiguration sequence.

We first prove the only-if direction. Consider the first edge in an $(f_0 \to f_r)$-reconfiguration sequence $\mathcal{S}$ on $\mathcal{C}_{R_L}(G)$. Suppose that it corresponds to recoloring a vertex $v \in V(G)$ from the color $f_0(v) = i$ to another color j. Then, the list recolorability $R_L(v)$ of v contains an edge ij. Recall that $R_L(v)$ is a subgraph of R. Therefore, $ij \in E(R)$. Since $\mathcal{S}$ correctly recolors v from i to j, any vertex in $N(G, v)$ is not colored with j in f_0. However, there may exist a vertex w in $N(G', v) \setminus V(G)$ which is colored with j in f'_0. By the construction of G', such a vertex w must be contained in V_F. Since $f'_0(w) = j$, we can assume $w = v_{i'j}$ for some color i'. Notice that $i' \neq i$ holds, because $ij \in E(R_L(v))$. Therefore, we can recolor w from j to another color i' $(\neq i)$, and then recolor v to j. In this way, $\mathcal{C}_R(G')$ contains a path corresponding to the first edge in $\mathcal{S}$ on $\mathcal{C}_{R_L}(G)$. By repeatedly applying the procedure above, we can obtain an coloring f satisfying $f(v) = f'_r(v)$ for any vertex $v \in V(G') \setminus V_F$. Then we can directly recolor each $v \in V_F$ from $f(v)$ to $f'_r(v)$ if $f(v) \neq f'_r(v)$ since no neighbor vertex in $N(G', v) \subseteq V(G) \cup V(K_k)$ is colored neither $f(v)$ nor $f'_r(v)$ and $f(v)f'_r(v) \in E(R(v))$. Therefore there exists an $(f'_0 \to f'_r)$-reconfiguration sequence on $\mathcal{C}_R(G')$.

We then prove the if direction. Consider the first edge in an $(f'_0 \to f'_r)$-reconfiguration sequence $\mathcal{S}'$ on $\mathcal{C}_R(G')$. Suppose that it corresponds to recoloring a vertex $v \in V(G')$ from the color $f'_0(v) = i$ to another color j. Since any vertex in $V(K_k)$ is frozen on f'_0, the vertex v must be in $V(G') \setminus V(K_k) = V(G) \cup V_F$. If $v \in V_F$, then we simply ignore the edge and repeat. We thus consider the case where $v \in V(G)$. If $ij \in E(R_L(v))$, then we can simply recolor v from $f_0(v) = i$ to j; recall that $f_0(w) = f'_0(w)$ for all vertices $w \in V(G)$, and hence no vertex in $N(G, v)$ is colored with j on f_0 if $\mathcal{S}'$ can recolor v from $f'_0(v)$ to j. Finally, we thus consider the case where $v \in V(G)$ and $ij \notin E(R_L(v))$. However, we claim that this case does not

happen, as follows. In this case, G' contains a vertex $v_{ij} \in V_F$ which is adjacent with all vertices in $V(K_k) \setminus \{r_i, r_j\}$. Since every vertex r_l in $V(K_k)$ is frozen on f'_0 and is colored with l, the vertex v_{ij} must be colored with either i or j in any k-coloring which is reachable from f'_0 on $\mathcal{C}_R(G')$. Since v is adjacent with v_{ij}, this contradicts the assumption that $\mathcal{S}'$ recolors v from i to j directly. Therefore, the final case does not happen. In this way, $\mathcal{C}_{R_L}(G)$ contains a path (an edge) corresponding to the first edge in $\mathcal{S}'$ on $\mathcal{C}_R(G')$. By repeatedly applying the procedure above, we can obtain an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_{R_L}(G)$. □

Recall that for any supergraph R' of R, any list R-recolorability is also a list R'-recolorability, therefore we obtain the following corollary:

Corollary 1. *Let R' be an arbitrary supergraph of a recolorability graph R. Then,* Coloring reconfiguration under list R-recolorability *can be reduced to* coloring reconfiguration under R'-recolorability *in time polynomial in $|V(G)|$ and $|V(R')|$, where G is an input graph of the list variant.*

3.2 Recolorability graphs of maximum degree at least four

We first consider the case where a recolorability graph is of maximum degree at least four. We emphasize again that the following theorem holds for an arbitrary recolorability graph R as long as the maximum degree of R is at least four.

Theorem 2. *Let R' be any recolorability graph whose maximum degree is at least four. Then,* coloring reconfiguration under R'-recolorability *is PSPACE-complete.*

Proof. Observe that the problem can be solved in (most conveniently, nondeterministic [24]) polynomial space, and hence it is in PSPACE. Therefore, we show that the problem is PSPACE-hard for such a recolorability graph R'. Notice that, since R' is of maximum degree at least four, R' is a supergraph of a star $K_{1,4}$. Therefore, by Corollary 1 it suffices to prove that the list variant remains PSPACE-hard even for a list R-recolorability such that $R = K_{1,4}$. (See Figure 3.2(a).) To show this, we give a polynomial-time reduction from 4-COLORING RECONFIGURATION, which is known to be PSPACE-complete [8].

Construction.

Let G be an input graph for 4-COLORING RECONFIGURATION, and let f_0 and f_r be a two given 4-colorings of G; we assume the color set $C = \{1, 2, 3, 4\}$. As a corresponding instance of the list variant, we take the same graph G in which the list recolorability $R_L(v)$ of each vertex $v \in V(G)$ is a star $K_{1,4}$ such that its center is a new color 5 and its leaves are the four colors $1, 2, 3$ and 4. Then, both f_0 and f_r are list colorings of the corresponding graph G, and we take the 4-colorings f_0 and f_r as the corresponding list colorings. This completes the construction of the corresponding instance, and hence it is clearly done in polynomial time.

Correctness.

We prove that there exists an $(f_0 \to f_r)$-reconfiguration sequence for 4-COLORING RECONFIGURATION if and only if there exists an $(f_0 \to f_r)$-reconfiguration sequence for the list variant for the list R-recolorability R_L, where $R = K_{1,4}$.

We first prove the only-if direction. Suppose that an $(f_0 \to f_r)$-reconfiguration sequence exists for 4-COLORING RECONFIGURATION, and let $\langle f_0, f_1, \ldots, f_\ell \rangle$ be such

a sequence, where $f_\ell = f_r$. Then, for each $i \in \{0, 1, \ldots, \ell - 1\}$, there exists exactly one vertex $v_{p_i} \in V(G)$ such that $f_i(v_{p_i}) \neq f_{i+1}(v_{p_i})$. In addition, we know that both $f_i(v_{p_i})$ and $f_{i+1}(v_{p_i})$ are contained in $\{1, 2, 3, 4\}$. For each $i \in \{0, 1, \ldots, \ell - 1\}$, we define a list coloring f'_i of G, as follows: for each vertex $v \in V(G)$,

$$f'_i(v) = \begin{cases} 5 & \text{if } v = v_{p_i}; \\ f_i(v) & \text{otherwise.} \end{cases}$$

Recall that, for all vertices $v \in V(G)$, the color 5 is contained in $R_L(v)$ and is adjacent with all the other colors $1, 2, 3, 4$ in $R_L(v)$. (See Figure 3.2(a).) Therefore, the sequence $\langle f_0, f'_0, f_1, f'_1, \ldots, f_{\ell-1}, f'_{\ell-1}, f_\ell \rangle$ of list colorings of G forms an $(f_0 \to f_r)$-reconfiguration sequence for the list variant.

We then prove the if direction. Suppose that an $(f_0 \to f_r)$-reconfiguration sequence exists for the list variant for the list R-recolorability R_L, and let $\langle f_0, f_1, \ldots, f_\ell \rangle$ be such a sequence, where $f_\ell = f_r$. For each $i \in \{0, 1, \ldots, \ell\}$, we define a mapping $f'_i : V(G) \to \{1, 2, 3, 4\}$, as follows: for each vertex $v \in V(G)$,

$$f'_i(v) = \begin{cases} f_i(v) & \text{if } f_i(v) \in \{1, 2, 3, 4\}; \\ f'_{i-1}(v) & \text{if } f_i(v) = 5. \end{cases}$$

Note that, since f_0 and f_r $(= f_\ell)$ are 4-colorings of G, both $f'_0 = f_0$ and $f'_\ell = f_\ell = f_r$ hold. We now claim that the sequence $\langle f'_0, f'_1, \ldots, f'_\ell \rangle$ forms an $(f_0 \to f_r)$-reconfiguration sequence for 4-COLORING RECONFIGURATION (if needed, we delete the redundant 4-colorings) by proving the following (a) and (b):

(a) f'_i is a 4-coloring of G for each $i \in \{0, 1, \ldots, \ell\}$; and

(b) $\left|\{v \in V(G) : f'_{i-1}(v) \neq f'_i(v)\}\right| \leq 1$ for each $i \in \{1, 2, \ldots, \ell\}$.

We first prove the claim (a) above. Let w be any vertex in G such that $f_i(w) = 5$. Then, in the corresponding mapping f'_i, the vertex w is colored with some color $f_q(w) \in \{1, 2, 3, 4\}$ such that $q = \max\{0 \leq j \leq i - 1 : f_j(w) \neq 5\}$. Then, $f_j(w) = 5$

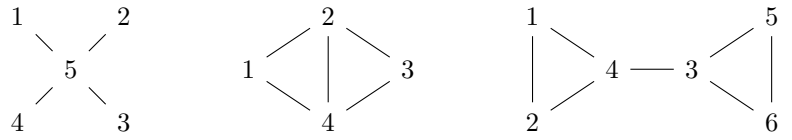


Figure 3.2: (a) Recolorability graph $K_{1,4}$. In our reduction, the star $K_{1,4}$ with center color 5 is the list recolorability of all vertices $v \in V(G)$. (b) Recolorability graph which is a diamond graph. (c) Recolorability graph which is a $2K_3 + e$ graph.

for all $j \in \{q+1, q+2, \ldots, i\}$, and hence every vertex $v \in N(G, w) \cup \{w\}$ is colored with the same color $f_q(v)$, because any recoloring must be made via the color 5. (See Figure 3.2(a).) Since f_q is a list (proper) coloring of G, no vertex in $N(G, w)$ is colored with $f_q(w) = f'_i(w)$. This argument holds for all vertices in G which is colored with 5 in f_i, and hence f'_i is a (proper) 4-coloring of G.

We finally prove the claim (b) above. Let v be the vertex which is recolored between f_{i-1} and f_i, $i \in \{1, 2, \ldots, \ell\}$. If v is recolored from a color in $\{1, 2, 3, 4\}$ to 5, then we have $f'_i = f'_{i-1}$ and hence the claim holds. Note that v stays with the same color $f'_i(v) = f_{i-1}(v)$ until it is recolored to some color in $\{1, 2, 3, 4\}$. Therefore, the claim holds also for the case where v is recolored from a color in $\{1, 2, 3, 4, 5\}$ to another color in $\{1, 2, 3, 4\}$, because only v is recolored between f_{i-1} and f_i. □

3.3 Recolorability graphs with more than one cycle

In this subsection, we consider the case where a recolorability graph R having more than one cycle. Our result is expressed as follows:

Theorem 3. *Let R be a recolorability graph such that $|E(R)| > |V(R)|$. Then* COLORING RECONFIGURATION UNDER R-RECOLORABILITY *is PSPACE-complete.*

By Theorem 1, it suffices to prove that the list variant remains PSPACE-hard

for a list R-recolorability, such that $|E(R)| > |V(R)|$. We first characterize the structure of R by two small graphs: A graph is called a *diamond* graph if it can be obtained by deleting exactly one edge from a complete graph K_4 of size four. (See Figure 3.2(b)); a $2K_3 + e$ graph is a graph obtained by adding exactly one edge to disjoint union of two triangles K_3. (See Figure 3.2(c).) The following lemma shows that any graph R with $|E(R)| > |V(R)|$ can be characterized by the maximum degree of R and these two small graphs.

Lemma 1. *Let R be a connected graph such that $|E(R)| > |V(R)|$. Then, R satisfies at least one of the following statements*:

(a) *R has a vertex whose degree is at least four*;

(b) *R is a supergraph of some subdivision of a diamond graph*; *and*

(c) *R is a supergraph of some subdivision of a $2K_3 + e$ graph.*

Proof. We first remove vertices of degree one and their incident edges from R repeatedly until all the vertices of the resulting graph are of degree at least two. From now on, we denote the resulting graph by R. Observe that R is connected and satisfies $|E(R)| > |V(R)|$, because we delete only degree-one vertices and the same number of edges. It suffices to prove that such a graph R satisfies one of the three properties (a)–(c).

Since all vertices of R are of degree at least two, R has a cycle C. Moreover, since $|E(R)| > |V(R)|$, C has a vertex v of degree at least three. Then we traverse vertices in $V(R)$ by the depth-first search which starts from v and secondary visit any vertex in $N(R, v) \setminus N(C, v)$, until we reach a vertex that is already visited or is contained in the cycle C. Since R has no degree-one vertex, there are the following three cases as the result of the search:

- If the search reaches v, then the degree of v is at least four; the case (a) holds.
- If the search reaches a vertex in $V(C) \setminus \{v\}$, then R contains a subdivision of diamond; the case (b) holds.
- If the search reaches an already visited vertex which is not in $V(C)$, then R contains a subdivision of $2K_3 + e$; the case (c) holds.

Thus, the lemma holds. □

If Lemma 1(a) holds for the recolorability graph R, then COLORING RECONFIGURATION UNDER R-RECOLORABILITY is PSPACE-complete by Theorem 4.4. Therefore it suffices to prove the PSPACE-completeness for the cases (b) and (c), i.e., we will show that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is PSPACE-complete if the recolorability graph R is any subdivision of diamond or $2K_3 + e$.

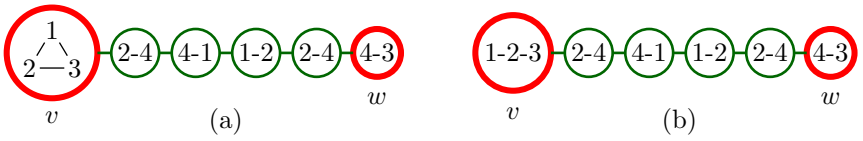


Figure 3.3: (a) An example of $(2, 4)$-forbidding path from v to w and, (b) an example of $(2, 4)$-forbidding path from v to w with a recolorability constraint

In the rest of section we use the notion of (a, b)-forbidding paths which was introduced in [8]. A path graph from v to w and its list is called (a, b)-*forbidding path from* v *to* w if the color combination (a, b) can not be assigned to the vertices u and v, respectively at the same time. In forbidding path, any other color combinations are called *admissible*. Figure 3.3(a) shows an example of a forbidding path. We cannot assign the colors 2 and 3 to the vertices u and v respectively at same time.

Indeed, this notion was defined for conventional coloring reconfiguration problem without recolorability. In our situation, besides forbidding the inadmissible pairs, we also have to consider the structure of the recolorability. Two admissible pairs (x, y) and (x', y') are called *adjacent* if $x = x'$ and $yy' \in E(R)$, of $xx' \in E(R)$ and $y = y'$. We must ensure that two colorings f, f' corresponding two adjacent admissible pairs $(x, y), (x', y')$ are reachable, that is, there is a sequence $f_0, f_1, \ldots, f_t$ such that $f_0 = f$, $f_t = f'$ and for some $i \in \{0, 1, \ldots, t-1\}$, colorings $f_0, f_1, \ldots, f_i$ correspond to the admissible pair (x, y), and the rest colorings $f_{i+1}, f_{i+2}, \ldots, f_t$ correspond to the admissible pair (x', y'). To deal with such a situation the conventional definition of the forbidding path is insufficient. For our purpose we give generalized definition of forbidding path for coloring reconfiguration with the recolorability constraint. A path graph G from u to w with its list R-recolorability R_L are called (a, b)-*forbidding path from* u *to* v if the following properties hold:

(a) There exists no list coloring f of G such that $f(u) = a$ and $f(v) = b$.

(b) For any admissible pair (x, y) the subgraph of R_L-reconfiguration graph induced by the colorings f such that $f(u) = x$ and $f(v) = y$, is connected.

(c) Let (x, y) and (x', y') be two admissible pairs. If $x = x'$ and $yy' \in E(R_L(v))$, or $xx' \in E(R_L(u))$ and $y = y'$, then there exist two colorings f and f' such that $(f(u), f(v)) = (x, y), (f'(u), f'(v)) = (x', y')$ and f, f' are adjacent on $\mathcal{C}_{R_L}(G)$.

In our proof we only deal with (a, b)-forbidding paths from u to v where the list recolorability of u, v are paths of length one or two, as shown in Figure 3.3(b). Therefore we often use a notation like "$(2, 4)$-forbidding path from 1-2-3 to 4-3" where "1-2-3" means some vertex whose list recolorability is a path consists of the colors $1, 2, 3$.

A typical (a, b)-forbidding path from u to v consists of a path $u, w_1, w_2, \ldots, w_p, v$ whose all intermediate vertices have list recolorability $R_L(w_i)$ of size two such that:

(i) There is a walk $c_1, c_2, \ldots, c_{p+1}$ on R, satisfying $c_i \neq c_{i+2}$ for all $i \in \{1, \ldots, p-1\}$;

(ii) for any $i \in \{1, \ldots, p\}$, $V(R_L(w_i)) = \{c_i, c_{i+1}\}$ and $E(R_L(w_i)) = \{c_i c_{i+1}\}$; and

(iii) $c_1 = a, c_2 \notin V(R_L(u)), c_p \notin V(R_L(v)), c_{p+1} = b$ hold.

Lemma 2. *A path $u, w_1, w_2, \ldots, w_p, v$ and its list recolorability R_L satisfying the above conditions* (i)(ii)(iii) *is an (a, b)-forbidding path from u to v.*

Proof. We show that all of the conditions (a)(b)(c) of the definition of the (a, b)-forbidding path hold for such a path. The proof depends on the induction on the length p of the path $w_1, w_2, \ldots, w_p$.

First consider the case where $p = 1$, then $V(R_L(w_1)) = \{a, b\}$ and $E(R_L(w_1)) = \{ab\}$ hold. Moreover, by the condition (iii), $b \notin R_L(u)$ and $a \notin R_L(v)$. For this case the condition (a) holds since if the colors a, b are assigned to the vertices u, v respectively, the vertex w_1 cannot be colored by neither a or b.

To show that the condition (b) holds for any admissible pair of the case $p = 1$, we consider two cases. Let $(x, y) \in V(R_L(u)) \times V(R_L(v)) \setminus \{(a, b)\}$ be an admissible pair. If $x \neq a$ and $y \neq b$, then the colorings which map (u, v) to (x, y) are limited to only two colorings which map w_1 to a and b respectively. Obviously they are adjacent under R_L. Otherwise, one of the cases $x = a$ and $y = b$ holds, then the color of w_1 uniquely determined. Therefore the condition (b) holds for the case $p = 1$.

Let (x, y) and (x', y') be two admissible pairs such that $x = x'$ and $yy' \in E(R_L(v))$. For the condition (c), we consider two cases: For the case $x \neq a$, two

adjacent colorings f, f' such that $f(w_1) = f'(w_1) = a$ exist for each admissible pair, since y, y' cannot be the color a by the condition (iii). Otherwise we have $x = a$ and then, two adjacent colorings f, f' such that $f(w_1) = f'(w_1) = b$ exist for each admissible pair, since y, y' cannot be b because of the definition of constraint of admissible pair.

For the case $p > 1$, we focus on the subpath $u, w_1, \ldots, w_p$. For notational convenience we denote whole path $u, w_1, \ldots, w_p, v$ by G, and denote the subpath $u, w_1, \ldots, w_p$ by $G \setminus v$. Notice that $G \setminus v$ satisfies the conditions (i)(ii)(iii) therefore is a (a, c_p)-forbidding path from u to w_p by the induction hypothesis. Then the following hold:

(a′) There is no coloring f such that $f(u) = a$ and $f(w_p) = c_p$.

(b′) $\mathcal{C}_{R_L}(G \setminus v)[\{f : f(u) = x, f(w_p) = y\}]$ is connected for any admissible pair (x, y).

(c′) For any two admissible pairs (x, y) and (x', y') such that $x = x'$ and $yy' \in E(R_L(v))$, or $xx' \in E(R_L(u))$ and $y = y'$, there are two coloring f, f' satisfying $f(u) = x, f(v) = y, f'(u) = x', f'(v) = y'$, which are adjacent on $\mathcal{C}_{R_L}(G \setminus v)$.

Consider the condition (a) for the case $p > 1$. If there is a coloring f satisfying $f(u) = a$ and $f(v) = b$, then the color of the vertex w_p must be $f(w_p) = c_p$, this is forbidden by the property (a′).

To show that the condition (b) holds for the case $p > 1$, consider two cases: Let (x, y) be an admissible pair. If a coloring f satisfies $f(u) = a$, then w_p must have the color $f(w_p) = c_{p+1} = b$, and by property (b′), $\mathcal{F}_{a,c_{p+1}} = \mathcal{C}_{R_L}(G \setminus v)[\{f : f(u) = a, f(w_p) = c_{p+1}\}]$ is connected (indeed it consists of single coloring). Therefore

$\mathcal{C}_{R_L}(G)[\{f : f(u) = a, f(v) = y\}]$ is connected for any admissible pair (a, y). Otherwise we have $x = f(u) \neq a$ and we have two possibility of the color of the vertex w_p, $f(w_p) = c_p$ or $f(w_p) = c_{p+1}$. $\mathcal{F}_{x,c_p} = \mathcal{C}_{R_L}(G \setminus v)[\{f : f(u) = x, f(w_p) = c_p\}]$ and $\mathcal{F}_{x,c_{p+1}} = \mathcal{C}_{R_L}(G \setminus v)[\{f : f(u) = x, f(w_p) = c_{p+1}\}]$ are also connected by the property (b′), and by the property (c′) there are two coloring $f_{(x,c_p),(x,c_{p+1})} \in V(\mathcal{F}_{x,p})$ and $f_{(x,c_{p+1}),(x,c_p)} \in V(\mathcal{F}_{x,c_{p+1}})$ which are adjacent on $\mathcal{C}_{R_L}(G \setminus v)$. Therefore $\mathcal{C}_{R_L}(G)[\{f : f(u) = x, f(v) = y\}]$ is connected for any admissible pair (x, y).

Finally we show that the condition (c) holds for the case $p > 1$. Let (x, y) and (x', y') be admissible pairs of G. We have the following cases:

- For the case $y = y'$,
  - if x or x' is a, then y must not be b. For this case two adjacent colorings f, f' such that $f(u) = x, f'(u) = x'$, $f(w_i) = f'(w_i) = c_{i+1}$ for all $i \in \{1, \ldots, p\}$, and $f(v) = f'(v) = y$ exist.
  - if x, x' are not a, there are two adjacent colorings f, f' such that $f(u) = x, f'(u) = x'$, $f(w_i) = f'(w_i) = c_i$ for all $i \in \{1, \ldots, p\}$, and $f(v) = f'(v) = y$ exist.
- For the case $x = x'$,
  - if $x = a$ then y, y' cannot be b. For this case two adjacent colorings f, f' such that $f(u) = f'(u) = a$, $f(w_i) = f'(w_i) = c_{i+1}$ for all $i \in \{1, \ldots, p\}$, and $f(v) = y, f'(v) = y'$ exist.
  - Otherwise, $x \neq a$ then case two adjacent colorings f, f' such that $f(u) = f'(u) = x$, $f(w_i) = f'(w_i) = c_i$ for all $i \in \{1, \ldots, p\}$, and $f(v) = y, f'(v) = y'$ exist.

□

We call such a path (a, b)-*forbidding path from u to v induced by walk* $c_1, c_2, \ldots, c_{p+1}$. Fig. 3.3 shows an example of such $(2, 4)$-forbidding path from 1-2-3 to 2-4 induced by walk $2, 4, 1, 2, 4$, where the recolorability graph R is the diamond shown in Figure 3.2.

In the following lemma we show that the PSPACE-completeness holds for the case of Lemma 1(b).

Lemma 3. *Let R be any recolorability graph which is obtained by subdividing the edges of diamond. Then,* COLORING RECONFIGURATION UNDER LIST R-RECOLORABILITY *is PSPACE-complete.*

Proof. According to Theorem 1, it suffices to prove that the list variant under list R-recolorability remains PSPACE-hard.

A subdivision of diamond consists of two degree three vertices c_0 and c_∞, and three edge disjoint paths X, Y, Z connecting c_0 and c_∞. (See Fig. 3.4(a).)

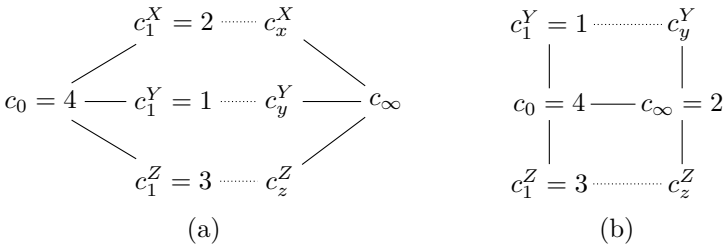


Figure 3.4: Subdivisions of diamond determined by tuple (x, y, z) for the case where (a) $x > 0$, and (b) $x = 0$.

If subdivision R is obtained without subdividing the edge between two degree three vertices, then at most one of the paths X, Y, Z is not exist, and in such a case c_0 and c_∞ are adjacent. (See Fig. 3.4(b).) Without loss of generality, we assume only the path X may not exist. We identify a subdivision of diamond by 3-tuple of integers (x, y, z), where $x \geq 0, y, z \geq 1$ and $x \leq y \leq z$. Three paths

X, Y, Z connecting c_0 and c_∞ are consists of x, y, z vertices respectively. We denote the vertices of path X by $c_1^X, c_2^X, \ldots, c_x^X$ along the path. c_1^X, c_x^X are adjacent to c_0, c_∞ respectively. $c_1^Y, \ldots, c_y^Y$ and $c_1^Z, \ldots, c_z^Z$ are defined analogously. In the case of $x = 0$, the path X is removed and c_0 and c_∞ are adjacent. For instance, diamond is determined by 3-tuple $(0, 1, 1)$.

If R is determined by 3-tuple (x, y, z), we assume $c_1^X = 2, c_1^Y = 1, c_1^Z = 3$, and $c_0 = 4$ for the case $x > 0$, otherwise we let $c_\infty = 2$ by appropriate renaming of colors. Notice that the color 4 is of degree three and the colors $1, 2, 3$ are adjacent to 4.

To show PSPACE-completeness, we give a polynomial-time reduction from NONDETERMINISTIC CONSTRAINT LOGIC (NCL, for short) [16].

Nondeterministic constraint logic.

An NCL "machine" is specified by an undirected graph together with an assignment of weights from $\{1, 2\}$ to each edge of the graph. An (*NCL*) *configuration* of this machine is an orientation (direction) of the edges such that the sum of weights of in-coming arcs at each vertex is at least two. Fig. 3.5(a) illustrates a configuration of an NCL machine, where each weight-2 edge is depicted by a thick (blue) line and each weight-1 edge by a thin (orange) line. Then, two NCL configurations are *adjacent* if they differ in a single edge direction. Given an NCL machine and its two configurations, it is known to be PSPACE-complete to determine whether there exists a sequence of adjacent NCL configurations which transforms one into the other [16].

In fact, the problem remains PSPACE-complete even for AND/OR *constraint graphs*, which consist only of two types of vertices, called "NCL AND vertices" and "NCL OR vertices." A vertex of degree three is called an *NCL* AND *vertex* if its

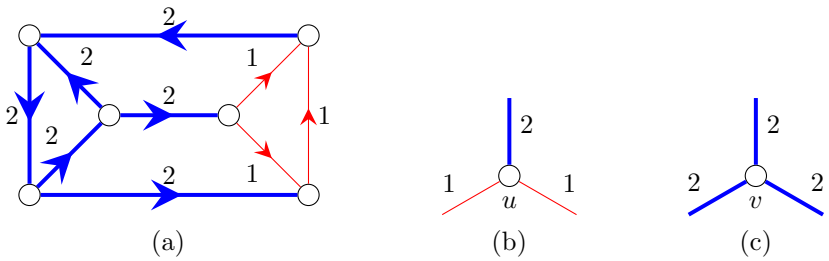


Figure 3.5: (a) A configuration of an NCL machine, (b) NCL AND vertex u, and (c) NCL OR vertex v.

three incident edges have weights 1, 1 and 2. (See Figure 3.5(b).) An NCL AND vertex u behaves as a logical AND, in the following sense: the weight-2 edge can be directed outward for u if and only if both two weight-1 edges are directed inward for u. Note that, however, the weight-2 edge is not necessarily directed outward even when both weight-1 edges are directed inward. A vertex of degree three is called an *NCL* OR *vertex* if its three incident edges have weights 2, 2 and 2. (See Figure 3.5(c).) An NCL OR vertex v behaves as a logical OR: one of the three edges can be directed outward for v if and only if at least one of the other two edges is directed inward for v. It should be noted that, although it is natural to think of NCL AND/OR vertices as having inputs and outputs, there is nothing enforcing this interpretation; especially for NCL OR vertices, the choice of input and output is entirely arbitrary because an NCL OR vertex is symmetric. For example, the NCL machine in Figure 3.5(a) is an AND/OR constraint graph. From now on, we call an AND/OR constraint graph simply an *NCL machine*, and call an edge in an NCL machine an *NCL edge*.

Gadgets.

We first subdivide every NCL edge vw into a path $vv'w'w$ of length three by adding two new vertices v' and w'; the newly added vertices v' and w' are called *connectors* for v and w, respectively. (See Figure 3.6(a) and (b).) We call the edge

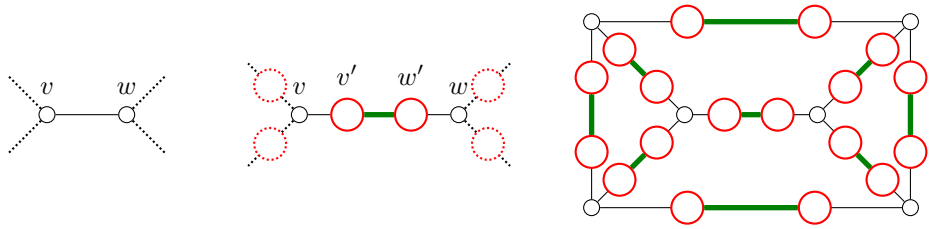


Figure 3.6: (a) An NCL edge vw, (b) its subdivision into a path $vv'w'w$, and (c) the resulting graph which corresponds to the NCL machine in Figure 3.5(a), where each connector is depicted by a (red) large circle and each link edge by a thick (green) line.

$v'w'$ a *link edge* between two NCL vertices v and w, and call the edges vv' and ww' *NCL one-third edges* for v and w, respectively. Notice that every vertex in the resulting graph belongs to exactly one of stars $K_{1,3}$ such that the center v of each $K_{1,3}$ corresponds to an NCL AND/OR vertex and the three leaves are connectors for v. Furthermore, these stars are all mutually disjoint, and joined together by link edges. (See Figure 3.6(c) as an example.)

Therefore, our reduction involves constructing three types of gadgets which correspond to link edges and stars of NCL AND/OR vertices. In our gadgets, all connectors v' for NCL AND/OR vertices v have the same list recolorability $L_R(v')$ such that $V(L_R(v')) = \{2, 4\}$ and $E(L_R(v')) = \{24\}$. Then, in our reduction, assigning the color 4 to v' always corresponds to directing the NCL one-third edge vv' from v' to v (i.e., the inward direction for v), while assigning the color 2 to v' always corresponds to directing vv' from v to v' (i.e., the outward direction for v).

(i) Link edge gadget.

Let $v'w'$ be a link edge, where v' and w' are connectors for NCL AND/OR vertices v and w, respectively. Our link edge gadget $G_{v'w'}$ is a $(4, 4)$-forbidding path from 2-4 to 4-2 induced by walk $c_0, c_1^Y, c_2^Y, \ldots, c_y^Y, c_\infty, c_z^Z, c_{z-1}^Z, \ldots, c_1^Z, c_0$. Fig. 3.7(a) illustrates

an overview of link edge gadget, which is a $(4, 4)$-forbidding path from 2-4 to 4-2. Fig. 3.7(b) is an instantiation of link edge gadget for the case R is diamond.

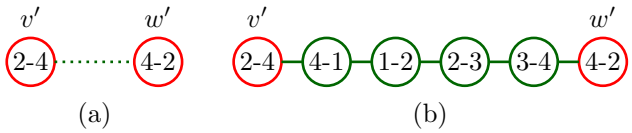


Figure 3.7: (a) An overview of link edge gadget. (b) Link edge gadget for the case R is diamond.

If we assign 4 to v' (the inward direction for v), then w' must be colored with 2 (the outward direction for w); conversely, v' must be colored with 2 if we assign 4 to w'. In particular, the gadget must forbid a list coloring which assigns 4 to both v' and w' (the inward directions for both v and w), because such a list coloring corresponds to the direction which contributes to both v and w illegally. On the other hand, assigning 2 to both v' and w' (the outward directions for both v and w) corresponds to the *neutral* orientation of the NCL edge vw which contributes to neither v nor w, and hence we simply do not care such an orientation.

Fig. 3.8(c) illustrates an example of the R_L-reconfiguration graph $\mathcal{C}_{R_L}(G_{v'w'})$ on the link edge gadget $G_{v'w'}$ where R is diamond. Each rectangle corresponds to a node of $\mathcal{C}_{R_L}(G_{v'w'})$, that is, a list coloring of $G_{v'w'}$, where the underlined bold number represents the color assigned to the vertex. Then, $\mathcal{C}_{R_L}(G_{v'w'})$ is connected, and there is no list coloring which assigns 4 to both v' and w', as claimed above. Furthermore, the reversal of the NCL edge vw can be simulated by the path on $\mathcal{C}_{R_L}(G_{v'w'})$ via the neutral orientation of vw, as illustrated in Figure 3.8(c). Thus, this gadget works correctly as a link edge.

(ii) AND gadget. Our AND gadget G_{and} for each NCL AND vertex v has three vertices v_a, v_b, v_c having the same list recolorability 2-4, which correspond to the

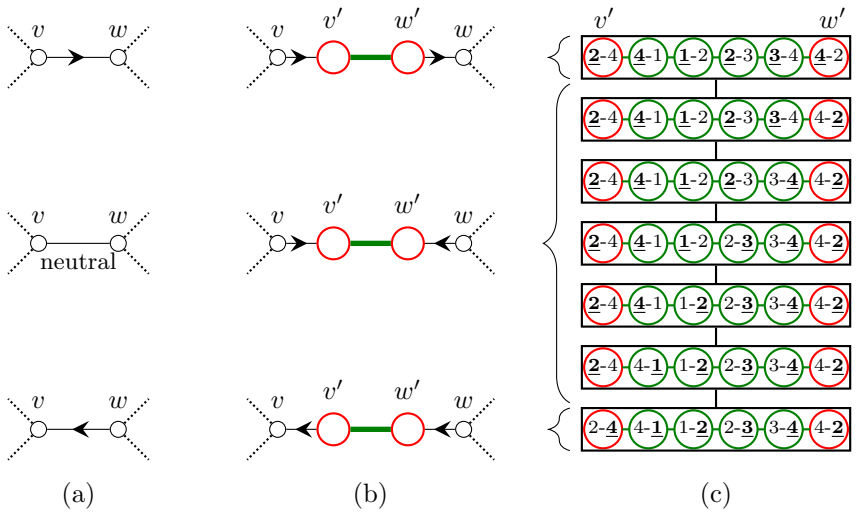


Figure 3.8: (a) Three orientations of an NCL edge vw, (b) their corresponding orientations of the NCL one-third edges vv' and ww', and (c) all list colorings of the link edge gadget $G_{v'w'}$ in the R_L-reconfiguration graph $\mathcal{C}_{R_L}(G_{v'w'})$.

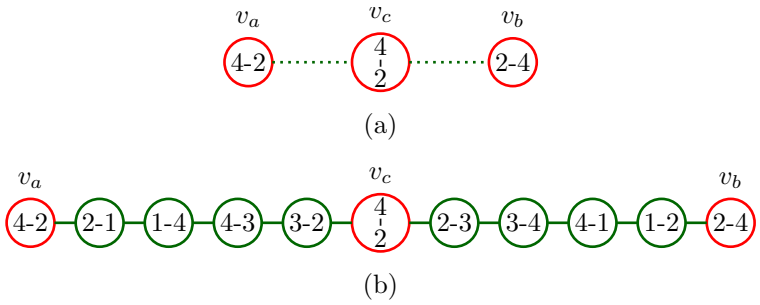


Figure 3.9: (a) An overview of AND gadget. (b) G_{and} for the case R is diamond.

three connectors for v. v_a, v_c and v_c, v_b are joined by $(2,2)$-forbidding path from 4-2 to 2-4 induced by walk W, where W is $(c_1^X, c_2^X, \ldots, c_x^X,)c_\infty, c_y^Y, c_{y-1}^Y, \ldots, c_1^Y$, $c_0, c_1^Z, c_2^Z, \ldots, c_z^Z, c_\infty, c_x^X, c_{x-1}^X, \ldots, c_1^X$, where the parenthesized part is omitted if $x = 0$.

Fig. 3.9(a) illustrates an overview of AND gadget, where two $(2,2)$-forbidding paths from 4-2 to 2-4 are indicated by dotted lines. Fig. 3.9(b) is an instantiation of AND gadget for the case where R is diamond.

In the figure, the connectors v_a and v_b come from the two weight-1 NCL edges,

while the connector v_c comes from the weight-2 NCL edge. We now explain this gadget works as an NCL AND vertex. The AND gadget must forbid the case where all the connectors v_a, v_b and v_c are colored with 2 at the same time (i.e., all NCL one-third edges vv_a, vv_b and vv_c take the outward direction for v). In addition, the gadget must simulate the following situation: v_c can be colored with 2 (i.e., the weight-2 edge vv_c can take the outward direction for v) only when both v_a and v_b are colored with 4 at the same time (i.e., both the weight-1 edges vv_a and vv_b take the inward direction for v).

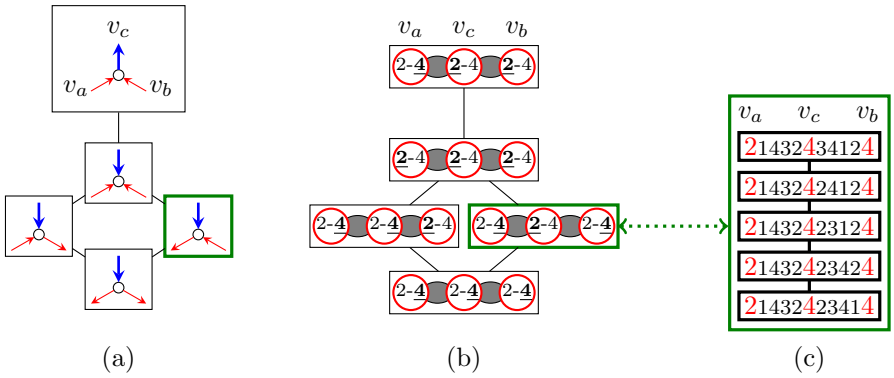
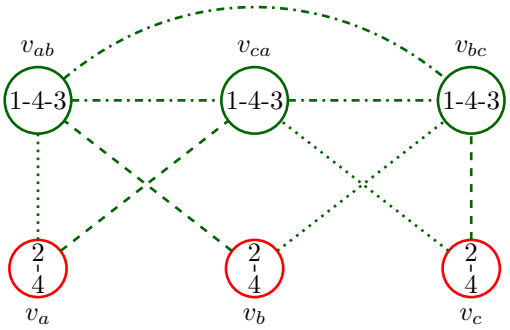


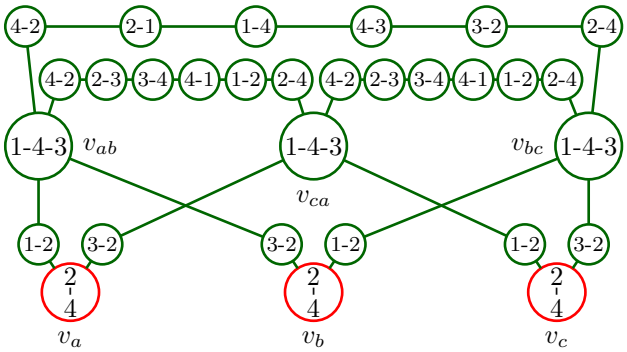
Figure 3.10: (a) All feasible orientations of the three NCL one-third edges incident to an NCL AND vertex together with their adjacency, (b) image of R_L-reconfiguration graph $\mathcal{C}_{R_L}(G_{\mathsf{and}})$ on the AND gadget G_{and}, and (c) the inside of the rightmost (green) thick box in the image which corresponds to assigning the colors 2, 4 and 4 to v_a, v_c and v_b, respectively, where we simply write the colors assigned to G_{and} by a sequence of colors.

Fig. 3.10(a) illustrates all feasible orientations of the three NCL one-third edges vv_a, vv_b and vv_c, whose corresponding assignments of colors to the connectors are depicted in Figure 3.10(b). Due to the space limitation, in Figure 3.10(b), we only indicate the colors assigned to v_a, v_c and v_b. As an example, for the case R is diamond we show all list (proper) colorings of G_{and} that assign the colors 2, 4 and 4 to v_a, v_c and v_b, respectively. Then, as illustrated in Figure 3.10(c), these list colorings

are "internally connected," that is, any two list colorings are reconfigurable with each other without recoloring any connector of G_{and}. This property is due to the condition (b) of the definition of forbidding path. Furthermore, this gadget preserves the "external adjacency" in the following sense: if we contract the list colorings in $\mathcal{C}_{R_L}(G_{\mathsf{and}})$ having the same color assignments to the connectors into a single vertex, then the resulting graph is exactly the graph depicted in Figure 3.10(a) and (b). This property is due to the condition (c) of the definition of forbidding path. Therefore, we can conclude that our AND gadget correctly works as an NCL AND vertex.



(a)



(b)

Figure 3.11: (a) An overview of out OR gadget. (b) An instantiation of OR gadget for the case R is diamond.

(iii) OR gadget.

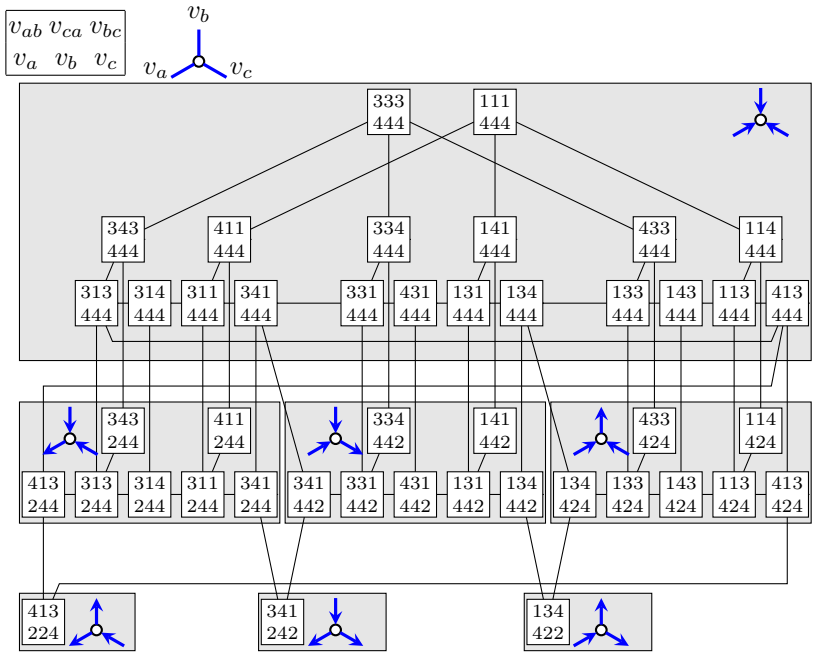


Figure 3.12: Outline of R_L-reconfiguration graph $\mathcal{C}_{R_L}(G_{\mathsf{or}})$ on the OR gadget G_{or}.

Fig. 3.11(a) illustrates an overview of our OR gadget G_{or} for each NCL OR vertex v, where v_a, v_b and v_c correspond to the three connectors for v.

Our OR gadget consists of six vertices and nine forbidding paths. Three vertices v_{ab}, v_{bc}, v_{ca} have the same list recolorability 1-4-3. G_{or} has three types of forbidding paths:

- $(1, 2)$-forbidding path from 1-4-3 to 2-4 (denoted by dotted line);
- $(3, 2)$-forbidding path from 1-4-3 to 2-4 (denoted by dashed line); and
- $(4, 4)$-forbidding path from 1-4-3 to 1-4-3 (denoted by dotted dashed line)

Each forbidding path can be induced by the walk on R as follows:

- $(1, 2)$-forbidding path from 1-4-3 to 2-4 can be induced by the walk $c_1^Y, c_2^Y, \ldots, c_y^Y, c_\infty(, c_x^X, c_{x-1}^X, \ldots, c_1^X)$;

- $(3,2)$-forbidding path from 1-4-3 to 2-4 can be induced by the walk $c_1^Z, c_2^Z, \ldots, c_z^Z, c_\infty(, c_x^X, c_{x-1}^X, \ldots, c_1^X)$;

- $(4,4)$-forbidding path from 1-4-3 to 1-4-3 can be induced by the walk $c_0, (c_1^X, c_2^X, \ldots, c_x^X,)c_\infty, c_y^Y, c_{y-1}^Y, \ldots, c_1^Y, c_0, c_1^Z, c_2^Z, \ldots, c_z^Z, c_\infty(, c_x^X, c_{x-1}^X, \ldots, c_1^X), c_0,$

where parenthesized parts are omitted if $x = 0$. Figure 3.11(b) shows an instantiation of G_{or} for the case R is diamond.

We now explain this gadget works as an NCL OR vertex. For each NCL OR vertex v, it suffices that at least one of the three NCL edges take the inward direction for v. Thus, the OR gadget must forbid only the case where all the connectors v_a, v_b and v_c are colored with 2 at the same time. Indeed, our gadget in Figure 3.11 forbids such the case, because otherwise all three vertices v_{ab}, v_{bc} and v_{ca} must be colored with 4 and such a case is forbidden by $(4,4)$-forbidding paths between v_{ab}, v_{bc}, v_{ca}.

Fig. 3.12 illustrates (an outline of) the R_L-reconfiguration graph $\mathcal{C}_{R_L}(G_{\mathsf{or}})$ on the OR gadget G_{or}, where we indicate only the colors assigned to the vertices v_{ab}, v_{bc}, v_{ca}, v_a, v_b and v_c. All list colorings of G_{or} in each shaded box assign the same three colors to the three connectors v_a, v_b and v_c, and hence they correspond to the same orientations of the three NCL one-third edges vv_a, vv_b and vv_c.

Reduction and its correctness.

As we have mentioned above, we first subdivide every NCL edge vw into a path $vv'w'w$ of length three by adding two connectors v' and w'. (See Figure 3.6.) Then, we replace each of link edges and NCL AND/OR vertices with its corresponding gadget; let G be the resulting graph. In addition, we construct two list colorings of G which correspond to two given configurations C_0 and C_r of the NCL machine.

Note that there are (in general, exponentially) many list colorings which correspond to the same NCL configuration. However, by the construction of the three gadgets, no two distinct NCL configurations correspond to the same list coloring of G. We thus choose any two list colorings f_0 and f_r of G which correspond to C_0 and C_r, respectively. This completes the construction of the corresponding instance for the list variant under list R-recolorability. Clearly, the construction can be done in polynomial time.

We now prove that there exists a desired sequence of NCL configurations if and only if there exists an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_{R_L}(G)$.

We first prove the only-if direction. Suppose that there exists a desired sequence of NCL configurations, and consider any two adjacent NCL configurations C_{i-1} and C_i in the sequence. Then, only one NCL edge vw changes its orientation between C_{i-1} and C_i. Notice that, since both C_{i-1} and C_i are feasible NCL configurations, the NCL AND/OR vertices v and w have enough in-coming arcs even without vw. Therefore, we can simulate this reversal by the reconfiguration sequence of list colorings in Figure 3.8(c) which passes through the neutral orientation of vw as illustrated in Figure 3.8(a). Recall that both AND and OR gadgets are internally connected, and preserve the external adjacency. Therefore, any reversal of an NCL edge can be simulated by a reconfiguration sequence of list colorings of G, and hence there exists an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_{R_L}(G)$.

We finally prove the if direction. Suppose that there exists an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_{R_L}(G)$. Notice that, by the construction of gadgets, any list coloring of G corresponds to a feasible NCL configuration such that some NCL edges may take the neutral orientation. Pick the first index i in the $(f_0 \to f_r)$-reconfiguration sequence $\langle f_0, f_1, \ldots, f_\ell \rangle$ which corresponds to changing the direction

of an NCL edge vw from the neutral orientation to another one. Then, we regard that all list colorings $f_0, f_1, \ldots, f_{i-1}$ correspond to the initial NCL configuration C_0, and that f_i corresponds to the NCL configuration C_1 such that only the NCL edge vw changes its orientation from C_0; note that no NCL edge takes the neutral orientation in C_1. We repeat this process, and obtain a sequence of feasible NCL configurations $C_0, C_1, \ldots, C_{\ell'}$ such that $C_{\ell'} = C_r$ and no NCL edge takes the neutral orientation. Then, $\langle C_0, C_1, \ldots, C_{\ell'} \rangle$ forms the desired sequence of NCL configurations. □

Lemma 1(c) is proved in similar way.

Lemma 4. *Let R be a graph obtained by subdividing $2K_3$+e graph. Then,* COLORING RECONFIGURATION UNDER LIST R-RECOLORABILITY *is PSPACE-complete.*

Proof. As same as the proof of Lemma 3, we show the PSPACE-completeness by the reduction from NCL. Since the proof is similar to the proof of Lemma 3, we only show how to construct link edge,AND,OR gadgets.

A subdivision of $2K_3 + e$ graph consists of two vertex disjoint cycles and a path joining them. Similarly to the graph R in the proof of Lemma 3, we determine the graph R by 3-tuple of integers (x, y, z), where $x \geq y \geq 2, z \geq 2$. R has two cycles which consist of $x + 1$ and $y + 1$ colors respectively, We let the colors on the cycle consists of $x+1$ (respectively, $y+1$) colors $c_1^Z, c_1^X, c_2^X, \ldots, c_x^X$ (respectively, $c_z^Z, c_1^Y, c_2^Y, \ldots, c_y^Y$) along the cycle. The two cycles are joined by a path $c_1^Z, c_2^Z, \ldots, c_z^Z$. Fig. 3.13 illustrates an overview of R.

By appropriate renaming of color labels, we assume $c_1^X = 1, c_x^X = 2, c_2^Z = 3, c_1^Z = 4$. Then we can construct list recolorabilities 2-4 and 1-4-3 used in the construction of link edge, AND, OR gadgets in the proof of Lemma 3. All we have to do is to construct forbidding paths which constitute link edge, AND, OR gadgets. In the

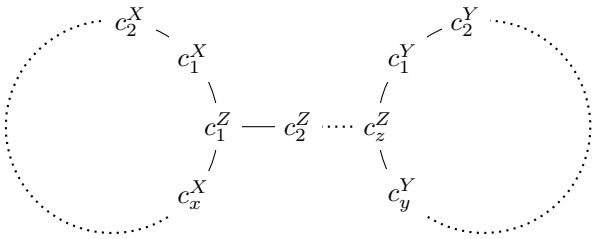


Figure 3.13: An overview of R.

proof of Lemma 3 we introduced five types of forbidding paths:

- $(4, 4)$-forbidding path from 2-4 to 4-2;
- $(2, 2)$-forbidding path from 4-2 to 2-4;
- $(2, 1)$-forbidding path from 2-4 to 1-4-3;
- $(2, 3)$-forbidding path from 2-4 to 1-4-3; and
- $(4, 4)$-forbidding path from 1-4-3 to 1-4-3

to construct three types of gadgets. All of them can be induced by walks on R as follows:

- $(4, 4)$-forbidding path from 2-4 to 4-2 can be induced by the walk
  $c_1^Z, c_2^Z, \ldots, c_z^Z, c_1^Y, c_2^Y, \ldots, c_y^Y, c_z^Z, c_{z-1}^Z, \ldots, c_1^Z$.
- $(2, 2)$-forbidding path from 4-2 to 2-4 can be induced by the walk
  $c_x^X, c_{x-1}^X, \ldots, c_1^X, c_1^Z, c_2^Z, \ldots, c_z^Z, c_1^Y, c_2^Y, \ldots, c_y^Y, c_z^Z, c_{z-1}^Z, \ldots,$
  $c_1^Z, c_1^X, c_2^X, \ldots, c_x^X$.
- $(2, 1)$-forbidding path from 2-4 to 1-4-3 can be induced by the walk
  $c_x^X, c_{x-1}^X, \ldots, c_1^X, c_1^Z, c_2^Z, \ldots, c_z^Z, c_1^Y, c_2^Y, \ldots, c_y^Y, c_z^Z, c_{z-1}^Z, \ldots,$
  $c_1^Z, c_x^X, c_{x-1}^X, \ldots, c_1^X$.

- $(2,3)$-forbidding path from 2-4 to 1-4-3 can be induced by the walk $c_x^X, c_{x-1}^X, \ldots, c_1^X, c_1^Z, c_2^Z, \ldots, c_z^Z, c_1^Y, c_2^Y, \ldots, c_y^Y, c_z^Z, c_{z-1}^Z, \ldots, c_2^Z$.

- $(4,4)$-forbidding path from 1-4-3 to 1-4-3 can be induced by the walk $c_1^Z, c_x^X, c_{x-1}^X, \ldots, c_1^X, c_1^Z, c_2^Z, \ldots, c_z^Z, c_1^Y, c_2^Y, \ldots, c_y^Y, c_z^Z, c_{z-1}^Z, \ldots, c_1^Z$, $c_1^X, c_2^X, \ldots, c_x^X, c_1^Z$.

□

Now we can complete the proof of Theorem 3.

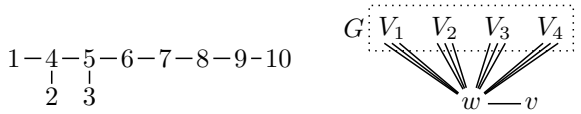
Theorem 3. R satisfies at least one of three conditions of Lemma 1. If the condition (a) holds, then the problem is PSPACE-complete by Theorem 4.4. Otherwise, if the condition (b) holds, then the problem is PSPACE-complete for some recolorability graph $R' \subseteq R$ by Lemma 3. Finally, if the condition (c) holds, then the problem is PSPACE-complete for some recolorability graph $R' \subseteq R$ by Lemma 4. For each case, by Corollary 1 the problem for the recolorability graph R is PSPACE-complete. □

3.4 Recolorability graph which is a binary tree

In this section we show that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is NP-hard even if the recolorability graph R is a binary tree. Our claim is stated as follows:

Theorem 4. *There is a binary tree R such that* COLORING RECONFIGURATION UNDER R-RECOLORABILITY *is NP-hard.*

Proof. We prove by reduction from a known NP-complete problem a 3-coloring of planar graphs [13]. It is well-known that every planar graph is 4-colorable [1], furthermore a 4-coloring of a planar graph can be computed in polynomial time [23].



(a) (b)

Figure 3.14: (a) R which is a binary tree, and (b) a graph G'.

Let G be a given planar graph and we first give a 4-coloring $f_F : V(G) \to \{1, 2, 3, 4\}$ of G in polynomial-time. Then we separate the vertex set $V(G)$ into four subsets V_1, V_2, V_3, V_4, where $V_i = \{v \in V(G) \mid f_F(v) = i\}$.

We give an recolorability graph R defined as follows (see also Figure 3.14(a)):

$$V(R) = \{1, 2, \ldots, 10\}$$

$$E(R) = \{\{1, 4\}, \{2, 4\}, \{3, 5\}, \{4, 5\}, \{5, 6\}, \{6, 7\}, \{7, 8\}, \{8, 9\}, \{9, 10\}\}$$

Then we construct an instance of COLORING RECONFIGURATION UNDER LIST R-RECOLORABILITY. To prove the claim it suffices to prove COLORING RECONFIGURATION UNDER LIST R-RECOLORABILITY is NP-hard by Theorem 1. We make a graph G' as follows: we add a vertex v to G as an isolated vertex, and then add w and join it with all other vertices, as depicted in Figure 3.14. We define a list R-recolorability R_L:

$$R_L(u) = \begin{cases} R[1, 2, 3, 4, 5, 6, 7, 8] & \text{if } u \in V(G) \\ R[4, 5, 6, 7, 8, 9] & \text{if } u = w \\ R[3, 5, 6, 7, 8, 9, 10] & \text{if } u = v \end{cases}$$

Then we define initial and target colorings f_0, f_r:

$$f_0(u) = \begin{cases} i + 4 & \text{if } u \in V_i \\ 9 & \text{if } u = w \\ 10 & \text{if } u = v \end{cases} \quad \text{and} \quad f_r(u) = \begin{cases} i + 4 & \text{if } u \in V_i \\ 9 & \text{if } u = w \\ 3 & \text{if } u = v \end{cases}$$

We show there is $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_{R_L}(G')$ if and only if there is a 3-coloring of G. We first prove the if part. Let f_T be a 3-coloring of G.

First we recolor all vertices $u \in V_1$ from 5 to $f_T(u) \in \{1, 2, 3\}$. This recoloring can be done since there is no vertex colored $1, 2, 3, 4$, and since all vertices in V_1 are not adjacent. Next, we recolor all vertices $u \in V_2$ from 6 to $f_T(u) \in \{1, 2, 3\}$. This can be done since there is no vertex colored $4, 5$, and the vertices colored $f_T(u)$ are in V_1 and are not adjacent to u since f_T is a coloring. We recolor the vertices u in V_3, V_4 to $f_T(u)$ in similar way. Then there is no vertex colored $4, 5, 6, 7, 8$, therefore we can recolor the vertex w from 9 to 4, and then recolor the vertex v from 10 to 3. Now all vertices except for v (already has its target color) can be reach their target colors, equivalently, initial colors, by backtracking their recoloring process.

We then prove the only-if part. Let $\mathcal{S}$ be an $(f_0 \to f_r)$-reconfiguration sequence. By the structure of list R-recolorability, we can see that the vertex w must be recolored to 4 in order to recolor the vertex v to 3 for the first time. Similarly, the vertex $u \in V(G)$ must be recolored to $1, 2$ or 3 in order to recolor the vertex w to 4 for the first time. Therefore, in the $(f_0 \to f_r)$-reconfiguration sequence $\mathcal{S}$, there is a coloring f before the first recoloring of v from 5 to 3 such that $f|_{V(G)}$ is a 3-coloring of G. □

3.5 Recolorability graphs which are unicyclic graphs

In this section we show the computational hardness of COLORING RECONFIGURATION UNDER R-RECOLORABILITY where R is a unicyclic graph of maximum degree three. In Section 5, we show that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is PSPACE-complete even if the recolorability graph R is an unicyclic graph of maximum degree three. Actually, in the proof of PSPACE-completeness the recolorability graph has four vertices of degree three, therefore

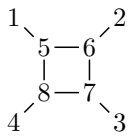


Figure 3.15: The recolorability graph R which is unicyclic.

one may think the problem may be easy for the case where the recolorability graph has less than four degree three vertices. In Section 3.5.2 we contradict it by proving that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is NP-hard even if R is a unicyclic graph having exactly degree one vertex.

3.5.1 PSPACE-completeness

In this section we show that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is PSPACE-complete even if R is an unicyclic graph. Our statement is as follows:

Theorem 5. *There is a unicyclic graph R such that* COLORING RECONFIGURATION UNDER R-RECOLORABILITY *is PSPACE-complete.*

Proof. We prove by the reduction from 4-COLORING RECONFIGURATION, which is a PSPACE-complete problem [8]. Let G be a graph, and f_0, f_r be initial and target colorings of a given instance of 4-COLORING RECONFIGURATION. Let R be a recolorability graph defined as follows (See also Figure3.15):

$$V(R) = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$E(R) = \{\{1, 5\}, \{2, 6\}, \{3, 7\}, \{4, 8\}, \{5, 6\}, \{6, 7\}, \{7, 8\}, \{8, 5\}\}$$

To prove the claim it suffices to prove that COLORING RECONFIGURATION UNDER LIST R-RECOLORABILITY is PSPACE-complete by Theorem 1. We construct an instance of COLORING RECONFIGURATION UNDER LIST R-RECOLORABILITY. For

each edge $uw \in E(G)$, we add two vertices x, y and join then with all other vertices. We also join x and y themselves. Let G' be the resulting graph:

$$V(G') = V(G) \cup \{x, y\}$$
$$E(G') = E(G) \cup \{xy\} \cup \{xv \mid v \in V(G)\} \cup \{yv \mid v \in V(G)\}$$

We define list R-recolorability L_R of G' as follows:

$$L_R(v) = \begin{cases} R & \text{if } v \in V(G); \\ R[5,6,7,8] & \text{otherwise.} \end{cases}$$

We also define initial and target colorings f'_0, f'_r of G':

$$f'_0(v) = \begin{cases} f_0(v) & \text{if } v \in V(G) \\ 5 & \text{if } v = x \\ 6 & \text{if } v = y \end{cases}$$
$$f'_r(v) = \begin{cases} f_r(v) & \text{if } v \in V(G) \\ 5 & \text{if } v = x \\ 6 & \text{if } v = y \end{cases}$$

Then we show there is an $(f'_0 \to f'_r)$-reconfiguration sequence on $\mathcal{C}_{R_L}(G')$ if and only if $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_4(G)$. We first prove the if part. We show that we can simulate any one step recoloring in 4-COLORING RECONFIGURATION by the constructed instance. For a 4-coloring f of G, we define f' as well as f'_0, f'_r:

$$f'(u) = \begin{cases} f(u) & \text{if } u \in v(g) \\ 5 & \text{if } u = v_e \text{ for some } e \in e(g) \\ 6 & \text{if } u = v'_e \text{ for some } e \in e(g) \end{cases}$$

Let f_1 be a coloring which is adjacent to f_0 in $\mathcal{C}_4(G)$, and let v the unique vertex such that $f_0(v) \neq f_1(v)$. We construct a reconfiguration sequence from f'_0 to f'_1. First, we try to recolor the vertex v on G' from $f'_0(v)$ to $f'_0(v) + 4$. Though there may be an adjacent vertex x or y which is colored $f'_0(v) + 4$, it can be recolored to other color. Then we can recolor the vertex v to $f'_1(v) + 4$ by an appropriate

recoloring process in which vertices x, y may also be recolored. As an example of such a recoloring process, consider a situation where g is current coloring and $g(v) = 4, g(x) = 5, g(y) = 6$, and we want to recolor v to 5. In order to recolor v, we need to recolor the vertex x to 7, and then recolor y to 6. Then v is recolored to $f'_1(v)$. After that vertices x, y can be recolored to $f_1(x) = 5, f'_1(y) = 6$ and we reach the coloring f'_1.

Next, we prove the only-if part. Let $\langle g_0, g_1, \ldots, g_t \rangle$ be an $(f'_0 \to f'_r)$-reconfiguration sequence where $g_0 = f'_0, g_t = f'_r$. We construct a sequence $\langle h_0, h_1, \ldots, h_t \rangle$ of 4-colorings of G as follows:

$$h_0 = g_0|_{V(G)}$$
$$h_{i+1}(v) = \begin{cases} g_{i+1}(v) - 4 & \text{if condition (A) holds;} \\ h_i(v) & \text{otherwise;} \end{cases}$$

where condition (A) is as follows: $g_i(v), g_{i+1}(v) \in \{5, 6, 7, 8\}$ and $g_{i+1}(v) \neq g_i(v)$ and there is no adjacent vertex $w \in V(G)$ of v colored $g_{i+1}(v) - 4$. We first notice that if $g_i(v) \in \{1, 2, 3, 4\}$ then $g_i(v) = h_i(v)$ since such a vertex must have been recolored only between $g_i(v)$ and $g_i(v) + 4$, or have been recolored from a color $c \in \{5, 6, 7, 8\}$ other from $g_i(v) + 4$, when v has no adjacent vertex colored $g_i(v)$. Since h_i and h_{i+1} may differ on exactly one vertex, all we have to do is to show that all h_i are proper 4-colorings of G. We prove it by induction on the index i. If $i = 0$, h_0 is trivially a 4-coloring of G. Otherwise $i > 0$, there are four cases of recoloring:

(a) x or y is recolored

(b) $v \in V(G)$ is recolored from $g_{i-1}(v) \in \{1, 2, 3, 4\}$ to $g_i(v) = g_{i-1} + 4$

(c) $v \in V(G)$ is recolored from $g_{i-1}(v) \in \{5, 6, 7, 8\}$ to $g_i(v) = g_{i-1} - 4$

(d) $v \in V(G)$ is recolored from $g_{i-1}(v) \in \{5, 6, 7, 8\}$ to $g_i(v) \in \{5, 6, 7, 8\}$

Notice that for the cases(a)(b)(c) $h_i = h_{i-1}$ therefore h_i is a 4-coloring of G by induction hypothesis. Therefore we consider the remaining case (d). If there is an adjacent vertex w colored $g_i(v)$, then $h_i = h_{i-1}$ also holds. Otherwise, $h_i(v) = h_i(w)$ may be holds for an adjacent vertex w of v. Such a vertex must have a color in $\{5, 6, 7, 8\}$ since if $g_i(w) \in \{1, 2, 3, 4\}$ then $g_i(w) = h_i(w) = h_i(v)$ must holds. However, if two adjacent vertices v, w have color in $\{5, 6, 7, 8\}$, v can not be recolored between $\{5, 6, 7, 8\}$ since $G'[v, w, x, y]$ is a clique in which each vertex has each color in $\{5, 6, 7, 8\}$, which contradicts the assumption v has been recolored. Therefore no adjacent vertex w satisfies $h_i(w) = h_i(v)$ and h_i is a proper 4-coloring. □

3.5.2 NP-hardness for the case where R has exactly one degree three vertex

In this section we prove that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is still NP-hard even if the recolorability graph R is an unicyclic graph having exactly one vertex of degree three.

Theorem 6. *There is a unicyclic graph R having exactly one vertex of degree three such that* COLORING RECONFIGURATION UNDER R-RECOLORABILITY *is NP-hard.*

Proof. Similar to the proof of Theorem 4, we prove by a reduction from a known NP-complete problem 4-coloring of planar graphs [13]. Let G be a given planar graph. We first give a 4-coloring $f_F : V(G) \to \{1, 2, 3, 4\}$ of G in polynomial-time by a known algorithm [23]. Then we separate $V(G)$ into four subsets $V_i = \{v \in V(G) \mid f_F(v) = i\}$. Let R be a recolorability graph defined as follows (See also

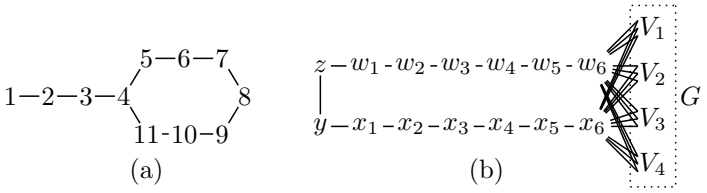


Figure 3.16: (a) The recolorability graph R which is unicyclic graph having exactly one vertex of degree three, and (b) The graph G' which is the constructed instance.

Figure 3.16(a)).

$$V(R) = \{1, 2, \ldots, 11\}$$

$$E(R) = \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,6\},\{6,7\},$$
$$\{7,8\},\{8,9\},\{9,10\},\{10,11\},\{11,4\}\}$$

By Theorem 1, to prove the claim it suffice to prove that COLORING RECONFIGURATION UNDER LIST R-RECOLORABILITY is NP-hard. We construct a graph G' by adding some vertices as follows (See also Figure 3.16(b)):

$$V(G') = V(G) \cup \{w_1, w_2, w_3, w_4, w_5, w_6, x_1, x_2, x_3, x_4, x_5, x_6, y, z\}$$

$$E(G') = E(G) \cup \{w_6 v \mid v \in V(G)\} \cup \{x_6 v \mid v \in V(G)\} \cup$$
$$\{w_1w_2, w_2w_3, w_3w_4, w_4w_5, w_5w_6, x_1x_2, x_2x_3, x_3x_4, x_4x_5, x_5x_6, zw_1, yx_1\}$$

Then we define list R-recolorability R_L of G' as in Figure 3.17. And we define initial and target colorings f_0, f_r as in Figure 3.18

f_0, f_r are differ only on the colors of two vertices y, z.

We show that there is an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_{R_L}(G')$ if and only if there is a 3-coloring of G. We first prove the if part. Let f_T be a 3-coloring of G. Then an $(f_0 \to f_r)$-reconfiguration sequence can be obtained as follows:

1. Recolor all vertices v in V_1 satisfying $f_T(v) = 1$ from $4 \to 3 \to 2$.

$$R_L(v) = \begin{cases} R[2,3,4,5,6,7,9,10,11] & \text{if } v \in V(G) \\ R[3,4,5,6] & \text{if } v = w_1 \\ R[4,5,6,7] & \text{if } v = w_2 \\ R[5,6,7,8] & \text{if } v = w_3 \\ R[6,7,8,9] & \text{if } v = w_4 \\ R[7,8,9,10] & \text{if } v = w_5 \\ R[8,9,10,11] & \text{if } v = w_6 \\ R[3,4,11,10] & \text{if } v = x_1 \\ R[4,11,10,9] & \text{if } v = x_2 \\ R[11,10,9,8] & \text{if } v = x_3 \\ R[10,9,8,7] & \text{if } v = x_4 \\ R[9,8,7,6] & \text{if } v = x_5 \\ R[8,7,6,5] & \text{if } v = x_6 \\ R[1,2,3,4,11] & \text{if } v = y \\ R[1,2,3,4,5] & \text{if } v = z \end{cases}$$

Figure 3.17: The list recolorability R_L.

2. Recolor all vertices v in V_1 satisfying $f_T(v) = 2$ or $f_T(v) = 3$ from $4 \to 11 \to 10 \to 9$.

3. Recolor all vertices v in V_2 satisfying $f_T(v) = 1$ from $5 \to 4 \to 3 \to 2$.

4. Recolor all vertices v in V_2 satisfying $f_T(v) = 2$ or $f_T(v) = 3$ from $5 \to 4 \to 11 \to 10$.

5. Recolor all vertices v in V_3 satisfying $f_T(v) = 1$ from $6 \to 5 \to 4 \to 3 \to 2$.

6. Recolor all vertices v in V_3 satisfying $f_T(v) = 2$ or $f_T(v) = 3$ from $6 \to 5 \to 4 \to 11$.

7. Recolor all vertices v in V_4 satisfying $f_T(v) = 1$ from $7 \to 6 \to 5 \to 4 \to 3 \to 2$.

$$f_0(v) = \begin{cases} 1 & \text{if } v = y \\ 2 & \text{if } v = z \\ 3 & \text{if } v = w_1 \\ 4 & \text{if } v = w_2 \\ 5 & \text{if } v = w_3 \\ 6 & \text{if } v = w_4 \\ 7 & \text{if } v = w_5 \\ 8 & \text{if } v = w_6 \\ 3 & \text{if } v = x_1 \\ 4 & \text{if } v = x_2 \\ 11 & \text{if } v = x_3 \\ 10 & \text{if } v = x_4 \\ 9 & \text{if } v = x_5 \\ 8 & \text{if } v = x_6 \\ 4 & \text{if } v \in V_1 \\ 5 & \text{if } v \in V_2 \\ 6 & \text{if } v \in V_3 \\ 7 & \text{if } v \in V_4 \end{cases} \quad \text{and} \quad f_r(v) = \begin{cases} 2 & \text{if } v = y \\ 1 & \text{if } v = z \\ 3 & \text{if } v = w_1 \\ 4 & \text{if } v = w_2 \\ 5 & \text{if } v = w_3 \\ 6 & \text{if } v = w_4 \\ 7 & \text{if } v = w_5 \\ 8 & \text{if } v = w_6 \\ 3 & \text{if } v = x_1 \\ 4 & \text{if } v = x_2 \\ 11 & \text{if } v = x_3 \\ 10 & \text{if } v = x_4 \\ 9 & \text{if } v = x_5 \\ 8 & \text{if } v = x_6 \\ 4 & \text{if } v \in V_1 \\ 5 & \text{if } v \in V_2 \\ 6 & \text{if } v \in V_3 \\ 7 & \text{if } v \in V_4 \end{cases}.$$

Figure 3.18: The initial and target colorings.

8. Recolor all vertices v in V_3 satisfying $f_T(v) = 2$ or $f_T(v) = 3$ from $11 \to 4 \to 5 \to 6$.

9. Recolor all vertices v in V_2 satisfying $f_T(v) = 2$ or $f_T(v) = 3$ from $10 \to 11 \to 4 \to 5$.

10. Recolor all vertices v in V_1 satisfying $f_T(v) = 2$ or $f_T(v) = 3$ from $9 \to 10 \to 11 \to 4$.

11. Recolor all vertices v in V_1 satisfying $f_T(v) = 2$ from $4 \to 3$.

12. Recolor all vertices v in V_1 satisfying $f_T(v) = 3$ from $4 \to 11 \to 10 \to 9$.

13. Recolor all vertices v in V_2 satisfying $f_T(v) = 2$ from $5 \to 4 \to 3$.

14. Recolor all vertices v in V_2 satisfying $f_T(v) = 3$ from $5 \to 4 \to 11 \to 10$.

15. Recolor all vertices v in V_3 satisfying $f_T(v) = 2$ from $6 \to 5 \to 4 \to 3$.

16. Recolor all vertices v in V_3 satisfying $f_T(v) = 3$ from $6 \to 5 \to 4 \to 11$.

17. Recolor all vertices v in V_4 satisfying $f_T(v) = 2$ from $7 \to 6 \to 5 \to 4 \to 3$.

18. Recolor all vertices v in V_3 satisfying $f_T(v) = 3$ from $11 \to 4 \to 5 \to 6$.

19. Recolor all vertices v in V_2 satisfying $f_T(v) = 3$ from $10 \to 11 \to 4 \to 5$.

20. Recolor all vertices v in V_1 satisfying $f_T(v) = 3$ from $9 \to 10 \to 11 \to 4$.

21. Recolor all vertices v in V_2 satisfying $f_T(v) = 3$ from $5 \to 4$.

22. Recolor all vertices v in V_3 satisfying $f_T(v) = 3$ from $6 \to 5 \to 4$.

23. Recolor all vertices v in V_4 satisfying $f_T(v) = 3$ from $7 \to 6 \to 5 \to 4$.

24. Recolor w_6 from $8 \to 9 \to 10 \to 11$.

25. Recolor w_5 from $7 \to 8 \to 9 \to 10$.

26. Recolor w_4 from $6 \to 7 \to 8 \to 9$.

27. Recolor w_3 from $5 \to 6 \to 7 \to 8$.

28. Recolor w_2 from $4 \to 5 \to 6 \to 7$.

29. Recolor w_1 from $3 \to 4 \to 5 \to 6$.

30. Recolor x_6 from $8 \to 7 \to 6 \to 5$.

31. Recolor x_5 from $9 \to 8 \to 7 \to 6$.

32. Recolor x_4 from $10 \to 9 \to 8 \to 7$.

33. Recolor x_3 from $11 \to 10 \to 9 \to 8$.

34. Recolor x_2 from $4 \to 11 \to 10 \to 9$.

35. Recolor x_1 from $3 \to 4 \to 11 \to 10$.

36. Recolor z from $2 \to 3 \to 4 \to 5$.

37. Recolor y from $1 \to 2 \to 3 \to 4 \to 11$.

38. Recolor z from $5 \to 4 \to 3 \to 2 \to 1$.

39. Recolor z from $11 \to 4 \to 3 \to 2$.

40. Execute Step 1-35 backwardly.

Through the Steps from 1 to 23, we recolor the vertices in $V(G)$ to have a 3-coloring $f'_T : V(G) \to \{2, 3, 4\}$ such that $f_T(v) + 1 = f'_T(v)$. Then we exchange the colors of the vertices y, z and recolor all other vertices to their initial colors, and then we can reach the target coloring.

We then prove the only-if part. To reach the target coloring we must exchange the colors of the vertices y and z. To do this, we have to have a coloring f_M in $(f_0 \to f_r)$-reconfiguration sequence such that $f_M(y) = 11$ and $f_M(z) = 5$. Such a

coloring must be the following pattern:

$$f_M(v) = \begin{cases} 11 & \text{if } v = y \\ 5 & \text{if } v = z \\ 6 & \text{if } v = w_1 \\ 7 & \text{if } v = w_2 \\ 8 & \text{if } v = w_3 \\ 9 & \text{if } v = w_4 \\ 10 & \text{if } v = w_5 \\ 11 & \text{if } v = w_6 \\ 10 & \text{if } v = x_1 \\ 9 & \text{if } v = x_2 \\ 8 & \text{if } v = x_3 \\ 7 & \text{if } v = x_4 \\ 6 & \text{if } v = x_5 \\ 5 & \text{if } v = x_6 \\ 2 \text{ or } 3 \text{ or } 4 & \text{if } v \in V(G) \end{cases}$$

Therefore if an $(f_0 \to f_r)$-reconfiguration sequence exists then a 3-coloring of G exists. □

Chapter 4

Polynomial-Time Algorithms

In this chapter, we study the polynomial-time solvability of our problem, and generalize the known algorithmic results from the viewpoint of the graph structure of recolorability graphs. Specifically, our main result can be stated as the following theorem:

Theorem 7. *Suppose that a recolorability graph R is of maximum degree at most two, and let $k = |V(R)|$. For any graph G with n vertices and m edges,* COLORING RECONFIGURATION UNDER R-RECOLORABILITY *can be solved in $O(k+n+m)$ time. Furthermore, if an $(f_0 \to f_r)$-reconfiguration sequence exists for two k-colorings f_0 and f_r of G, then*

- *its shortest length can be computed in $O(k + n + m)$ time; and*
- *a shortest $(f_0 \to f_r)$-reconfiguration sequence can be output in $O(kn(n + m))$ time.*

We emphasize that Theorem 7 holds for any graph G, and only the structure of R is restricted. Since K_3 is of maximum degree two, Theorem 7 generalizes the known positive results for COLORING RECONFIGURATION [12, 19]. Note that k is not always a constant (indeed, can be larger than n).

We finally note that our results give a new insight for deeper understandings of tractable/intractable cases of COLORING RECONFIGURATION. Notice that the maximum degree of a recolorability graph R gives an upper bound on the number of "choices for the next step" (i.e., the number of recolorable colors from the currently assigned color) for every vertex in G. Then, together with the hardness results in Chapter 3, we know that the problem is PSPACE-complete if every vertex in G has more than two choices, but is solvable for any graph in polynomial time if the number of choices is bounded by at most two. This insight generalizes the known results: COLORING RECONFIGURATION is PSPACE-complete if $k \geq 4$ (and hence each vertex has more than two choices), and is solvable for any graph in polynomial time if $k \leq 3$ (and hence each vertex has at most two choices). Therefore, we can conclude that the essential for tractable/intractable cases of COLORING RECONFIGURATION is not the number k of colors but the number of such choices (i.e., the maximum degree of R).

In this chapter, we prove Theorem 7 as follows. Since the maximum degree of R is two and we can assume that the recolorability graph is connected, R is either a path or a cycle. In Section 4.1, we will prove Theorem 7 for the case where R is a path. Sections 4.2 and 4.3 are devoted to the case where R is a cycle. In Section 4.2, we will give an algorithm to solve COLORING RECONFIGURATION UNDER RECOLORABILITY for such a case; here, we do not care the shortest length of $(f_0 \to f_r)$-reconfiguration sequences, but only check its existence. In Section 4.3, we will give an algorithm to compute the shortest length for a yes-instance.

We also an polynomial-time algorithm of COLORING RECONFIGURATION UNDER R-RECOLORABILITY where R is a claw graph in Section 4.4.

4.1 Algorithms for Path Recolorability

In this section, we consider the case where R is a path. We first prove that the existence of an $(f_0 \to f_r)$-reconfiguration sequence can be checked in linear time, as follows.

Theorem 8. Coloring reconfiguration under R-recolorability *for any graph G can be solved in $O(k+n+m)$ time if a recolorability graph R is a path.*

We prove Theorem 8 by giving such an algorithm. We first rename the colors in R so that the colors $1, 2, \ldots, k$ appear in a numerical order along the path R, and modify two k-colorings f_0 and f_r accordingly; this can be done in $O(k+n)$ time. Then, the most important property for the path recolorability is that any recoloring step preserves the "order" of colors assigned to two adjacent vertices in G: If a k-coloring f of G assigns colors to two adjacent vertices $v, w \in V(G)$ such that $f(v) < f(w)$, then $f'(v) < f'(w)$ holds for any k-coloring f' such that an $(f \to f')$-reconfiguration sequence exists. Indeed, this property yields the following necessary and sufficient condition, which can be checked in $O(m)$ time; and hence Theorem 8 holds.

Lemma 5. *An $(f_0 \to f_r)$-reconfiguration sequence exists on $\mathcal{C}_R(G)$ if and only if $f_r(v) < f_r(w)$ holds for any $vw \in E(G)$ such that $f_0(v) < f_0(w)$.*

Proof. Since the ordering cannot be changed by any recoloring process, the only-if direction is trivial. Therefore, we only prove the if direction here.

We show that for any k-coloring f $(\neq f_r)$ of G, satisfying $f(v) < f(w) \Leftrightarrow f_r(v) < f_r(w)$ for each edge $vw \in E(G)$, there exists a k-coloring f' satisfying the following two conditions:

(a) f and f' are adjacent under R; and

(b) $\left(\sum_{v\in V(G)} |f_r(v) - f'(v)|\right) = \left(\sum_{v\in V(G)} |f_r(v) - f(v)|\right) - 1$.

Note that by condition (a), we have $f'(v) < f'(w) \Leftrightarrow f_r(v) < f_r(w)$ for each edge $vw \in E(G)$. Since we assume that $f_0(v) < f_0(w) \Leftrightarrow f_r(v) < f_r(w)$ holds for any $vw \in E(G)$, we can obtain an $(f_0 \to f_r)$-reconfiguration sequence by applying this fact to f_0 recursively.

Now we show that the existence of such an f'. Let $\overrightarrow{H_f}$ be a digraph with vertex set $V(G)$ and arc set $A(\overrightarrow{H_f}) = \{(v, w) : vw \in E(G), f(v)+1 = f(w)\}$. Since $f \neq f_r$, there exists at least one vertex v such that $f(v) \neq f_r(v)$. We consider only the case where $f(v) < f_r(v)$, because the other case is symmetric. There are following two cases:

Case 1: v is a sink vertex on $\overrightarrow{H_f}$

Since v is a sink vertex, any neighbor of v is not colored with $f(v)+1$. Therefore, v can be recolored to $f(v)+1$ and we obtain a desired f'.

Case 2: v is not a sink vertex on $\overrightarrow{H_f}$

Since v is not a sink vertex, there exists a forward walk from v. Let v^* be a end of the forward walk. Since the forward walk is ended at v^*, it is sink vertex. Therefore, v^* can be recolored to $f(v^*)+1$ and we obtain a desired f', if $f(v^*) < f_r(v^*)$ (and hence $f(v^*) < k$).

Finally, we show that v^* in above Case 2 fulfills $f(v^*) < f_r(v^*)$. Since there is a forward walk from v to v^*, it is enough to show that if $f(v) < f_r(v)$ and there is an arc $(v, w) \in A(\overrightarrow{H_f})$, then $f(w) < f_r(w)$. Since there is an arc (v, w), we have $f(v) + 1 = f(w)$, and hence by assumption, we have $f_r(v) < f_r(w)$. Then

$f_r(w) > f_r(v) > f(v) = f(w) - 1$ and we have $f(w) < f_r(w)$. This complete the proof. □

We next give a linear-time algorithm to compute $\mathsf{dist}(f_0, f_r)$ if an $(f_0 \to f_r)$-reconfiguration sequence exists on $\mathcal{C}_R(G)$.

Theorem 9. *Suppose that a recolorability graph R is a path, and let f_0 and f_r be two k-colorings of a graph G such that an $(f_0 \to f_r)$-reconfiguration sequence exists on $\mathcal{C}_R(G)$. Then,*

(*a*) $\mathsf{dist}(f_0, f_r) = \sum_{v \in V(G)} |f_r(v) - f_0(v)|$ *holds*;

(*b*) $\mathsf{dist}(f_0, f_r)$ *can be computed in $O(k + n + m)$ time*; *and*

(*c*) *a shortest $(f_0 \to f_r)$-reconfiguration sequence can be output in $O(kn(n + m))$ time.*

We check if an $(f_0 \to f_r)$-reconfiguration sequence exists on $\mathcal{C}_R(G)$ by Theorem 8; this can be done in $O(k + n + m)$ time. Then, Theorem 9(b) immediately follows from Theorem 9(a). Therefore, we will prove Theorem 9(a) and (c), as follows: Observe that $\mathsf{dist}(f_0, f_r) \geq \sum_{v \in V(G)} |f_r(v) - f_0(v)|$ holds, because each recoloring step can change the current color of a vertex $v \in V(G)$ to its adjacent color in R, and hence each vertex $v \in V(G)$ requires at least $|f_r(v) - f_0(v)|$ recoloring steps. Therefore, the following lemma completes the proof of Theorem 9.

Lemma 6. *There exists an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_R(G)$ of length $\sum_{v \in V(G)} |f_r(v) - f_0(v)|$. Furthermore, it can be output in $O(kn(n + m))$ time.*

Proof. We can obtain an $(f_0 \to f_r)$-reconfiguration sequence with length $\sum_{v \in V(G)} |f_r(v) - f_0(v)|$ by the procedure given in the proof of Lemma 5. Note that since this length achieves the claimed lower bound, it is the shortest one.

Now we estimate the running time of this procedure. The construction of $\overrightarrow{H_f}$ takes $O(m)$ time, and then we can find its sink (resp. source) vertex v satisfying $f(v) < f_r(v)$ (resp. $f(v) > f_r(v)$) in $O(n)$ time. Therefore each recursive step takes $O(n+m)$ time. Since the upper bound of the length of the shortest $(f_0 \to f_r)$-reconfiguration sequence is kn, it can be output in $O(n+m) \cdot kn = O(kn(n+m))$ time. □

4.2 Algorithm for Reachability on Cycle Recolorability

In this section, we consider the case where R is a cycle, and show that the existence of an $(f_0 \to f_r)$-reconfiguration sequence can be checked in linear time; the shortest length will be discussed in the next section. We prove the following theorem in this section.

Theorem 10. Coloring reconfiguration under recolorability *for any graph G can be solved in $O(k+n+m)$ time if a recolorability graph R is a cycle.*

Since K_3 is a cycle, Theorem 10 immediately implies the following corollary, which is restatement of the result by Johnson et al. [19].

Corollary 2. 3-Coloring reconfiguration *can be solved in linear time.*

We will prove Theorem 10 by giving such an algorithm, as follows. In Section 4.2.1, we give a simple necessary condition for a yes-instance based on the concept of frozen vertices; the idea is simple, but we need a nice characterization of frozen vertices for checking the condition in linear time. In Section 4.2.2, we then give a necessary and sufficient condition for a yes-instance by defining a potential

function which appropriately characterizes the reconfigurability of k-colorings; however, this condition cannot be checked in linear time by a naive way. In Section 4.2.3, we thus explain how to check the condition in linear time.

We rename the colors in R so that the colors $1, 2, \ldots, k$ appear in a numerical order along the cycle R, and modify two k-colorings f_0 and f_r accordingly; this can be done in $O(k+n)$ time. For notational convenience, we define the *successor* color c^+ and the *predecessor* color c^- for a color $c \in V(R)$, as follows:

$$c^+ = \begin{cases} c+1 & \text{if } c < k; \\ 1 & \text{if } c = k, \end{cases} \qquad \text{and} \qquad c^- = \begin{cases} c-1 & \text{if } c > 1; \\ k & \text{if } c = 1. \end{cases}$$

We use this notation also for a color assigned by a k-coloring: For a k-coloring f of a graph G and a vertex v in G, we denote by $f(v)^+$ and $f(v)^-$ the successor and predecessor colors for $f(v)$, respectively. In this and next sections, we call a k-coloring of G simply a *coloring*.

4.2.1 Characterization of frozen vertices

We now generalize the characterization of frozen vertices [12] from the viewpoint of cycle recolorability, which plays an important role in our algorithm. The following lemma gives a simple necessary condition, which immediately follows from the definition of frozen vertices.

Lemma 7. *Suppose that there exists an $(f \to f')$-reconfiguration sequence for two colorings f and f' of a graph G. Then, $\mathsf{Frozen}(f) = \mathsf{Frozen}(f')$, and $f(v) = f'(v)$ holds for every vertex v in $\mathsf{Frozen}(f)$.*

Note that it is not trivial to compute $\mathsf{Frozen}(f)$ for a coloring f in linear time. However, we will give a characterization of frozen vertices (in Lemma 8), which enables us to compute all of them in linear time (as proved in Lemma 9). We

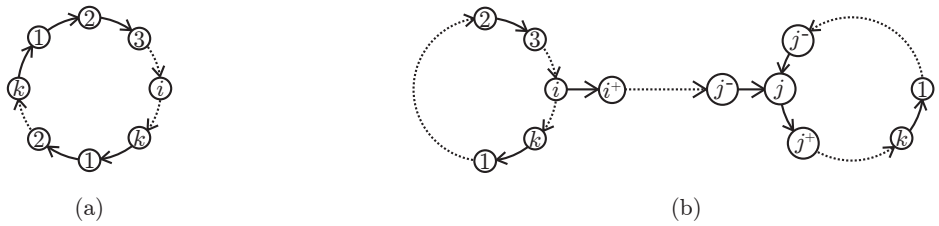


Figure 4.1: Characterization of frozen vertices.

note that Lemma 8 generalizes the characterization of frozen vertices on COLORING RECONFIGURATION with $k = 3$ given by Cereceda et al. [12].

To characterize the frozen vertices, we introduce some notation and terms. For a graph G and its coloring f, let $\overrightarrow{H}_f$ be the digraph with vertex set $V(\overrightarrow{H}_f) = V(G)$ and arc set

$$A(\overrightarrow{H}_f) \;=\; \{(v, w) : vw \in E(G) \text{ and } f(v)^+ = f(w)\}.$$

Notice that an arc $(v, w) \in A(\overrightarrow{H}_f)$ implies that $f(v) = f(w)^-$, and represents that, if we wish to recolor v from $f(v)$ to $f(v)^+$, we need to recolor w from $f(w)$ $(= f(v)^+)$ to $f(w)^+$ in advance. The *forward blocking graph from v on a coloring f*, denoted by $\overrightarrow{B}^+(v, f)$, is the subgraph of $\overrightarrow{H}_f$ consisting of all forward walks from v on $\overrightarrow{H}_f$. Similarly, the *backward blocking graph to v on a coloring f*, denoted by $\overrightarrow{B}^-(v, f)$, is the subgraph of $\overrightarrow{H}_f$ consisting of all backward walks to v on $\overrightarrow{H}_f$. Then, we have the following lemma. (See also Figure 4.1.)

Lemma 8. *A vertex $v \in V(G)$ is frozen on f if and only if it satisfies at least one of the following two conditions (a) and (b):*

- *(a) v is contained in a directed cycle in $\overrightarrow{H}_f$; and*
- *(b) $\overrightarrow{H}_f$ has both a forward walk from v and a backward walk to v, each of which contains a vertex in a directed cycle.*

Proof. Let S be the set of all vertices in G that satisfy at least one of the two conditions (a) and (b) above. Then, we will prove that $S = \mathsf{Frozen}(f)$.

We first prove that $S \subseteq \mathsf{Frozen}(f)$ holds. Let v be an arbitrary vertex in S, then we show that $v \in \mathsf{Frozen}(f)$. Since $v \in S$, it satisfies at least one of the two conditions (a) and (b). By the definition of $\overrightarrow{H}_f$, any vertex on a directed cycle in $\overrightarrow{H}_f$ cannot change its color. Therefore, if v satisfies the condition (a), we have $v \in \mathsf{Frozen}(f)$.

Next we consider the case where v satisfies only the condition (b). Then, $\overrightarrow{H}_f$ has a forward walk from v which ends in a vertex w contained in a directed cycle. Note that w is frozen on f, because it satisfies the condition (a). This implies that any vertex z (including v) cannot be recolored to its successor color $f(z)^+$. At the same time, $\overrightarrow{H}_f$ has a backward walk to v which ends in a vertex contained in a directed cycle, and hence v cannot be recolored to its predecessor color $f(v)^-$, too. Thus, v is frozen on f, as claimed.

We then prove that $\mathsf{Frozen}(f) \subseteq S$ holds by taking its contraposition. Let v be any vertex which is not in S, then we show that $v \notin \mathsf{Frozen}(f)$. Since $v \notin S$, at least one of $\overrightarrow{B}^+(v, f)$ and $\overrightarrow{B}^-(v, f)$ is an acyclic digraph. Assume that $\overrightarrow{B}^+(v, f)$ is acyclic; it is symmetric to prove the case where $\overrightarrow{B}^-(v, f)$ is acyclic. Then, we show that v can be recolored to the successor color $f(v)^+$ by the induction on the number of arcs in $\overrightarrow{B}^+(v, f)$. If $|A(\overrightarrow{B}^+(v, f))| = 0$, then v can be recolored immediately to $f(v)^+$ because any neighbor of v is not colored with $f(v)^+$. Therefore, consider the case where $|A(\overrightarrow{B}^+(v, f))| > 0$. Then, we obtain a new coloring f' of G by recoloring an arbitrary sink vertex w in $\overrightarrow{B}^+(v, f)$ to $f(w)^+$. Note that we can recolor w directly to $f(w)^+$, since it has no out-going arc in $\overrightarrow{B}^+(v, f)$. Furthermore, since $\overrightarrow{B}^+(v, f)$ is connected, w has at least one in-coming arc in $\overrightarrow{B}^+(v, f)$; observe that $\overrightarrow{B}^+(v, f')$

does not have such an in-coming arc of w, because w is colored with $f^+(w)$ in f'. We thus have $|A(\overrightarrow{B}^+(v, f'))| \leq |A(\overrightarrow{B}^+(v, f))| - 1$, and hence by applying the induction hypothesis the claim holds. □

Based on Lemma 8, we now prove that $\mathsf{Frozen}(f)$ can be computed in linear time.

Lemma 9. *For any coloring f of a graph G, $\mathsf{Frozen}(f)$ can be computed in $O(m)$ time.*

Proof. One can construct the digraph $\overrightarrow{H}_f$ in $O(m)$ time, by checking each edge vw in G. For convenience, we denote $\overrightarrow{H}'_f$ be a digraph obtained from $\overrightarrow{H}_f$ by deleting vertices which are not incident to any arc of $\overrightarrow{H}_f$. Notice that no vertex in $\overrightarrow{H}_f \setminus \overrightarrow{H}'_f$ is frozen on f by Lemma 8. To achive linear-time calculation, we compute outdegree $d^+(v)$ and indegree $d^-(v)$ of each vertex $v \in V(\overrightarrow{H'}_f)$ and keep them as variables. Then, we can compute $\mathsf{Frozen}(f)$ in $O(n + m)$ time as follows. First, we construct a queue consisting of all vertex v such that $d^+(v) = 0$ or $d^-(v) = 0$ in $O(n)$ time. Then we repeat the following:

1. Take a vertex v from the head of the queue.
2. If $d^+(v) = 0$, for each vertex $w \in V(\overrightarrow{H'}_f)$ such that $(w, v) \in A(\overrightarrow{H'}_f)$, decrement $d^+(w)$, and if it reaches 0, add w to the queue.
3. Otherwise ($d^-(v) = 0$), for each vertex $w \in V(\overrightarrow{H'}_f)$ such that $(v, w) \in A(\overrightarrow{H'}_f)$, decrement $d^-(w)$, and if it reaches 0, add w to the queue.
4. Remove v from the queue.

Each iteration step remove one vertex whose outdegree or indegree is 0, and refresh the indegree/outdegree of its neighbor. Step 1 and 4 takes $O(1)$ time for

each step. The computation time of Step 2 and 3 is bounded by degree of the vertex v, hence the summation of this computation time through the algorithm is $O(m)$. Therefore this algorithm takes $O(n+m)$ time.

The remaining digraph has no source/sink vertex therefore any of whose vertex is frozen on f. Therefore, all frozen vertices on f can be found in $O((n+m)+n+(n+m)) = O(n+m)$ time. Since G is connected in this paper, $m \geq n-1$ and hence $O(n+m) = O(m)$. □

4.2.2 Necessary and sufficient condition

In the remainder of this section, by Lemma 7 we assume that $\mathsf{Frozen}(f_0) = \mathsf{Frozen}(f_r)$ and $f_0(v) = f_r(v)$ for each vertex $v \in \mathsf{Frozen}(f_0)$; otherwise it is a no-instance. In this subsection, we will give a necessary and sufficient condition for a yes-instance.

We define notation to describe the condition. Let G be an undirected graph, and let $\overrightarrow{H}$ be any digraph whose underlying graph is a subgraph of G. For a coloring f of G and each arc $(u,v) \in A(\overrightarrow{H})$, we define the *potential* $\mathsf{p}_f((u,v))$ *of* (u,v) *on* f, as follows:

$$\mathsf{p}_f((u,v)) = \begin{cases} f(v) - f(u) & \text{if } f(v) > f(u); \\ f(v) - f(u) + k & \text{if } f(v) < f(u). \end{cases} \tag{4.1}$$

Note that $f(u) \neq f(v)$ holds since $uv \in E(G)$. In addition, observe that

$$\mathsf{p}_f((u,v)) + \mathsf{p}_f((v,u)) = k \tag{4.2}$$

holds for any pair of parallel arcs (u,v) and (v,u) if such a pair exists. Then, the *potential* $\mathsf{p}_f(\overrightarrow{H})$ *of* $\overrightarrow{H}$ *on* f is defined to be the sum of potentials of all arcs of $\overrightarrow{H}$ on f, that is, $\mathsf{p}_f(\overrightarrow{H}) = \sum_{(u,v)\in A(\overrightarrow{H})} \mathsf{p}_f((u,v))$.

Let C be a cycle in an undirected graph G. Then, there are only two possible orientations of C such that they form directed cycles, that is, either the clockwise

direction or the anticlockwise direction; we always denote by $\overrightarrow{C}$ and $\overleftarrow{C}$ such the two possible orientations of C. The following lemma immediately follows from Eq. (4.2).

Lemma 10. *Let f be a coloring of an undirected graph G. Then, $\mathsf{p}_f(\overrightarrow{C}) + \mathsf{p}_f(\overleftarrow{C}) = k|E(C)|$ for every cycle C in G.*

For a coloring f of an undirected graph G, we define a new (undirected) graph G^f as follows[1]: let $V(G^f) = V(G)$, and we arbitrarily add new edges between frozen vertices on G so that $\mathsf{Frozen}(f)$ induces a connected subgraph in the resulting graph. Then, since there are at most $|V(G)|$ frozen vertices, G^f has $|V(G)|$ vertices and at most $|E(G)| + |V(G)| - 1$ edges. Note that $G^f = G$ if $\mathsf{Frozen}(f) = \emptyset$. Recall that two given colorings f_0 and f_r of G are assumed to satisfy $\mathsf{Frozen}(f_0) = \mathsf{Frozen}(f_r)$ and $f_0(v) = f_r(v)$ for every vertex v in $\mathsf{Frozen}(f_0)$. We can thus suppose $G^{f_0} = G^{f_r}$, and hence simply denote it by G^{f}. Furthermore, since newly added edges join only frozen vertices, we have the following lemma.

Lemma 11. *There exists an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_R(G)$ if and only if there exists an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_R(G^{\mathsf{f}})$.*

We are now ready to claim our necessary and sufficient condition, as follows.

Theorem 11. *Let f_0 and f_r be two colorings of a graph G such that $\mathsf{Frozen}(f_0) = \mathsf{Frozen}(f_r)$, and $f_0(v) = f_r(v)$ for all vertices $v \in \mathsf{Frozen}(f_0)$. Then, an $(f_0 \to f_r)$-reconfiguration sequence exists on $\mathcal{C}_R(G)$ if and only if $\mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_r}(\overrightarrow{C})$ holds for every cycle C in G^{f}.*

Before proving the theorem, we note that Theorem 11 is independent from the choice of the orientations of a cycle C, because Lemma 10 implies that $\mathsf{p}_{f_0}(\overrightarrow{C}) =$

[1]We note that our construction of G^f is different from that by Cereceda et al. [12] so that the running time of our algorithm does not depend on k.

$\mathsf{p}_{f_r}(\overrightarrow{C})$ holds if and only if $\mathsf{p}_{f_0}(\overleftarrow{C}) = \mathsf{p}_{f_r}(\overleftarrow{C})$ holds. We also note that Theorem 11 does not directly yield a linear-time algorithm.

We first prove the only-if direction of Theorem 11. Suppose that there exists an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_R(G)$. Then, Lemma 11 implies that $\mathcal{C}_R(G^{\mathsf{f}})$ contains an $(f_0 \to f_r)$-reconfiguration sequence $\langle f_0, f_1, \ldots, f_\ell \rangle$, where $f_\ell = f_r$, and hence the only-if direction of Theorem 11 can be obtained from the following lemma.

Lemma 12. *Suppose that two colorings f and f' are adjacent on $\mathcal{C}_R(G^{\mathsf{f}})$. Then, $\mathsf{p}_f(\overrightarrow{C}) = \mathsf{p}_{f'}(\overrightarrow{C})$ holds for every cycle C in G^{f}.*

Proof. Let C be any cycle in G^{f}. Since f and f' are adjacent on $\mathcal{C}_R(G^{\mathsf{f}})$, there exists exactly one vertex $v \in V(G^{\mathsf{f}})$ such that $f(v) \neq f'(v)$. If v is not contained in C, then $\mathsf{p}_f(\overrightarrow{C}) = \mathsf{p}_{f'}(\overrightarrow{C})$ trivially holds. We thus consider the case where v is contained in C. Let (u, v) and (v, w) be the in-coming and out-going arcs of v in $\overrightarrow{C}$, respectively. Then, for any other arc $\overrightarrow{a} \in A(\overrightarrow{C}) \setminus \{(u, v), (v, w)\}$, we have

$$\mathsf{p}_f(\overrightarrow{a}) = \mathsf{p}_{f'}(\overrightarrow{a}). \tag{4.3}$$

Note that the color $f'(v)$ is either the successor or predecessor color for $f(v)$. We may assume that $f'(v)$ is the successor color for $f(v)$, that is, $f'(v) = f(v)^+$; the proof for the other case is symmetric. Then, in order to show $\mathsf{p}_f(\overrightarrow{C}) = \mathsf{p}_{f'}(\overrightarrow{C})$, it suffices to prove that both

$$\mathsf{p}_f((u, v)) = \mathsf{p}_{f'}((u, v)) - 1 \tag{4.4}$$

and

$$\mathsf{p}_f((v, w)) = \mathsf{p}_{f'}((v, w)) + 1 \tag{4.5}$$

hold, because Eqs. (4.3), (4.4) and (4.5) yield that

$$\begin{aligned}
\mathsf{p}_f(\overrightarrow{C}) =& \mathsf{p}_f((u,v)) + \mathsf{p}_f((v,w)) + \sum\{\mathsf{p}_f(\overrightarrow{a}) : \overrightarrow{a} \in A(\overrightarrow{C}) \setminus \{(u,v),(v,w)\}\} \\
=& \big(\mathsf{p}_{f'}((u,v)) - 1\big) + \big(\mathsf{p}_{f'}((v,w)) + 1\big) \\
&+ \sum\{\mathsf{p}_{f'}(\overrightarrow{a}) : \overrightarrow{a} \in A(\overrightarrow{C}) \setminus \{(u,v),(v,w)\}\} \\
=& \mathsf{p}_{f'}((u,v)) + \mathsf{p}_{f'}((v,w)) + \sum\{\mathsf{p}_{f'}(\overrightarrow{a}) : \overrightarrow{a} \in A(\overrightarrow{C}) \setminus \{(u,v),(v,w)\}\} \\
=& \mathsf{p}_{f'}(\overrightarrow{C})
\end{aligned}$$

as claimed. We consider the following two cases:

Case 1: $f(v) = k$.

In this case, $f'(v) = f(v)^+ = 1$. Since u is adjacent with v in G^{f}, both $f'(u) \neq f'(v)$ and $f(u) \neq f(v)$ hold. Therefore, we have $1 = f'(v) < f'(u) = f(u) < f(v) = k$. Then, Eq. (4.4) follows from Eq. (4.1) as follows:

$$\begin{aligned}
\mathsf{p}_f((u,v)) &= f(v) - f(u) \\
&= k - f(u) + 1 - 1 \\
&= f'(v) - f'(u) + k - 1 \\
&= \mathsf{p}_{f'}((u,v)) - 1.
\end{aligned}$$

Similarly, $1 = f'(v) < f'(w) = f(w) < f(v) = k$ holds, and hence Eq. (4.5) follows from Eq. (4.1) as follows:

$$\begin{aligned}
\mathsf{p}_f((v,w)) &= f(w) - f(v) + k \\
&= f(w) - k + k - 1 + 1 \\
&= f'(w) - f'(v) + 1 \\
&= \mathsf{p}_{f'}((v,w)) + 1.
\end{aligned}$$

Case 2: $f(v) < k$.

In this case, $f'(v) = f(v)^+ = f(v)+1$. We verify only Eq. (4.4); one can similarly verify Eq. (4.5). Furthermore, we consider only the case where $f'(u) = f(u) < f(v)$ holds; the proof is similar for the case where $f'(v) < f'(u) = f(u)$ holds. Then, Eq. (4.4) follows from Eq. (4.1) as follows:

$$\mathsf{p}_f((u,v)) = f(v) - f(u) = f'(v) - 1 - f'(u) = \mathsf{p}_{f'}((u,v)) - 1.$$

This completes the proof of the lemma. □

We then prove the if direction of Theorem 11: If $\mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_r}(\overrightarrow{C})$ holds for every cycle C in G^{f}, then an $(f_0 \to f_r)$-reconfiguration sequence exists on $\mathcal{C}_R(G^{\mathsf{f}})$; Lemma 11 then implies that $\mathcal{C}_R(G)$ contains an $(f_0 \to f_r)$-reconfiguration sequence.

Our proof is constructive, that is, we give an algorithm which indeed finds an $(f_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_R(G^{\mathsf{f}})$. We say that a vertex v is *fixed* if it is colored with $f_r(v)$ and our algorithm decides not to recolor v anymore. Thus, all frozen vertices are fixed. Our algorithm maintains the set of fixed vertices, denoted by F. The following Algorithm 1 transforms f_0 into a coloring f'_0 of G^{f} so that $F \neq \emptyset$, as the initialization.

Algorithm 1 (Initialization for Algorithm 2)

1. If $\mathsf{Frozen}(f_0) \neq \emptyset$, then let $F = \mathsf{Frozen}(f_0)$ and $f'_0 = f_0$.
2. Otherwise let $F = \{v\}$ for an arbitrarily chosen vertex $v \in V(G)$. Let $f = f_0$, and obtain f'_0 such that $f'_0(v) = f_r(v)$, as follows:
   - 2-1. If $f(v) = f_r(v)$, then let $f'_0 = f$ and stop the algorithm.
   - 2-2. Otherwise recolor a sink vertex w (possibly v itself) of $\overrightarrow{B}^+(v, f)$ to $f(w)^+$. Let f be the resulting coloring, and go to Step 2-1.

Note that we can always find a sink vertex w in Step 2-2 of Algorithm 1, because otherwise $\overrightarrow{B}^+(v, f)$ contains a directed cycle; by Lemma 8 the vertices in the directed cycle are frozen, and hence this contradicts the assumption that $\mathsf{Frozen}(f_0) = \emptyset$ holds in Step 2. Furthermore, since an $(f_0 \to f'_0)$-reconfiguration sequence exists on $\mathcal{C}_R(G^{\mathsf{f}})$, by Lemma 12 we have $\mathsf{p}_{f'_0}(\overrightarrow{C}) = \mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_r}(\overrightarrow{C})$ for any cycle C in G^{f}. We now give the following lemma.

Lemma 13. *Let F be the vertex subset obtained by Algorithm* 1*. Then, the induced subgraph $G^{\mathsf{f}}[F]$ is connected.*

Proof. If $\mathsf{Frozen}(f_0) = \emptyset$, then F consists of a single vertex v and hence the lemma clearly holds. Therefore, consider the case where $\mathsf{Frozen}(f_0) \neq \emptyset$. In this case, $G^{\mathsf{f}}[F] = G^{\mathsf{f}}[\mathsf{Frozen}(f_0)]$. Recall that G^{f} was obtained by adding new edges to G so that $G^{\mathsf{f}}[\mathsf{Frozen}(f_0)]$ is connected. Thus, $G^{\mathsf{f}}[F]$ is connected also in this case. □

We give our main procedure, called Algorithm 2, which finds an $(f'_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_R(G^{\mathsf{f}})$. The algorithm attempts to extend the vertex set F to $V(G^{\mathsf{f}})$ so that each vertex v in F is fixed (and hence is colored with $f_r(v)$); we eventually obtain the target coloring f_r when $F = V(G^{\mathsf{f}})$. Recall that our algorithm never recolors any vertex v in F, and all frozen vertices are contained in F. Let $f = f'_0$, and apply the following procedure.

Algorithm 2 (Finding an $(f'_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_R(G^{\mathsf{f}})$.)

1. If $F = V(G^{\mathsf{f}})$ holds, then stop the algorithm.
2. Otherwise pick an arbitrary vertex $v \in V(G^{\mathsf{f}}) \setminus F$ which is adjacent with at least one vertex $u \in F$.
   - 2-1. If $f(v) = f_r(v)$, then add v to F and go to Step 1.
   - 2-2. Otherwise

- if $\mathsf{p}_f((u,v)) < \mathsf{p}_{f_r}((u,v))$, then recolor a sink vertex w (possibly v itself) of $\overrightarrow{B}^+(v,f)$ to $f(w)^+$; and
- if $\mathsf{p}_f((u,v)) > \mathsf{p}_{f_r}((u,v))$, then recolor a source vertex w (possibly v itself) of $\overrightarrow{B}^-(v,f)$ to $f(w)^-$.

Let f be the resulting coloring, and go to Step 2-1.

To prove that Algorithm 2 correctly finds an $(f'_0 \to f_r)$-reconfiguration sequence on $\mathcal{C}_R(G^{\mathsf{f}})$, it suffices to show that there always exists a non-fixed sink/source vertex in Step 2-2 under the condition that $\mathsf{p}_{f'_0}(\overrightarrow{C}) = \mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_r}(\overrightarrow{C})$ holds for any cycle C in G^{f}. Therefore, the following lemma completes the proof of the if direction of Theorem 11.

Lemma 14. *Every application of Step* 2 *of Algorithm* 2 *produces a set F of fixed vertices and a coloring f of G^{f} satisfying the following* (*a*) *and* (*b*)*: For each edge $uv \in G^{\mathsf{f}}$ such that $u \in F$ and $v \notin F$,*

(*a*) *if $\mathsf{p}_f((u,v)) < \mathsf{p}_{f_r}((u,v))$, then $\overrightarrow{B}^+(v,f)$ is a directed acyclic graph such that no vertex in $\overrightarrow{B}^+(v,f)$ is contained in F; and*

(*b*) *if $\mathsf{p}_f((u,v)) > \mathsf{p}_{f_r}((u,v))$, then $\overrightarrow{B}^-(v,f)$ is a directed acyclic graph such that no vertex in $\overrightarrow{B}^-(v,f)$ is contained in F.*

Proof. By Lemma 12 we first note that

$$\mathsf{p}_f(\overrightarrow{C}) = \mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_r}(\overrightarrow{C}) \tag{4.6}$$

holds for any cycle C in G^{f}. We prove only the claim (a); the proof for the claim (b) is similar.

We first prove that no vertex in $\overrightarrow{B}^+(v,f)$ is contained in F if $\mathsf{p}_f((u,v)) < \mathsf{p}_{f_r}((u,v))$. Suppose for a contradiction that $\overrightarrow{B}^+(v,f)$ contains a vertex in F, and

let w be a fixed vertex in $\overrightarrow{B}^+(v,f)$ which is closest to v, that is, $\overrightarrow{B}^+(v,f)$ contains a directed path from v to w which passes through only non-fixed vertices except for w. Then, consider a directed cycle $\overrightarrow{C}$ consisting of the following three directed paths (i)–(iii):

(i) $\overrightarrow{P}_{u,v}$ is a directed path consisting of the single arc (u,v)

By the assumption, we have $\mathsf{p}_f(\overrightarrow{P}_{u,v}) < \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v})$.

(ii) $\overrightarrow{P}_{v,w}$ is the directed path in $\overrightarrow{B}^+(v,f)$ from v to w

By the definition of a forward blocking graph, notice that $\mathsf{p}_f(\overrightarrow{a}) = 1$ holds for any arc $\overrightarrow{a}$ in $\overrightarrow{P}_{v,w}$. Equation (4.1) implies that $\mathsf{p}_{f'}(\overrightarrow{a'}) \geq 1$ holds for any coloring f' of G^{f} and any arc $\overrightarrow{a'}$. Therefore, we have $\mathsf{p}_f(\overrightarrow{P}_{v,w}) \leq \mathsf{p}_{f_r}(\overrightarrow{P}_{v,w})$.

(iii) $\overrightarrow{P}_{w,u}$ is a directed path from w to u such that $V(\overrightarrow{P}_{w,u}) \subseteq F$

Lemma 13 ensures that such a path $\overrightarrow{P}_{w,u}$ exists. Since $V(\overrightarrow{P}_{w,u}) \subseteq F$, we have $f(z) = f_r(z)$ for any vertex z in $\overrightarrow{P}_{w,u}$. Thus, $\mathsf{p}_f(\overrightarrow{P}_{w,u}) = \mathsf{p}_{f_r}(\overrightarrow{P}_{w,u})$ holds.

Then, we have the following inequality:

$$\begin{aligned}\mathsf{p}_f(\overrightarrow{C}) &= \mathsf{p}_f(\overrightarrow{P}_{u,v}) + \mathsf{p}_f(\overrightarrow{P}_{v,w}) + \mathsf{p}_f(\overrightarrow{P}_{w,u}) \\ &< \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) + \mathsf{p}_{f_r}(\overrightarrow{P}_{v,w}) + \mathsf{p}_{f_r}(\overrightarrow{P}_{w,u}) = \mathsf{p}_{f_r}(\overrightarrow{C}).\end{aligned}$$

This inequality contradicts Eq. (4.6), and hence we can conclude that no vertex in $\overrightarrow{B}^+(v,f)$ is contained in F if $\mathsf{p}_f((u,v)) < \mathsf{p}_{f_r}((u,v))$.

Finally, we prove that $\overrightarrow{B}^+(v,f)$ is a directed acyclic graph. Suppose for a contradiction that $\overrightarrow{B}^+(v,f)$ contains a directed cycle $\overrightarrow{C}$. Then, by Lemma 8 any vertex v in $\overrightarrow{C}$ is frozen on f. By Lemma 7 such a vertex v is frozen also on f_0. Therefore, v must be included in F initially. This contradicts the fact that no vertex in $\overrightarrow{B}^+(v,f)$ is contained in F if $\mathsf{p}_f((u,v)) < \mathsf{p}_{f_r}((u,v))$. □

4.2.3 Proof of Theorem 10

We finally prove Theorem 10 by giving such an algorithm. Our algorithm first checks the simple necessary condition described in Lemma 7. By Lemma 9 this step can be done in $O(m)$ time. Note that we can obtain the vertex subsets $\mathsf{Frozen}(f_0)$ and $\mathsf{Frozen}(f_r)$ in this running time. Then, we determine whether a given instance is a yes-instance or not, based on the necessary and sufficient condition described in Theorem 11. However, recall that the condition in Theorem 11 cannot be checked in linear time by a naive way. Below, we give a linear-time algorithm to check the condition.

Let T be an arbitrary spanning tree of the graph G^{f}. For an edge $e \in E(G^{\mathsf{f}}) \setminus E(T)$, we denote by $C_{T,e}$ the unique cycle obtained by adding the edge e to T. The following lemma shows that it suffices to check the necessary and sufficient condition only for the number $|E(G^{\mathsf{f}}) \setminus E(T)|$ of cycles.

Lemma 15. *Let T be any spanning tree of G^{f}. Then, $\mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_r}(\overrightarrow{C})$ holds for every cycle C of G^{f} if and only if $\mathsf{p}_{f_0}(\overrightarrow{C_{T,e}}) = \mathsf{p}_{f_r}(\overrightarrow{C_{T,e}})$ holds for every edge $e \in E(G^{\mathsf{f}}) \setminus E(T)$.*

Proof. The only-if direction clearly holds, and hence we prove the if direction by the induction on the number of edges in $E(C) \setminus E(T)$ for a cycle C of G^{f}.

We first consider any cycle C of G^{f} such that $|E(C) \setminus E(T)| = 1$. Let e' be the edge in $E(C) \setminus E(T)$, then $C_{T,e'} = C$. By the assumption, we have $\mathsf{p}_{f_0}(\overrightarrow{C_{T,e'}}) = \mathsf{p}_{f_r}(\overrightarrow{C_{T,e'}})$ and hence $\mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_0}(\overrightarrow{C_{T,e'}}) = \mathsf{p}_{f_r}(\overrightarrow{C_{T,e'}}) = \mathsf{p}_{f_r}(\overrightarrow{C})$, as claimed.

We then consider any cycle C of G^{f} such that $|E(C) \setminus E(T)| > 1$. Then, C contains at least two edges in $E(C) \setminus E(T)$. Pick an arbitrary edge uv in $E(C) \setminus E(T)$, and let wx be the edge in $E(C) \setminus E(T)$ that first appears after uv when we traverse C

along the direction $\overrightarrow{C}$; note that $v = w$ may hold, and that all edges between v and w are contained in $E(T)$ if exist. (See Figure 4.2.) For two vertices $a, b \in V(C)$, we denote by $\overrightarrow{P}_{a,b}$ the directed path in $\overrightarrow{C}$ from a to b. We divide $\overrightarrow{C}$ into four directed paths $\overrightarrow{P}_{u,v}$, $\overrightarrow{P}_{v,w}$, $\overrightarrow{P}_{w,x}$ and $\overrightarrow{P}_{x,u}$. Then, since both uv and wx are contained in $E(C) \setminus E(T)$, there exist two vertices $y \in V(\overrightarrow{P}_{v,w})$ and $z \in V(\overrightarrow{P}_{x,u})$ such that the unique path on T between y and z does not pass through any edge in C. (See Figure 4.2.) Let $\overrightarrow{P}_{y,z}$ be the orientation from y to z for such a path, while let $\overrightarrow{P}_{z,y}$ be the other orientation of the path. Then, we define two directed cycles $\overrightarrow{C_1}$ and $\overrightarrow{C_2}$, as follows:

- $\overrightarrow{C_1} = \overrightarrow{P}_{u,v} \cup \overrightarrow{P}_{v,y} \cup \overrightarrow{P}_{y,z} \cup \overrightarrow{P}_{z,u}$; and
- $\overrightarrow{C_2} = \overrightarrow{P}_{w,x} \cup \overrightarrow{P}_{x,z} \cup \overrightarrow{P}_{z,y} \cup \overrightarrow{P}_{y,w}$.

Since $\overrightarrow{C_1}$ and $\overrightarrow{C_2}$ pass through the unique path in T between y and z in the opposite directions, the arcs in $\overrightarrow{C_1}$ and $\overrightarrow{C_2}$ are all mutually disjoint. Now both $|E(C_1) \setminus E(T)|$ and $|E(C_2) \setminus E(T)|$ are strictly smaller than $|E(C) \setminus E(T)|$. We thus apply the induction hypothesis to $\overrightarrow{C_1}$ and $\overrightarrow{C_2}$, and have $\mathsf{p}_{f_0}(\overrightarrow{C_1}) = \mathsf{p}_{f_r}(\overrightarrow{C_1})$ and $\mathsf{p}_{f_0}(\overrightarrow{C_2}) = \mathsf{p}_{f_r}(\overrightarrow{C_2})$. Therefore, by Eq. (4.2) we have

$$\begin{aligned} \mathsf{p}_{f_0}(\overrightarrow{C_1}) + \mathsf{p}_{f_0}(\overrightarrow{C_2}) &= \mathsf{p}_{f_0}(\overrightarrow{C_1} \cup \overrightarrow{C_2}) \\ &= \mathsf{p}_{f_0}(\overrightarrow{C}) + \mathsf{p}_{f_0}(\overrightarrow{P}_{y,z}) + \mathsf{p}_{f_0}(\overrightarrow{P}_{z,y}) \\ &= \mathsf{p}_{f_0}(\overrightarrow{C}) + k|A(\overrightarrow{P}_{y,z})| \end{aligned}$$

and

$$\begin{aligned} \mathsf{p}_{f_r}(\overrightarrow{C_1}) + \mathsf{p}_{f_r}(\overrightarrow{C_2}) &= \mathsf{p}_{f_r}(\overrightarrow{C_1} \cup \overrightarrow{C_2}) \\ &= \mathsf{p}_{f_r}(\overrightarrow{C}) + \mathsf{p}_{f_r}(\overrightarrow{P}_{y,z}) + \mathsf{p}_{f_r}(\overrightarrow{P}_{z,y}) \\ &= \mathsf{p}_{f_r}(\overrightarrow{C}) + k|A(\overrightarrow{P}_{y,z})|. \end{aligned}$$

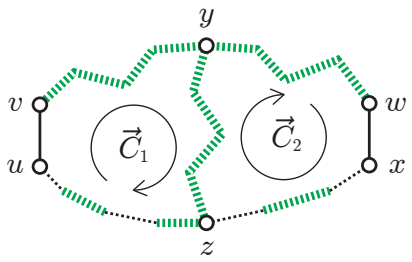


Figure 4.2: Illustration for Lemma 15, where the edges in a spanning tree T are depicted by (green) dotted thick lines and the edges in $E(C) \setminus E(T)$ by thin lines.

By the induction hypothesis, we thus have $\mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_r}(\overrightarrow{C})$, as claimed. □

Lemma 15 and the following lemma imply that there is a linear-time algorithm to check the necessary and sufficient condition described in Theorem 11. Therefore, the following lemma completes the proof of Theorem 10.

Lemma 16. *Let T be any spanning tree of G^{f}. Then, we can check whether $\mathsf{p}_{f_0}(\overrightarrow{C_{T,e}}) = \mathsf{p}_{f_r}(\overrightarrow{C_{T,e}})$ for all $e \in E(G^{\mathsf{f}}) \setminus E(T)$ can be computed in $O(n+m)$ time in total.*

Proof. We show that equivalence between $\mathsf{p}_{f_0}(\overrightarrow{C_{T_e}})$ and $\mathsf{p}_{f_r}(\overrightarrow{C_{T_e}})$ can be computed in $O(1)$ time for any $e \in E(G^{\mathsf{f}}) \setminus E(T)$ by $O(|V(G^{\mathsf{f}})| + |E(G^{\mathsf{f}})|)$ time preprocessing, then we can compute the equivalence between $\mathsf{p}_{f_0}(\overrightarrow{C_{T_e}})$ and $\mathsf{p}_{f_r}(\overrightarrow{C_{T_e}})$ in $O(|V(G^{\mathsf{f}})| + |E(G^{\mathsf{f}})|) + O(1) \cdot |E(G^{\mathsf{f}}) \setminus E(T)| = O(n + (m+n)) + O(1) \cdot (O(m+n) - (n-1)) = O(n+m)$ time in total for all $e \in E(G^{\mathsf{f}}) \setminus E(T)$.

First, we construct a spanning tree T of G^{f} by $O(|V(G^{\mathsf{f}})| + |E(G^{\mathsf{f}}|)$ time traversing. We regard T as a rooted tree and take arbitrary vertex r as its root. For convenience we give some notation. We call a vertex of degree one and degree at least three on T as *leaf* and *branch* vertex respectively. Notice that any edge $e \in E(G^{\mathsf{f}}) \setminus E(T)$

incidents to two leaf vertices of T. For any two vertices $v, w \in V(G^{\mathsf{f}})$, there is an unique path $P_{v,w}$ between v and w on T. We denote a partially orientation $\overrightarrow{P}_{v,w}$ as a directed path from v to w. We also denote unique lowest common ancestor of v and w on the rooted tree T by $lca(v, w)$.

For each leaf and branch vertex v, we compute $\mathsf{p}_{f_0}(\overrightarrow{P}_{r,v})$ and $\mathsf{p}_{f_0}(\overrightarrow{P}_{v,r})$. Since $\mathsf{p}_{f_0}(\overrightarrow{P}_{r,v})$ (resp. $\mathsf{p}_{f_0}(\overrightarrow{P}_{v,r})$) is the sum of potentials of arcs in directed path from r to v (resp. from v to r), we can compute these potentials for all leaf and branch vertex v in $O(|E(G^{\mathsf{f}})|)$ time by depth-first search with successive addition of potentials of arcs. Together with this computation, we construct an array which is indexed by leaf and branch vertices and returns $\mathsf{p}_{f_0}(\overrightarrow{P}_{v,r})$ and $\mathsf{p}_{f_0}(\overrightarrow{P}_{r,v})$, in order to compute $\mathsf{p}_{f_0}(\overrightarrow{P}_{v,r})$ and $\mathsf{p}_{f_0}(\overrightarrow{P}_{r,v})$ for any leaf and branch vertex in $O(1)$ time. Construction of this array can be done $O(|E(G^{\mathsf{f}})|)$ time.

Let C be unique cycle obtained by adding an edge $vw \in E(G^{\mathsf{f}}) \setminus E(T)$ to T and $\overrightarrow{C}$ be an orientation which forms directed cycle, in which the edge vw is oriented as an arc (v, w). Then the following holds:

$$
\begin{aligned}
&\mathsf{p}_{f_0}(\overrightarrow{P}_{r,v}) + \mathsf{p}_{f_0}((v,w)) + \mathsf{p}_{f_0}(\overrightarrow{P}_{w,r}) \\
=&\mathsf{p}_{f_0}(\overrightarrow{P}_{r,lca(v,w)}) + \mathsf{p}_{f_0}(\overrightarrow{P}_{lca(v,w),v}) + \mathsf{p}_{f_0}((v,w)) + \mathsf{p}_{f_0}(\overrightarrow{P}_{w,lca(v,w)}) + \mathsf{p}_{f_0}(\overrightarrow{P}_{lca(v,w),r}) \\
=&(\mathsf{p}_{f_0}(\overrightarrow{P}_{lca(v,w),v}) + \mathsf{p}_{f_0}((v,w)) + \mathsf{p}_{f_0}(\overrightarrow{P}_{w,lca(v,w)})) + (\mathsf{p}_{f_0}(\overrightarrow{P}_{r,lca(v,w)}) + \mathsf{p}_{f_0}(\overrightarrow{P}_{lca(v,w),r})) \\
=&\mathsf{p}_{f_0}(\overrightarrow{C}) + k|A(\overrightarrow{P}_{r,lca(v,w)})|
\end{aligned}
$$

This value can be computed in $O(1)$ time, and we can also compute the value $\mathsf{p}_{f_r}(\overrightarrow{C}) + k|A(\overrightarrow{P}_{r,lca(v,w)})|$ in $O(1)$ time in similar way.

These two values $\mathsf{p}_{f_0}(\overrightarrow{C}) + k|A(\overrightarrow{P}_{r,lca(v,w)})|, \mathsf{p}_{f_r}(\overrightarrow{C}) + k|A(\overrightarrow{P}_{r,lca(v,w)})|$ are equal if and only if $\mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_r}(\overrightarrow{C})$, therefore we can check whether $\mathsf{p}_{f_0}(\overrightarrow{C}) = \mathsf{p}_{f_r}(\overrightarrow{C})$ in $O(1)$ time. To check equivalence between $\mathsf{p}_{f_0}(\overrightarrow{C_{T,e}})$ and $\mathsf{p}_{f_r}(\overrightarrow{C_{T,e}})$ for all edges

$e \in E(G^{\mathsf{f}}) \setminus E(T)$ we need $O(1) \cdot (|E(G^{\mathsf{f}})| - |V(G^{\mathsf{f}})| + 1) = O(1) \cdot O(m - n + 1) = O(m)$ time. The entire computing time takes $O(n + m) + O(m) = O(n + m)$ time. □

4.3 Algorithm for Shortest Sequence on Cycle Recolorability

In this section, we consider the case where R is a cycle, and explain how to compute the length of (or how to output) a shortest reconfiguration sequence.

Let $P_{u,v}$ be a path in an undirected graph G connecting vertices u and v. We denote by $\overrightarrow{P}_{u,v}$ the directed path from u to v. The following theorem characterizes the shortest length of an $(f_0 \to f_r)$-reconfiguration sequence, which generalizes the characterization for COLORING RECONFIGURATION with $k = 3$ [12, 19].

Theorem 12. *Suppose that a recolorability graph R is a cycle, and let f_0 and f_r be two colorings of a graph G such that an $(f_0 \to f_r)$-reconfiguration sequence exists on $\mathcal{C}_R(G)$. Then, the following (a) and (b) hold:*

(a) *If $\mathsf{Frozen}(f_0) \neq \emptyset$, then it holds for an arbitrary chosen vertex $u \in \mathsf{Frozen}(f_0)$ that*

$$\mathsf{dist}(f_0, f_r) = \sum_{v \in V(G)} \left| \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) \right|,$$

where $P_{u,v}$ is an arbitrary chosen path in G connecting u and v.

(b) *If $\mathsf{Frozen}(f_0) = \emptyset$, then there exist two integers $\rho_{u,1}$ and $\rho_{u,2}$ for an arbitrary chosen vertex $u \in V(G)$ such that*

$$\mathsf{dist}(f_0, f_r) = \min \left\{ \sum_{v \in V(G)} \left| \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho_{u,1} \right|, \sum_{v \in V(G)} \left| \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho_{u,2} \right| \right\},$$

where $P_{u,v}$ is an arbitrary chosen path in G connecting u and v.

To prove Theorem 12 we show the following two cases. If $\mathsf{Frozen}(f_0) \neq \emptyset$ then the length of the shortest $(f_0 \to f_r)$-reconfiguration sequence is

$$\sum_{v \in V(G)} \left| \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) \right| \tag{4.7}$$

where u is arbitrary frozen vertex and $\overrightarrow{P}_{u,v}$ is arbitrary directed path. Otherwise if $\mathsf{Frozen}(f_0) = \emptyset$ then the length of the shortest $(f_0 \to f_r)$-reconfiguration sequence is

$$\min \left\{ \sum_{v \in V(G)} \left| \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho \right| : \rho \equiv f_r(u) - f_0(u) \pmod{k} \right\} \tag{4.8}$$

where u is arbitrary vertex and $\overrightarrow{P}_{u,v}$ is arbitrary directed path.

In Section 4.3 we show that Theorem 12 holds for the case $\mathsf{Frozen}(f_0) \neq \emptyset$, i.e., Eq. 4.7 holds. In Section 4.3 we show that Theorem 12 holds for the other case $\mathsf{Frozen}(f_0) \neq \emptyset$, i.e., the choice of the value of $\rho \equiv f_r(u) - f_0(u) \pmod{k}$ of Eq. 4.7 can be reduced into only two choices $\rho_{u,1}$ and $\rho_{u,2}$.

For remaining proofs we need some more notations and lemmas. For any two reachable colorings f and f', we denote an $(f \to f')$-reconfiguration sequence by $\overrightarrow{S}_{f,f'}$. Sometimes we append some coloring f to the head of $(f' \to f'')$-reconfiguration sequence $\overrightarrow{S}_{f',f''}$, we denote such a sequence by $f\overrightarrow{S}_{f',f''}$ The length of $\overrightarrow{S}_{f_0,f_\ell} = \langle f_0, f_1, \ldots, f_\ell \rangle$ is denoted by $\mathsf{len}(\overrightarrow{S}_{f_0,f_\ell}) = \ell$. For a $(f_0 \to f_\ell)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_\ell} = \langle f_0, f_1, \ldots, f_\ell \rangle$ and a vertex $v \in V(G)$, *Rotation* $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell})$ of v is defined as follows:

- If $\ell = 0$ (sequence contains exactly one coloring) then $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell}) = 0$
- Otherwise $\overrightarrow{S}_{f_0,f_\ell} = f_0\overrightarrow{S}_{f_1,f_\ell}$ for some f_1. There are three cases:
  - If $f_0(v) = f_1(v)$ then $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell}) = \mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell})$.
  - If $f_0(v)^+ = f_1(v)$ then $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell}) = \mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell}) + 1$.

– If $f_0(v)^- = f_1(v)$ then $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell}) = \mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell}) - 1$.

Intuitively the rotation of a vertex v corresponds to the total movement of the color of v on the recolorability cycle through the reconfiguration sequence $\overrightarrow{S}_{f_0,f_\ell}$. For example, if the rotation $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})$ is 3, $f_r(v)$ is at a distance 3 from $f_0(v)$, in the direction of increasing order of the labels of the recolorability cycle.

For the case $\mathsf{Frozen}(f_0) \neq \emptyset$ the rotation of each vertex is uniquely determined. Actually in this case the length of the shortest reconfiguration sequence equals the sum of absolute values of rotations of each vertices. On the other hand, for the case $\mathsf{Frozen}(f_0) = \emptyset$ the vertices may have infinitely possibility of the values of their rotations. For this case we choose arbitrary vertex u and determine its rotation, then the rotations of all other vertices are uniquely determined. Actually, the variable ρ occuring Eq. 4.8 is the rotation of u.

The following lemma shows an inequality between the length of reconfiguration sequence and rotation. This is the basis for lower bound of the length of $(f_0 \to f_r)$-reconfiguration sequence.

Lemma 17. $\mathsf{len}(\overrightarrow{S}_{f,f'}) \geq \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f,f'})|$ *holds for any reconfiguration sequence* $\overrightarrow{S}_{f,f'}$ *on* $\mathcal{C}_R(G)$.

Proof. We prove by the induction on the length $\mathsf{len}(\overrightarrow{S}_{f,f'})$. By definition of rotation, if $\mathsf{len}(\overrightarrow{S}_{f,f'}) = 0$, i.e., $f = f'$, then $\mathsf{r}_v(\overrightarrow{S}_{f,f'}) = 0$ holds for any vertex $v \in V(G)$, therefore the inequation holds.

Otherwise, we consider the case $\mathsf{len}(\overrightarrow{S}_{f,f'}) > 0$ holds. Let $\overrightarrow{S}_{f,f'} = \langle f_0, f_1, \ldots, f_\ell \rangle$, where $f_0 = f$ and $f_\ell = f'$. Then $\overrightarrow{S}_{f_0,f_\ell} = f_0 \overrightarrow{S}_{f_1,f_\ell}$ holds for some coloring f_1. Let $w \in V(G)$ be an unique vertex such that $f_0(w) \neq f_1(w)$. There are two cases of the

value of $\mathsf{r}_w(\overrightarrow{S}_{f_0,f_\ell})$:

$$\mathsf{r}_w(\overrightarrow{S}_{f_0,f_\ell}) = \begin{cases} \mathsf{r}_w(\overrightarrow{S}_{f_1,f_\ell}) + 1 & \text{if } f_0(w)^+ = f_1(w) \\ \mathsf{r}_w(\overrightarrow{S}_{f_1,f_\ell}) - 1 & \text{if } f_0(w)^- = f_1(w) \end{cases}$$

Notice that in both of cases $|\mathsf{r}_w(\overrightarrow{S}_{f_0,f_\ell})| \leq |\mathsf{r}_w(\overrightarrow{S}_{f_1,f_\ell})| + 1$ holds. By definition of rotation, we have $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell}) = \mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell})$ for any other vertex $v \in V(G) \setminus \{w\}$ since $f_0(v) = f_1(v)$. By induction hypothesis we have the following inequation:

$$\mathsf{len}(\overrightarrow{S}_{f_1,f_\ell}) \geq \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell})|$$

Therefore we obtain the desired inequation as follows:

$$\begin{aligned} \mathsf{len}(\overrightarrow{S}_{f_0,f_\ell}) &= \mathsf{len}(\overrightarrow{S}_{f_1,f_\ell}) + 1 \\ &\geq \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell})| + 1 \\ &= |\mathsf{r}_w(\overrightarrow{S}_{f_1,f_\ell})| + 1 + \sum_{v \in V(G)\setminus\{w\}} |\mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell})| \\ &\geq |\mathsf{r}_w(\overrightarrow{S}_{f_0,f_\ell})| + \sum_{v \in V(G)\setminus\{w\}} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell})| \\ &= \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell})| \end{aligned}$$

□

Our purpose is to describe rotations of each vertex by a function of two colorings f_0 and f_r. The following lemma shows the correspondence between rotation and coloring.

Lemma 18. *Let $\overrightarrow{S}_{f,f'}$ be any reconfiguration sequence on $\mathcal{C}_R(G)$ and $\overrightarrow{P}_{u,v}$ be any directed path from a vertex $u \in V(G)$ to a vertex $v \in V(G)$. Then the following formula holds:*

$$\mathsf{p}_f(\overrightarrow{P}_{u,v}) + \mathsf{r}_v(\overrightarrow{S}_{f,f'}) - \mathsf{p}_{f'}(\overrightarrow{P}_{u,v}) - \mathsf{r}_u(\overrightarrow{S}_{f,f'}) = 0.$$

Proof. We prove by the induction of the length $\mathsf{len}(\overrightarrow{S}_{f,f'})$. If $\mathsf{len}(\overrightarrow{S}_{f,f'}) = 0$, i.e., $f = f'$, then $\mathsf{p}_f(\overrightarrow{P}_{u,v}) = \mathsf{p}_{f'}(\overrightarrow{P}_{u,v})$ and $\mathsf{r}_v(\overrightarrow{S}_{f,f'}) = \mathsf{r}_u(\overrightarrow{S}_{f,f'}) = 0$ and the claimed formula holds.

Otherwise, we consider the case $\mathsf{len}(\overrightarrow{S}_{f,f'}) > 0$ holds. Let $\overrightarrow{S}_{f,f'} = \langle f_0, f_1, \ldots, f_\ell \rangle$ where $f_0 = f$ and $f_\ell = f'$. Then $\overrightarrow{S}_{f_0,f_\ell} = f_0 \overrightarrow{S}_{f_1,f_\ell}$ holds for some coloring f_1. By induction hypothesis we have the following formula:

$$\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + \mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell}) - \mathsf{p}_{f_\ell}(\overrightarrow{P}_{u,v}) - \mathsf{r}_u(\overrightarrow{S}_{f_1,f_\ell}) = 0$$

There are three cases of the choice of vertex recoloring from f_0 to f_1:

- If $f_0(u)^{\pm} = f_1(u)$,

$$\begin{aligned}
&\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell}) - \mathsf{p}_{f_\ell}(\overrightarrow{P}_{u,v}) - \mathsf{r}_u(\overrightarrow{S}_{f_0,f_\ell}) \\
=&(\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) \pm 1) + \mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell}) - \mathsf{p}_{f_\ell}(\overrightarrow{P}_{u,v}) - (\mathsf{r}_u(\overrightarrow{S}_{f_1,f_\ell}) \pm 1) \\
=&\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + \mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell}) - \mathsf{p}_{f_\ell}(\overrightarrow{P}_{u,v}) - \mathsf{r}_u(\overrightarrow{S}_{f_1,f_\ell}) = 0
\end{aligned}$$

- If $f_0(v)^{\pm} = f_1(v)$,

$$\begin{aligned}
&\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell}) - \mathsf{p}_{f_\ell}(\overrightarrow{P}_{u,v}) - \mathsf{r}_u(\overrightarrow{S}_{f_0,f_\ell}) \\
=&(\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) \mp 1) + (\mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell}) \pm 1) - \mathsf{p}_{f_\ell}(\overrightarrow{P}_{u,v}) - \mathsf{r}_u(\overrightarrow{S}_{f_1,f_\ell}) \\
=&\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + \mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell}) - \mathsf{p}_{f_\ell}(\overrightarrow{P}_{u,v}) - \mathsf{r}_u(\overrightarrow{S}_{f_1,f_\ell}) = 0
\end{aligned}$$

- If $f_0(w)^{\pm} = f_1(w)$ where w is neither u nor v,

$$\begin{aligned}
&\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \mathsf{r}_v(\overrightarrow{S}_{f_0,f_\ell}) - \mathsf{p}_{f_\ell}(\overrightarrow{P}_{u,v}) - \mathsf{r}_u(\overrightarrow{S}_{f_0,f_\ell}) \\
=&\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + \mathsf{r}_v(\overrightarrow{S}_{f_1,f_\ell}) - \mathsf{p}_{f_\ell}(\overrightarrow{P}_{u,v}) - \mathsf{r}_u(\overrightarrow{S}_{f_1,f_\ell}) = 0
\end{aligned}$$

□

In the rest of this section, we consider two case whether $\mathsf{Frozen}(f_0)$ is empty or not.

The case $\mathsf{Frozen}(f_0) \neq \emptyset$

In this subsection we show that Algorithm 2 always finds the shortest reconfiguration sequence for the case where $\mathsf{Frozen}(f_0) = \mathsf{Frozen}(f_r) \neq \emptyset$. Let u be a frozen vertex. Then u is never recolored through $(f_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_r}$. Therefore we have $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_r}) = 0$. Then by Lemma 18 we have the following statement:

Corollary 3. *Let $\overrightarrow{S}_{f,f'}$ be an arbitrary reconfiguration sequence on $\mathcal{C}_R(G)$ and $\overrightarrow{P}_{u,v}$ be a directed path from $u \in V(G)$ to $v \in V(G)$. If $u \in \mathsf{Frozen}(f)$ then the following formula holds:*

$$\mathsf{p}_{f'}(\overrightarrow{P}_{u,v}) - \mathsf{p}_f(\overrightarrow{P}_{u,v}) = \mathsf{r}_v(\overrightarrow{S}_{f,f'})$$

The following lemma shows that we can always finds the shortest reconfiguration sequence in the case $\mathsf{Frozen}(f_0) \neq \emptyset$.

Lemma 19. *If $\mathsf{Frozen}(f_0) \neq \emptyset$, then the length of the shortest $(f_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_r}$ is $\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) = \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})|$.*

Proof. We show that Algorithm 2 always find an $(f_0 \to f_r)$-reconfiguration sequence of the claimed length. By assumption there exist an $(f_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_r}$ which is found by Algorithm 2. We prove that $\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) = \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})|$ holds by the induction on $\mathsf{len}(\overrightarrow{S}_{f_0,f_r})$.

If $\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) = 0$, then $f_0 = f_r$ and $\sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})| = 0$ trivially holds. Otherwise, we consider the case $\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) > 0$. In this case we have some coloring f_1 such that $\overrightarrow{S}_{f_0,f_r} = f_0 \overrightarrow{S}_{f_1,f_r}$. Then we have some unique vertex z such that $f_0(z) \neq f_1(z)$. f_0 is recolored by the step 2-2 of Algorithm 2 and then new coloring f_1 is obtained. Then there are two vertices $x \in F$ and $y \notin F$ they are adjacent and the step 2-2 is separated into two cases:

- $\mathsf{p}_{f_0}((x,y)) < \mathsf{p}_{f_r}((x,y))$ and the sink vertex z of $\overrightarrow{B}^+(x, f_0)$ is recolored to $f(z)^+$.
- $\mathsf{p}_{f_0}((x,y)) > \mathsf{p}_{f_r}((x,y))$ and the source vertex z of $\overrightarrow{B}^-(x, f_0)$ is recolored to $f(z)^-$.

We discuss only the former case because the latter case is symmetric. In the remaining process Algorithm 2 finds an $(f_1 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_1,f_r}$ whose length $\mathsf{len}(\overrightarrow{S}_{f_1,f_r})$ is exactly $\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) - 1$. So by the induction hypothesis the following formula holds:

$$\mathsf{len}(\overrightarrow{S}_{f_1,f_r}) = \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_1,f_r})|$$

In the former case, we have $\mathsf{r}_z(\overrightarrow{S}_{f_0,f_r}) = \mathsf{r}_z(\overrightarrow{S}_{f_1,f_r}) + 1$ while $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) = \mathsf{r}_v(\overrightarrow{S}_{f_1,f_r})$ for any other vertex $v \in V(G) \setminus \{z\}$. Therefore we have the following formula:

$$\begin{aligned}
\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) - 1 &= \mathsf{len}(\overrightarrow{S}_{f_1,f_r}) \\
&= \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_1,f_r})| \\
&= \sum_{v \in V(G)\setminus\{z\}} |\mathsf{r}_v(\overrightarrow{S}_{f_1,f_r})| + |\mathsf{r}_z(\overrightarrow{S}_{f_1,f_r})| \\
&= \sum_{v \in V(G)\setminus\{z\}} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})| + |\mathsf{r}_z(\overrightarrow{S}_{f_0,f_r}) - 1|
\end{aligned}$$

Here if a property $\mathsf{r}_z(\overrightarrow{S}_{f_0,f_r}) > 0$ holds, then we have the following succeeding formula:

$$\begin{aligned}
&\sum_{v \in V(G)\setminus\{z\}} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})| + |\mathsf{r}_z(\overrightarrow{S}_{f_0,f_r}) - 1| \\
&= \sum_{v \in V(G)\setminus\{z\}} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})| + |\mathsf{r}_z(\overrightarrow{S}_{f_0,f_r})| - 1 \\
&= \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})| - 1
\end{aligned}$$

Then we have $\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) = \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})|$. So we prove that the inequation $\mathsf{r}_z(\overrightarrow{S}_{f_0,f_r}) > 0$ holds.

Let $u \in \mathsf{Frozen}(f_0)$ be some frozen vertex. By Corollary 3 we have

$$\mathsf{p}_{f_r}(\overrightarrow{P}_{u,z}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,z}) = \mathsf{r}_z(\overrightarrow{S}_{f_0,f_r})$$

for any directed path $\overrightarrow{P}_{u,z}$ from u to z. $\overrightarrow{P}_{u,z}$ consists of the following three parts:

- $\overrightarrow{P}_{u,x}$ is a directed path whose vertices are all fixed (possible $u = x$). We have $\mathsf{p}_{f_r}(\overrightarrow{P}_{u,x}) = \mathsf{p}_{f_0}(\overrightarrow{P}_{u,x})$ since $f_0(v) = f_r(v)$ holds for any fixed vertex v.

- $\overrightarrow{P}_{x,y}$ is just an arc (x, y). We have $\mathsf{p}_{f_r}(\overrightarrow{P}_{x,y}) > \mathsf{p}_{f_0}(\overrightarrow{P}_{x,y})$ by the assumption of recoloring of step 2-2 of Algorithm 2.

- $\overrightarrow{P}_{y,z}$ is contained in $\overrightarrow{B}^+(x, f_0)$ (possibly $y = z$). By the definition of forward blocking graph, $\mathsf{p}_{f_0}(\overrightarrow{e}) = 1$ holds for any arc $\overrightarrow{e} \in A\overrightarrow{P}_{y,z}$, which is minimum potential value for an arc. Therefore $\mathsf{p}_{f_r}(\overrightarrow{P}_{y,z}) \geq \mathsf{p}_{f_0}(\overrightarrow{P}_{y,z})$ holds.

By the above three inequations, we have $\mathsf{r}_z(\overrightarrow{S}_{f_0,f_r}) = \mathsf{p}_{f_r}(\overrightarrow{P}_{u,z}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,z}) > 0$. Therefore Algorithm 2 finds an $(f_0 \to f_r)$-reconfiguration sequence whose length is $\mathsf{len}(()\overrightarrow{S}_{f_0,f_r} = \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})|$. □

Indeed, we can compute the value $\sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})| = \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v})|$ in $O(m)$ time.

The case $\mathsf{Frozen}(f_0) = \emptyset$

In this subsection we show how to find the shortest reconfiguration sequence in the case where $\mathsf{Frozen}(f_0) = \mathsf{Frozen}(f_r) = \emptyset$. By Lemma 17 and 18 we have the following

inequation for any $(f_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_r}$ on a graph G:

$$\begin{aligned}\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) &\geq \sum_{v \in V(G)} |\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})| \\ &= \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})| \end{aligned} \tag{4.9}$$

where u is arbitrary vertex and $\overrightarrow{P}_{u,v}$ is some arbitrary path from u to v. Notice that the lower bound of length $\mathsf{len}(\overrightarrow{S}_{f_0,f_r})$ depends on only colorings f_0, f_r and the rotation $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})$ of the vertex u. In the rest of this subsection we first show an algorithm which takes an integer parameter ρ as an aimed value of the rotation of the vertex u, and finds an $(f_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_r}$, satisfying $\rho = \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})$, whose length equals the lower bound shown in Eq. 4.9. Then we also show how to find the value of the parameter $\rho = \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})$ that minimizes right hand side of Eq. 4.9.

We first characterize the possible value of the parameter $\rho = \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})$ by the following lemma:

Lemma 20. *For any $(f_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_r}$ and any vertex v, $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r})$ is congruent to $f_r(v) - f_0(v)$ modulo k.*

Proof. We prove by the induction on the length $\mathsf{len}(\overrightarrow{S}_{f_0,f_r})$. If $\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) = 0$ then $f_0 = f_r$ therefore $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) = f_r(v) - f_0(v) = 0$ holds for any vertex v and congruence holds.

Otherwise, we consider the case $\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) > 0$. Then we have a coloring f_1 such that $\overrightarrow{S}_{f_0,f_r} = f_0 \overrightarrow{S}_{f_1,f_r}$. By induction hypothesis $\mathsf{r}_v(\overrightarrow{S}_{f_1,f_r}) \equiv f_r(v) - f_1(v) \pmod{k}$ i.e., there exists some integer q such that $\mathsf{r}_v(\overrightarrow{S}_{f_1,f_r}) + qk = f_r(v) - f_1(v)$ holds for any vertex $v \in V(G)$. Notice that We consider the following three cases:

- If $f_0(v)^+ = f_1(v)$ then $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) = \mathsf{r}_v(\overrightarrow{S}_{f_1,f_r}) + 1$. Then we consider the following two subcases:

  - If $f_0(v) = k$ and $f_1(v) = 1$, then

$$\begin{aligned} f_r(v) - f_0(v) =& f_r(v) - f_1(v) + 1 - k \\ =& \mathsf{r}_v(\overrightarrow{S}_{f_1,f_r}) + 1 + (q-1)k = \mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) + (q-1)k \end{aligned}$$

  - Otherwise $f_0(v) + 1 = f_1(v)$, then

$$\begin{aligned} f_r(v) - f_0(v) =& f_r(v) - f_1(v) + 1 \\ =& \mathsf{r}_v(\overrightarrow{S}_{f_1,f_r}) + 1 + qk = \mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) + qk \end{aligned}$$

- If $f_0(v)^- = f_1(v)$ then $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) = \mathsf{r}_v(\overrightarrow{S}_{f_1,f_r}) - 1$. Then we consider the following two subcases:

  - If $f_0(v) = 1$ and $f_1(v) = k$, then

$$\begin{aligned} f_r(v) - f_0(v) =& f_r(v) - f_1(v) - 1 + k \\ =& \mathsf{r}_v(\overrightarrow{S}_{f_1,f_r}) - 1 + (q+1)k = \mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) + (q+1)k \end{aligned}$$

  - Otherwise $f_0(v) - 1 = f_1(v)$, then

$$\begin{aligned} f_r(v) - f_0(v) =& f_r(v) - f_1(v) - 1 \\ =& \mathsf{r}_v(\overrightarrow{S}_{f_1,f_r}) - 1 + qk = \mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) + qk \end{aligned}$$

- If $f_0(v) = f_1(v)$ then $\mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) = \mathsf{r}_v(\overrightarrow{S}_{f_1,f_r})$. Then,

$$\begin{aligned} f_r(v) - f_0(v) =& f_r(v) - f_1(v) \\ =& \mathsf{r}_v(\overrightarrow{S}_{f_1,f_r}) + qk = \mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) + qk \end{aligned}$$

Therefore in every cases $f_r(v) - f_0(v) \equiv \mathsf{r}_v(\overrightarrow{S}_{f_0,f_r}) \pmod{k}$ holds. □

Indeed, for any vertex u and integer $\rho \equiv f_r(u) - f_0(u) \pmod{k}$, we can obtain an $(f_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_r}$ such that

$$\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) = \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})|$$

and $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_r}) = \rho$.

Lemma 21. *For any vertex u and integer $\rho \equiv f_r(u) - f_0(u) \pmod{k}$, there is a $(f_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_r}$ satisfying $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_r}) = \rho$ and*

$$\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) = \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})|.$$

We give an algorithm that gives initial fixed set F and coloring f_0' for the Algorithm 2 for the case $\mathsf{Frozen}(f_0) = \emptyset$. It takes the vertex u and an integer $\rho \equiv f_r(u) - f_0(u) \pmod{k}$ as parameter. Soundness of this algorithm is shown in the same way as the Algorithm 1.

Algorithm 3 (Initialization for Algorithm 2, with parameters u and ρ)

1. Let f initially be the coloring f_0.
2. If $\rho = 0$ then output f and $F = \{u\}$.
3. Otherwise,
   - if $\rho > 0$ then recolor some sink vertex v of $\overrightarrow{B}^+(u, f)$ from $f(v)$ to $f(v)^+$. Let f be newly obtained coloring and if $v = u$ then let ρ be $\rho - 1$; or
   - if $\rho < 0$ then recolor some source vertex v of $\overrightarrow{B}^-(u, f)$ from $f(v)$ to $f(v)^-$. Let f be newly obtained coloring and if $v = u$ then let ρ be $\rho + 1$.

Then return to the step 2.

Note that Algorithm 3 always outputs $\{u\}$ as fixed vertices for Algorithm 2.

Lemma 22. *For parameters $u \in V(G)$ and $\rho \equiv f_r(u) - f_0(u) \pmod{k}$, Algorithm 3 finds an $(f_0 \to f_0')$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_0'}$ such that $f_0'(u) = f_r(u)$ and $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_0'}) = \rho$.*

Proof. We prove first prove Algorithm 3 properly finds an $\overrightarrow{S}_{f_0,f_0'}$ such that $f_0'(u) = f_r(u)$ by the induction on $|\rho|$. If $\rho = 0$ then $f_r(u) - f_0(u) = 0$ and Algorithm 3 simply outputs $f_0' = f_0$. Therefore $f_r(u) = f_0(u) = f_0'(u)$. Otherwise, if $\rho > 0$ (respectively, $\rho < 0$) then Algorithm 3 recolors some sink (respectively, source) vertex v of $\overrightarrow{B}^+(u, f_0)$ to $f_0(v)^+$ (respectively, $\overrightarrow{B}^-(u, f_0)$ to $f_0(v)^-$). The existence of such a vertex is ensured by absence of frozen vertices, i.e., if $\overrightarrow{B}(u, f_0)$ contains some directed cycle then vertices contained in such a cycle must be frozen.

We treat only the case $\rho > 0$ because the case $\rho < 0$ is symmetric. In this case Algorithm 3 recolors some sink vertex v of $\overrightarrow{B}^+(u, f)$ from $f(v)$ to $f(v)'$. If $v \neq u$ then Algorithm 3 repeats until $|A(\overrightarrow{B}^+(u, f))| = 0$. Otherwise, Algorithm 3 recolors u from $f(u)$ to $f(u)'$ and decreases r to $r - 1$. Let f' be newly obtained coloring and $\rho' = \rho - 1$ where ρ is the value before recoloring. We consider the following two cases:

- If $f(u) = k$ and $f'(u) = 1$ then $f_r(u) - f'(u) = f_r(u) - f(u) + k - 1$.
- Otherwise $f(u) + 1 = f'(u)$ then $f_r(u) - f'(u) = f_r(u) - f(u) - 1$.

In both case the following congruence holds:

$$f_r(u) - f'(u) \equiv f_r(u) - f(u) - 1 \equiv \rho - 1 \equiv \rho' \pmod{k}$$

Therefore by the induction hypothesis in the remaining process Algorithm 3 outputs f'_0 such that $f'_0(u) = f_r(u)$ holds.

Then we prove $\mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0})$ by induction of the length $\mathsf{len}(\overrightarrow{S}_{f_0,f'_0})$. If $\mathsf{len}(\overrightarrow{S}_{f_0,f'_0}) = 0$ then we can assume that $\rho = 0$ and $\mathsf{len}(\overrightarrow{S}_{f_0,f'_0}) = \rho$ holds. Otherwise, we can assume $|r| > 0$ and there are some cases of recoloring of f_0:

- If $r > 0$ and there is some sink vertex v of $\overrightarrow{B}^+(u, f_0)$ that is not u, then $f_0(u) = f_1(u)$ and parameter ρ is unchanged therefore $\mathsf{r}_u(\overrightarrow{S}_{f_1,f'_0}) = \mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0})$ and $f_r(u) - f_1(u) = f_r(u) - f_0(u) \equiv \rho \pmod{k}$. By induction hypothesis $\mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0}) = \mathsf{r}_u(\overrightarrow{S}_{f_1,f'_0}) = \rho$ holds.

- If $r > 0$ and there is only one sink vertex $\overrightarrow{B}^+(u, f_0)$ namely u, then u is recolored from $f_0(u)$ to $f_0(u)^+ = f_1(u)$ and parameter ρ is decreased into $\rho - 1$. There are two subcases:

  - If $f_0(u) = k$ and $f_1(u) = 1$, then $f_r(u) - f_1(u) = f_r(u) - f_0(u) + k - 1 \equiv \rho - 1 \pmod{k}$. Then by induction hypothesis $\mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0}) - 1 = \mathsf{r}_u(\overrightarrow{S}_{f_1,f'_0}) = \rho - 1$ holds.

  - Otherwise, if $f_0(u) + 1 = f_1(u)$, then $f_r(u) - f_1(u) = f_r(u) - f_0(u) - 1 \equiv \rho - 1 \pmod{k}$. Then by induction hypothesis $\mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0}) - 1 = \mathsf{r}_u(\overrightarrow{S}_{f_1,f'_0}) = \rho - 1$ holds.

  In both cases we have $\mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0}) = \rho$.

- If $r < 0$ then we also have $\mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0}) = \rho$ but we omit the proof since it is symmetric.

□

Let f'_0 be the coloring output by Algorithm 3 and $\overrightarrow{S}_{f_0,f'_0}$ be the $(f_0 \to f'_0)$-reconfiguration sequence produced by Algorithm 3. Now we have fixed vertices $F = \{u\}$ for Algorithm 2, and then we can obtain an $(f'_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f'_0,f_r}$. We omit the proof of the following lemma because it can be proved in the same way as Lemma 19.

Lemma 23. *Let f'_0 and $F = \{u\}$ are coloring and fixed vertices which are output by Algorithm 3. For these parameters Algorithm 2 finds an $(f'_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f'_0,f_r}$ such that,*

$$\mathsf{len}(\overrightarrow{S}_{f'_0,f_r}) = \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f'_0}(\overrightarrow{P}_{u,v})|$$

Let $\overrightarrow{S}_{f_0,f_r}$ be a concatenation of two subsequences $\overrightarrow{S}_{f_0,f'_0}\overrightarrow{S}_{f'_0,f_r}$. Through the execution of Algorithm 2 vertex u is always fixed, therefore we have $\rho = \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r}) = \mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0}) + \mathsf{r}_u(\overrightarrow{S}_{f'_0,f_r}) = \mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0}) + 0 = \mathsf{r}_u(\overrightarrow{S}_{f_0,f'_0})$. The following lemma and its corollary shows that concatenation of Algorithm 3 and Algorithm 2 outputs $\overrightarrow{S}_{f_0,f_r}$ such that

$$\mathsf{len}(\overrightarrow{S}_{f_0,f_r}) = \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})|$$

which is the shortest $(f_0 \to f_r)$ with respect to fixed rotation $\rho = \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})$ of the vertex u.

Lemma 24. *Let f'_0 be the coloring obtained by Algorithm 3 with parameter $u \in V(G)$ and $\rho \equiv f_r(u) - f_0(u) \pmod{k}$. Then,*

$$\sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f'_0}(\overrightarrow{P}_{u,v})| = \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| - \mathsf{len}(\overrightarrow{S}_{f_0,f'_0}).$$

Proof. We prove by the induction on the length $\mathsf{len}(\overrightarrow{S}_{f_0,f'_0})$. If $\mathsf{len}(\overrightarrow{S}_{f_0,f'_0}) = 0$ i.e., $f_0 = f'_0$, then by the definition of Algorithm 3, ρ must be 0 and the formula

trivially holds. Otherwise if $\mathsf{len}(\overrightarrow{S}_{f_0,f_0'}) > 0$, then there is some coloring f_1 such that $\overrightarrow{S}_{f_0,f_0'} = f_0 \overrightarrow{S}_{f_1,f_0'}$. In this case $\rho \neq 0$ and we consider some cases of the execution of Algorithm 3.

- If $\rho > 0$ and u is recolored from $f_0(u)$ to $f_0(u)^+ = f_1(u)$, Then the rest execution of Algorithm 3 takes f_1 as an initial coloring and $\rho - 1$ as parameters. By induction hypothesis we have the following formula:

$$\begin{aligned}
&\sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0'}(\overrightarrow{P}_{u,v})| \\
&= \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + (\rho - 1)| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'})
\end{aligned}$$

Since we have $\mathsf{p}_f(\overrightarrow{P}_{u,u}) - \mathsf{p}_{f'}(\overrightarrow{P}_{u,u}) = 0$ for any coloring f, f', we obtain the following transformation:

$$\begin{aligned}
&\sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + (\rho - 1)| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) \\
&= \sum_{v \in V(G)\setminus\{u\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + (\rho - 1)| \\
&+ |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,u}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,u}) + (\rho - 1)| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) \\
&= \sum_{v \in V(G)\setminus\{u\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + (\rho - 1)| + |(\rho - 1)| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'})
\end{aligned}$$

Since we have $\rho > 0$, we obtain the following transformation:

$$\begin{aligned}
&\sum_{v \in V(G)\setminus\{u\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + (\rho - 1)| + |(\rho - 1)| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) \\
&= \sum_{v \in V(G)\setminus\{u\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - (\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + 1) + \rho| + |\rho| - (\mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) + 1)
\end{aligned}$$

Finally, since $\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + 1 = \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v})$ and $\mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) + 1 = \mathsf{len}(\overrightarrow{S}_{f_0,f_0'})$, we

have the desired formula:

$$\begin{aligned}
&\sum_{v\in V(G)\setminus\{u\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - (\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v})+1) + \rho| + |\rho| - (\mathsf{len}(\overrightarrow{S}_{f_1,f_0'})+1)\\
=&\sum_{v\in V(G)\setminus\{u\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| + |\rho| - \mathsf{len}(\overrightarrow{S}_{f_0,f_0'})\\
=&\sum_{v\in V(G)\setminus\{u\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho|\\
+&|\mathsf{p}_{f_r}(\overrightarrow{P}_{u,u}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,u}) + \rho| - \mathsf{len}(\overrightarrow{S}_{f_0,f_0'})\\
=&\sum_{v\in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| - \mathsf{len}(\overrightarrow{S}_{f_0,f_0'})
\end{aligned}$$

- If $\rho > 0$ and some vertex $w \neq u$ is recolored from $f_0(w)$ to $f_0(w)^+ = f_1(w)$, then the rest execution of Algorithm 3 takes f_1 as an initial coloring and ρ as parameters. By induction hypothesis we have the following formula:

$$\begin{aligned}
&\sum_{v\in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0'}(\overrightarrow{P}_{u,v})|\\
=&\sum_{v\in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) - \rho| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'})
\end{aligned}$$

Since we have $\mathsf{p}_{f_1}(\overrightarrow{P}_{u,w}) = \mathsf{p}_{f_0}(\overrightarrow{P}_{u,w}) + 1$ and $\mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) = \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v})$ for any vertex $v \neq w$, the following transformation holds:

$$\begin{aligned}
&\sum_{v\in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + \rho| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'})\\
=&\sum_{v\in V(G)\setminus\{w\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,v}) + \rho|\\
+&|\mathsf{p}_{f_r}(\overrightarrow{P}_{u,w}) - \mathsf{p}_{f_1}(\overrightarrow{P}_{u,w}) + \rho| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'})\\
=&\sum_{v\in V(G)\setminus\{w\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho|\\
+&|\mathsf{p}_{f_r}(\overrightarrow{P}_{u,w}) - (\mathsf{p}_{f_0}(\overrightarrow{P}_{u,w})+1) + \rho| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'})\\
=&\sum_{v\in V(G)\setminus\{w\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho|\\
+&|(\mathsf{p}_{f_r}(\overrightarrow{P}_{u,w}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,w})) + (\rho-1)| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'})
\end{aligned}$$

In the step 2 of Algorithm 3, there is a directed path from u to v contained in forward blocking graph $\overrightarrow{B}^+(u, f_0)$. Let $\overrightarrow{P}_{u,w}$ be such a directed path, then $\mathsf{p}_{f_0}(\overrightarrow{e}) = 1$ holds for any $\overrightarrow{e} \in A(\overrightarrow{P}_{u,w})$, which is minimum potential of a directed edge. Therefore we have $\mathsf{p}_{f_r}(\overrightarrow{P}_{u,w}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,w}) \geq 0$. Together with the assumption $r > 0$, we have the following transformation:

$$
\begin{aligned}
&\sum_{v \in V(G)\setminus\{w\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| \\
&+|(\mathsf{p}_{f_r}(\overrightarrow{P}_{u,w}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,w})) + (\rho - 1)| - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) \\
&= \sum_{v \in V(G)\setminus\{w\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| \\
&+(\mathsf{p}_{f_r}(\overrightarrow{P}_{u,w}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,w})) + (\rho - 1) - \mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) \\
&= \sum_{v \in V(G)\setminus\{w\}} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| \\
&+|\mathsf{p}_{f_r}(\overrightarrow{P}_{u,w}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,w}) + \rho| - (\mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) + 1) \\
&= \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| - (\mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) + 1)
\end{aligned}
$$

Since we have $\mathsf{len}(\overrightarrow{S}_{f_0,f_0'}) = \mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) + 1$, we obtain the desired formula:

$$
\begin{aligned}
&\sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| - (\mathsf{len}(\overrightarrow{S}_{f_1,f_0'}) + 1) \\
&= \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| - \mathsf{len}(\overrightarrow{S}_{f_0,f_0'})
\end{aligned}
$$

- The case $r < 0$ is symmetric therefore we omit the proof.

In all case we have the desired formula. □

By Lemma 22, 23, 24 and Eq. 4.9 we have the following corollary:

Corollary 4. *Let $\overrightarrow{S}_{f_0,f_0'}$ be a reconfiguration sequence obtained by Algorithm 3 with initial parameters $u \in V(G)$ and $\rho \equiv f_r(u) - f_0(u) \pmod{k}$, and $\overrightarrow{S}_{f_0',f_r}$ be a reconfiguration sequence obtained by Algorithm 2 with initial fixed vertices $F = \{u\}$.*

Then we have

$$\mathsf{len}(\overrightarrow{S}_{f_0,f_0'}) + \mathsf{len}(\overrightarrow{S}_{f_0',f_r}) = \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho|$$

and the reconfiguration sequence $\overrightarrow{S}_{f_0,f_0'}\overrightarrow{S}_{f_0',f_r}$ *is the shortest one among all reconfiguration sequence such that* $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_r}) = \rho$.

Proof. The formula follows from Lemma 23 and 24. By Lemma 22 we have $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_0'}) = \rho$. And by the fact that the vertex u is fixed through the execution of Algorithm 2, we have $\mathsf{r}_u(\overrightarrow{S}_{f_0',f_r}) = 0$. Therefore $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_r}) = \mathsf{r}_u(\overrightarrow{S}_{f_0,f_0'}) = \mathsf{r}_u(\overrightarrow{S}_{f_0',f_r}) = \rho$ holds. Then

$$\mathsf{len}(\overrightarrow{S}_{f_0,f_0'}) + \mathsf{len}(\overrightarrow{S}_{f_0',f_r}) = \sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})|$$

fits to the lower bound of the length of the reconfiguration sequence whose rotation $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_r})$ of u is ρ given by Eq. 4.9. □

Now we can compute the length of the shortest $(f_0 \to f_r)$-reconfiguration sequence $\overrightarrow{S}_{f_0,f_r}$ satisfying $\mathsf{r}_u(\overrightarrow{S}_{f_0,f_r}) = \rho$ for any integer $\rho \equiv f_r(u) - f_0(u) \pmod{k}$. To find the shortest $(f_0 \to f_r)$-reconfiguration sequence, we have to find the value of ρ which minimizes $\sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho|$. To find this value we exploit an well-known property in statistics, such that the median of real numbers minimizes the sum of absolute errors. Let $X' = (x_{i_1}, x_{i_2}, \ldots, x_{i_n})$ be a sequence obtained by sorting a sequence $X = (x_1, x_2, \ldots, x_n)$ of real number by increasing order. Then its *median* $Med(X)$ of the sequence X is defined as follows:

$$Med(X) = \begin{cases} x_{i_{\frac{n+1}{2}}} & \text{if } n \text{ is odd} \\ \left(x_{i_{\frac{n}{2}}} + x_{i_{\frac{n}{2}+1}}\right)/2 & \text{otherwise if } n \text{ is even.} \end{cases}$$

The following lemma shows that the function which takes real number y and returns the sum of absolute error of the sequence X from y is a convex function whose minimum is $y = Med(X)$

Lemma 25. *Let $X = (x_1, \ldots, x_n)$ be a sequence of real number. Then $f(y) = \sum_{x \in X} |x - y|$ is monotonically increasing on $y \geq Med(X)$ and monotonically decreasing on $y \leq Med(X)$.*

Proof. We only show the monotonically increasingness of $f(y)$ on $y \geq Med(X)$, because the proof of monotonically decreasingness is symmetric. Let a, b be real numbers such that $Med(X) \leq a \leq b$. In the rest of proof we show $f(a) \leq f(b)$. Let L_a, R_a, L_b, R_b be the sets $\{x \in X \mid x \leq a\}, \{x \in X \mid x > a\}, \{x \in X \mid x \leq b\}$ and $\{x \in X \mid x > b\}$ respectively.

$$
\begin{aligned}
& f(b) - f(a) \\
=& \sum_{x \in X} |x - b| - \sum_{x \in X} |x - a| \\
=& \left(\sum_{x \in L_b} (b - x) + \sum_{x \in R_b} (x - b) \right) - \left(\sum_{x \in L_a} (a - x) + \sum_{x \in R_a} (x - a) \right) \\
=& \left(\sum_{x \in L_b} (b - x) + \sum_{x \in L_a} (x - a) \right) - \left(\sum_{x \in R_a} (x - a) + \sum_{x \in R_b} (b - x) \right) \\
=& \left(\sum_{x \in L_b \setminus L_a} (b - x) + \sum_{x \in L_a} (b - x) + \sum_{x \in L_a} (x - a) \right) \\
& - \left(\sum_{x \in R_a \setminus R_b} (x - a) + \sum_{x \in R_b} (x - a) + \sum_{x \in R_b} (b - x) \right) \\
=& \left(\sum_{x \in L_b \setminus L_a} (b - x) + \sum_{x \in L_a} (b - a) \right) - \left(\sum_{x \in R_a \setminus R_b} (x - a) + \sum_{x \in R_b} (b - a) \right)
\end{aligned}
$$

Since $L_b \setminus L_a = R_a \setminus R_b = X \setminus (L_a \uplus R_b)$ and b is maximum of such a set, we have

the following succeeding transformation:

$$
\begin{aligned}
&\left(\sum_{x\in L_b\setminus L_a}(b-x)+\sum_{x\in L_a}(b-a)\right)-\left(\sum_{x\in R_a\setminus R_b}(x-a)+\sum_{x\in R_b}(b-a)\right)\\
=&\sum_{x\in X\setminus(L_a\uplus R_b)}(a+b-2x)+\sum_{x\in L_a}(b-a)-\sum_{x\in R_b}(b-a)\\
\geq&\sum_{x\in X\setminus(L_a\uplus R_b)}(a+b-2b)+\sum_{x\in L_a}(b-a)-\sum_{x\in R_b}(b-a)\\
=&\sum_{x\in X\setminus(L_a\uplus R_b)}(a-b)+\sum_{x\in L_a}(b-a)-\sum_{x\in R_b}(b-a)\\
=&(|X|-|L_a|-|R_b|)(a-b)+|L_a|(b-a)-|R_b|(b-a)\\
=&(2|L_a|-|X|)(b-a)
\end{aligned}
$$

Since $a \geq Med(X)$, we have $|L_a| \geq |X|/2$ by the property of median. Thus, we have $(2|L_a| - |X|)(b-a) \geq 0$. By above discussion, we have $f(b) - f(a) \geq 0$. Therefore monotonically increasingness hold for $y \geq Med(X)$.

□

The following lemma complete the proof of Theorem 12.

Lemma 26. *Let $V(G) = \{v_1, v_2, \ldots, v_n\}$ and u be arbitrary vertex, and let $X = (x_1, x_2, \ldots, x_n)$ be a sequence where $x_i = \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v})$. Then*

$$
\begin{aligned}
&\min\left\{\sum_{v\in V(G)}\left|\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v})-\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v})+\rho\right| : \rho\equiv f_r(u)-f_0(u) \pmod{k}\right\}\\
=&\min\left\{\sum_{v\in V(G)}\left|\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v})-\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v})+\rho\right| :\right.\\
&\left.\rho\in\left\{\left\lfloor\frac{Med(X)-(f_r(u)-f_0(u))}{k}\right\rfloor,\left\lceil\frac{Med(X)-(f_r(u)-f_0(u))}{k}\right\rceil\right\}\right\}
\end{aligned}
$$

Proof. We want to obtain the value $\rho \equiv f_r(u) - f_0(u) \pmod{k}$ which minimizes the value $\sum_{v\in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho| = \sum_{v\in V(G)} |\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \rho|$ of Eq. 4.8. By Lemma 25, $\rho = Med(X)$ minimizes this value, however $\rho \equiv (f_r(u) -$

$f_0(u))$ (mod k) is cannot always be $Med(X)$. Therefore we choose ρ as either least $\rho \equiv (f_r(u) - f_0(u))$ (mod k) which is at least $Med(X)$, or greatest $\rho \equiv (f_r(u) - f_0(u))$ (mod k) which is at most $Med(X)$, that is

$$\rho \in \left\{ \left\lfloor \frac{Med(X) - (f_r(u) - f_0(u))}{k} \right\rfloor, \left\lceil \frac{Med(X) - (f_r(u) - f_0(u))}{k} \right\rceil \right\}.$$

□

We finally claim that $\mathsf{dist}(f_0, f_r)$ can be computed in linear time, based on Theorem 12, and that a shortest $(f_0 \to f_r)$-reconfiguration sequence can be output in polynomial time.

Lemma 27. *For any vertex $u \in V(G)$, two integers $\rho_{u,1}$ and $\rho_{u,2}$ of Theorem 12(b) can be obtained in $O(n+m)$ time. Furthermore,*

(*a*) $\mathsf{dist}(f_0, f_r)$ *can be computed in $O(n+m)$ time; and*

(*b*) *a shortest $(f_0 \to f_r)$-reconfiguration sequence can be output in $O(kn(n+m))$ time.*

We prove Lemma 27 by the following three lemmas. First we show (a) for the case where $\mathsf{Frozen}(f_0) \neq \emptyset$.

Lemma 28. *The value of Eq. 4.7 can be computed in $O(n+m)$ time.*

Proof. It suffices to show that we can compute the values $\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v})$ for all vertices $v \in V(G)$ in $O(m)$ total time. Notice that the starting point u is fixed, then we can use the same technique used in Lemma 16. We construct a spanning rooted tree rooted by the vertex u in $O(n+m)$ time. Then we can compute $\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v})$ for all vertices in $O(n)$ time by successive addition of potentials of arcs. The whole computation takes $O(n+m)$ time. □

For the other case where $\mathsf{Frozen}(f_0) = \emptyset$ we exploit the median of medians algorithm [2] to compute the value of Eq. 4.8 in $O(n+m)$ time. Finally we show that we can compute the value of Eq. 4.8 in $O(m)$ time using well-known the median of medians algorithm [2].

Lemma 29. *The values $\rho_{u,1}$ and $\rho_{u,2}$ of Theorem 12(b) and the value of Eq. 4.8 can be computed in $O(n+m)$ time.*

Proof. Let $V(G) = \{v_1, v_2, \ldots, v_n\}$. We first compute the values $\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v})$ and $\mathsf{p}_{f_0}(\overrightarrow{P}_{u,v})$ for all vertices in $O(n+m)$ total time using the same method as Lemma 28. Then we compute the median $Med(X)$ of the sequence $X = (x_1, x_2, \ldots, x_n)$ where $x_i = \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_r}(\overrightarrow{P}_{u,v})$ in $O(n)$ time, using median of medians algorithm [2] which finds ith largest element of given sequence of n elements of total ordered set in $O(n)$ time for any $i \leq n$. Then by Lemma 26 we can determine two choices $\rho_{u,1}$ and $\rho_{u,2}$ of ρ which minimizes $\sum_{v \in V(G)} |\mathsf{p}_{f_r}(\overrightarrow{P}_{u,v}) - \mathsf{p}_{f_0}(\overrightarrow{P}_{u,v}) + \rho|$ in $O(n)$ time. Therefore computation of $\rho_{u,1}$ and $\rho_{u,2}$ takes $O(n+m)$ time.

Once $\rho_{u,1}$ and $\rho_{u,2}$ is determined, the value of Eq. 4.8 can be computed in $O(n+m)$ time by the same way as the proof of Lemma 28. □

Finally we show that we can obtain the actual shortest $(f_0 \to f_r)$-reconfiguration sequence in $O(kn(n+m))$ time.

Lemma 30. *Then a shortest $(f_0 \to f_r)$-reconfiguration sequence can be output in $O(kn(n+m))$ time.*

Proof. For the case $\mathsf{Frozen}(f_0) \neq \emptyset$, by the proof of Lemma 19 we can obtain the shortest $(f_0 \to f_r)$-reconfiguration sequence by Algorithm 2. Its running time is estimated as follows: Since the step 2 decreases the size of $V(G^{\mathsf{f}}) \setminus F$ exactly one,

the iteration of step 1 to 2 occurs at most n times. For the iteration of the step 2-1 to 2-2 we need to go into detail. Recoloring of the picked vertex v occurs at most k time. We need to recolor a sink (source) vertex w of forward blocking graph $\overrightarrow{B}^+(v,f)$ (backward blocking graph $\overrightarrow{B}^-(v,f)$). However we need not to reconstruct the blocking graph at each iteration of the step 2-2. While the color of the vertex v has been unchanged, the recoloring can be done by the following procedure:

1. Let B be forward blocking graph $\overrightarrow{B}^+(v,f)$ (backward blocking graph $\overrightarrow{B}^-(v,f)$) if $f(v) < f_r(v)$ ($f(v) > f_r(v)$).

2. Choose a sink (source) vertex w and recolor it to $f(w)^+$ ($f(w)^-$), then delete w from B.

3. Repeat the step 2 until the vertex v is recolored.

Therefore we need to construct the blocking graph at most k time. The construction of B takes $O(n+m)$ time and after construction of B, at most $|V(B)| = O(n)$ step recoloring occurs until v is recolored, therefore whole recoloring step takes $n \cdot k \cdot (O(n+m) + O(n)) = O(kn(n+m))$ time.

For the case $\mathsf{Frozen}(f_0) = \emptyset$, the recoloring algorithm consists of three parts: Finding the value ρ which minimizes the value of Eq. 4.8, Algorithm 3, and Algorithm 2. As mentioned above, Algorithm 2 can be computed in $O(kn(n+m))$ time. By Lemma 29 the value of ρ can be found in $O(n+m)$ time. The running time of Algorithm 3 can be estimated as $O(k(n+m))$ in the same way as Algorithm 2. Therefore for the case we need $O(n+m) + O(k(n+m)) + O(kn(n+m))$ time. □

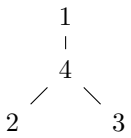


Figure 4.3: A claw graph.

4.4 Algorithm for Claw Recolorability

In this section we show that COLORING RECONFIGURATION UNDER R-RECOLORABILITY can be solved in polynomial time if the recolorability graph R is *claw*, which is a complete bipartite graph $K_{1,3}$, shown in Figure .

Theorem 13. COLORING RECONFIGURATION UNDER R-RECOLORABILITY *can be solved in polynomial time if R is a claw graph.*

Proof. Let $R = (\{1,2,3,4\}, \{\{1,4\},\{2,4\},\{3,4\}\})$. We show that any instance of COLORING RECONFIGURATION UNDER R-RECOLORABILITY can be transformed into an instance of 3-COLORING RECONFIGURATION. Let f_0, f_r be the initial and target colorings a graph G of COLORING RECONFIGURATION UNDER R-RECOLORABILITY. Let u be a vertex in $\{v \in V(G) \mid f_0(v) = 4\}$. We recolor u to arbitrary color in $\{1,2,3\} \setminus \{f_0(v) \mid v \in N(G,u)\}$, if $\{1,2,3\} \setminus \{f_0(v) \mid v \in N(G,u)\} \neq \emptyset$. We repeat it until any further recoloring of a vertex occurs. Let f_0' be the obtained coloring. In the same way f_r' can be obtained. Notice that a vertex colored 4 and its neighbor vertices are frozen on f_0', f_r', therefore we assume that $\bigcup\{N(G,v) \mid v \in V(G), f_0'(v) = 4\} = \bigcup\{N(G,v) \mid v \in V(G), f_r'(v) = 4\}$ and for any $v \in \bigcup\{N(G,v) \mid v \in V(G), f_0'(v) = 4\}$, $f_0'(v) = f_r'(v)$ holds, otherwise the instance is no-instance. Let $V_i = \{v \mid v \in N(G,u), f_0'(v) = i, f_0'(u) = 4\}$. We contract the vertices in V_i into a vertex v_i, and construct a clique by joining v_1, v_2, v_3 by edges. Then we remove all vertices colored 4 on f_0', and we obtain an instance

of 3-COLORING RECONFIGURATION. Let G_T be the obtained graph and f'_{0T}, f'_{rT} be the obtained colorings.

We show that the instance of COLORING RECONFIGURATION UNDER R-RECOLORABILITY is reachable if and only if the obtained instance of 3-COLORING RECONFIGURATION is reachable. The proof is similar to the proof of Theorem .

We first prove if part. Let $\langle f'_{0T}, f'_{1T}, \ldots, f'_{\ell T}\rangle$ be a reconfiguration sequence for 3-COLORING RECONFIGURATION, where $f'_{\ell T} = f'_{rT}$. Since v_1, v_2, v_3 are frozen and only the color of $V(G_T) \setminus \{v_1, v_2, v_3\}$ changes, for each f'_{iT} there is a corresponding coloring f'_i of G such that $f'_i|_{V(G)\setminus\{v\in V(G)|f'_0(v)=4\}} = f'_{iT}|_{V(G_T)\setminus\{v_1,v_2,v_3\}}$. For each $i \in \{0, 1, \ldots, \ell-1\}$ we can construct a coloring $f'_{i,i+1}$ such that

$$f'_{i,i+1}(v) = \begin{cases} 4 & \text{if } f'_i(v) \neq f'_{i+1}(v); \\ f'_i(v) & \text{otherwise} \end{cases}$$

which is adjacent to f'_i and f'_{i+1}. Therefore $\langle f'_0, f'_{0,1}, f'_1, \ldots, f'_{\ell-1,\ell}, f'_\ell\rangle$ is an reconfiguration sequence of COLORING RECONFIGURATION UNDER R-RECOLORABILITY.

We then prove the only-if direction. Let $\langle f'_0, f'_1, \ldots, f'_\ell\rangle$ be a reconfiguration sequence where $f'_\ell = f'_r$. We construct a sequence of colorings $\langle f''_0, f''_1, \ldots, f''_\ell\rangle$ of G_T as follows:

$$f''_i(v) = \begin{cases} i & \text{if } v = v_i \in \{v_1, v_2, v_3\}; \\ f'_i(v) & \text{if } v \notin \{v_1, v_2, v_3\} \text{ and } f_i(v) \in \{1, 2, 3\}; \\ f'_{i-1}(v) & \text{if } v \notin \{v_1, v_2, v_3\} \text{ and } f_i(v) = 4. \end{cases}$$

Note that $f''_0 = f'_{0T}$ and $f''_\ell = f'_{\ell T}$ hold. We now claim that the sequence $\langle f''_0, f''_1, \ldots, f''_\ell\rangle$ is a (redundant) reconfiguration sequence of G_T, by proving the following (a) and (b):

(a) f''_i is a 4-coloring of G for each $i \in \{0, 1, \ldots, \ell\}$; and

(b) $\left|\{v \in V(G) : f''_i(v) \neq f''_{i+1}(v)\}\right| \leq 1$ for each $i \in \{0, 1, \ldots, \ell-1\}$.

We first prove the claim (a) above. If adjacent vertices v, w are colored $1, 2$ or 3 in f'_i, they have different colors not only in f'_i but also in f''_i. Let w be a vertex in G such that $f'_i(w) = 4$. The vertex w is colored with $f'_q(w) \in \{1, 2, 3\}$ such that $q = \max\{j \mid 0 \leq j \leq i-1, f'_j(w) \neq 4\}$. Then, $f'_j(w) = 4$ for all $j \in \{q+1, q+2, \ldots, i\}$ and hence every vertex $v \in N(G, w)$ is colored with the same color $f'_q(v)$, because any recoloring must be made via the color 4. Therefore no neighbor of w has the same color of w in f''_i and f''_i is a proper 3-coloring.

We finally prove the claim (b) above. Let v be the vertex which is recolored between f'_i and f'_{i+1} for $i \in \{0, 1, \ldots, \ell-1\}$. If v is recolored from a color in $\{1, 2, 3\}$ to 4, then we have $f''_i = f''_{i+1}$ and the claim holds. Note that v stays with the same color $f''_i(v) = f''_{i+1}(v)$ until it is recolored some color in $\{1, 2, 3\}$. Therefore, the claim holds also for the case where v is recolored from a color in $\{1, 2, 3, 4\}$ to another color $\{1, 2, 3\}$, because only v is recolored between f'_i and f'_{i+1}.

□

Chapter 5

Directed recolorability

In this chapter we generalize the concept of recolorability graph into digraph and study the complexity status of the generalized problem. For a color set $C = \{1, 2, \ldots, k\}$, *directed recolorability graph* $\overrightarrow{R}$ is a digraph whose vertex set $V(G)$ is the color set C. We define COLORING RECONFIGURATION UNDER DIRECTED $\overrightarrow{R}$-RECOLORABILITY by means of reconfiguration graph; let G be a graph and $\overrightarrow{R}$ be a directed recolorability graph which is defined on a color set C of size k, then *directed $\overrightarrow{R}$-reconfiguration graph* $\overrightarrow{\mathcal{C}}_{\overrightarrow{R}}(G)$ is a digraph defined as follows: the vertex set $V(\overrightarrow{\mathcal{C}}_{\overrightarrow{R}}(G))$ is the set of all k-colorings of G, and for two k-colorings f, f' of G, there is an arc from $(f, f') \in A(\overrightarrow{\mathcal{C}}_{\overrightarrow{R}}(G))$ if the following two conditions hold:

- $|\{v \in V(G) \mid f(v) \neq f'(v)\}| = 1$; and
- if $f(v) \neq f'(v)$ then $(f(v), f'(v)) \in A(\overrightarrow{R})$.

Informally, the color c of a vertex can be changed into other color c' if the resulting color assignment is a proper k-coloring of G, and there is an arc $(c, c') \in A(\overrightarrow{R})$. For two colorings f, f', a $(f \to f')$-*reconfiguration sequence* is a directed path from f to f' on $\overrightarrow{\mathcal{C}}_{\overrightarrow{R}}(G)$.

In Section 5.1 we show that COLORING RECONFIGURATION UNDER DIRECTED

$\overrightarrow{R}$-RECOLORABILITY is NP-hard even if the directed recolorability graph is a polytree of maximum outdegree two and maximum indegree two. On the other hand, in Section 5.2 we give a polynomial-time algorithm of the problem for the case where the directed recolorability graph is a rooted tree. Note that if the directed recolorability graph is acyclic, then the problem is in NP since no vertex is colored to a same color twice, i.e., there is no sequence of coloring $\langle f_0, f_1, \ldots, f_\ell \rangle$ such that $\langle f_0(v), f_1(v), \ldots, f_\ell(v) \rangle = \langle c, c', \ldots, c \rangle$ for a vertex v and two different colors c, c', then the length of any reconfiguration sequence is at most $|V(G)| \cdot (|V(\overrightarrow{R})| - 1) = n(k-1)$.

5.1 NP-hardness for polytree

In this section we show the following:

Theorem 14. *There is a polytree of maximum outdegree two and maximum indegree two such that* COLORING RECONFIGURATION UNDER DIRECTED $\overrightarrow{R}$-RECOLORABILITY *is NP-hard.*

Proof. As well as the proof of Theorem 4, we show the NP-hardness by a polynomial-time reduction from 3-coloring of planar graphs [13]. Every planar graph is 4-colorable [1], and 4-coloring of a planar graph can be computed in polynomial-time [23]. Let G be a given planar graph and we first give a 4-coloring $f_F : V(G) \to \{1, 2, 3, 4\}$ of G in polynomial-time. Then we separate the vertex set $V(G)$ into four subsets V_1, V_2, V_3, V_4, where $V_i = \{v \in V(G) \mid f_F(v) = i\}$.

We construct an instance of COLORING RECONFIGURATION UNDER DIRECTED R-RECOLORABILITY fro m the graph G. Let G' be an undirected graph defined as

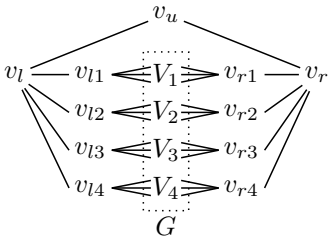


Figure 5.1: A graph G'.

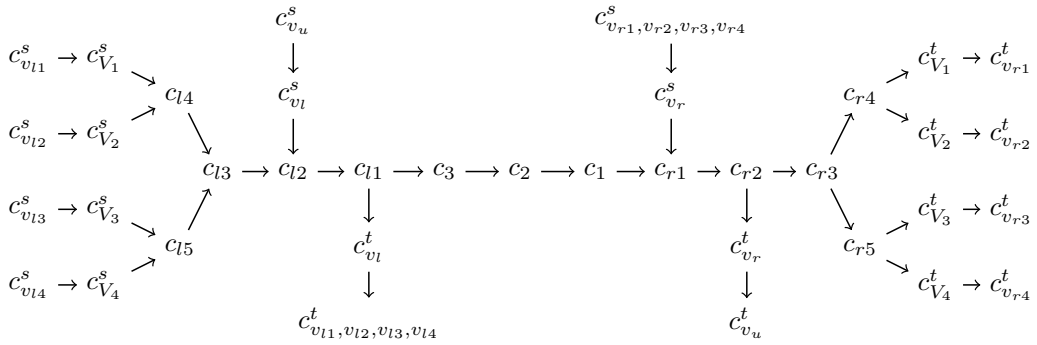


Figure 5.2: Directed recolorability graph which is a polytree.

follows (see also Figure 5.1):

$$V(G') = V(G) \cup \{v_{l1}, v_{l2}, v_{l3}, v_{l4}, v_l, v_u, v_r, v_{r1}, v_{r2}, v_{r3}, v_{r4}\}$$

$$E(G') = E(G) \cup \bigcup_{i\in\{1,2,3,4\}} \{v_{li}v \mid v \in V_i\} \cup \bigcup_{i\in\{1,2,3,4\}} \{v_{ri}v \mid v \in V_i\}$$
$$\cup \{v_{l1}v_l, v_{l2}v_l, v_{l3}v_l, v_{l4}v_l, v_lv_u, v_uv_r, v_rv_{r1}, v_rv_{r2}, v_rv_{r3}, v_rv_{r4}\}$$

We also define a directed recolorability graph $\overrightarrow{R}$ as follows (see also Figure 5.2):

$$
\begin{aligned}
V(\overrightarrow{R}) = \{&c^s_{v_{l1}}, c^s_{v_{l2}}, c^s_{v_{l3}}, c^s_{v_{l4}}, c^t_{v_{l1},v_{l2},v_{l3},v_{l4}}, c^s_{V_1}, c^s_{V_2}, c^s_{V_3}, c^s_{V_4}, c^s_{v_l}, c^t_{v_l},\\
&c_{l1}, c_{l2}, c_{l3}, c_{l4}, c_{l5}, c^s_{v_u}, c_1, c_2, c_3, c^t_{v_u}, c_{r1}, c_{r2}, c_{r3}, c_{r4}, c_{r5},\\
&c^s_{v_r}, c^t_{v_r}, c^t_{V_1}, c^t_{V_2}, c^t_{V_3}, c^t_{V_4}, c^s_{v_{r1},v_{r2},v_{r3},v_{r4}}, c^t_{v_{r1}}, c^t_{v_{r2}}, c^t_{v_{r3}}, c^t_{v_{r4}}\}\\
A(\overrightarrow{R}) = \{&(c^s_{v_{l1}}, c^s_{V_1}), (c^s_{v_{l2}}, c^s_{V_2}), (c^s_{v_{l3}}, c^s_{V_3}), (c^s_{v_{l4}}, c^s_{V_4}),\\
&(c^s_{V_1}, c_{l4}), (c^s_{V_2}, c_{l4}), (c^s_{V_3}, c_{l5}), (c^s_{V_4}, c_{l5}),\\
&(c_{l4}, c_{l3}), (c_{l5}, c_{l3}), (c_{l3}, c_{l2}), (c_{l2}, c_{l1}),\\
&(c^s_{v_u}, c^s_{v_l}), (c^s_{v_l}, c_{l2}), (c_{l1}, c^t_{v_l}), (c^t_{v_l}, c^t_{v_{l1},v_{l2},v_{l3},v_{l4}}),\\
&(c_{l1}, c_3), (c_3, c_2), (c_2, c_1), (c_1, c_{r1}),\\
&(c^s_{v_{r1},v_{r2},v_{r3},v_{r4}}, c^s_{v_r}), (c^s_{v_r}, c_{r1}), (c_{r2}, c^t_{v_r}), (c^t_{v_r}, c^t_{v_u}),\\
&(c_{r1}, c_{r2}), (c_{r2}, c_{r3}), (c_{r3}, c_{r4}), (c_{r3}, c_{r5}),\\
&(c_{r4}, c^t_{V_1}), (c_{r4}, c^t_{V_2}), (c_{r5}, c^t_{V_3}), (c_{r5}, c^t_{V_4}),\\
&(c^t_{V_1}, c^t_{v_{r1}}), (c^t_{V_2}, c^t_{v_{r2}}), (c^t_{V_3}, c^t_{v_{r3}}), (c^t_{V_4}, c^t_{v_{r4}})\},
\end{aligned}
$$

where each element in $V(\overrightarrow{R})$ represents some color in $C = \{1, 2, \ldots, 37\}$. Then we define initial and target colorings f_0, f_t of G' as follows:

$$
f_0(v) = \begin{cases} c^s_{V_i} & \text{if } v \in V_i \\ c^s_{v_{r1},v_{r2},v_{r3},v_{r4}} & \text{if } v \in \{v_{r1}, v_{r2}, v_{r3}, v_{r4}\} \\ c^s_v & \text{otherwise;} \end{cases}
$$

$$
f_t(v) = \begin{cases} c^t_{V_i} & \text{if } v \in V_i \\ c^t_{v_{l1},v_{l2},v_{l3},v_{l4}} & \text{if } v \in \{v_{l1}, v_{l2}, v_{l3}, v_{l4}\} \\ c^t_v & \text{otherwise.} \end{cases}
$$

We now prove that there is an $(f_0 \to f_r)$-reconfiguration sequence on $\overrightarrow{\mathcal{C}}_{\overrightarrow{R}}(G')$ if and only if there is a 3-coloring of G. We first prove the if part. For a vertex v and two colors c, c' such that there is an directed path $\langle c = c_0, c_1, \ldots, c_\ell = c' \rangle$ from c to

c' on $\overrightarrow{\mathcal{C}}_{\overrightarrow{R}}(G')$, we say *recolor v from c to c'* to express a successive ℓ recoloring steps of v from $c = c_0$ to $c_1, c_2, \ldots, c_\ell = c'$. Let $f_T : V(G) \to \{1, 2, 3\}$ be a 3-coloring of G. An $(f_0 \to f_r)$-reconfiguration sequence can be obtained by following steps:

1. Recolor vertices in $\{v \in V(G) \mid f_T(v) = 1\}$ from $c^s_{V_i}$ to c_1 for each $i \in \{1, 2, 3, 4\}$.

2. Recolor vertices in $\{v \in V(G) \mid f_T(v) = 2\}$ from $c^s_{V_i}$ to c_2 for each $i \in \{1, 2, 3, 4\}$.

3. Recolor vertices in $\{v \in V(G) \mid f_T(v) = 3\}$ from $c^s_{V_i}$ to c_3 for each $i \in \{1, 2, 3, 4\}$.

4. Recolor v_{li} from $c^s_{v_{li}}$ to $c^t_{v_{l1}, v_{l2}, v_{l3}, v_{l4}}$, for each $i \in \{1, 2, 3, 4\}$.

5. Recolor v_l from $c^s_{v_l}$ to $c^t_{v_l}$.

6. Recolor v_u from $c^s_{v_u}$ to $c^t_{v_u}$.

7. Recolor v_r from $c^s_{v_r}$ to $c^t_{v_r}$.

8. Recolor v_{ri} from $c^t_{v_{r1}, v_{r2}, v_{r3}, v_{r4}}$, to $c^s_{v_{ri}}$ for each $i \in \{1, 2, 3, 4\}$.

9. Recolor vertices in $\{v \in V(G) \mid f_T(v) = 1\}$ from c_1 to $c^t_{V_i}$ for each $i \in \{1, 2, 3, 4\}$.

10. Recolor vertices in $\{v \in V(G) \mid f_T(v) = 2\}$ from c_2 to $c^t_{V_i}$ for each $i \in \{1, 2, 3, 4\}$.

11. Recolor vertices in $\{v \in V(G) \mid f_T(v) = 3\}$ from c_3 to $c^t_{V_i}$ for each $i \in \{1, 2, 3, 4\}$.

For each step we take a vertex v and recolor it from some color c to another color c'. Notice that there is no adjacent vertex of v having any color on the directed path from c to c' on $\overrightarrow{\mathcal{C}}_{\overrightarrow{R}}(G')$, therefore we can execute this recoloring process.

Then we prove the only-if part. We assume there is an $(f_0 \to f_r)$-reconfiguration sequence $\mathcal{S}$. Let $v \in V_1$. We pick some colorings from $\mathcal{S}$ as follows:

- f_{v,c_3} is the first coloring on the sequence $\mathcal{S}$ such that $f_{v,c_3}(v) = c_3$.
- $f_{v_{l1}}$ is the first coloring on the sequence $\mathcal{S}$ such that $f_{v_{l1}}(v_{l1}) = c^t_{v_{l1},v_{l2},v_{l3},v_{l4}}$.
- f_{v_l} is the first coloring on the sequence $\mathcal{S}$ such that $f_{v_l}(v_l) = c^t_{v_l}$.
- f_{v_u} is the first coloring on the sequence $\mathcal{S}$ such that $f_{v_u}(v_u) = c^t_{v_u}$.
- f_{v_r} is the first coloring on the sequence $\mathcal{S}$ such that $f_{v_r}(v_r) = c^t_{v_r}$.
- $f_{v_{r1}}$ is the first coloring on the sequence $\mathcal{S}$ such that $f_{v_{r1}}(v_{r1}) = c_{r2}$.
- $f_{v,c_{r1}}$ is the first coloring on the sequence $\mathcal{S}$ such that $f_{v,c_{r1}}(v) = c_{r1}$.

For convenience we denote $f_i < f_{i'}$ for two colorings $f_i, f_{i'} \in \mathcal{S} = \langle f_0, f_1, \ldots, f_t \rangle$ if $i < i'$. The following relationships hold for the above colorings:

- $f_{v_{l1}}$ must occur after $f_{v,c3}$ on $\mathcal{S}$. $(f_{v,c3} < f_{v_{l1}})$
- f_{v_l} must occur after $f_{v_{l1}}$ on $\mathcal{S}$. $(f_{v_{l1}} < f_{v_l})$
- f_{v_u} must occur after f_{v_l} on $\mathcal{S}$. $(f_{v_l} < f_{v_u})$
- f_{v_r} must occur f_{v_u} on $\mathcal{S}$. $(f_{v_u} < f_{v_r})$
- $f_{v_{r1}}$ must occur f_{v_r} on $\mathcal{S}$. $(f_{v_r} < f_{v_{r1}})$
- $f_{v,c_{r1}}$ must occur $f_{v_{r1}}$ on $\mathcal{S}$. $(f_{v_{r1}} < f_{v,c_{r1}})$

Therefore $f_{v,c3} < f_{v_{l1}} < f_{v_l} < f_{v_u} < f_{v_r} < f_{v_{r1}} < f_{v,c_{r1}}$ holds. Between $f_{v,c3}$ and $f_{v,c_{r1}}$ on $\mathcal{S}$ the color of the vertex v is either c_1, c_2 or c_3, hence $f_{v_u}(v) \in \{c_1, c_2, c_3\}$. For vertices $w \in V_2 \cup V_3 \cup V_4$, we can prove $f_{v_u}(w) \in \{c_1, c_2, c_3\}$ in similar way. Then $f_{v_u}|_{V(G)}$ is a 3-coloring of G. □

Notice that in the proof of Theorem 14 the color of a vertex traverses the shortest path between initial color and target color of the vertex, hence the number of steps required to reach from initial coloring to target coloring can be calculated: it is $12 \cdot |V(G)| + 7 \cdot 4 + 3 + 10 + 3 + 7 \cdot 4 = 12 \cdot |V(G)| + 72$ and depends only on the size of the planar graph G. Conversely, for an instance of COLORING RECONFIGURATION UNDER R-RECOLORABILITY where R is the underlying graph of Figure 5.2 and the graph and its initial and target coloring can be obtained as same as the proof of Theorem 14 from a planar graph G, if we give a restriction of the number of steps to be $12 \cdot |V(G)| + 72$, to reach from the initial coloring to the target coloring we must trace the same reconfiguration sequence as the proof of Theorem 14. Therefore we obtain the following corollary:

Corollary 5. COLORING RECONFIGURATION UNDER R-RECOLORABILLITY *is NP-complete if the number of steps is restricted by unary number and even if the recolorability graph R is a tree of maximum degree three.*

5.2 Algorithm for rooted tree

By Theorem 14 COLORING RECONFIGURATION UNDER DIRECTED $\overrightarrow{R}$-RECOLORABILITY is NP-hard for polytree. In this section we show that the problem is polynomial-time solvable for rooted tree, more restricted class of polytree.

Lemma 31. *Let $\overrightarrow{R}$ be a rooted tree and $f_0, f_r : V(G) \to V(\overrightarrow{R})$ be colorings of a graph G. Then there is an $(f_0 \to f_r)$-reconfiguration sequence on $\overrightarrow{\mathcal{C}}_{\overrightarrow{R}}(G)$ if and only if the following two statements hold:*

(a) *for each vertex $v \in V(G)$, there is a directed path from $f_0(v)$ to $f_r(v)$ on $\overrightarrow{\mathcal{C}}_{\overrightarrow{R}}(G)$; and*

(b) *for each edge $vw \in E(G)$, the directed path from $f_0(v)$ to $f_r(v)$ does not contain the directed path from $f_0(w)$ to $f_r(w)$.*

Proof. Let f_0, f_r be initial and target colorings of the graph G satisfying two conditions (a) and (b). A directed path $\overrightarrow{P}_{f_0(v)\to f_r(v)}$ from $f_0(v)$ to $f_r(v)$ on $\overrightarrow{R}$ is uniquely specified for each vertex $v \in V(G)$. We define *distance* between f_0 and f_r as $\sum\{\text{length of } \overrightarrow{P}_{f_0(v)\to f_r(v)} \mid v \in V(G)\}$. We prove the lemma by the induction on the distance between f_0 and f_r. Clearly $f_0 = f_r$ if and only if the distance between f_0 and f_r is 0. If $\overrightarrow{P}_{f_0(v)\to f_r(v)} = \langle c_0, c_1, \ldots, c_\ell \rangle$ is of length at least 1, we define $next(v) = c_1$, which is, roughly speaking, the color which the vertex to be recolored to. Notice that the distance between current coloring and target coloring decreases exactly 1 if we recolor a vertex v to $next(v)$. We prove that if $f_0 \neq f_r$ then there exists a vertex v which can be recolored $next(v)$ and the resulting coloring also satisfies the conditions (a) and (b), and then by induction hypothesis the claim holds.

We construct a directed graph $\overrightarrow{G'}$ as follows:

$$V(\overrightarrow{G'}) = V(G)$$
$$A(\overrightarrow{G'}) = \{(v, w) \mid v, w \in V(G), (f_0(v), f_0(w)) \in A(\overrightarrow{R})\}$$

There is no directed cycle in G' since if it exists then $\overrightarrow{R}$ must have a directed cycle.

Therefore there are two cases: (A) $\overrightarrow{G'}$ has no arc, or (B) $\overrightarrow{G'}$ has a sink vertex of indegree at least one. If the case (A) holds, any vertex for which $next(v)$ is defined has no neighbor colored $next(v)$, therefore we can recolor a vertex v to $next(v)$. Otherwise, if case (B) holds, let v be some sink vertex of indegree at least one on $\overrightarrow{G'}$, and let w be a vertex such that $(w, v) \in \overrightarrow{G'}$. If $next(v)$ is not defined, then $\overrightarrow{P}_{f_0(v)\to f_r(v)} = \langle f_0(v)\rangle$. Since $(w, v) \in \overrightarrow{G'}$, $\overrightarrow{P}_{f_0(w)\to f_r(w)} = \langle f_0(w), next(w) = f_0(v), \dots\rangle$ therefore $\overrightarrow{P}_{f_0(w)\to f_r(w)}$ contains $\overrightarrow{P}_{f_0(w)\to f_r(w)}$, which contradicts the assumption (a). Therefore $next(v)$ is defined and v can be recolored to $next(v)$.

Then we prove the recoloring preserves the conditions (a) and (b). Let f'_0 be the coloring obtained by recoloring a vertex v to $next(v)$ on f_0. Since we recolor a vertex v to $next(v)$, the condition (a) also holds on f_0. We prove that the condition (b) holds for f'_0 by contradiction. There are two cases:

- If $\overrightarrow{P}_{f'_0(v)\to f_r(v)}$ contains $\overrightarrow{P}_{f'_0(w)\to f_r(w)}$ for neighbor w of v, since $(f_0(v), f'_0(v)) \in A(\overrightarrow{R})$, $\overrightarrow{P}_{f_0(v)\to f_r(v)}$ contains $\overrightarrow{P}_{f'_0(v)\to f_r(v)}$, and since $f'_0(w) = f_0(w)$, $\overrightarrow{P}_{f_0(v)\to f_r(v)}$ contains $\overrightarrow{P}_{f_0(w)\to f_r(w)}$ which contradicts the assumption (b).

- If $\overrightarrow{P}_{f'_0(v)\to f_r(v)}$ is contained by $\overrightarrow{P}_{f'_0(w)\to f_r(w)} = \overrightarrow{P}_{f_0(w)\to f_r(w)} = \langle c_0, c_1, \dots, c_\ell\rangle$ for neighbor w of v, since v, w are adjacent, $f'_0(v) \neq f'_0(w)$. Then $f'_0(v) = c_i$ for some $i \in \{1, \dots, \ell\}$. Since $\overrightarrow{R}$ is a rooted tree $f_0(v)$ such that $(f_0(v), f'_0(v)) \in A(\overrightarrow{R})$ is uniquely specified as $f_0(v) = c_{i-1}$. Therefore $\overrightarrow{P}_{f_0(v)\to f_r(v)}$ is contained $\overrightarrow{P}_{f_0(w)\to f_r(w)}$ which contradicts the assumption (b).

□

Chapter 6

Edge-coloring reconfiguration

In this chapter we deal with EDGE-COLORING RECONFIGURATION, which is COLORING RECONFIGURATION where the input graph is restricted to line graphs. Ito et al. [18] showed that LIST EDGE-COLORING RECONFIGURATION is PSPACE-complete even for planar graphs of maximum degree three, using six colors. Since 3-COLORING RECONFIGURATION is solved in linear time, LIST EDGE COLORING RECONFIGURATION is solved in polynomial time if the number of colors is at most three, using polynomial time reduction of Theorem 1.

In this chapter we show two results. First, we show that LIST EDGE-COLORING RECONFIGURATION is PSPACE-complete even for planar graphs of maximum degree three and bounded bandwidth, if the number of colors is four. As the second result we show that EDGE-COLORING RECONFIGURATION is PSPACE-completeness even for planar graph and bounded bandwidth quadratic to k, where k is the number of colors at least 5.

We prove the first claim in the next theorem:

Theorem 15. *For every number of k of colors at least four, the* LIST EDGE-COLORING RECONFIGURATION *problem for planar graphs of bounded bandwidth and maximum degree three.*

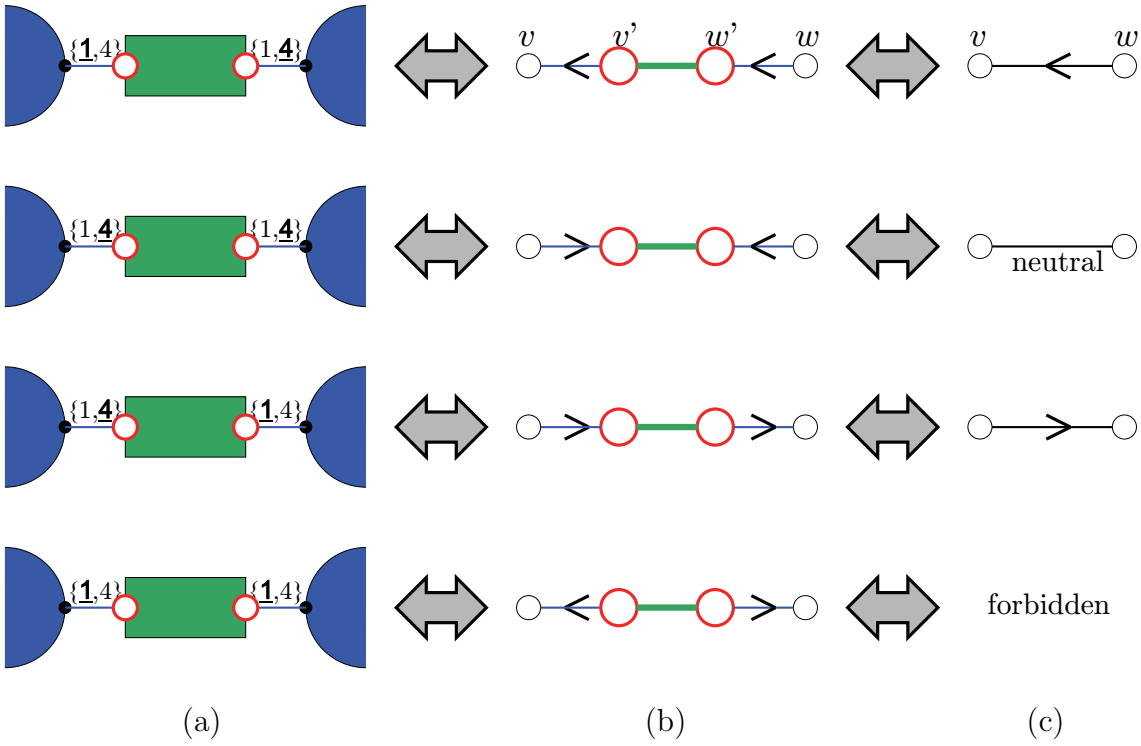


Figure 6.1: (a) Color assignments to connector edges, (b) their corresponding orientations of the edges vv' and ww', and (c) the corresponding orientations of an NCL edge vw.

Proof. Similarly as in 3, we prove by a reduction from NCL.

Link edge gadget

Recall that, in a given NCL machine, two NCL vertices v and w are joined by a single NCL edge vw. Therefore, the link edge gadget between v and w should be consistent with the orientations of the NCL edge vw, as follows (see also Figure 6.1):

If we assign the color 1 to the connector edge vv' (i.e., the inward direction for v), then ww' must be colored with 4 (i.e., the outward direction for w); conversely, vv' must be colored with 4 if we assign 1 to ww'. In particular, the gadget must forbid a list edge-coloring which assigns 1 to both vv' and ww' (i.e., the inward directions for both v and w), because such a coloring corresponds to the direction which illegally contributes to both v and w at the same time. On the other hand, assigning 4 to

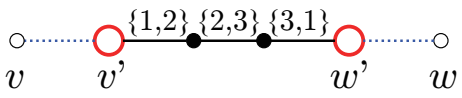


Figure 6.2: Link edge gadget with two connector edges vv' and ww' for LIST EDGE-COLORING RECONFIGURATION.

both vv' and ww' (i.e., the outward directions for both v and w) corresponds to the *neutral* orientation of the NCL edge vw which contributes to neither v nor w, and hence we simply do not care such an orientation.

Figure 6.2 illustrates our link edge gadget between two NCL vertices v and w. Figure 6.3(b) illustrates the "reconfiguration graph" of this link edge gadget together with two connector edges vv' and ww': each rectangle represents a node of the reconfiguration graph, that is, a list edge-coloring of the gadget, where the underlined bold number represents the color assigned to the edge, and two rectangles are joined by an edge in the reconfiguration graph if their corresponding list edge-colorings are adjacent. Then, the reconfiguration graph is connected as illustrated in Figure 6.3(b), and the link edge gadget has no list edge-coloring which assigns 1 to the two connector edges vv' and ww' at the same time, as required. Furthermore, the reversal of the NCL edge vw can be simulated by the path via the neutral orientation of vw, as illustrated in Figure 6.3(a). Thus, this link edge gadget works correctly.

AND gadget

Consider an NCL AND vertex v. Figure 6.4(a) illustrates all valid orientations of the three connector edges for v; each box represents a valid orientation of the three connector edges for v, and two boxes are joined by an edge if their orientations are adjacent. We construct our AND gadget so that it correctly simulates this reconfiguration graph in Figure 6.4(a).

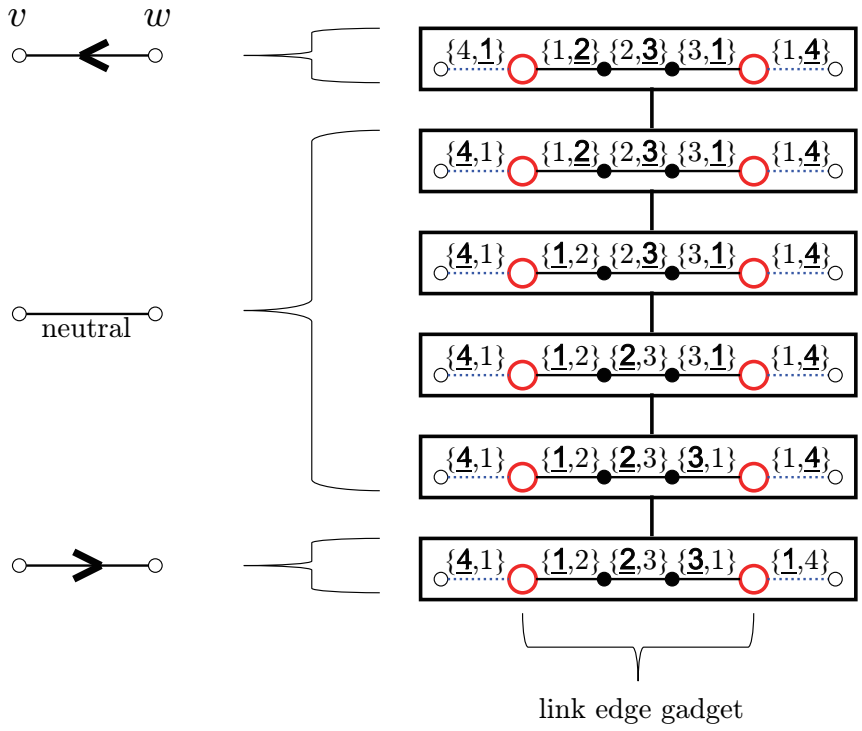


Figure 6.3: (a) Three orientations of an NCL edge vw, and (b) all list edge-colorings of the link edge gadget with two connector edges.

Figure 6.5 illustrates our AND gadget for each NCL AND vertex v, where e_1, e_2 and e_a correspond to the three connector edges for v such that e_1 and e_2 come from the two weight-1 NCL edges and e_a comes from the weight-2 NCL edge. Figure 6.4(b) illustrates the reconfiguration graph for all list edge-colorings of the AND gadget, where each large dotted box surrounds all colorings having the same color assignments to the three connector edges for v. Then, we can see that these list edge-colorings are "internally connected," that is, any two list edge-colorings in the same dotted box are reconfigurable with each other without recoloring any connector edge. Furthermore, this gadget preserves the "external adjacency" in the following sense: if we contract the list edge-colorings in the same dotted box in Figure 6.4(b)

into a single vertex, then the resulting graph is exactly the graph depicted in Figure 6.4(a). Therefore, we can conclude that our AND gadget correctly works as an NCL AND vertex.

OR gadget

Figure 6.6 illustrates our OR gadget for each NCL OR vertex v, where e_1, e_2 and e_3 correspond to the three connector edges for v. To verify that this OR gadget correctly simulates an NCL OR vertex, it suffices to show that this gadget satisfies both the internal connectedness and the external adjacency. Since this gadget has 1575 list edge-colorings, we have checked these sufficient conditions by a computer search of all list edge-colorings of the gadget.

Reduction

As we have explained before, we replace each of link edges and stars of NCL AND/OR vertices with its corresponding gadget; let G be the resulting graph. Since NCL remains PSPACE-complete even if an input NCL machine is planar, bounded bandwidth and of maximum degree three [26], the resulting graph G is also planar, bounded bandwidth and of maximum degree three; notice that, since each gadget consists of only a constant number of edges, the bandwidth of G is also bounded.

In addition, we construct two list edge-colorings of G which correspond to two given NCL configurations C_0 and C_r of the NCL machine. Note that there are (in general, exponentially) many list edge-colorings which correspond to the same NCL configuration. However, by the construction of the three gadgets, no two distinct NCL configurations correspond to the same list edge-coloring of G. We arbitrarily choose two list edge-colorings f_0 and f_r of G which correspond to C_0 and C_r, respectively.

This completes the construction of our corresponding instance of LIST EDGE-

COLORING RECONFIGURATION. Clearly, the construction can be done in polynomial time.

Correctness

We now prove that there exists a desired sequence of NCL configurations between C_0 and C_r if and only if there exists a reconfiguration sequence between f_0 and f_r.

We first prove the only-if direction. Suppose that there exists a desired sequence of NCL configurations between C_0 and C_r, and consider any two adjacent NCL configurations C_{i-1} and C_i in the sequence. Then, only one NCL edge vw changes its orientation between C_{i-1} and C_i. Notice that, since both C_{i-1} and C_i are valid NCL configurations, the NCL AND/OR vertices v and w have enough in-coming arcs even without vw. Therefore, we can simulate this reversal by the reconfiguration sequence of list edge-colorings in Figure 6.3(b) which passes through the neutral orientation of vw as illustrated in Figure 6.3(a). Recall that both AND and OR gadgets are internally connected, and preserve the external adjacency. Therefore, any reversal of an NCL edge can be simulated by a reconfiguration sequence of list edge-colorings of G, and hence there exists a reconfiguration sequence between f_0 and f_r.

We now prove the if direction. Suppose that there exists a reconfiguration sequence $\langle f_0, f_1, \ldots, f_\ell \rangle$ from f_0 to $f_\ell = f_r$. Notice that, by the construction of gadgets, any list edge-coloring of G corresponds to a valid NCL configuration such that some NCL edges may take the neutral orientation. In addition, f_0 and f_r correspond to valid NCL configurations without any neutral orientation. Pick the first index i in the reconfiguration sequence $\langle f_0, f_1, \ldots, f_\ell \rangle$ which corresponds to changing the direction of an NCL edge vw to the neutral orientation. Then, since the neutral orientation contributes to neither v nor w, we can simply ignore the change of the

NCL edge vw and keep the direction of vw as the same as the previous direction. By repeating this process and deleting redundant orientations if needed, we can obtain a sequence of valid adjacent orientations between C_0 and C_r such that no NCL edge takes the neutral orientation.

□

Then we prove the following theorem for the non-list variant.

Theorem 16. *For every integer $k \geq 5$, the* EDGE-COLORING RECONFIGURATION *problem is PSPACE-complete for planar graphs of bandwidth quadratic in k and maximum degree k.*

To prove the theorem, similarly as in the previous theorem, we will construct three types of gadgets corresponding to a link edge and stars of NCL AND/OR vertices. However, since we deal with the non-list variant, every edge has all k colors as its available colors. Thus, we construct one more gadget, called a *color gadget*, which restricts the colors available for the edge. The gadget is simply a star having k leaves, and we assign the k colors to the edges of the star in both f_0 and f_r. (See Figure 6.7(a) as an example for $k = 5$.) Note that, since the color set consists of only k colors, these k edges must stay the same colors in any reconfiguration sequence. Thus, if we do not want to assign a color c to an edge e, then we connect the leaf edge with the color c to an endpoint v of e. (See Figure 6.7(b).) In this way, we can treat the edge e as if it has the list $L(e)$ of available colors. However, we need to pay attention to the fact that all edges e' sharing the endpoint v cannot receive the color c by connecting such color gadgets to v. Therefore, our gadgets are constructed so that all endpoints of connector edges shared by other gadgets are attached with the same color gadgets which forbid colors 2 and $5, 6, \ldots, k$, and

hence we can connect the gadgets consistently.

Figures 6.8 and 6.9 illustrate all gadgets for the non-list variant, where the available colors for each edge is attached as the list of the edge. Notice that the gadgets in Figure 6.8 forbid the colors i (and j), and $6, 7, \ldots, k$, where $i, j \leq 5$. We again emphasize that all (red) endpoints of connector edges shared by other gadgets are attached with the same color gadgets which forbid colors 2 and $5, 6, \ldots, k$, and hence we can connect the gadgets consistently. Then, the link edge gadget and AND/OR gadgets have 10, 40, 297752 edge-colorings, respectively. We have checked that all gadgets satisfy both the internal connectedness and the external adjacency by a computer search of all edge-colorings of the gadgets. Therefore, by the same arguments as the proof of Theorem 15, we can conclude that an instance of NCL is a yes-instance if and only if the corresponding instance of EDGE-COLORING RECONFIGURATION is a yes-instance.

Recall that NCL remains PSPACE-complete even if an input NCL machine is planar, bounded bandwidth, and of maximum degree three [26]. Thus, the resulting graph G is also planar. Notice that only the size of the color gadget depends on k, and the other (parts of) gadgets are of constant sizes. Since $k \geq 5$, the maximum degree of G is k, i.e., the degree of the center of each color gadget. In addition, since each gadget in Figure 6.8 contains $O(k)$ color gadgets, it contains $O(k^2)$ edges. Therefore, the number of edges in each gadget in Figure 6.9 can be bounded by a quadratic in k. Since the bandwidth of the input NCL machine is a constant, that of G can be bounded by a quadratic in k.

This completes the proof of Theorem 16. □

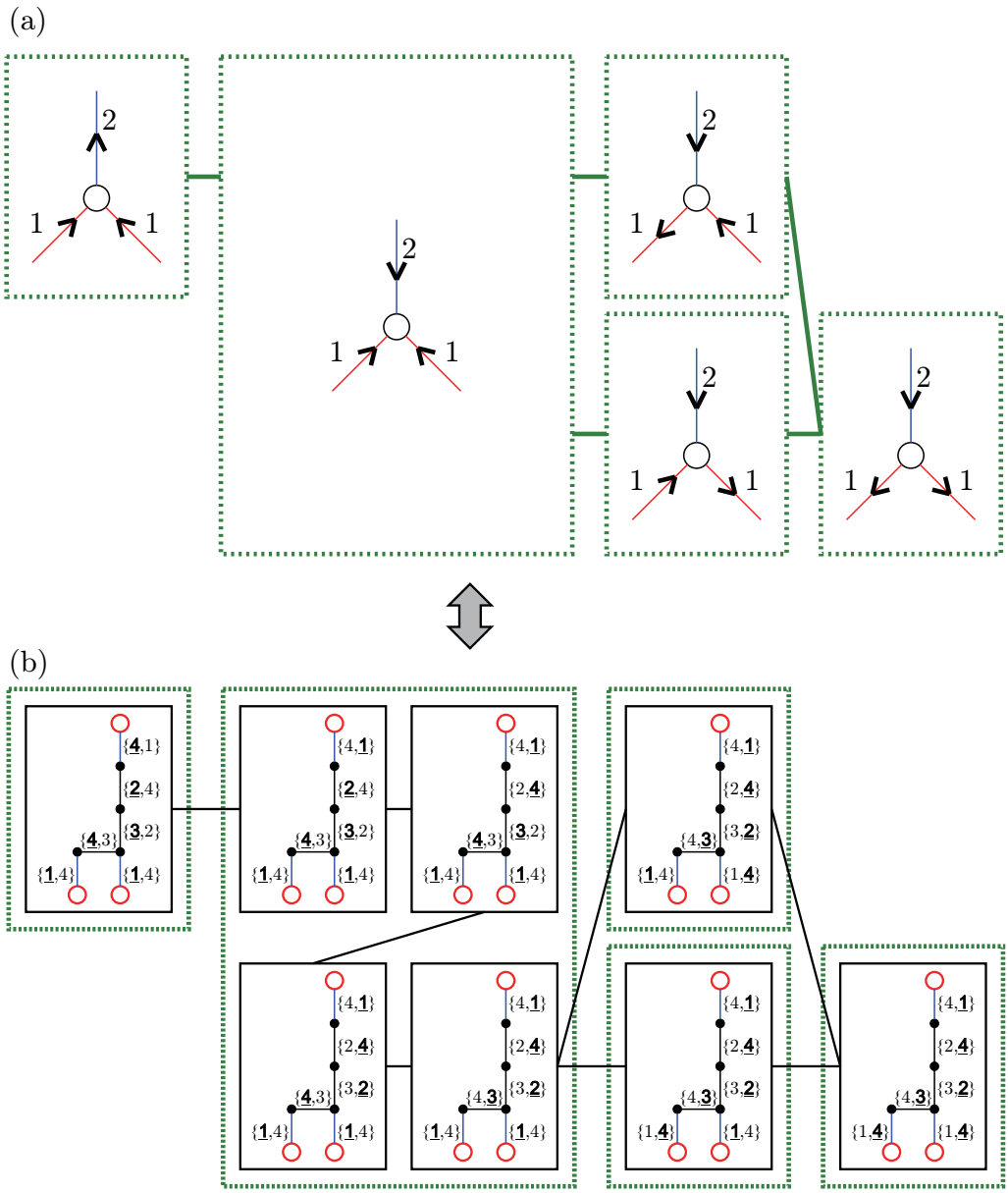


Figure 6.4: (a) All valid orientations of the three connector edges for an NCL AND vertex v, and (b) all list edge-colorings of the AND gadget in Figure 6.5.

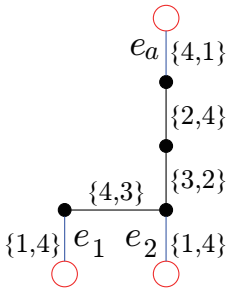


Figure 6.5: AND gadget for LIST EDGE-COLORING RECONFIGURATION.

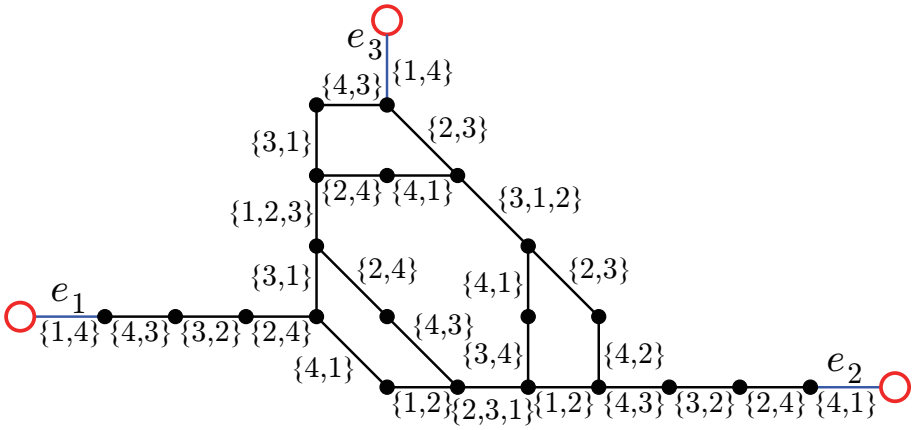


Figure 6.6: OR gadget for LIST EDGE-COLORING RECONFIGURATION.

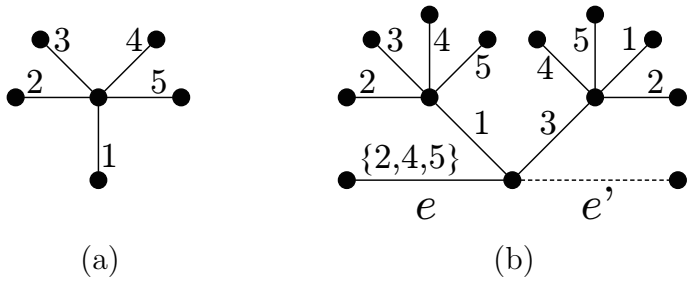


Figure 6.7: (a) Gadget for restricting a color ($k = 5$), and (b) edge e whose available colors are restricted to $\{2, 4, 5\}$.

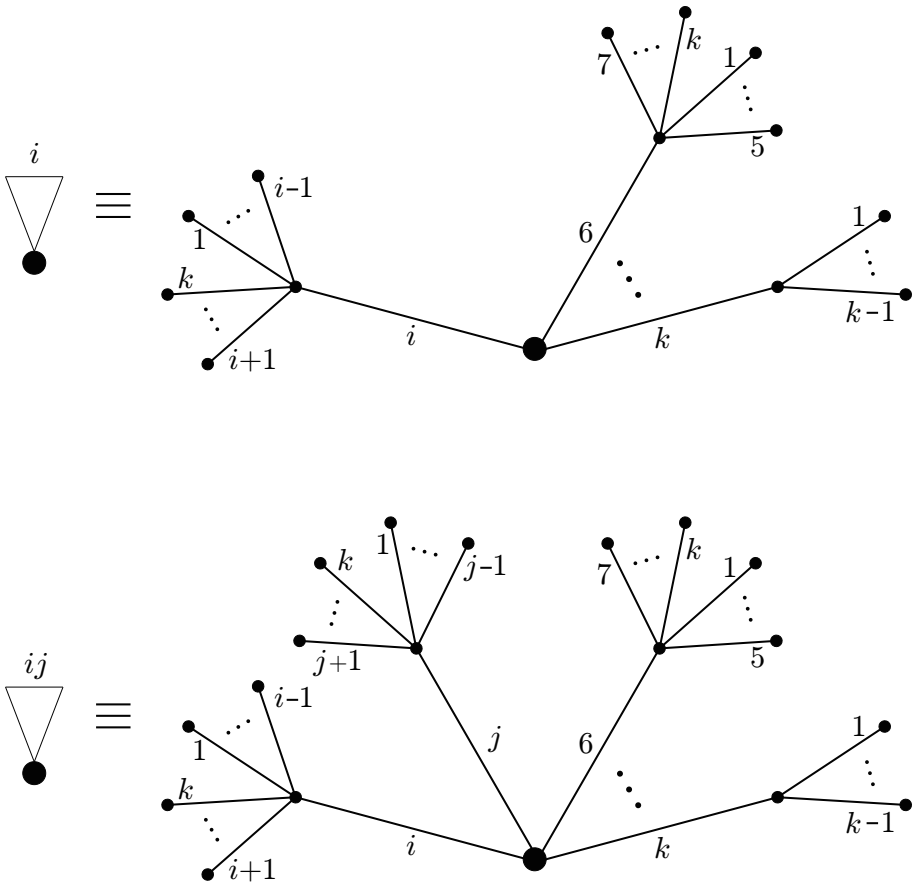
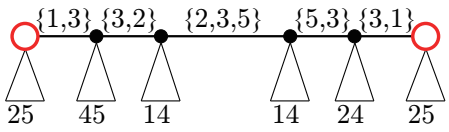
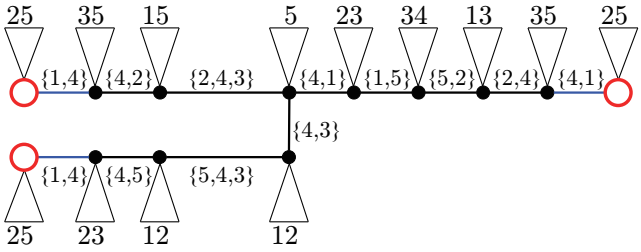


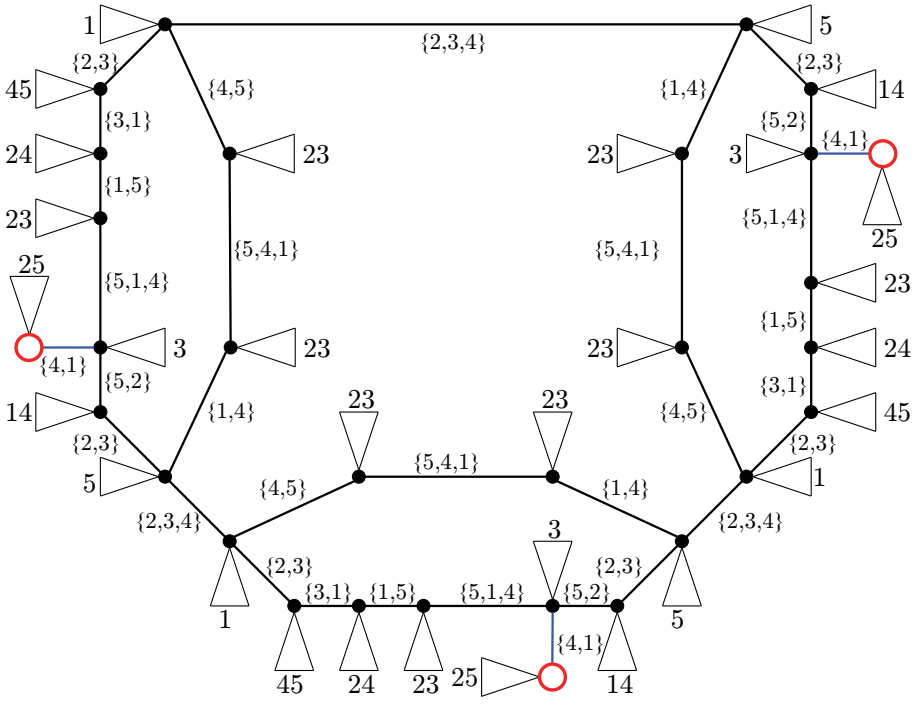
Figure 6.8: Explanatory note for the gadgets in Figure 6.9.



link edge gadget



AND gadget



OR gadget

Figure 6.9: Link edge, AND, and OR gadgets for the non-list variant.

Chapter 7

Conclusions

In this thesis, we studied COLORING RECONFIGURATION problem and its generalizations from the viewpoints of recolorability and irreversible rules.

In Chapter 3 and 4 we investigated the complexity status of COLORING RECONFIGURATION UNDER R-RECOLORABILITY from the viewpoint of the structure of the recolorability graph R. We showed that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is linear-time solvable for any recolorability graph R of maximum degree two, while there is a recolorability graph R of maximum degree three such that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is PSPACE-complete. We also showed that not all recolorability graph R of maximum degree three makes COLORING RECONFIGURATION UNDER R-RECOLORABILITY PSPACE-complete: in Section 4.4 we showed that COLORING RECONFIGURATION UNDER R-RECOLORABILITY is polynomial-time solvable if R is claw graph.

In Chapter 5 we studied the irreversible rules of COLORING RECONFIGURATION problem by a further generalization of recolorability graph, directed recolorability graph. We showed that COLORING RECONFIGURATION UNDER $\overrightarrow{R}$-RECOLORABILITY is NP-hard if R is polytree, while it is polynomial-time solvable if R is rooted tree. As a corollary of NP-hardness, we also showed that length re-

stricted version of COLORING RECONFIGURATION UNDER R-RECOLORABILITY is NP-complete.

In Chapter 6 we studied the computational hardness of LIST EDGE-COLORING RECONFIGURATION and EDGE-COLORING RECONFIGURATION. We sharply classified the complexity status of LIST EDGE-COLORING RECONFIGURATION from the viewpoint of the number of colors. We also gave the first hardness result for EDGE-COLORING RECONFIGURATION. As an open question, the complexity status of EDGE-COLORING RECONFIGURATION with number of colors four is still unknown.

In this thesis we introduced the idea of recolorability which parameterizes the structure of transformation rules, and we explored the complexity status of COLORING RECONFIGURATION problem with respect to the recolorability. In the past investigation of the reconfiguration problem, such a parameterization has been studied in limited way. Most of reconfiguration problems have single transformation rule, and few one has multiple, but finite number of transformation rules. For instance, INDEPENDENT SET RECONFIGURATION has three types of transformation rule [20]. Transformation rules for a reconfiguration problem is selected in "suitable" way, but their validity has not been argued. Our future work is to give exhaustive analysis of transformation rules of such problems by parameterizing the transformation rules.

Appendix A

Source codes for Chapter 6

We list source files of the program of verification of OR-gadgets of Theorem 15 and Theorem 16. All of the source codes are written in Haskell, and confirmed to run in ghc version 8.6.5. **ListEdgeColoring.hs** and **EdgeColoring.hs** correspond to Theorem 15 and Theorem 16. Each files import the common module **Crur.hs**. In this program the color is treated as an integer, and the colorings of the gadgets are lists of integer indexed by edge. For example, the list $[1, 2]$ assigns the color one to first edge, and assigns color 2 to second edge. The numberings of the edges of OR-gadgets are drawn in Figure A.1 and Figure A.2.

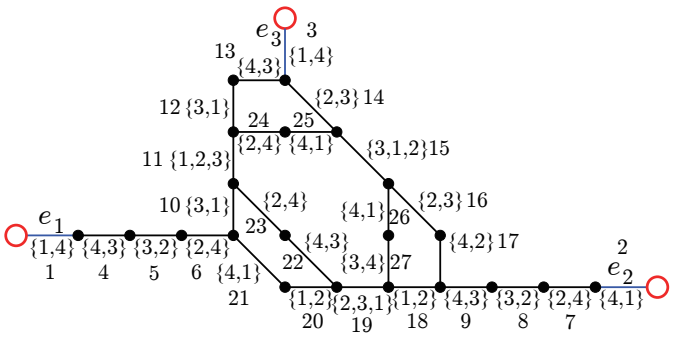


Figure A.1: Indices of edges of OR-gadget of LIST EDGE-COLORING RECONFIGURATION.

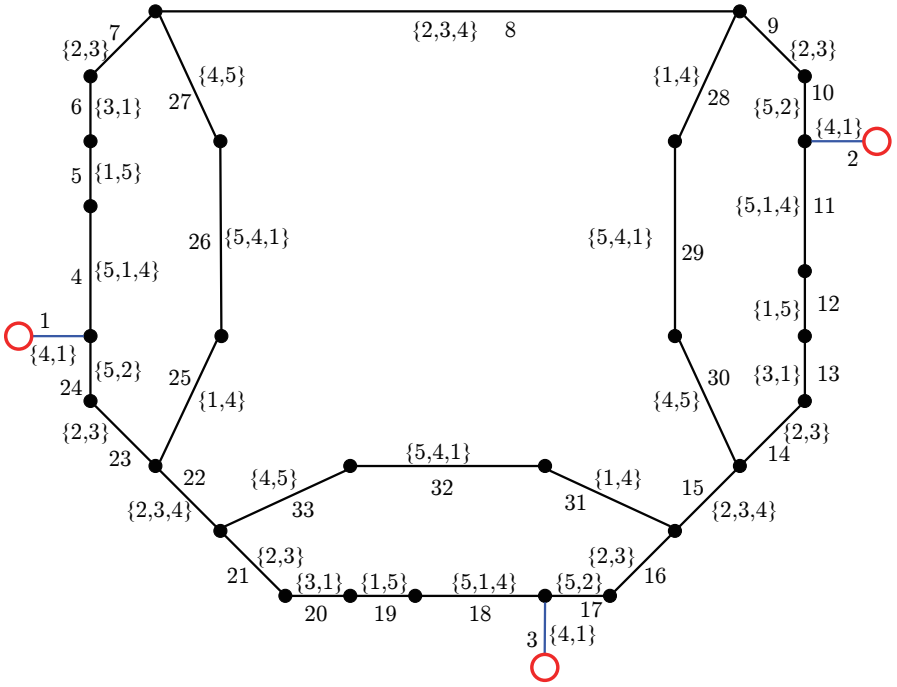


Figure A.2: Indices of edges of OR-gadget of EDGE-COLORING RECONFIGURATION.

Crur.hs

```
1  module Crur where
2  import Data.Array (Array,(!),bounds,indices,listArray)
3  import Data.Maybe (fromJust)
4  import Data.Set (Set, empty, insert, notMember)
5
6  type Vertex = Int
7  type Adjacency = [Vertex]
8  type GraphAL = [Adjacency] -- Adjacency list
9  type GraphAA = Array Int Adjacency -- Adjacency array
10
11 type Color = Int
12 type Rec = [Maybe [Color]]
13 type LRec = [Rec]
14 type LRecA = Array Int (Array Int (Maybe [Color]))
15
16 type Coloring = [Color]
17 type SemiColoring = [Maybe Color]
18
19 showColoring :: Coloring -> String
20 showColoring = concat . map show
21
22 gAL2GAA :: GraphAL -> GraphAA
23 gAL2GAA l = listArray (1,length l) l
24
25 lRec2LRecA :: LRec -> LRecA
26 lRec2LRecA l =
27   listArray (1,length l)
28   $ map (\l'->listArray (1,length l') l') l
29
30 isProperColoring :: GraphAA -> Coloring -> Bool
31 isProperColoring g c =
32   let
33     ca = listArray (bounds g) c
34     il = indices g
35   in
36     all (\i -> all (\j -> ca!i /= ca!j)
37      (g!i::[Int])) il
38
39 isProperSemiColoring :: GraphAA -> SemiColoring -> Bool
40 isProperSemiColoring g c =
41   let
```

```
42      ca = listArray (bounds g) c
43      il = indices g
44    in
45      all (\i -> all (\j -> ca!i /= ca!j || ca!i
46      == Nothing || ca!j == Nothing) (g!i::[Int])) il
47
48  enumNeighborColoring
49    :: GraphAA -> LRecA -> Coloring -> [Coloring]
50  enumNeighborColoring g l c =
51    filter (isProperColoring g)
52      $ eNCaux (snd $ bounds g) c
53    where
54      eNCaux :: Int -> [Color] -> [[Color]]
55      eNCaux 0 _ = []
56      eNCaux n (d:ds) =
57        let
58          (_,end) = bounds g
59        in
60          (map (:) (fromJust ((l!(end-n+1))!d)) <*> [ds])
61          ++ (map (d:) $ eNCaux (n-1) ds)
62
63  enumConColoring
64    :: GraphAA -> LRecA -> Coloring -> Set Coloring
65  enumConColoring g l c = eCCaux empty c
66    where
67      eCCaux :: Set Coloring -> Coloring -> Set Coloring
68      eCCaux s d =
69        let s' = insert d s
70        in foldl eCCaux s' (filter (flip notMember s')
71        $ enumNeighborColoring g l d)
72
73  enumConColoringProp
74    :: GraphAA -> LRecA -> (Coloring -> Bool)
75    -> Coloring -> Set Coloring
76  enumConColoringProp g l p c = eCCaux empty c
77    where
78      eCCaux :: Set Coloring -> Coloring -> Set Coloring
79      eCCaux s d =
80        if p d then
81          let s' = insert d s
82          in foldl eCCaux s' (filter (flip notMember s')
83          $ enumNeighborColoring g l d)
```

```
84        else s
```

ListEdgeColoring.hs

```
 1 import Crur
 2 import Data.Char (intToDigit)
 3 import Data.Maybe (catMaybes)
 4 import Data.Set as DS (elems, filter, findMin, size)
 5
 6 orGad :: GraphAL
 7 orGad =
 8   [
 9     [4], -- 1
10     [7], -- 2
11     [13,14], -- 3
12     [1,5], -- 4
13     [4,6], -- 5
14     [5,10,21], -- 6
15     [2,8], -- 7
16     [7,9], -- 8
17     [8,17,18], -- 9
18     [6,11,21,23], -- 10
19     [10,12,23,24], -- 11
20     [11,13,24], -- 12
21     [3,12,14], -- 13
22     [3,13,15,25], -- 14
23     [14,16,25,26], -- 15
24     [15,17,26], -- 16
25     [9,16,18], -- 17
26     [9,17,19,27], -- 18
27     [18,20,22,27], -- 19
28     [19,21,22], -- 20
29     [6,10,20], -- 21
30     [19,20,23], -- 22
31     [10,11,22], -- 23
32     [11,12,25], -- 24
33     [14,15,24], -- 25
34     [15,16,27], -- 26
35     [18,19,26] -- 27
36   ]
37
38 orGadLRec :: LRec
39 orGadLRec =
```

```
40    [
41      [             -- 1
42        Just [4],
43        Nothing,
44        Nothing,
45        Just [1]
46      ],
47      [           -- 2
48        Just [4],
49        Nothing,
50        Nothing,
51        Just [1]
52      ],
53      [             -- 3
54        Just [4],
55        Nothing,
56        Nothing,
57        Just [1]
58      ],
59      [             -- 4
60        Nothing,
61        Nothing,
62        Just [4],
63        Just [3]
64      ],
65      [             -- 5
66        Nothing,
67        Just [3],
68        Just [2],
69        Nothing
70      ],
71      [             -- 6
72        Nothing,
73        Just [4],
74        Nothing,
75        Just [2]
76      ],
77      [             -- 7
78        Nothing,
79        Just [4],
80        Nothing,
81        Just [2]
```

```
82        ],
83        [               -- 8
84          Nothing,
85          Just [3],
86          Just [2],
87          Nothing
88        ],
89        [               -- 9
90          Nothing,
91          Nothing,
92          Just [4],
93          Just [3]
94        ],
95        [               -- 10
96          Just [3],
97          Nothing,
98          Just [1],
99          Nothing
100       ],
101       [               -- 11
102         Just [2,3],
103         Just [1,3],
104         Just [1,2],
105         Nothing
106       ],
107       [               -- 12
108         Just [3],
109         Nothing,
110         Just [1],
111         Nothing
112       ],
113       [               -- 13
114         Nothing,
115         Nothing,
116         Just [4],
117         Just [3]
118       ],
119       [               -- 14
120         Nothing,
121         Just [3],
122         Just [2],
123         Nothing
```

```
124     ],
125     [               -- 15
126       Just [2,3],
127       Just [1,3],
128       Just [1,2],
129       Nothing
130     ],
131     [               -- 16
132       Nothing,
133       Just [3],
134       Just [2],
135       Nothing
136     ],
137     [               -- 17
138       Nothing,
139       Just [4],
140       Nothing,
141       Just [2]
142     ],
143     [               -- 18
144       Just [2],
145       Just [1],
146       Nothing,
147       Nothing
148     ],
149     [               -- 19
150       Just [2,3],
151       Just [1,3],
152       Just [1,2],
153       Nothing
154     ],
155     [               -- 20
156       Just [2],
157       Just [1],
158       Nothing,
159       Nothing
160     ],
161     [               -- 21
162       Just [4],
163       Nothing,
164       Nothing,
165       Just [1]
```

```
166       ],
167       [             -- 22
168         Nothing,
169         Nothing,
170         Just [4],
171         Just [3]
172       ],
173       [             -- 23
174         Nothing,
175         Just [4],
176         Nothing,
177         Just [2]
178       ],
179       [             -- 24
180         Nothing,
181         Just [4],
182         Nothing,
183         Just [2]
184       ],
185       [             -- 25
186         Just [4],
187         Nothing,
188         Nothing,
189         Just [1]
190       ],
191       [             -- 26
192         Just [4],
193         Nothing,
194         Nothing,
195         Just [1]
196       ],
197       [             -- 27
198         Nothing,
199         Nothing,
200         Just [4],
201         Just [3]
202       ]
203     ]
204
205   orGadColoring :: Coloring
206   orGadColoring = [
207       1,1,1,4,3,2,4,2,3,3,1,3,4,2,
```

```
208       3,2,4,1,2,1,4,3,2,2,1,1,3
209     ]
210
211 main :: IO ()
212 main = do
213   let
214     g = gAL2GAA orGad
215     l = lRec2LRecA orGadLRec
216     -- Enumerate colorings reachable from orGadColoring
217     colorings = enumConColoring g l orGadColoring
218   putStrLn
219     $ "or-gadget has "
220     ++ (show $ length colorings)
221     ++ " colorings connected to "
222     ++ showColoring orGadColoring
223   -- Verification of internal connectedness
224   let
225     -- Returns True
226     -- when 1st, 2nd, 3rd edges have color 1,1,1
227     pro111 = (==[1,1,1]) . take 3
228     -- Verify connectedness of colorings
229     -- satisfying pro111
230     con111 = DS.filter pro111 colorings
231       == enumConColoringProp g l pro111
232         (findMin $ DS.filter pro111 colorings)
233   putStrLn $
234     "111 is "
235     ++ (if con111 then "connected" else "disconnected")
236     ++ " and has "
237     ++ show (DS.size $ DS.filter pro111 colorings)
238     ++ " colorings"
239   -- Verify for the case 114,141,144,411,441
240   let
241     pro114 = (==[1,1,4]) . take 3
242     con114 = DS.filter pro114 colorings
243       == enumConColoringProp g l pro114
244       (findMin $ DS.filter pro114 colorings)
245   putStrLn $
246     "114 is "
247     ++ (if con114 then "connected" else "disconnected")
248     ++ " and has "
249     ++ show (DS.size $ DS.filter pro114 colorings)
```

```
250       ++ " colorings"
251     let
252       pro141 = (==[1,4,1]) . take 3
253       con141 = DS.filter pro141 colorings
254         == enumConColoringProp g l pro141
255         (findMin $ DS.filter pro141 colorings)
256     putStrLn $
257       "141 is "
258       ++ (if con141 then "connected" else "disconnected")
259       ++ " and has "
260       ++ show (DS.size $ DS.filter pro141 colorings)
261       ++ " colorings"
262     let
263       pro144 = (==[1,4,4]) . take 3
264       con144 = DS.filter pro144 colorings
265         == enumConColoringProp g l pro144
266         (findMin $ DS.filter pro144 colorings)
267     putStrLn $
268       "144 is "
269       ++ (if con144 then "connected" else "disconnected")
270       ++ " and has "
271       ++ show (DS.size $ DS.filter pro144 colorings)
272       ++ " colorings"
273     let
274       pro411 = (==[4,1,1]) . take 3
275       con411 = DS.filter pro411 colorings
276         == enumConColoringProp g l pro411
277         (findMin $ DS.filter pro411 colorings)
278     putStrLn $
279       "411 is "
280       ++ (if con411 then "connected" else "disconnected")
281       ++ " and has "
282       ++ show (DS.size $ DS.filter pro411 colorings)
283       ++ " colorings"
284     let
285       pro414 = (==[4,1,4]) . take 3
286       con414 = DS.filter pro414 colorings
287         == enumConColoringProp g l pro414
288         (findMin $ DS.filter pro414 colorings)
289     putStrLn $
290       "414 is "
291       ++ (if con414 then "connected" else "disconnected")
```

```
292     ++ " and has "
293     ++ show (DS.size $ DS.filter pro414 colorings)
294     ++ " colorings"
295   let
296     pro441 = (==[4,4,1]) . take 3
297     con441 = DS.filter pro441 colorings
298       == enumConColoringProp g l pro441
299       (findMin $ DS.filter pro441 colorings)
300   putStrLn $
301     "441 is "
302     ++ (if con441 then "connected" else "disconnected")
303     ++ " and has "
304     ++ show (DS.size $ DS.filter pro441 colorings)
305     ++ " colorings"
306   -- Case 444 does not exist
307   let
308     pro444 = (==[4,4,4]) . take 3
309   putStrLn $ "444 does "
310     ++ (if null
311       $ DS.filter pro444 colorings then "not " else "")
312     ++ "exist"
313   -- Verification of external adjacency
314   let
315     -- Returns adjacent coloring of col
316     -- satisfying pro if exist, as a pair
317     test pro col =
318       let
319         nei =
320           Prelude.filter pro
321             $ enumNeighborColoring g l col
322       in if null nei then Nothing
323          else Just (col,head nei)
324   -- Verify adjacency between 111 and 114
325   let
326     con111_114 =
327       catMaybes $ map (test pro114)
328       $ DS.elems $ DS.filter pro111 colorings
329   putStrLn $ "111 and 114 are "
330     ++ if null con111_114 then "not adjacent"
331     else "adjacent by "
332     ++ ("\n"++(map intToDigit$fst$head con111_114))
333     ++ ("\n"++(map intToDigit$snd$head con111_114))
```

```
334   -- Verify adjacency between 111 and 141
335   let
336     con111_141 =
337       catMaybes $ map (test pro141)
338       $ DS.elems $ DS.filter pro111 colorings
339   putStrLn $ "111 and 141 are "
340     ++ if null con111_141 then "not adjacent"
341     else "adjacent by "
342     ++ ("\n"++ (map intToDigit$fst$head con111_141))
343     ++ ("\n"++ (map intToDigit$snd$head con111_141))
344   -- Verify adjacency between 111 and 411
345   let
346     con111_411 =
347       catMaybes $ map (test pro411)
348       $ DS.elems $ DS.filter pro111 colorings
349   putStrLn $ "111 and 411 are "
350     ++ if null con111_411 then "not adjacent"
351     else "adjacent by "
352     ++ ("\n"++(map intToDigit$fst$head con111_411))
353     ++ ("\n"++(map intToDigit$snd$head con111_411))
354   -- Verify adjacency between 114 and 144
355   let
356     con114_144 =
357       catMaybes $ map (test pro144)
358       $ DS.elems $ DS.filter pro114 colorings
359   putStrLn $ "114 and 144 are "
360     ++ if null con114_144 then "not adjacent"
361     else "adjacent by "
362     ++ ("\n"++(map intToDigit$fst$head con114_144))
363     ++ ("\n"++(map intToDigit$snd$head con114_144))
364   -- Verify adjacency between 114 and 414
365   let
366     con114_414 =
367       catMaybes $ map (test pro414)
368       $ DS.elems $ DS.filter pro114 colorings
369   putStrLn $ "114 and 414 are "
370     ++ if null con114_414 then "not adjacent"
371     else "adjacent by "
372     ++ ("\n"++(map intToDigit$fst$head con114_414))
373     ++ ("\n"++(map intToDigit$snd$head con114_414))
374   -- Verify adjacency between 141 and 144
375   let
```

```
376     con141_144 =
377       catMaybes $ map (test pro144)
378       $ DS.elems $ DS.filter pro141 colorings
379   putStrLn $ "141 and 144 are "
380     ++ if null con141_144 then "not adjacent"
381     else "adjacent by "
382     ++ ("\n"++(map intToDigit$fst$head con141_144))
383     ++ ("\n"++(map intToDigit$snd$head con141_144))
384   -- Verify adjacency between 141 and 441
385   let
386     con141_441 =
387       catMaybes $ map (test pro441)
388       $ DS.elems $ DS.filter pro141 colorings
389   putStrLn $ "141 and 441 are "
390     ++ if null con141_441 then "not adjacent"
391     else "adjacent by "
392     ++ ("\n"++(map intToDigit$fst$head con141_441))
393     ++ ("\n"++(map intToDigit$snd$head con141_441))
394   -- Verify adjacency between 411 and 414
395   let
396     con411_414 =
397       catMaybes $ map (test pro414)
398       $ DS.elems $ DS.filter pro411 colorings
399   putStrLn $ "411 and 414 are "
400     ++ if null con411_414 then "not adjacent"
401     else "adjacent by "
402     ++ ("\n"++(map intToDigit$fst$head con411_414))
403     ++ ("\n"++(map intToDigit$snd$head con411_414))
404   -- Verify adjacency between 411 and 441
405   let
406     con411_441 =
407       catMaybes $ map (test pro441)
408       $ DS.elems $ DS.filter pro411 colorings
409   putStrLn $ "411 and 441 are "
410     ++ if null con411_441 then "not adjacent"
411     else "adjacent by "
412     ++ ("\n"++(map intToDigit$fst$head con411_441))
413     ++ ("\n"++(map intToDigit$snd$head con411_441))
```

EdgeColoring.hs

```
1 import Crur
2 import Data.Char (intToDigit)
```

```
 3 import Data.Maybe (catMaybes)
 4 import Data.Set as DS (elems, filter, findMin, size)
 5
 6 orGad :: GraphAL
 7 orGad =
 8   [
 9     [4,24], -- 1
10     [10,11], -- 2
11     [17,18], -- 3
12     [1,5,24], -- 4
13     [4,6], -- 5
14     [5,7], -- 6
15     [6,8,27], -- 7
16     [7,9,27,28], -- 8
17     [8,10,28], -- 9
18     [2,9,11], -- 10
19     [2,10,12], -- 11
20     [11,13], -- 12
21     [12,14], -- 13
22     [13,15,30], -- 14
23     [14,16,30,31], -- 15
24     [15,17,31], -- 16
25     [3,16,18], -- 17
26     [3,17,19], -- 18
27     [18,20], -- 19
28     [19,21], -- 20
29     [20,22,33], -- 21
30     [21,23,25,33], -- 22
31     [22,24,25], -- 23
32     [1,4,23], -- 24
33     [22,23,26], -- 25
34     [25,27], -- 26
35     [7,8,26], -- 27
36     [8,9,29], -- 28
37     [28,30], -- 29
38     [14,15,29], -- 30
39     [15,16,32], -- 31
40     [31,33], -- 32
41     [21,22,32] -- 33
42   ]
43
44 orGadLRec :: LRec
```

```
45 orGadLRec =
46   [
47     [           -- 1
48       Just [4],
49       Nothing,
50       Nothing,
51       Just [1],
52       Nothing
53     ],
54     [           -- 2
55       Just [4],
56       Nothing,
57       Nothing,
58       Just [1],
59       Nothing
60     ],
61     [           -- 3
62       Just [4],
63       Nothing,
64       Nothing,
65       Just [1],
66       Nothing
67     ],
68     [           -- 4
69       Just [4,5],
70       Nothing,
71       Nothing,
72       Just [1,5],
73       Just [1,4]
74     ],
75     [           -- 5
76       Just [5],
77       Nothing,
78       Nothing,
79       Nothing,
80       Just [1]
81     ],
82     [           -- 6
83       Just [3],
84       Nothing,
85       Just [1],
86       Nothing,
```

```
87          Nothing
88        ],
89        [             -- 7
90          Nothing,
91          Just [3],
92          Just [2],
93          Nothing,
94          Nothing
95        ],
96        [             -- 8
97          Nothing,
98          Just [3,4],
99          Just [2,4],
100         Just [2,3],
101         Nothing
102       ],
103       [             -- 9
104         Nothing,
105         Just [3],
106         Just [2],
107         Nothing,
108         Nothing
109       ],
110       [             -- 10
111         Nothing,
112         Just [5],
113         Nothing,
114         Nothing,
115         Just [2]
116       ],
117       [             -- 11
118         Just [4,5],
119         Nothing,
120         Nothing,
121         Just [1,5],
122         Just [1,4]
123       ],
124       [             -- 12
125         Just [5],
126         Nothing,
127         Nothing,
128         Nothing,
```

```
129         Just [1]
130       ],
131       [             -- 13
132         Just [3],
133         Nothing,
134         Just [1],
135         Nothing,
136         Nothing
137       ],
138       [             -- 14
139         Nothing,
140         Just [3],
141         Just [2],
142         Nothing,
143         Nothing
144       ],
145       [             -- 15
146         Nothing,
147         Just [3,4],
148         Just [2,4],
149         Just [2,3],
150         Nothing
151       ],
152       [             -- 16
153         Nothing,
154         Just [3],
155         Just [2],
156         Nothing,
157         Nothing
158       ],
159       [             -- 17
160         Nothing,
161         Just [5],
162         Nothing,
163         Nothing,
164         Just [2]
165       ],
166       [             -- 18
167         Just [4,5],
168         Nothing,
169         Nothing,
170         Just [1,5],
```

```
171         Just [1,4]
172       ],
173       [             -- 19
174         Just [5],
175         Nothing,
176         Nothing,
177         Nothing,
178         Just [1]
179       ],
180       [             -- 20
181         Just [3],
182         Nothing,
183         Just [1],
184         Nothing,
185         Nothing
186       ],
187       [             -- 21
188         Nothing,
189         Just [3],
190         Just [2],
191         Nothing,
192         Nothing
193       ],
194       [             -- 22
195         Nothing,
196         Just [3,4],
197         Just [2,4],
198         Just [2,3],
199         Nothing
200       ],
201       [             -- 23
202         Nothing,
203         Just [3],
204         Just [2],
205         Nothing,
206         Nothing
207       ],
208       [             -- 24
209         Nothing,
210         Just [5],
211         Nothing,
212         Nothing,
```

```
213         Just [2]
214       ],
215       [             -- 25
216         Just [4],
217         Nothing,
218         Nothing,
219         Just [1],
220         Nothing
221       ],
222       [             -- 26
223         Just [4,5],
224         Nothing,
225         Nothing,
226         Just [1,5],
227         Just [1,4]
228       ],
229       [             -- 27
230         Nothing,
231         Nothing,
232         Nothing,
233         Just [5],
234         Just [4]
235       ],
236       [             -- 28
237         Just [4],
238         Nothing,
239         Nothing,
240         Just [1],
241         Nothing
242       ],
243       [             -- 29
244         Just [4,5],
245         Nothing,
246         Nothing,
247         Just [1,5],
248         Just [1,4]
249       ],
250       [             -- 30
251         Nothing,
252         Nothing,
253         Nothing,
254         Just [5],
```

```
255         Just [4]
256       ],
257       [             -- 31
258         Just [4],
259         Nothing,
260         Nothing,
261         Just [1],
262         Nothing
263       ],
264       [             -- 32
265         Just [4,5],
266         Nothing,
267         Nothing,
268         Just [1,5],
269         Just [1,4]
270       ],
271       [             -- 33
272         Nothing,
273         Nothing,
274         Nothing,
275         Just [5],
276         Just [4]
277       ]
278     ]
279
280 orGadColoring :: Coloring
281 orGadColoring = [
282       1,1,1,4,5,1,3,2,3,2,4,5,1,3,2,3,
283       2,4,5,1,3,2,3,2,4,5,4,4,5,4,4,5,4
284     ]
285
286 main :: IO ()
287 main = do
288   let
289     g = gAL2GAA orGad
290     l = lRec2LRecA orGadLRec
291     -- Enumerate colorings reachable from orGadColoring
292     colorings = enumConColoring g l orGadColoring
293   putStrLn
294     $ "or-gadget has "
295     ++ (show $ length colorings)
296     ++ " colorings connected to "
```

```
297     ++ showColoring orGadColoring
298   -- Verification of internal connectedness
299   let
300     -- Returns True
301     -- when 1st, 2nd, 3rd edges have color 1,1,1
302     pro111 = (==[1,1,1]) . take 3
303     -- Verify connectedness of colorings
304     -- satisfying pro111
305     con111 = DS.filter pro111 colorings
306       == enumConColoringProp g l pro111
307         (findMin $ DS.filter pro111 colorings)
308   putStrLn $
309     "111 is "
310     ++ (if con111 then "connected" else "disconnected")
311     ++ " and has "
312     ++ show (DS.size $ DS.filter pro111 colorings)
313     ++ " colorings"
314   -- Verify for the case 114,141,144,411,441
315   let
316     pro114 = (==[1,1,4]) . take 3
317     con114 = DS.filter pro114 colorings
318       == enumConColoringProp g l pro114
319       (findMin $ DS.filter pro114 colorings)
320   putStrLn $
321     "114 is "
322     ++ (if con114 then "connected" else "disconnected")
323     ++ " and has "
324     ++ show (DS.size $ DS.filter pro114 colorings)
325     ++ " colorings"
326   let
327     pro141 = (==[1,4,1]) . take 3
328     con141 = DS.filter pro141 colorings
329       == enumConColoringProp g l pro141
330       (findMin $ DS.filter pro141 colorings)
331   putStrLn $
332     "141 is "
333     ++ (if con141 then "connected" else "disconnected")
334     ++ " and has "
335     ++ show (DS.size $ DS.filter pro141 colorings)
336     ++ " colorings"
337   let
338     pro144 = (==[1,4,4]) . take 3
```

```
339       con144 = DS.filter pro144 colorings
340         == enumConColoringProp g l pro144
341         (findMin $ DS.filter pro144 colorings)
342     putStrLn $
343       "144 is "
344       ++ (if con144 then "connected" else "disconnected")
345       ++ " and has "
346       ++ show (DS.size $ DS.filter pro144 colorings)
347       ++ " colorings"
348     let
349       pro411 = (==[4,1,1]) . take 3
350       con411 = DS.filter pro411 colorings
351         == enumConColoringProp g l pro411
352         (findMin $ DS.filter pro411 colorings)
353     putStrLn $
354       "411 is "
355       ++ (if con411 then "connected" else "disconnected")
356       ++ " and has "
357       ++ show (DS.size $ DS.filter pro411 colorings)
358       ++ " colorings"
359     let
360       pro414 = (==[4,1,4]) . take 3
361       con414 = DS.filter pro414 colorings
362         == enumConColoringProp g l pro414
363         (findMin $ DS.filter pro414 colorings)
364     putStrLn $
365       "414 is "
366       ++ (if con414 then "connected" else "disconnected")
367       ++ " and has "
368       ++ show (DS.size $ DS.filter pro414 colorings)
369       ++ " colorings"
370     let
371       pro441 = (==[4,4,1]) . take 3
372       con441 = DS.filter pro441 colorings
373         == enumConColoringProp g l pro441
374         (findMin $ DS.filter pro441 colorings)
375     putStrLn $
376       "441 is "
377       ++ (if con441 then "connected" else "disconnected")
378       ++ " and has "
379       ++ show (DS.size $ DS.filter pro441 colorings)
380       ++ " colorings"
```

```
381  -- Case 444 does not exist
382  let
383    pro444 = (==[4,4,4]) . take 3
384  putStrLn $ "444 does "
385    ++ (if null $ DS.filter pro444 colorings then "not "
386          else "")
387    ++ "exist"
388  -- Verification of external adjacency
389  let
390    -- Returns adjacent coloring of col
391    -- satisfying pro if exist, as a pair
392    test pro col =
393      let
394        nei =
395          Prelude.filter pro
396            $ enumNeighborColoring g l col
397      in if null nei then Nothing
398         else Just (col,head nei)
399  -- Verify adjacency between 111 and 114
400  let
401    con111_114 =
402      catMaybes $ map (test pro114)
403      $ DS.elems $ DS.filter pro111 colorings
404  putStrLn $ "111 and 114 are "
405    ++ if null con111_114 then "not adjacent"
406    else "adjacent by "
407    ++ ("\n"++(map intToDigit$fst$head con111_114))
408    ++ ("\n"++(map intToDigit$snd$head con111_114))
409  -- Verify adjacency between 111 and 141
410  let
411    con111_141 =
412      catMaybes $ map (test pro141)
413      $ DS.elems $ DS.filter pro111 colorings
414  putStrLn $ "111 and 141 are "
415    ++ if null con111_141 then "not adjacent"
416    else "adjacent by "
417    ++ ("\n"++(map intToDigit$fst$head con111_141))
418    ++ ("\n"++(map intToDigit$snd$head con111_141))
419  -- Verify adjacency between 111 and 411
420  let
421    con111_411 =
422      catMaybes $ map (test pro411)
```

```
423       $ DS.elems $ DS.filter pro111 colorings
424     putStrLn $ "111 and 411 are "
425       ++ if null con111_411 then "not adjacent"
426       else "adjacent by "
427       ++ ("\n"++(map intToDigit$fst$head con111_411))
428       ++ ("\n"++(map intToDigit$snd$head con111_411))
429     -- Verify adjacency between 114 and 144
430     let
431       con114_144 =
432         catMaybes $ map (test pro144)
433         $ DS.elems $ DS.filter pro114 colorings
434     putStrLn $ "114 and 144 are "
435       ++ if null con114_144 then "not adjacent"
436       else "adjacent by "
437       ++ ("\n"++(map intToDigit$fst$head con114_144))
438       ++ ("\n"++(map intToDigit$snd$head con114_144))
439     -- Verify adjacency between 114 and 414
440     let
441       con114_414 =
442         catMaybes $ map (test pro414)
443         $ DS.elems $ DS.filter pro114 colorings
444     putStrLn $ "114 and 414 are "
445       ++ if null con114_414 then "not adjacent"
446       else "adjacent by "
447       ++ ("\n"++(map intToDigit$fst$head con114_414))
448       ++ ("\n"++(map intToDigit$snd$head con114_414))
449     -- Verify adjacency between 141 and 144
450     let
451       con141_144 =
452         catMaybes $ map (test pro144)
453         $ DS.elems $ DS.filter pro141 colorings
454     putStrLn $ "141 and 144 are "
455       ++ if null con141_144 then "not adjacent"
456       else "adjacent by "
457       ++ ("\n"++(map intToDigit$fst$head con141_144))
458       ++ ("\n"++(map intToDigit$snd$head con141_144))
459     -- Verify adjacency between 141 and 441
460     let
461       con141_441 =
462         catMaybes $ map (test pro441)
463         $ DS.elems $ DS.filter pro141 colorings
464     putStrLn $ "141 and 441 are "
```

```
465     ++ if null con141_441 then "not adjacent"
466     else "adjacent by "
467     ++ ("\n"++(map intToDigit$fst$head con141_441))
468     ++ ("\n"++(map intToDigit$snd$head con141_441))
469   -- Verify adjacency between 411 and 414
470   let
471     con411_414 =
472       catMaybes $ map (test pro414)
473       $ DS.elems $ DS.filter pro411 colorings
474   putStrLn $ "411 and 414 are "
475     ++ if null con411_414 then "not adjacent"
476     else "adjacent by "
477     ++ ("\n"++(map intToDigit$fst$head con411_414))
478     ++ ("\n"++(map intToDigit$snd$head con411_414))
479   -- Verify adjacency between 411 and 441
480   let
481     con411_441 =
482       catMaybes $ map (test pro441)
483       $ DS.elems $ DS.filter pro411 colorings
484   putStrLn $ "411 and 441 are "
485     ++ if null con411_441 then "not adjacent"
486     else "adjacent by "
487     ++ ("\n"++(map intToDigit$fst$head con411_441))
488     ++ ("\n"++(map intToDigit$snd$head con411_441))
```

Bibliography

[1] K. Appel and W. Haken. Every planar map is four colorable. part i: Discharging. *Illinois Journal of Mathematics*, 21(3):429–490, 1977.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

[3] M. Bonamy and N. Bousquet. Recoloring bounded treewidth graphs. *Electronic Notes in Discrete Mathematics*, 44:257–262, 2013.

[4] M. Bonamy and N. Bousquet. Recoloring graphs via tree decompositions. *Eur. J. Comb.*, 69:200–213, 2018.

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

[6] P. S. Bonsma. The complexity of rerouting shortest paths. *Theor. Comput. Sci.*, 510:1–12, 2013.

[7] P. S. Bonsma. Rerouting shortest paths in planar graphs. *Discrete Applied Mathematics*, 231:95–112, 2017.

[8] P. S. Bonsma and L. Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theor. Comput. Sci.*, 410(50):5215–5226, 2009.

[9] P. S. Bonsma, A. E. Mouawad, N. Nishimura, and V. Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In M. Cygan and P. Heggernes, editors, *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers*, volume 8894 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2014.

[10] P. S. Bonsma and D. Paulusma. Using contracted solution graphs for solving reconfiguration problems. In P. Faliszewski, A. Muscholl, and R. Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, volume 58 of *LIPIcs*, pages 20:1–20:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[11] R. C. Brewster, S. McGuinness, B. Moore, and J. A. Noel. A dichotomy theorem for circular colouring reconfiguration. *Theor. Comput. Sci.*, 639:1–13, 2016.

[12] L. Cereceda, J. van den Heuvel, and M. Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011.

[13] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[14] T. Hatanaka, T. Ito, and X. Zhou. The list coloring reconfiguration problem for bounded pathwidth graphs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98-A(6):1168–1178, 2015.

[15] T. Hatanaka, T. Ito, and X. Zhou. Parameterized complexity of the list coloring reconfiguration problem with graph parameters. *Theor. Comput. Sci.*, 739:65–79, 2018.

[16] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005.

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

[18] T. Ito, M. Kaminski, and E. D. Demaine. Reconfiguration of list edge-colorings in a graph. *Discrete Applied Mathematics*, 160(15):2199–2207, 2012.

[19] M. Johnson, D. Kratsch, S. Kratsch, V. Patel, and D. Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016.

[20] D. Lokshtanov and A. E. Mouawad. The complexity of independent set reconfiguration on bipartite graphs. *ACM Trans. Algorithms*, 15(1):7:1–7:19, 2019.

[21] A. E. Mouawad, N. Nishimura, V. Pathak, and V. Raman. Shortest reconfiguration paths in the solution space of boolean formulas. *SIAM J. Discrete Math.*, 31(3):2185–2200, 2017.

[22] D. Ratner and M. K. Warmuth. N×N puzzle and related relocation problem. *J. Symb. Comput.*, 10(2):111–138, 1990.

[23] N. Robertson, D. P. Sanders, P. D. Seymour, and R. Thomas. Efficiently four-coloring planar graphs. In G. L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 571–575. ACM, 1996.

[24] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[25] K. W. Schwerdtfeger. A computational trichotomy for connectivity of boolean satisfiability. *JSAT*, 8(3/4):173–195, 2014.

[26] T. C. van der Zanden. Parameterized complexity of graph constraint logic. In T. Husfeldt and I. A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 282–293. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[27] M. Wrochna. Homomorphism reconfiguration via homotopy. In E. W. Mayr and N. Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 730–742. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[28] M. Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *J. Comput. Syst. Sci.*, 93:1–10, 2018.

List of publications

Refereed papers in journals

1. Hiroki Osawa, Akira Suzuki, Takehiro Ito, and Xiao Zhou. The complexity of (list) edge-coloring reconfiguration problem. *IEICE Transactions*, 101–AA(1):232–238, 2018.

Refereed papers in international conferences

1. Hiroki Osawa, Akira Suzuki, Takehiro Ito, and Xiao Zhou. Algorithms for coloring reconfiguration under recolorability constraints. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 37:1–37:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

2. Hiroki Osawa, Akira Suzuki, Takehiro Ito, and Xiao Zhou. Complexity of coloring reconfiguration under recolorability constraints. In Yoshio Okamoto and Takeshi Tokuyama, editors, *28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand*, volume 92 of LIPIcs, pages 62:1–62:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

3. Hiroki Osawa, Akira Suzuki, Takehiro Ito, and Xiao Zhou. The complexity of (list) edge-coloring reconfiguration problem. In Sheung-Hung Poon, Md. Saidur Rahman, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation, 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29-31, 2017, Proceedings.*, volume 10167 of *Lecture Notes in Computer Science*, pages 347–358. Springer, 2017.