Reconfiguring Graph Colorings

by

Krishna Vaidyanathan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Graph coloring has been studied for a long time and continues to receive interest within the research community [43]. It has applications in scheduling [46], timetables, and compiler register allocation [45]. The most popular variant of graph coloring, k-coloring, can be thought of as an assignment of k colors to the vertices of a graph such that adjacent vertices are assigned different colors.

Reconfiguration problems, typically defined on the solution space of search problems, broadly ask whether one solution can be transformed to another solution using step-by-step transformations, when constrained to one or more specific transformation steps [55]. One well-studied reconfiguration problem is the problem of deciding whether one k-coloring can be transformed to another k-coloring by changing the color of one vertex at a time, while always maintaining a k-coloring at each step.

We consider two variants of graph coloring: acyclic coloring and equitable coloring, and their corresponding reconfiguration problems. A k-acyclic coloring is a k-coloring where there are more than two colors used by the vertices of each cycle, and a k-equitable coloring is a k-coloring such that each color class, which is defined as the set of all vertices with a particular color, is nearly the same size as all others.

We show that reconfiguration of acyclic colorings is PSPACE-hard, and that for non-bipartite graphs with chromatic number 3 there exist two k-acyclic colorings f_s and f_e such that there is no sequence of transformations that can transform f_s to f_e. We also consider the problem of whether two k-acyclic colorings can be transformed to each other in at most ℓ steps, and show that it is in XP, which is the class of algorithms that run in time $O(n^{f(k)})$ for some computable function f and parameter k, where in this case the parameter is defined to be the length of the reconfiguration sequence plus the length of the longest induced cycle.

We also show that the reconfiguration of equitable colorings is PSPACE-hard and W[1]-hard with respect to the number of vertices with the same color. We give polynomial-time algorithms for Reconfiguration of Equitable Colorings when the number of colors used is two and also for paths when the number of colors used is three.

Acknowledgements

I would like to thank my supervisor, Professor Naomi Nishimura, for her support, guidance, and patience. This work would not have been possible without her suggestions and feedback.

I would also like to thank Professor Therese Biedl and Professor Anna Lubiw for agreeing to read my thesis and providing valuable comments.

I am grateful to Kevin Yeo, Vijay Subramanya, Hicham El-Zein, Tatsuhiko Hatanaka, Haruka Mizuta, Tesshu Hanaka, and Kshitij Jain for the many discussions that I have had with them.

I am also grateful to my family and friends for their support.

Lastly, I am thankful to Professor Lekshmi for encouraging me to pursue graduate studies.

Table of Contents

List of Figures vii

1 Introduction 1

1.1 Organization . . . 3

2 Preliminaries 4

2.1 Graphs . . . 4

2.2 Graph classes . . . 5

2.3 Coloring . . . 6

2.3.1 Acyclic colorings . . . 7

2.3.2 Equitable colorings . . . 8

2.4 Reconfiguration . . . 11

2.5 Parameterized complexity . . . 13

3 Literature Survey 15

3.1 Colorings . . . 15

3.1.1 Acyclic colorings . . . 16

3.1.2 Equitable colorings . . . 16

3.1.3 List colorings . . . 17

3.1.4 Kempe chains . . . 17

3.1.5 Edge-colorings . . . 18

3.2 Reconfiguration of graph colorings 18
3.2.1 Recoloring a single vertex 18
3.2.2 Recoloring a single edge 21
3.2.3 Kempe chain recoloring 22
3.2.4 Reconfiguring list $L(2,1)$-labelings 22

4 Acyclic Coloring Reconfiguration **24**
4.1 Non-bipartite graphs of acyclic chromatic number 3 24
4.2 k-ACR REACH is PSPACE-hard 27
4.3 k-ACR BOUND is in XP 34

5 Equitable Coloring Reconfiguration **38**
5.1 General properties of k-ECR REACH 38
5.2 k-ECR REACH is PSPACE-hard 40
5.3 2-equitable colorings 46
5.4 Paths 50

6 Conclusions and Future Work **80**

References **82**

List of Figures

2.1 Canonical form for a path of odd length ($|colorclass(1, f)| = 4, |colorclass(2, f)| = 3, |colorclass(3, f)| = 4$) 10

2.2 Canonical form for a path of even length ($|colorclass(1, f)| = 4, |colorclass(2, f)| = 3, |colorclass(3, f)| = 3$) 10

3.1 Two non-Kempe equivalent 3-colorings of the triangular prism. 22

5.1 The figure shows a sample input and output 3-equitable coloring of a path for Algorithm 4. 53

5.2 One iteration of Algorithm 6 57

5.3 One iteration of Algorithm 7. 58

5.4 Operations of Algorithm 6 for iteration $p + 2$. 59

5.5 Example of one iteration of Algorithm 8. 63

5.6 An example of the operations in Algorithm 10. 69

5.7 An example of the operations in Algorithm 11. 72

5.8 An example for Algorithm 13, where Figure 5.8a and Figure 5.8b show a subpath and a 3-equitable colorings before and after, respectively. 75

5.9 An example of the operations in Algorithm 14. 77

6.1 A possible canonical form for paths when $k > 3$. 81

Chapter 1

Introduction

Reconfiguration problems consider finding step-by-step transformations between feasible solutions of search problems. A few well-known examples of reconfiguration problems include the 15-puzzle, Rush Hour, and Rubik's cube [15, 50]. Also, some of the search problems on which reconfiguration problems have been studied are: graph coloring, independent set, and Boolean satisfiability [55].

Another way of formulating reconfiguration problems is through the *reconfiguration graph*. The reconfiguration graph of an instance of a search problem has as its vertex set all feasible solutions, and two vertices in the reconfiguration graph are connected by an edge if the feasible solution associated with one vertex can be transformed to the feasible solution associated with the other vertex by a transformation step. Different adjacency relations or, equivalently, transformation steps can be defined on the reconfiguration graph of the search problem under consideration. When considering a search problem where a feasible solution is a subset of the set of all vertices, three popular transformation steps are token jumping (TJ), token sliding (TS), and token addition and removal (TAR) [30].

We explain the three transformation steps (TJ, TS, and TAR) in the context of Independent Set Reconfiguration. An independent set of a graph is defined as a set of vertices such that no two vertices in the set are adjacent. In all the three transformation steps, we can model an independent set of a graph as a set of tokens placed on the vertices of the independent set. In TJ, an independent set can be transformed by moving a token from a vertex of the independent set to another vertex. In TS, an independent set can be transformed by sliding a token along an edge from a vertex of the independent set. In TAR, a token can be added or removed as long as there are at least $m - 1$ tokens (where m is some positive integer).

Usually, three types of problems are raised in reconfiguration.

1. The Reachability problem, which takes as input two feasible solutions of an instance of a search problem, and asks if the two feasible solutions are connected in the reconfiguration graph.

2. The Shortest Transformation (or BOUND) problem, which takes as input two feasible solutions of an instance of a search problem and a positive integer ℓ, and asks if the two feasible solutions are connected in the reconfiguration graph by a path of length ℓ.

3. The Connectivity problem, which asks if the reconfiguration graph is connected.

In this thesis, we are primarily interested in the Reachability problem, and more specifically, reconfiguration of two variants of graph coloring (acyclic and equitable colorings).

Reconfiguration of graph colorings has real-world applications in the field of radio-communication networks for radio channel reassignment [4] and in statistical physics for single-site Glauber dynamics (single-site Glauber dynamics can be represented as a Markov chain whose state space is the set of all k-colorings of a graph) [35] [1]. They also have applications in frequency assignment [28] and wireless local area networks [31].

The problem of whether two graph colorings are connected in the reconfiguration graph has received significant attention, and some consider it to be the most well-studied problem in reconfiguration [55]. Different variants of graph coloring such as proper coloring, list coloring, list labeling, and edge-coloring have been considered. In addition, for the variants of graph coloring mentioned, transformation steps such as changing the color of a vertex, Kempe chain recoloring, and changing the color of an edge have been studied.

We consider two variants of graph coloring that have not been studied in reconfiguration: acyclic colorings and equitable colorings. In reconfiguration of acyclic colorings, we consider that the transformation step is to change the color of a vertex and study the Reachability, Shortest Transformation, and Connectivity problems. In reconfiguration of equitable colorings, we consider that the transformation step is to swap the colors of two vertices, and study the Reachability problem.

[1] These applications were first reported by Cereceda [14].

1.1 Organization

The organization of the rest of the thesis is as follows.

In Chapter 2, we give terminology and notation that we use throughout this thesis. We outline a brief history of graph coloring in Chapter 3. Then we survey results in reconfiguration of variants of graph coloring when the transformation step is to change the color of a single vertex, and also other transformation steps. We also provide the history of acyclic colorings and equitable colorings.

We study Acyclic Coloring Reconfiguration in Chapter 4. We consider the Connectivity problem and show that for any non-bipartite graph of acyclic chromatic number 3 there exist two k-acyclic colorings that can not be transformed to each other by a sequence of transformation steps. We also prove that Acyclic Coloring Reconfiguration (k-ACR REACH) is PSPACE-hard, and consider the related Shortest Transformation problem (k-ACR BOUND) and show that it is in XP with respect to the length of the reconfiguration sequence plus the length of the longest induced cycle.

In Chapter 5, we show that Equitable Coloring Reconfiguration (k-ECR REACH) is PSPACE-hard and also W[1]-hard with respect to the size of a color class (the number of vertices with the same color). Then we give an algorithm that runs in linear time for 2-ECR Reach. Also, for paths, we show that 3-ECR Reach can be solved in polynomial time.

Finally, in Chapter 6, we raise the question of whether the W-hardness of k-ACR BOUND can be proved with respect to the length of the reconfiguration sequence plus the length of the longest induced cycle. We also ask the question of whether k-ECR REACH is in FPT for some parameter. Lastly, we conjecture that k-ECR REACH can be solved in polynomial time for paths when the number of colors used is greater than three and provide a sketch of a possible algorithm.

Chapter 2

Preliminaries

In this chapter, we present some definitions and notation that we use throughout the thesis. In Section 2.1, we give standard graph theoretic definitions. In Section 2.2, we define classes of graphs that we look at in this thesis. In Section 2.3, we provide formal definitions related to coloring, and since this thesis deals specifically with acyclic and equitable colorings, Sections 2.3.1 and 2.3.2 elaborate on definitions related to them, respectively. We are then ready to introduce, in Section 2.4, definitions for reconfiguration problems in general and specific definitions for the reconfiguration problems that we discuss in this thesis. Since we discuss parameterized algorithms for k-ACR BOUND, which we define in Section 2.4, we discuss parameterized complexity in Section 2.5.

2.1 Graphs

We refer the reader to the book by Diestel [18] for graph theoretic definitions that we do not define. Unless otherwise specified, we consider only simple, undirected graphs and denote the vertex and edge sets of a graph G as $V(G)$ and $E(G)$, respectively. We typically denote by n the *size* of graph G, $|V(G)|$. Two vertices u and v are said to be *adjacent* if $uv \in E(G)$. The neighborhood of a vertex v in graph G, denoted by $N_G(v)$, is the set $\{u \mid uv \in E(G)\}$. The *degree* of a vertex v is defined as $d(v) = |\{u \mid u \in V(G), uv \in E(G)\}|$ and the *maximum degree* $\Delta(G) = \max_{v \in V(G)} d(v)$. A graph is *bipartite* if there exists a partition of the vertex set into two disjoint sets such that no edge exists between two vertices in the same set. The largest length of the shortest path between any two vertices of a graph is called the *diameter* of the graph. The *line graph of a graph* G is defined

as the graph whose vertex set corresponds to $E(G)$ and two vertics are adjacent if their corresponding edges in $E(G)$ share a common vertex [18].

An *induced subgraph* of G for a subset $S \subseteq V(G)$, denoted as $G[S]$, is the graph with vertex set S and edge set $\{uv \mid u, v \in S, uv \in E(G)\}$. For a path P, we define a *subpath of* P as a connected subgraph of P. We also say that two vertices are *connected* if there exists a path between them or if they are the same vertex. A subgraph A of a graph is said to be *connected* if between any two vertices of A, there exists a path P such that $V(P) \subseteq V(A)$. For a connected graph G, we call a set $S \subset V(G)$ a *separator* if the induced graph on $V(G) \setminus S$ has two vertices that are not connected.

A graph is *k-chordal* if there exists no induced cycle of length greater than k. For a graph G, let $\Gamma(G)$ denote the minimum positive integer t for which G is t-chordal. Thus, $\Gamma(G) = 3$ when G is chordal.

An *identification of two vertices u and v of a graph G* is a transformation where u and v are replaced by a vertex w and each edge wx is added, where x is adjacent to either u or v in G.

We consider directed graphs in some of our proofs by directing the edges of an undirected graph, and refer to the set of arcs of a directed graph $\overrightarrow{G}$ as $E(\overrightarrow{G})$. We denote an arc from u to v in a directed graph $\overrightarrow{G}$ as $\overrightarrow{uv}$, where $u, v \in V(\overrightarrow{G})$. Alternatively, we also denote the arc from u to v as $\overleftarrow{vu}$. For a subset S of $V(\overrightarrow{G})$, we denote the directed induced subgraph by $\overrightarrow{G}[S]$. For ease of reading, in this thesis, we also denote the directed induced subgraph for the subset S by $\overrightarrow{S}$.

2.2 Graph classes

We define a few classes of graphs that are looked at in the results surveyed in Chapter 3.

Definition 1. A graph G is *cubic* if each vertex in G has degree exactly three.

We define two similar decompositions, tree-decompositions and then path-decompositions.

Definition 2. A *tree-decomposition* of a graph G is defined as a pair (T, X) where T is a tree and X maps a node of T to a vertex subset of G, satisfying the conditions below.

1. $\bigcup_{v \in V(T)} X(v) = V(G)$.

2. For each edge $uv \in E(G)$, there is a node y in T such that $u, v \in X(y)$.

3. For each vertex $v \in V(G)$, the induced graph on the set of nodes y in T such that $v \in X(y)$ is connected.

The *width of a tree-decomposition* (T, X) is defined as $\max_{y \in V(T)} |X(y)| - 1$, and the *treewidth of a graph* G, $tw(G)$, is defined as the least integer t such that there is a tree-decomposition of width t [18].

Definition 3. A *path-decomposition* of a graph G is a tree-decomposition (P, X) where P is a path.

The *width of a path-decomposition* (P, X), similar to the width of a tree-decomposition, is defined as $\max_{y \in V(P)} |X(y)| - 1$, and the *pathwidth of* G is defined as the least integer t such that there is a path-decomposition of width t [18].

The following graph class was introduced by Bonamy *et al.* to study the structure of the reconfiguration graph for Coloring Reconfiguration [9]. The following graph class requires knowledge of k-colorings, which we define in Section 2.3.

Definition 4. We call a graph G *k-color-dense* if it has a k-coloring and satisfies either of the following conditions [9].

1. G is the disjoint union of cliques that have at most k vertices each.

2. G has a separator S, two disjoint subsets D_1, D_2 where $D_1 \cup D_2 = V(G) \setminus S$ and both $G[D_1]$, $G[D_2]$ are connected, and two vertices $u \in D_1$ and $v \in D_2$ such that the below conditions hold.

   (a) $|D_1| = 1$ or $|D_1 \cup S| \leq k$.

   (b) $S \subseteq N_G(v)$.

   (c) The graph obtained by identifying u and v is a k-color-dense graph.

2.3 Coloring

A *k-vertex coloring* of a graph G is a function $f : V(G) \to \{1, \ldots, k\}$. A k-vertex coloring f of G is *proper* or a *k-coloring* if for every edge uv in $E(G)$, $f(u) \neq f(v)$. Given a graph G

and a k-vertex coloring f, a subgraph H is said to be *2-chromatic* if there exist two colors c_1 and c_2 such that for every vertex u in $V(H)$, $f(u) = c_1$ or $f(u) = c_2$. We also say that two k-vertex colorings f_1 and f_2 *differ* by t, where $t = |\{u \mid u \in V(G), f_1(u) \neq f_2(u)\}|$.

Definition 5. We denote the set of vertices with color i in a k-vertex coloring f of G as $colorclass(i, f)$. We also refer to the set of all vertices with a color as a *color class*.

Definition 6. For any vertex set $A \subseteq V(G)$ of graph G and k-vertex coloring f of G, $colors(A, f)$ is the set $\{f(u) \mid u \in A\}$.

Definition 7. A k-vertex coloring f of a graph G is said to be *restricted* on an induced subgraph S to a k-vertex coloring $f|_S$ of S if $f|_S(u) = f(u)$ for every vertex u in $V(S)$.

2.3.1 Acyclic colorings

A k-coloring of a graph G is *acyclic* if there exists no 2-chromatic cycle in G [2]. The *acyclic chromatic number*, $A(G)$, of a graph G is the smallest integer k for which there is a k-coloring of G that is acyclic [2]. Note that when $\Gamma(G) = 3$, any proper coloring of G is also an acyclic coloring, as otherwise a C_3 that uses two or fewer colors would have an edge whose endpoints have the same color, contradicting the fact that the coloring is proper.

We provide some definitions that we use to prove that the reconfiguration graph of a non-bipartite graph, when the number of colors used is three, is not connected. We define weights on a graph by giving all its edges directions. In Section 4.1, we use a strategy of assigning weights to a directed graph depending on the colors of an edge's endpoints and the edge's direction to show a sufficiency condition for two 3-acyclic colorings being connected by a reconfiguration sequence. For this purpose, we typically direct the graph so that a particular cycle or path is directed. We assume without loss of generality that the colors used in all 3-acyclic colorings are 1, 2, and 3.

Definition 8. For an arc $\overrightarrow{uv}$ of a directed graph $\overrightarrow{G}$ with 3-acyclic coloring f, its weight is given by the following equation depending on the values of $f(u)$ and $f(v)$, and we denote it by $w(\overrightarrow{uv}, f)$. Note that for every edge uv and 3-acyclic coloring f of the graph, the vertices u and v do not have the same color in f.

	$f(v) = 1$	$f(v) = 2$	$f(v) = 3$
$f(u) = 1$		+1	-1
$f(u) = 2$	-1		+1
$f(u) = 3$	+1	-1	

(2.1)

Definition 9. The *weight* of a subgraph $\overrightarrow{S}$ of a directed graph $\overrightarrow{G}$ with a 3-acyclic coloring f, denoted by $W(\overrightarrow{S}, f)$, is the sum of the weights of its arcs. More formally, $W(\overrightarrow{S}, f) = \sum_{\overrightarrow{uv} \in E(\overrightarrow{S})} w(\overrightarrow{uv}, f)$.

2.3.2 Equitable colorings

In this section, we first define equitable colorings, then we give basic definitions that we use when discussing Equitable Coloring Reconfiguration. Finally, we give definitions and notation for paths when the number of colors used is three.

A k-coloring f of a graph G is *equitable* or a *k-equitable coloring* if for any two colors c_1, c_2 used by the k-coloring, $|colorclass(c_1, f)| - |colorclass(c_2, f)| \in \{-1, 1\}$ [48]. The *equitable chromatic number* is the smallest k such that a k-equitable coloring exists for a given graph [5].

We now define a transformation step for a k-equitable coloring of a graph, which we use when studying whether there exists a sequence of transformation steps between two k-equitable colorings of a graph.

Definition 10. Given a graph G and a k-equitable coloring f, for two vertices u, v of $V(G)$, we define the *swap operation* as changing the color of u to $f(v)$ and the color of v to $f(u)$. We denote this as *$swap(u, v)$ in f*.

Definition 11. Given two k-equitable colorings f and h of graph G, if f and h differ by 2, then we say that *the swap between f and h* is $swap(u, v)$ where u and v are the vertices whose colors are different in f and h.

Next, we define a necessary condition for when there can exist a sequence of swaps that transforms one k-equitable coloring to another k-equitable coloring of a graph.

Definition 12. Given a graph G and two k-equitable colorings f and h, f is *viable to h* (and vice-versa) if $|colorclass(i, f)| = |colorclass(i, h)|$ for $1 \leq i \leq k$.

Note that for a k-equitable coloring of a graph, the size of every color class remains the same after any number of swaps. Clearly, for any two k-equitable colorings f and h of a graph, if f is not viable to h then there can not exist a sequence of swaps that transforms f to h or vice-versa.

We formally characterize, given a k-equitable coloring and two vertices of a graph, when the k-vertex coloring that we obtain by swapping the colors of the two vertices is a proper

coloring. Intuitively, to characterize this, the colors of two vertices u and v of a graph can be swapped if and only if each vertex in the neighborhood of u does not have the same color as v and vice-versa.

Definition 13. Given a graph G, a k-equitable coloring f, and vertices u and v, we say that u *and* v *are valid for each other* if $colors(N_G(v)) \cap f(u) = \emptyset$, $colors(N_G(u)) \cap f(v) = \emptyset$, and $f(u) \neq f(v)$. We also say that $swap(u, v)$ *in* f *is valid* and denote by $valid(v, f)$ the set of all valid vertices for v in f.

Given source and target k-equitable colorings f_s and f_e, respectively, of a graph and a vertex v, if v has different colors in the source and target k-equitable colorings then, at some point in the reconfiguration sequence, the color of v is swapped with the color of a vertex, say w, with color $f_e(v)$. Ideally, the color of w in the target k-equitable coloring is not $f_e(v)$. Suppose that f_m is the k-equitable coloring in the reconfiguration sequence when the colors of v and w are swapped. We define below, given f_m, f_e, and v, the set of all such w.

Definition 14. Given a graph G, two k-equitable colorings f and h, and a vertex v such that $f(v) \neq h(v)$, we denote as $candidate(f, h, v)$ the set of vertices u such that $f(u) \neq h(u)$ and $h(v) = f(u)$.

We define below terms and notation for paths when $k = 3$, which we use in Section 5.4 where we show that the Reachability question for Equitable Coloring Reconfiguration can be solved in polynomial time. We first give definitions for ease of understanding when referring to vertices of a path. Next, we define a canonical k-equitable coloring for paths for specific sizes of color classes.

Definition 15. Given a path $P = v_1, v_2, \ldots, v_n$, a k-equitable coloring f, and a color c, we define the *rightmost* c as the vertex v_i, $1 \leq i \leq n$, where i is the largest number such that $f(v_i) = c$.

Definition 16. Given a path $P = v_1, v_2, \ldots, v_n$, we say that a vertex v_j is *left of* v_i, $1 \leq i, j \leq n$, if $i > j$, and we say that v_j is *right of* v_i if $i < j$. We also say that v_j is the *predecessor of* v_i if $j = i - 1$, and we say that v_j is the *successor of* v_i if $j = i + 1$.

Definition 17. Given a path $P = v_1, v_2, \ldots, v_n$ and a 3-equitable coloring f, we say that P is a *c-alternating path*, where c is a color used by f, if, for all i, exactly one vertex in the set $\{v_i, v_{i+1}\}$, $1 \leq i \leq n - 1$, has color c in f.

We define below a canonical 3-equitable coloring of a path $v_1, v_2, \ldots, v_n$ depending on whether n is odd or even. A high-level overview of the colors of vertices in the path is that, in the beginning, starting from v_1 it alternates between 2 and 3, and then alternates between 1 and 2, and finally alternates between 1 and 3. When n is odd the rightmost 1 is v_n, and when n is even the rightmost 1 is v_{n-1}. In the definition, notice that ℓ is the index of the leftmost 1 in the canonical 3-equitable coloring.

Definition 18. Given a path $P = v_1, v_2, \ldots, v_n$ and fixed sizes of color classes, we define a *canonical 3-equitable coloring* f of P such that the string $f(v_1)f(v_2)\ldots f(v_n)$ belongs to one of the below regular languages, depending on whether n is even or odd.

1. n is odd: $(23)^{k_1}(12)^{k_2}(13)^{k_3}1$ where, for $\ell = n - 2(|colorclass(1, f)| - 1)$, $k_1 = \frac{n-\ell+1}{2}$, $k_2 = |colorclass(2, f)| - k_1$, and $k_3 = n - (k_1 + k_2)$
2. n is even: $(23)^{k_1}(12)^{k_2}(13)^{k_3}$ where, for $\ell = n - 2\,|colorclass(1, f)| + 1$, $k_1 = \frac{n-\ell+1}{2}$, $k_2 = |colorclass(2, f)| - k_1$, and $k_3 = n - (k_1 + k_2)$

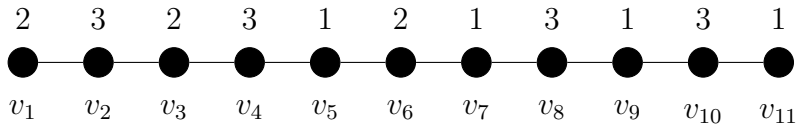


Figure 2.1: Canonical form for a path of odd length ($|colorclass(1, f)| = 4, |colorclass(2, f)| = 3, |colorclass(3, f)| = 4$)

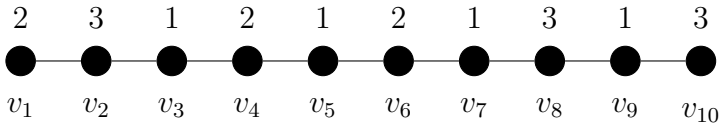


Figure 2.2: Canonical form for a path of even length ($|colorclass(1, f)| = 4, |colorclass(2, f)| = 3, |colorclass(3, f)| = 3$)

The following Observation follows from the definition of canonical 3-equitable colorings.

Observation 1. *Given a 3-equitable coloring f of a path P such that for v_i, $\ell \leq i \leq |V(P)|$, where v_ℓ is the leftmost 1 in f, $f(v_i)$ and the color of v_i in the canonical 3-equitable coloring of f are the same, then $\ell - 1$ is even.*

Proof. From Definition 18, when n is odd $\ell = n - 2(|colorclass(1, f)| - 1)$ and when n is even $\ell = n - 2\,|colorclass(1, f)| + 1$. The observation follows from this. □

2.4 Reconfiguration

In this section, we provide notation and terms for reconfiguration problems that we use throughout the thesis. We first provide a general framework and then give specific definitions. Note that reconfiguration problems are usually defined on the solution space of a problem. So, in general, the *reconfiguration graph* of a decision problem is defined to be the graph whose vertex set is the set of feasible solutions of the decision problem under consideration. We define the *reconfiguration step* of a reconfiguration problem to be a single transformation rule that transforms one feasible solution to another. We define an *adjacency relation* between two nodes of a reconfiguration graph if one node can be transformed to the other by the reconfiguration step. A *reconfiguration sequence* is defined as a path between two nodes in the reconfiguration graph. In this thesis, since we consider reconfiguration problems for different colorings, we define the nodes of a reconfiguration graph to be all possible colorings that satisfy a condition (e.g., colorings are proper for Coloring Reconfiguration).

Definition 19. Given a feasible solution A, we denote its node in the reconfiguration graph as the *node of* A.

Before describing Coloring Reconfiguration and Acyclic Coloring Reconfiguration, we define their reconfiguration step.

Definition 20. Given a graph G and a k-vertex coloring f, we *recolor* the vertex v in f if we change the color of v to obtain a k-vertex coloring that differs with f by 1.

Coloring Reconfiguration

Given a graph G and a positive integer k, we define the reconfiguration graph, $R_{k-COL}(G)$, to have its vertex set be all possible k-colorings of G. Two nodes in $R_{k-COL}(G)$ are said to be adjacent if the corresponding k-colorings differ by 1.

k-CR REACH

Input: A graph G and two k-colorings g_a and g_b.
Output: Is there a reconfiguration sequence between g_a and g_b?
Step: To recolor a vertex to obtain another coloring.

k-CR BOUND

Input: A graph G, two k-colorings g_a and g_b, and a positive integer ℓ.
Output: Is there a reconfiguration sequence of length at most ℓ between g_a and g_b?
Step: To recolor a vertex to obtain another coloring.

Acyclic Coloring Reconfiguration

Given a graph G and a positive integer k, we define the reconfiguration graph, $R_{k-ACR}(G)$, to have its vertex set be all possible k-acyclic colorings of G. In Acyclic Coloring Reconfiguration, similar to Coloring Reconfiguration, two nodes in $R_{k-ACR}(G)$ are adjacent if the corresponding k-acyclic colorings differ by 1.

k-ACR REACH

Input: A graph G and two k-acyclic colorings h_a and h_b.
Output: Is there a reconfiguration sequence between h_a and h_b?
Step: To recolor a vertex to obtain another acyclic coloring.

k-ACR BOUND

Input: A graph G, two k-acyclic colorings h_a and h_b, and a positive integer ℓ.
Output: Is there a reconfiguration sequence of length at most ℓ between h_a and h_b?
Step: To recolor a vertex to obtain another acyclic coloring.

Equitable Coloring Reconfiguration

Given a graph G and a positive integer k, we define the reconfiguration graph, $R_{k-ECR}(G)$, to have its vertex set be all possible k-equitable colorings of G. Two nodes of $R_{k-ECR}(G)$ are adjacent if the corresponding k-equitable colorings can be transformed to each other by swapping the colors of two vertices. Here, the reconfiguration step for some k-equitable coloring f is to swap the colors of two vertices.

k-ECR REACH

Input: A graph G with two k-equitable colorings f_s and f_e.
Output: Is there a reconfiguration sequence between f_s and f_e?
Step: To swap the colors of two vertices to obtain another equitable coloring.

k-ECR BOUND

Input: A graph G, two k-equitable colorings f_s and f_e, and a positive integer ℓ.
Output: Is there a reconfiguration sequence between f_s and f_e of length at most ℓ?
Step: To swap the colors of two vertices to obtain another equitable coloring.

Independent Set Reconfiguration

Given a graph G, we define the reconfiguration graph, $R_{IS-TJ}(G)$, to have its vertex set as the set of all possible independent sets. Two independent sets of equal size, say I_a and I_b, are adjacent if $|I_a \triangle I_b| = 2$ and two nodes in $R_{IS-TJ}(G)$ are adjacent if their corresponding independent sets are adjacent. (For two sets A and B, we denote as $A \triangle B$ the symmetric difference between A and B. More formally, $A \triangle B = (A \setminus B) \cup (B \setminus A)$.) Another way to formulate this variant of Independent Set Reconfiguration is through the TJ transformation rule.

TJ-IS REACH

Input: A graph G with two independent sets I_s and I_e.
Output: Is there a reconfiguration sequence between I_s and I_e?
Step: To replace a vertex in the independent set with another vertex to obtain another independent set.

2.5 Parameterized complexity

It is known that there can not be an algorithm that runs in polynomial time in terms of its input size for a problem that is NP-hard (unless P = NP). But, for many a problem there can be one or more secondary parameters other than input size that have a bearing on the computational complexity of the problem. In parameterized complexity, we try to take advantage of these secondary parameters while designing algorithms. Given a problem with parameter k, the problem is *fixed-parameter tractable for parameter* k if there exists an algorithm for the problem with running time in $O(f(k)n^c)$ where f is some computable function on k, n is the input size, and c is a constant independent of n and k [19]. The complexity class of all problems with fixed-parameter tractable algorithms is FPT.

To characterize when there is unlikely to exist a fixed-parameter tractable algorithm for a problem, Downey and Fellows defined the *W-hierarchy*, a hierarchy of complexity classes, as: FPT = W[0] $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \ldots \subseteq XP$ [19]. A problem of size n and parameter k is said to be in *XP for the parameter* k if there exists an algorithm with running time in $O(n^{f(k)})$. Note that it is believed that FPT $\neq$ W[1].

Chapter 3

Literature Survey

In this thesis, we study reconfiguration problems in graph coloring, specifically acylic colorings and equitable colorings. In Section 3.1, we discuss the background of some variants of colorings, including acyclic colorings and equitable colorings. Later, in Section 3.2, we look at previous work on reconfiguration problems in different variants of graph coloring.

3.1 Colorings

The history of the study of graph colorings can be traced back to the question by de Morgan, in 1852, asking if it is possible to color, with four colors, the counties of England in a map such that no two adjacent counties have the same color [43]. Since then, graph coloring has been studied by many mathematicians and is considered central in the field of discrete mathematics [14].

The question of de Morgan, which was generalized to ask for the minimum number of colors required to color a map, is widely called the four-color problem. Kempe proposed a solution for the four-color problem in 1879 [38], but it was disproved in 1890 by Heawood [34]. The four-color problem was finally solved by Appel and Haken using computers to perform extensive case-analysis in 1976 [3]. A simpler proof was given, later, by Robertson *et al.* [53].

The complexity of finding the minimum number of colors needed for a proper coloring of a graph was proved to be NP-hard by Karp [37]. Graph coloring continues to receive significant attention by researchers; for a comprehensive survey on graph coloring, we refer the reader to the book by Jensen and Toft [34].

3.1.1 Acyclic colorings

The acyclic coloring problem was first introduced by Grünbaum [25]. Since then it has been studied by many experts in graph coloring. Grünbaum, in the same paper, conjectured that for planar graphs the acyclic chromatic number is at most five, and this was later proved to be correct by Borodin [12]. Kostochka, in his thesis, proved that deciding whether a graph has an acyclic coloring using at most k colors is NP-complete by a reduction from the 3-colorability problem, by replacing each edge uv in a graph with three edge-disjoint paths from u to v of length two [40].

We use a similar method in Section 4.2 to show that k-ACR REACH is PSPACE-hard.

Erdős conjectured in 1976 that for a graph G with maximum degree $\Delta(G)$, when $\Delta(G)$ tends to infinity, $A(G) = O(\Delta(G)^{4/3})$ (where $A(G)$ is the minimum integer k such that G has a k-acyclic coloring); this conjecture was proved by Alon *et al.* using a probabilistic argument [2]. In the same paper, they also proved that $A(G)$ is in $\Omega(\Delta(G)^{4/3}/(\log \Delta(G))^{1/3})$.

3.1.2 Equitable colorings

The *equitable chromatic number* is the smallest k such that a k-equitable coloring exists for a given graph, and was first formalized by Meyer [48]. This problem has applications in load balancing [48], as well as scheduling and timetabling [24]. An application that was cited by Meyer [48] is the garbage collection routes problem, where the vertices of a graph correspond to garbage routes and an edge is drawn between two vertices if there is a constraint that the two corresponding garbage routes do not operate on the same day. Since it might be desirable to have approximately equal numbers of routes on any particular day of the week, the garbage collection routes problem becomes the same as finding a 6-equitable coloring (assuming a six-day workweek).

Erdős conjectured in 1964 that for any graph G, there exists a $(\Delta(G) + 1)$-equitable coloring [39], and this conjecture was proved in the Hajnal-Szemerédi theorem [26]. The Hajnal-Szemerédi theorem has spurred much research interest and has applications in finding bounds on sums of random variables that have limited dependence [33, 52]. Bodlaender and Fomin found that for bounded treewidth graphs, deciding if a graph has a k-equitable coloring can be solved in polynomial time by using Kostochka *et al.*'s results [42] to handle graphs with large maximum degree separately [5]. Fellows *et al.* used a reduction from the Multicolor Clique problem to show that deciding if a graph has a k-equitable coloring is W[1]-hard when parameterized by the treewidth of the graph plus the number of colors [22]. (The Multicolor Clique problem asks, given a graph and a k-coloring, if a

complete graph on k vertices such that all its vertices are colored distinctly is a subgraph.)

3.1.3 List colorings

We now consider another variant of colorings: list colorings. This variant of graph coloring has received attention from the research community and has seen the application of interesting techniques that combine different methods; for a more in-depth survey of list-coloring, we refer the reader to the survey by Alon [1].

Each vertex of a graph is assumed to have an associated list of permissible colors that the vertex can take. A *list coloring* of a graph is a k-coloring, for some k, where the color of each vertex belongs to the list associated with the vertex [18]. This variant was proposed, independently, by Vizing [57] and Erdős *et al.* [20].

Another motivation to study list coloring is the frequency assignment problem, where a vertex of a graph models a base station and an edge between two vertices represents the physical proximity of two base stations and hence the chance of interference, and the list associated with a vertex corresponds to the channels that can be assigned to it [28].

The *list-chromatic number of a graph* G is the smallest integer ℓ such that for any assignment of lists to vertices, each list is of size at least ℓ and G has a list coloring [34]. While the list-chromatic number of a graph is at least the chromatic number of a graph, it can be shown that, for some graphs, the difference between the chromatic number and the list-chromatic number can be arbitrarily large [1]. Given this, the conjecture by Bollobás and Harris that for any graph G, the list-chromatic number of the line graph of G is equal to the chromatic number of the line graph of G is surprising [6]. Vizing and Erdős *et al.* also, independently, proved a variant of Brooks' theorem that states that for any connected graph G that is not a complete graph or an odd cycle, the list-chromatic number of G is less than or equal to $\Delta(G)$ [6, 57]. Brooks' original theorem states that if a graph G is a complete graph or a cycle of odd length then the chromatic number is $\Delta(G) + 1$, and otherwise the chromatic number is at most $\Delta(G)$ [13].

3.1.4 Kempe chains

Kempe chains were first introduced by Alfred Kempe in his incorrect proof of the four-color problem [38]. Though Kempe's proof was incorrect, Kempe chains have proved to be a useful tool in graph coloring.

Given a graph G, a k-coloring, and any two colors c_1 and c_2, a (c_1, c_2)-*component* (or a Kempe chain) is defined as a connected subgraph of G whose vertices have color either c_1 or c_2.

We present some applications of Kempe chains in graph coloring. Kempe chains were used by Melnikov and Vizing to give a short proof of Brooks' theorem [47]. Las Vergnas and Meyniel also proved several theorems related to Hadwiger's conjecture using Kempe chains [44]. Hadwiger's conjecture states that every graph that does not have a K_{k+1} as a minor has a k-coloring, for any integer k [54].

Kempe chains also have applications in theoretical physics [56], timetables [51], and Markov chains [58].

3.1.5 Edge-colorings

An edge-coloring of a graph G is a function $f : E(G) \to \{1, 2, \ldots, k\}$ such that no two incident edges are assigned the same color by f [18].

We first note that, for a graph G, finding an edge-coloring of G is equivalent to finding a vertex-coloring on the line graph of G. Edge-colorings have not received as much attention as vertex colorings, but they still have connections to well-studied problems like coloring a map [34].

3.2 Reconfiguration of graph colorings

We give below related results on the reconfiguration of variants of graph coloring.

3.2.1 Recoloring a single vertex

We first consider the case when the reconfiguration step is to change the color of a single vertex. Consequently, the obvious reconfiguration problems are k-CR REACH and k-CR BOUND.

The problem of k-CR REACH was found to be PSPACE-complete for $k \geq 4$ by Bonsma and Cereceda [10]. For $k = 3$, Cereceda *et al.* described a polynomial-time algorithm that uses a characterization of when two 3-colorings are connected by a reconfiguration sequence [17]. Bonsma *et al.* considered the related k-CR BOUND problem and showed that it is NP-complete [11].

In terms of parameterized complexity, the following results are known for k-CR BOUND. Bonsma *et al.* proved that when parameterized by the length of the reconfiguration sequence ℓ, k-CR BOUND is W[1]-hard by a reduction from the Independent Set problem, which is known to be W[1]-hard with respect to the size of an independent set [11]. They also proved that k-CR BOUND is in XP when parameterized by the length of its reconfiguration sequence, ℓ, by a simple branching algorithm that considers, at each step, all possible k-colorings adjacent to the k-coloring under consideration, and the algorithm only considers a depth of recursion of ℓ. We use a similar technique in Section 4.3 to show an XP algorithm for k-ACR BOUND with respect to $\ell + \Gamma(G)$.

Bonsma *et al.* showed that when parameterized by $k+\ell$, the problem is fixed-parameter tractable [11]. Johnson *et al.* independently also showed that k-CR BOUND is fixed-parameter tractable for all fixed k when parameterized by the length of the reconfiguration sequence, but by a different algorithm [36].

The geometry of the reconfiguration graph, such as its diameter and whether it is connected, holds interest in the research community. So we consider the CONNECTIVITY variant of k-CR REACH. We say that a graph is *k-mixing* if any two k-colorings of the graph are connected by a reconfiguration sequence [9]. The mixing number of a graph G, $m(G)$, is defined as the smallest integer such that G is k-mixing for every $k \geq m(G)$.

An upper bound on the mixing number of a graph was given by Cereceda *et al.* using the degeneracy of a graph, which we define below [15]. The degeneracy of a graph, $deg(G)$, is the minimum positive integer such that every subgraph of G has a vertex of degree at most $deg(G)$. Cereceda *et al.* proved that for $k \geq deg(G) + 2$, G is k-mixing by using an inductive argument [15]. This implies that $m(G) \leq deg(G) + 2$.

Cereceda *et al.* also showed that for a graph G with chromatic number three, G is not 3-mixing (i.e., $R_{3-COL}(G)$ is not connected) [15]. They used a strategy that assigns a weight to an edge of a directed graph depending on the colors of its endpoints and the orientation of the edge, and proved that for any orientation where a cycle is directed the sum of the weights of the cycle's edges is the same for any two k-colorings if the corresponding vertices in the reconfiguration graph of both the k-colorings are connected. We use a similar strategy in Section 4.1 to prove that for non-bipartite graphs G with acyclic chromatic number 3, $R_{3-ACY}(G)$ is not connected.

Cereceda *et al.* proved results about bipartite graphs using the following transformation of a graph. A *pinch* is defined as the identification of two non-adjacent vertices that are at a distance of 2 from each other, and a graph G is *pinchable* to a graph H if there is a sequence of pinches that transforms G to H [16]. Cereceda *et al.* followed the same strategy as before to assign weights to directed edges to show that for a bipartite graph

G, the following statements are equivalent: G not being 3-mixing; the existence of a cycle in G where the sum of its edges, when the cycle is directed, is not equal to zero; and G being pinchable to C_6 [16]. They proved the equivalence of the three statements with the help of a *height function* that they define.

Cereceda *et al.* also showed that the problem of deciding whether a graph is 3-mixing is coNP-complete for bipartite graphs, but when restricted to planar bipartite graphs there exists a polynomial-time algorithm [16].

Bonamy *et al.* considered the structure of the reconfiguration graph of k-CR REACH and introduced a class of graphs that have chromatic number greater than or equal to k, the k-color-dense graphs [9]. They show that for $t \geq k+1$ and a k-color-dense graph G, the diameter of $R_{t-COL}(G)$ is in $O(|V(G)|^2)$. They also show that this bound is tight, by proving that there exists a chordal graph H and has chromatic number greater than or equal to k, that is also k-color-dense, such that $R_{(k+1)-COL}(H)$ has diameter in $\Theta(|V(G)|^2)$.

Bonamy and Bousquet proved an upper bound on the mixing number using the Grundy number, which we define below. Before defining the Grundy number, we define a greedy coloring. A *greedy coloring* of a graph is a k-coloring that is obtained by a greedy algorithm: the vertices of the graph are considered in some sequence and a vertex is assigned the least color that is not used in its neighborhood. The *Grundy number* of a graph G, $\chi_g(G)$, is the largest integer k for which the graph has a k-coloring that is a greedy coloring [34].

Bonamy and Bousquet showed by an inductive argument that when $k \geq \chi_g(G)+1$, then G is k-mixing and the diameter of $R_{k-COL}(G)$ is at most $4\chi_g(G)n$ [7]. They also proved that when $k \geq tw(G)+2$, G is k-mixing and the diameter of $R_{k-COL}(G)$ is at most $2(n^2+n)$, where $n = |V(G)|$ [7]. To prove this result, they define a class of canonical colorings called *coherent colorings*, which are dependent on a particular tree decomposition of the graph, and show that there is a reconfiguration sequence between any k-coloring and a coherent coloring, and that there is also a reconfiguration sequence between any two coherent colorings.

We next consider the reconfiguration problem for list colorings, k-LIST COLORING RECONFIGURATION. The reconfiguration step that has been studied in the literature is the same as k-CR REACH: to change the color of a vertex.

The k-LIST COLORING RECONFIGURATION problem, defined below, models situations where a feasible frequency assignment needs to be changed. Notice that since list colorings are a generalization of proper colorings, k-LIST COLORING RECONFIGURATION is a generalization of k-CR REACH. We now formally define k-LIST COLORING RECONFIGURATION, where the reconfiguration step is to change the color of a single vertex.

k-LIST COLORING RECONFIGURATION

Input: A graph G, a mapping that assigns a list of permissible colors to each vertex, and two list colorings g_a and g_b.

Output: Is there a reconfiguration sequence between g_a and g_b where each intermediate coloring is a list coloring?

Step: To recolor a vertex to obtain another list coloring.

Hatanaka *et al.* proved that k-LIST COLORING RECONFIGURATION is PSPACE-complete for graphs of pathwidth two. They also showed, through a dynamic-programming approach, a polynomial-time algorithm for graphs of pathwidth one [28].

3.2.2 Recoloring a single edge

The analogous *list edge-coloring* of a graph assumes that each edge e has an associated list of permissible colors, $L(e)$, and an edge-coloring is a *list edge-coloring* if each edge has a color from its associated list [18]. The k-LIST EDGE COLORING RECONFIGURATION can now be defined as below.

k-LIST EDGE COLORING RECONFIGURATION

Input: A graph G, a mapping that assigns a list of permissible colors to each edge, and two list edge-colorings g_a and g_b.

Output: Is there a reconfiguration sequence between g_a and g_b where each intermediate coloring is a list edge-coloring?

Step: To change the color of an edge to obtain another list edge-coloring.

Ito *et al.* considered k-LIST EDGE COLORING RECONFIGURATION and showed that the problem is PSPACE-complete even for planar graphs with maximum degree 2 and six colors [29]. An edge uv of G is classified as *tight* if $|L(e)| = \max\{d(u), d(v)\}$, *mild* if $|L(e)| = \max\{d(u), d(v)\} + 1$, and *slack* if $|L(e)| \geq \max\{d(u), d(v)\} + 2$. Ito *et al.* showed a polynomial-time algorithm for trees with the sufficient condition that each edge is either mild or slack, by employing breadth-first search to order the edges and recoloring an edge to its target color in this order by recoloring only edges that have not been encountered yet [29]. This sufficient condition was later improved by Ito, Kawamura, and Zhou [32], who showed that two list edge-colorings of a tree are connected by a reconfiguration sequence if either there is at most one tight edge or for each two tight edges e_1 and e_2, the path connecting them contains at least one slack edge.

3.2.3 Kempe chain recoloring

We consider a generalization of recoloring a single vertex, Kempe chain recoloring, which was first studied by Fisk [23]. A *Kempe change* for a graph G with a k-coloring is defined as an operation that interchanges the colors of vertices in a (c_1, c_2)-component (where c_1 and c_2 are any two colors) [21]. Two k-colorings are *Kempe equivalent* if there exists a sequence of Kempe changes that transforms one k-coloring to the other.

Mohar conjectured that for $k \geq 3$, all k-colorings of a graph are Kempe equivalent [49], but van den Heuvel showed a counterexample for $k = 3$ (see Figure 3.1) [55]. However, Mohar's conjecture was proved for $k \geq 4$ by Bonamy *et al.* [8], and Feghali *et al.* showed that when $k = 3$ and for cubic graphs, all 3-colorings are Kempe equivalent unless the graph is a complete graph on four vertices or a triangular prism (Figure 3.1) [21].

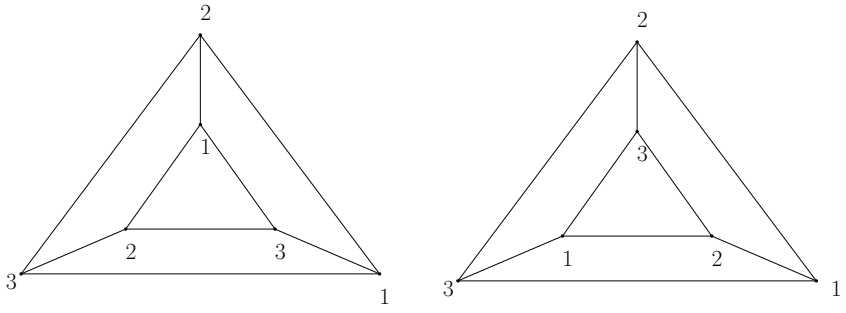


Figure 3.1: Two non-Kempe equivalent 3-colorings of the triangular prism.

3.2.4 Reconfiguring list $L(2,1)$-labelings

We next consider the list $L(2,1)$-labeling problem and its reconfiguration problem, k-LIST $L(2,1)$-LABELING RECONFIGURATION. An application for the list $L(2,1)$-labeling problem, defined later, is the assignment of channels to access points of a wireless local area network (WLAN), where there is a chance of interference between access points at close physical proximity. In situations where there is a requirement to have high throughput performance, the current channel assignment may need to be changed. These situations can be modeled by the k-LIST $L(2,1)$-LABELING RECONFIGURATION problem [31].

A *list* $L(2,1)$*-labeling* of a graph assumes that each vertex of a graph has a set of labels containing non-negative integers less than or equal to an integer k, called the *list* of the vertex, and assigns to each vertex a label from its list such that the labels of each pair of adjacent vertices differ by at least two and the labels of each pair of vertices that are at

distance two differ by at least one. The corresponding reconfiguration problem is defined as below, where the reconfiguration step is to change the label of a vertex while maintaining a list $L(2,1)$-labeling.

k-LIST $L(2,1)$-LABELING RECONFIGURATION

Input: A graph G and two list $L(2,1)$-labelings g_a and g_b.

Output: Is there a reconfiguration sequence between g_a and g_b where each intermediate coloring is a list $L(2,1)$-labeling?

Step: To recolor a vertex to obtain another list $L(2,1)$-labeling.

Ito *et al.* showed that k-LIST $L(2,1)$-LABELING RECONFIGURATION is PSPACE-complete even for bipartite planar graphs and $k \geq 6$ [31]. They proved PSPACE-completeness by a reduction from INDEPENDENT SET RECONFIGURATION under the token sliding adjacency relation. Ito *et al.* also showed that k-LIST COLORING RECONFIGURATION can be solved in linear time for $k \leq 4$. They showed this result by considering only when $k = 4$, as when $k \leq 3$ and the graph has a $L(2,1)$-labeling it can be proved that the graph has a constant number of vertices. Ito *et al.* then showed for a tree T, when each vertex v of T has more than $\max\{d(u) \mid u \in N_T(v)\} + 6$ labels in the list associated with it, a list $L(2,1)$-labeling can be reconfigured to any other list $L(2,1)$-labeling by performing a breadth-first search on T and relabeling a vertex to its target label as it is encountered through a sequence of relabelings. When a vertex v of T is relabeled to its target label, the vertices in the sequence of relabelings to relabel v are those vertices that have not been encountered yet [31].

Chapter 4

Acyclic Coloring Reconfiguration

In this chapter, we study reconfiguration problems in the acyclic coloring setting. We study whether results for k-CR REACH also apply to k-ACR REACH. In Section 4.1, we show that Cereceda *et al.*'s results [15] apply to k-ACR REACH, with modifications, by giving examples of graphs where the reconfiguration graph of k-ACR REACH is not connected. We also show that k-ACR REACH is PSPACE-hard through a reduction from k-CR REACH in Section 4.2. In Section 4.3, we also show that Bonsma *et al.*'s results [11] apply to k-ACR BOUND by showing that k-ACR BOUND is in XP when parameterized by the length of the reconfiguration sequence plus the length of the longest induced cycle ($\Gamma(G)$).

4.1 Non-bipartite graphs of acyclic chromatic number 3

A modification of Cereceda *et al.*'s result, which we summarize in Section 3.2.1, yields a similar result for k-ACR REACH [15]. We remind the reader that weights are assigned to a directed edge depending on the colors of the edge's endpoints and the direction of the edge (Definition 8), and the weight of a directed subgraph is the sum of the weights of the edges of the directed subgraph (Definition 9).

We prove, below, a necessary condition for two 3-acyclic colorings of a graph to be connected in the reconfiguration graph. We show that for two 3-acyclic colorings whose nodes are connected in the reconfiguration graph, the weights of any directed cycle must be equal.

Before stating the lemma, we give Cereceda *et al.*'s original lemma.

Lemma 1 ([15]). *Given two 3-colorings f and h of a graph G, and a cycle C in G, if the nodes of f and h are connected in R_{3-COL}, then $W(\overrightarrow{C}, f) = W(\overrightarrow{C}, h)$.*

Lemma 2. *Given two 3-acyclic colorings f and h of a graph H that contains a cycle C, if f and h are in a connected subgraph of $R_{3-ACR}(H)$, then on orienting H in any way such that C is directed, $W(\overrightarrow{C}, f) = W(\overrightarrow{C}, h)$.*

Proof. It is sufficient to prove that the lemma is true when f and h are adjacent; it is then true by transitivity for all connected 3-acyclic colorings in $R_{3-ACR}(H)$. Let v be the vertex where f and h differ by 1. We prove the lemma by showing that the weights of edges in C which are incident to v are of equal magnitude but opposite in sign in $W(\overrightarrow{C}, f)$ and $W(\overrightarrow{C}, h)$.

We first prove that for the vertices x and y in $V(C)$ that are adjacent to v, $f(x) = f(y) = h(x) = h(y)$. If x and y have different colors in f and h, then v can take only one color in both f and h. This contradicts $f(v) \neq h(v)$; so, it must be that x and y have the same color in f and h ($f(x) = h(y) = f(x) = h(y)$).

Since f and h differ by 1 at v, the only two edges of $\overrightarrow{C}$ where $W(\overrightarrow{C}, f)$ and $(W(\overrightarrow{C}, h)$ might differ are xv and vy. If we show that the weights of xv and vy are of equal magnitude but opposite in sign, then the lemma holds. We can also say that the directions of xv and vy in $\overrightarrow{C}$ can be either $\overrightarrow{xv}, \overrightarrow{vy}$ or $\overleftarrow{xv}, \overleftarrow{vy}$ as $\overrightarrow{C}$ is a directed cycle. Since the cases are symmetric, we assume without loss of generality that the directions of the edges are $\overrightarrow{xv}$ and $\overrightarrow{vy}$. By performing case analysis on all possible values of $f(v)$ and $f(x) = f(y)$, it can be seen that $w(\overrightarrow{xv}, f) + w(\overrightarrow{vy}, f) = 0$. Similarly, $w(\overrightarrow{xv}, h) + w(\overrightarrow{vy}, h) = 0$. Clearly $W(\overrightarrow{C}, f) = W(\overrightarrow{C}, h)$ must hold. □

The following corollary is immediate from Lemma 2.

Corollary 1. *Given any two 3-acyclic colorings f and h of a graph G with a cycle C, for any orientation of G such that C is directed, if $W(\overrightarrow{C}, f) \neq W(\overrightarrow{C}, h)$ then there exists no path between the nodes of f and h, respectively, in $R_{3-ACR}(G)$.*

We define a 3-vertex coloring $\beta_{f,H}$ of H from a 3-acyclic coloring f of H. We prove Theorem 2 using this result.

Definition 21. Given a 3-acyclic coloring f of graph H, let $\beta_{f,H}$ be a 3-vertex coloring

such that for every vertex v in H, $\beta_{f,H}(v)$ has value as defined in Equation 4.1.

$$\beta_{f,H}(v) = \begin{cases} 1 & \text{if } f(v) = 2 \\ 2 & \text{if } f(v) = 1 \\ 3 & \text{if } f(v) = 3 \end{cases} \tag{4.1}$$

We first show that $\beta_{f,H}$ is acyclic if f is acyclic.

Claim 1. *Given a graph H and a 3-acyclic coloring f of H, the 3-vertex coloring $\beta_{f,H}$ is a 3-acyclic coloring.*

Proof. Since $colorclass(1, f) = colorclass(2, \beta_{f,H})$, $colorclass(2, f) = colorclass(1, \beta_{f,H})$, and $colorclass(3, f) = colorclass(3, \beta_{f,H})$: if the endpoints of an edge have the same color in $\beta_{f,G}$, they also have the same color in f; and if a cycle is 2-chromatic in $\beta_{f,G}$, it is also 2-chromatic in f. Hence, as f is acyclic, $\beta_{f,G}$ is also acyclic. □

We are now ready to prove Theorem 2. Before stating the theorem, we give Cereceda *et al.*'s original theorem.

Theorem 1 ([15])**.** *For a graph G with chromatic number three, R_{3-COL} is not connected.*

Theorem 2. *For any graph H that is not bipartite and has acyclic chromatic number 3, $R_{3-ACR}(H)$ is not connected.*

Proof. We prove that $R_{3-ACR}(H)$ is not connected by showing that there exist two 3-acyclic colorings whose nodes are not connected in $R_{3-ACR}(H)$. We claim that f, any 3-acyclic coloring of H, and $\beta_{f,H}$, which is a 3-acyclic coloring by Claim 1, are two such 3-acyclic colorings.

It follows from Corollary 1 that showing the existence of a cycle whose weight is different in f and $\beta_{f,H}$ is sufficient to prove that the nodes of f and $\beta_{f,H}$ are not connected in R_{3-ACR}. Since H is not bipartite, there exists a cycle C in H of odd length. We also assume that H is oriented such that C is directed. For any edge uv in H, it can be verified for all possible values of $f(u), f(v), \beta_{f,H}(u), \beta_{f,H}(v)$ that $w(\overrightarrow{uv}, f) = -w(\overrightarrow{uv}, \beta_{f,H})$; since the weight of a directed cycle is the sum of the weights of its directed edges, $W(\overrightarrow{C}, f) = -W(\overrightarrow{C}, \beta_{f,H})$. Since C is of odd length and the weight of each edge can be either 1 or -1, $W(\overrightarrow{C}, f) \neq 0$ and $W(\overrightarrow{C}, \beta_{f,H}) \neq 0$. Clearly $W(\overrightarrow{C}, f) \neq W(\overrightarrow{C}, \beta_{f,H})$ since $W(\overrightarrow{C}, f), W(\overrightarrow{C}, \beta_{f,H}) \neq 0$ and $W(\overrightarrow{C}, f) = -W(\overrightarrow{C}, \beta_{f,H})$. □

4.2 k-ACR REACH is PSPACE-hard

We prove that k-ACR REACH is PSPACE-hard by reducing k-CR REACH to it. Since k-CR REACH is PSPACE-complete for $k \geq 4$, this suffices [10]. Before stating our reduction, we prove Theorem 3 using a technique similar to that used by Cereceda *et al.* when showing that any two k-colorings of a graph G are connected by a reconfiguration sequence when $k \geq deg(G) + 2$ [15].

Theorem 3. *Given a graph G and any two k-acyclic colorings f and h, if $k \geq n+2$, then there exists a reconfiguration sequence between f and h.*

Proof. We prove our theorem by induction on the number of vertices in G. The theorem is trivially true for one vertex. By the induction hypothesis, for a graph on $n-1$ vertices, any two k-acyclic colorings are connected in the reconfiguration graph. So, for a graph G on n vertices, we consider some vertex v. Let G_{n-1} denote the induced graph on the vertices $V(G) \setminus \{v\}$ and the reconfiguration sequence between $f|_{G_{n-1}}$ and $h|_{G_{n-1}}$ be $\gamma_1^{n-1}, \gamma_2^{n-1}, \ldots, \gamma_t^{n-1}$ for some t. We also set γ_{t+1}^{n-1} to $h|_{G_{n-1}}$.

We next prove that the k-acyclic colorings $\gamma_1^{n-1}, \ldots, \gamma_t^{n-1}, \gamma_{t+1}^{n-1}$ can be extended to a sequence of adjacent k-acyclic colorings of G. Let γ_i, $1 \leq i \leq t+1$, be the k-vertex coloring of G that is extended from γ_i^{n-1} by letting $\gamma_i(v) = f(v)$. We consider the smallest i, $1 \leq i \leq t+1$, for which γ_i is not acyclic and also let $\gamma_0 = f$. Note that the k-vertex colorings γ_{i-1} and γ_i use at maximum n colors, and so $|colors(V(G), \gamma_{i-1}) \cup colors(V(G), \gamma_i)| = n+1$. Since $k \geq n+2$, we can have an intermediate k-acyclic coloring α by recoloring v in γ_{i-1} to a color which is not used in either γ_{i-1} or γ_i. We also change the color of v for every k-vertex coloring in the set $\{\gamma_i, \ldots, \gamma_{t+1}\}$ to $\alpha(v)$. By placing α between γ_{i-1} and the modified γ_i we have a sequence of adjacent k-vertex colorings where γ_i is now acyclic. We proceed similarly for all k-vertex colorings in the sequence that are not acyclic. Since $\gamma_{t+1}^{n-1} = h|_{G_{n-1}}$, either $\gamma_{t+1} = h$ or γ_{t+1} and h are adjacent. □

We reduce k-CR REACH to k-ACR REACH by using a technique similar to that used by Kostochka when proving that deciding whether a graph has an acyclic coloring using at most k colors is NP-complete [41]. Kostochka reduced the problem of 3-colorability (deciding if a graph has a 3-coloring) to the problem of finding the acyclic chromatic number of a graph by replacing each edge with three internally disjoint paths of length two.

Given an instance (G, g_s, g_e) of k-CR REACH, we construct a corresponding instance (H, h_s, h_e) of k-ACR REACH. We replace each edge uv of G with k internally disjoint uv paths of length two. We call such internal vertices in these uv paths *subdivision* vertices

and other vertices *original* vertices. For each original vertex u in H, we denote the corresponding vertex in G as u_G. Similarly for each vertex u in G, we denote the corresponding vertex in H by u_H. The k-acyclic colorings h_s, h_e are any k-acyclic colorings of G such that, for every original vertex u in H, $h_s(u) = g_s(u_G)$ and $h_e(u) = g_e(u_G)$.

Observation 2. *The number of vertices and edges added to graph G in our reduction is in $O(n^2k)$, where $n = |V(G)|$.*

Proof. We prove the observation by first noting that, in our reduction, we add k vertices and $2k$ edges to G for each edge in $E(G)$. Since the number of edges in $E(G)$ is at most n^2, it follows that the number of vertices and edges added by our reduction is at most n^2k and $n^2 2k$ respectively. Clearly the number of vertices and edges added to G is in $O(n^2k)$. □

Observation 3. *Given any subdivision vertex w in H, we can say the following about w.*

1. *$|N_H(w)|= 2$.*
2. *The vertices in $N_H(w)$, say u and v, are original vertices.*
3. *For any k-acyclic coloring h of H, the vertices in $N_H(w)$, u and v, have $h(u) \neq h(v)$.*

Proof. The first two parts of the observation follow from the reduction. For the third part, we prove that there exists a 2-chromatic cycle otherwise. Since w is a subdivision vertex, $u_G v_G \in E(G)$. From our reduction, since $u_G v_G \in E(G)$, we can say that $|N_H(u) \cap N_H(v)|= k$. If $h(u) = h(v)$, since h is a k-coloring the size of the set $colors(N_H(u) \cap N_H(v), h)$ can be at most $k - 1$, and so there must exist two vertices in $N_H(u) \cap N_H(v)$ with the same color in h, which forms a 2-chromatic C_4. □

The following two observations follow from our reduction.

Observation 4. *For every edge uv in $E(H)$, the number of subdivision vertices in $\{u, v\}$ is exactly one.*

Observation 5. *For every induced cycle C in H, $|V(C)|\geq 4$.*

Proof. We prove by way of contradiction. Suppose there is a cycle C of length three. Then there is an edge e in C such that the endpoints of e are either both subdivision vertices or both original vertices. This contradicts Observation 4. □

Before stating the next observation, we make the following definition. For each k-coloring g of G, we denote as $R_{acyclic}(g)$ the set of all k-acyclic colorings h of H such that for every original vertex u in H, $h(u) = g(u_G)$ (we prove later in Claim 3 that such k-acyclic colorings must exist).

Observation 6. *Given a set $R_{acyclic}(g)$ for a k-coloring g in G, the following hold.*

1. *For each original vertex v and any two k-acyclic colorings in $R_{acyclic}(g)$, h_a and h_b, $h_a(v) = h_b(v)$.*
2. *For each induced cycle C in H and any k-acyclic coloring h in $R_{acyclic}(g)$, there exist two original vertices w, x in $V(C)$ such that $h(w) \neq h(x)$.*

Proof. Part one of the observation follows from the definition of $R_{acyclic}(g)$. We prove part two of the observation by way of contradiction. Suppose there exists a cycle C such that no two original vertices in $V(C)$ have the same color in h. As C is of length greater than or equal to four (Observation 5) and each edge has exactly one subdivision vertex (Observation 4), there are two original vertices w, x in $V(C)$ that are both adjacent to the same subdivision vertex in $V(C)$. But, as $h(w) = h(x)$, this contradicts part three of Observation 3. □

Claim 2. *Given a graph H with a k-acyclic coloring h and $k \geq 4$, the following hold for a subdivision vertex u.*

1. *There exists a color c such that $c \notin colors(N_H(u), h) \cup \{h(u)\}$.*
2. *Given such a color c, the k-vertex coloring f such that f and h differ by 1 at u with $f(u) = c$ is a k-acyclic coloring.*

Proof. To prove part one of the claim, we note that the neighborhood of u has only two vertices (Observation 3) and since $k \geq 4$, there exists at least one color c for which $c \notin colors(N_H(u), h) \cup \{h(u)\}$.

We prove part two of the claim by showing that f is a k-coloring and that it contains no 2-chromatic cycle. Since $c \notin colors(N_H(u), h)$, it is clear that f is a k-coloring. Since u is a subdivision vertex, $|N_H(u)| = 2$ and the vertices in $N_H(u)$ have distinct colors (Observation 3). It follows from u having exactly two neighbors that all cycles that contain u must include both the vertices in $N_H(u)$. Since both the vertices in $N_H(u)$ have distinct colors and $c \notin colors(N_H(u), h)$, all cycles containing u must be colored by at least three colors. □

Lemma 3. *Given a graph G and a k-coloring f of G, if there exists no 2-chromatic induced cycle, then there exists no 2-chromatic cycle in the graph.*

Proof. Towards contradiction, let there exist a 2-chromatic cycle C which is not an induced cycle. Assume without loss of generality that the two colors are 1 and 2. Let $C_{induced}$ be an induced cycle such that $V(C_{induced}) \subset V(C)$. Such a $C_{induced}$ exists since C is not an induced cycle. We know that for every vertex v in $V(C_{induced})$, $f(v) = 1$ or $f(v) = 2$. This would mean that there exists a 2-chromatic induced cycle in G. □

Claim 3. *We can say the following about $R_{k-COL}(G)$ and $R_{k-ACR}(G)$, for $k \geq 4$.*

1. *For every k-acyclic coloring h in $R_{k-ACR}(H)$, the k-vertex coloring g of G where for every vertex $u \in G$, $g(u) = g(u_H)$, is proper.*
2. *For every k-coloring g in $R_{k-COL}(G)$, there exists a k-acyclic coloring h in $R_{k-ACR}(H)$ such that $g(u_G) = h(u)$ for every original vertex u.*

Proof. We note that if the first part of the claim is false, then there is an edge uv in $E(G)$ for which $g(u) = g(v)$ and so $h(u_H) = h(v_H)$. To prove the first part of the claim, it thus suffices to prove that for no edge $uv \in E(G)$, $h(u_H) = h(v_H)$. By our reduction, there is a subdivision vertex that is adjacent to only u and v. Now, from Observation 3, it follows that $h(u) \neq h(v)$.

We prove the second part of our claim by constructing a k-acyclic coloring h satisfying $h(u) = g(u_G)$ for every original vertex u in H. For every original vertex u, let $g(u_G) = h(u)$. And for every subdivision vertex v with neighbors w and x, let $h(v)$ take a value such that $h(v) \neq g(w_G), h(v) \neq g(x_G)$ (this is possible since $k \geq 4$). Clearly h is a k-coloring, so what is left to prove is that there exists no 2-chromatic cycle. Towards contradiction, suppose there is a 2-chromatic cycle. We can conclude from Lemma 3 that there also exists an induced 2-chromatic cycle C. Since C is of length greater than or equal to four (Observation 5) and each edge has exactly one subdivision vertex (Observation 4), there are two original vertices p, q in C that are at a distance of two from each other. Also, as C is 2-chromatic and h is proper, $h(p) = h(q)$. For each original vertex z $g(z_G) = h(z)$, so $g(p_G) = g(q_G)$. But, since $p_G q_G \in E(G)$, this contradicts the fact that g is proper. □

The corollary below follows from part two of Claim 3.

Corollary 2. *Given a k-coloring g of G, the set $R_{acyclic}(g)$ is non-empty for $k \geq 4$.*

Claim 4. *Given a graph G, $k \geq 4$, for source and target k-colorings g_s and g_e of G there exist source and target k-acyclic colorings h_s and h_e of H such that for each original vertex u $g_s(u_G) = h_s(u)$, $g_e(u_G) = h_e(u)$ and vice-versa.*

Proof. The first part of the claim follows from part two of Claim 3, and the second part of the claim follows from part one of Claim 3. □

Claim 5. *For each k-coloring g of G, the induced subgraph on the nodes of $R_{acyclic}(g)$ in $R_{k-ACR}(H)$ is a connected subgraph.*

Proof. We can equivalently say that the nodes of any two k-acyclic colorings in $R_{acyclic}(g)$, say h_a and h_b, are connected in $R_{k-ACR}(H)$. The proof proceeds by induction on the cardinality of the set $\{v \in V(H) \mid h_a(v) \neq h_b(v)\}$, i.e., the number of vertices where h_a and h_b differ. When h_a and h_b differ by 1, the statement is trivially true. When h_a and h_b differ by t $(t > 1)$, take a vertex, say v, such that $h_a(v) \neq h_b(v)$. From part one of Observation 6, since each original vertex has the same color in any two k-acyclic colorings in $R_{acyclic}(g)$, v is a subdivision vertex as $h_a(v) \neq h_b(v)$. Let h_c be the k-vertex coloring that differs with h_a by 1 at v, where $h_c(v) = h_b(v)$. If h_c is acyclic, then since h_c differs with h_b by $t-1$, h_c and h_b are connected by our induction hypothesis; so, since h_a and h_c differ by 1, h_a and h_b are connected by transitivity.

We prove that h_c is not acyclic if and only if there is either a vertex that is adjacent to v with the same color as v in h_c or a 2-chromatic cycle that contains v. The if part is trivially true and the only if part follows as if this were not the case, since h_c and h_a differ by 1 at v, a 2-chromatic cycle or two adjacent vertices with the same color must also exist in h_a, which contradicts the fact that h_a is a k-acyclic coloring. We now prove that h_c is acyclic by showing that h_c is proper and that there is no 2-chromatic cycle in h_c.

We prove by way of contradiction that h_c is proper. From the previous paragraph, if h_c is not proper then there is a vertex adjacent to v that has the same color as v in h_c. Also, as v is a subdivision vertex, it has two neighbors (Observation 3), say w and x, and they are original vertices by Observation 4. Since $h_a, h_b \in R_{acyclic}(g)$, it follows from Observation 6 that $h_a(w) = h_b(w)$ and $h_a(x) = h_b(x)$. Since h_a and h_c differ by 1 at v, it also holds that $h_c(w) = h_a(w)$ and $h_c(x) = h_a(x)$. Consequently, $h_b(w) = h_c(w)$ and $h_b(x) = h_b(x)$. Since $h_c(v) = h_b(v)$, clearly the colors of w, x, and v are the same in h_c and h_b respectively $(h_b(w) = h_c(w), h_b(x) = h_c(x), h_b(v) = h_c(v))$. So, if a vertex adjacent to v has the same color in h_c, then it also has the same color in h_b, contradicting the fact that h_b is a k-coloring.

We now prove that h_c is acyclic by showing that there is no 2-chromatic cycle in h_c. To prove our statement, it is sufficient to show that there is no 2-chromatic cycle containing v

in h_c as we showed earlier that it would contradict the fact that h_b is acyclic. Since v has only two neighbors, w and x, any cycle containing v contains w and x as well. If we show that the colors of w, x, and v are distinct in h_c, then there can be no 2-chromatic cycle in h_c. As w and x are original vertices that are both adjacent to the same subdivision vertex, from part three of Observation 3, $h_c(w) \neq h_c(x)$. It also follows from h_c being a k-coloring that $h_c(v) \neq h_c(w)$ and $h_c(v) \neq h_c(x)$. Clearly w, x, and v have distinct colors in h_c. Thus, h_c is a k-acyclic coloring that is adjacent to h_a and differs from h_b by $t-1$. □

Claim 6. *For every two adjacent k-colorings g_a and g_b of G, $k \geq 4$, there exist two k-acyclic colorings in $R_{acyclic}(g_a)$ and $R_{acyclic}(g_b)$, respectively, whose nodes are connected in $R_{k-ACR}(H)$.*

Proof. We prove the claim by showing that for any k-acyclic coloring h_a in $R_{acyclic}(g_a)$ there is a reconfiguration sequence between h_a and some k-acyclic coloring in $R_{acyclic}(g_b)$. Let v be the original vertex in H for which $g_a(v_G) \neq g_b(v_G)$ and U be the set of neighbors of v with color $g_b(v_G)$ in h_a. Since the color of v in any k-acyclic coloring in $R_{acyclic}(g_b)$ is $g_b(v_G)$, the color of v should be changed at some point in the reconfiguration sequence between h_a and some k-acyclic coloring in $R_{acyclic}(g_b)$. Before this, all the vertices in the set U should be recolored to some color that is not $g_b(v_G)$. We first show the existence of a reconfiguration sequence that changes the color of each vertex in U, and then we show that the color of v can be changed to $g_b(v_G)$. Note that since only the colors of vertices in U, which are subdivision vertices (Observation 4), and v are changed, the k-acyclic coloring at the end of the reconfiguration sequence is in $R_{acyclic}(g_b)$.

We show the existence of a reconfiguration sequence that changes the color of each vertex in U. From Claim 2, any subdivision vertex can be recolored to some color so that the k-vertex coloring that we obtain is acyclic. It follows from this that we can recolor each vertex in U in any k-acyclic coloring to some color to obtain a k-acyclic coloring as all vertices in U are subdivision vertices. So, by iteratively recoloring vertices in U, we get a reconfiguration sequence between h_a and some k-acyclic coloring h_c such that for every u in U, $h_c(u) \neq g_b(v_G)$ and for every vertex $w \in V(H) \setminus U$, $h_c(w) = h_a(w)$. Clearly h_c is in $R_{acyclic}(g_a)$.

We now claim that by recoloring v in h_c to $g_b(v_G)$, the k-vertex coloring that we obtain, say h_b, is acyclic. Since h_b and h_c differ by 1 at v and no vertex adjacent to v has color $g_b(v_G)$ in h_c, it follows that h_b is proper. We can also say that if there is a 2-chromatic cycle in h_b, then the 2-chromatic cycle has to contain v as otherwise this would contradict the fact that h_c is acyclic.

We now show by way of contradiction that there is no 2-chromatic cycle in h_b. Suppose instead that there exists an induced 2-chromatic cycle C in h_b. We know that C has to

contain v. As C is 2-chromatic and h_b is proper $|C| \geq 4$. Hence, there exists a vertex y in C at a distance of two from v such that $h_b(v) = h_b(y)$. The vertex x in C that is adjacent to both v and y is a subdivision vertex as v is an original vertex (Observation 4). As x is a subdivision vertex, it follows that y is an original vertex. Since for each original vertex z $g_b(z_G) = h_b(z)$, $g_b(v_G) = g_b(y_G)$. But, since $v_G y_G \in E(G)$, this contradicts the fact that g_b is proper.

We have shown that h_b is acyclic, and since we have only changed the color of vertices in $U \cup \{v\}$, $h_b \in R_{acyclic}(g_b)$. Clearly there exists a reconfiguration sequence between h_a and h_b, a k-acyclic coloring in $R_{acyclic}(g_b)$. □

Claim 7. *For every two adjacent k-acyclic colorings h_a and h_b in $R_{k-ACR}(H)$, there exist two connected k-colorings g_a, g_b in $R_{k-COL}(G_c)$ such that for every vertex u in $V(G)$, $g_a(u) = h_a(u_H), g_b(u) = h_b(u_H)$.*

Proof. We need to show that the k-vertex colorings g_a and g_b where, for every original vertex u, $g_a(u) = h_a(u_H), g_b(u) = h_b(u_H)$, are k-colorings and their nodes in $R_{k-ACR}(H)$ are connected. From part one of Claim 3, we know that the k-vertex colorings g_a and g_b such that $g_a(u) = h_a(u_H), g_b(u) = h_b(u_H)$ for every every vertex $u \in V(G)$ are proper. To prove that g_a and g_b are connected in $R_{k-COL}(G)$, let us consider the vertex where h_a and h_b differ, say v. Note that such a v must exist since h_a and h_b are adjacent. If v is a subdivision vertex then for every vertex $u \in V(G)$, $g_a(u) = g_b(u)$, and so the corresponding vertices for g_a and g_b in $R_{k-COL}(G)$ are the same; if v is an original vertex, then g_a and g_b differ by 1 at v, and so the nodes of g_a and g_b are adjacent in $R_{k-COL}(G)$. □

We are now ready to prove Theorem 4.

Theorem 4. k-ACR REACH *is PSPACE-hard for* $4 \leq k < n + 2$.

Proof. We show that our reduction can be executed in polynomial time and space, and that an instance of k-CR REACH, (G, g_s, g_e), is a yes-instance if and only if the instance (H, h_s, h_e) of k-ACR REACH is a yes-instance. Since the number of vertices and edges we add to G in our reduction is in $O(n^2 k)$ (Observation 2) and $k < n+2$, our reduction can be executed using polynomial time and space. From Claim 4, each instance of k-CR REACH has a corresponding instance of k-ACR REACH and vice-versa. For the if part, if there exists a reconfiguration sequence between g_s and g_e, then there also exists a reconfiguration sequence between h_s and h_e in $R_{k-ACR}(H)$ (Claim 6). For the only if part, if there exists a reconfiguration sequence between h_s and h_e, then there exists a reconfiguration sequence between g_s and g_e in $R_{k-COL}(G)$ (Claim 7). □

It follows from Theorem 3 that when $k \geq n + 2$ any instance of k-ACR REACH is a yes-instance. Hence we can make the following note.

Note 1. As a consequence of Theorem 3, we can restrict our attention to $k < n + 2$ in k-ACR REACH.

4.3 k-ACR BOUND is in XP

In this section, we prove that k-ACR BOUND is in XP when parameterized by the length of the reconfiguration sequence plus the length of the longest induced cycle ($\Gamma(G)$). For a graph on n vertices there are at most k^n k-acyclic colorings and k-ACR REACH is PSPACE-hard. By fixing the length of the reconfiguration sequence in k-ACR BOUND to k^n, we establish the fact that k-ACR BOUND is weakly PSPACE-hard as k^n is exponential in terms of n. This implies that there is no polynomial-time algorithm for k-ACR BOUND when the data is encoded in unary, unless P = PSPACE. Our algorithm implies that k-ACR BOUND is in XP for parameter $\ell + \Gamma(G)$ as there is an algorithm that runs in time $O(n^{f(\ell+\Gamma(G))})$, where f is some computable function on $\ell + \Gamma(G)$. We first give a high-level description of our algorithm, then formally describe our algorithm and show a proof of correctness and running time.

Algorithm 1 is a simple branch-and-bound algorithm that is inspired by Bonsma *et al.*'s work [11], which we summarize in Section 3.2.1. For an instance (H, h_s, h_e) of k-ACR REACH, we start by considering all possible $(k-1)n$ k-vertex colorings that can be obtained by changing the color of a vertex v in h_s. For each such k-vertex coloring, it can be checked in linear time whether the k-vertex coloring is proper. To check for the existence of a 2-chromatic cycle, Algorithm 2, described later, is called. If the k-vertex coloring is acyclic, the algorithm checks if the k-vertex coloring is h_e, and if not recursively considers all possible adjacent k-vertex colorings. Since the length of the reconfiguration sequence can be at most ℓ, it suffices to consider a depth of recursion of ℓ. (The reader is reminded that ℓ is some positive integer taken as input by k-ACR BOUND.)

Since the length of the reconfiguration sequence is at most ℓ, Algorithm 1 also has depth of recursion of at most ℓ. For one k-acyclic coloring there are at most $(k-1)n$ adjacent k-acyclic colorings, so at each step of the algorithm we branch into $(k-1)n$ possible directions. At each branch, it can be checked if the k-vertex coloring is proper in $O(|V(H)|)$ time and if there is no 2-chromatic cycle in $O(\Delta(G)^{\Gamma(G)})$ time. Thus, this yields a $O((kn)^{\ell}(n + \Delta(G)^{\Gamma(G)}))$-time algorithm.

Algorithm 1 AcyclicRecolor

Input: A graph H, a source k-acyclic coloring h_s of H, a target k-acyclic coloring h_e of H, and a positive integer ℓ.

Output: Outputs YES if there exists a reconfiguration sequence of length less than or equal to ℓ and NO otherwise.

```
1: procedure AcyclicRecolor(H, h_s, h_e, ℓ)
2:     if AcyclicRecolorRecurse(H, h_s, h_e, 0, ℓ) = true then
3:         return YES
4:     else
5:         return NO
6:     end if
7: end procedure
8: procedure AcyclicRecolorRecurse(H, h_s, h_e, ℓ_current, ℓ)
9:     if ℓ_current ≥ ℓ then
10:        return false
11:    else if h_s = h_e then
12:        return true
13:    end if
14:    for each k-vertex coloring h_t that can be obtained by changing the color of a single vertex, say v, in h_s do
15:        if there exists an edge uv in E(H) such that h_t(u) = h_t(v) or TwoChromaticCycle(H, h_t, v) then
16:            continue
17:        else if AcyclicRecolorRecurse(H, h_t, h_e, ℓ_current + 1, ℓ) = true then
18:            return true
19:        end if
20:    end for
21:    return false
22: end procedure
```

Algorithm 2 takes as input a graph H, a k-acyclic coloring h of H, and a vertex v of H, and outputs true if there exists a 2-chromatic cycle containing v and false otherwise. Algorithm 2 calls an auxiliary procedure TwoChromaticCycleRecurse with parameters H, h, v, and a dynamic array containing only v. The algorithm enumerates all possible paths of length $\Gamma(H)$ from v, and this suffices to enumerate all possible cycles of length $\Gamma(H)$ from v as well. The algorithm does so with the help of a for loop that creates a new dynamic array, copying A to it, appending a neighbor of the vertex at the front of A and

recursively calling TwoChromaticCycleRecurse. The algorithm checks if the vertex at the front of A is v, and if so, checks if the cycle in the array is 2-chromatic.

Algorithm 2 TwoChromaticCycle

Input: A graph H, a k-acyclic coloring h of H, and a vertex v of H.
Output: Outputs true if there exists a 2-chromatic cycle containing v and false otherwise.

```
1: procedure TwoChromaticCycle(H, h, v)
2:     Create a dynamic array A and append v.
3:     return TwoChromaticCycleRecurse(H, h, v, A)
4: end procedure
5: procedure TwoChromaticCycleRecurse(H, h, v, A)
6:     if size(A) = Γ(H) + 1 then
7:         ▷ We assume that Γ(H) is known
8:         return false
9:     end if
10:    for each vertex w adjacent to A.front() in G do
11:        if w = v and the cycle formed by the vertices in array A
12: is 2-chromatic in h then
13:            return true
14:        end if
15:        Create a dynamic array A' and copy A to it.
16:        A'.append(w)
17:        if TwoChromaticCycleRecurse(H, h, v, A')=true then
18:            return true
19:        end if
20:    end for
21:    return false
22: end procedure
```

Lemma 4. *Algorithm 2 has running time $O(\Delta(H)^{\Gamma(H)})$ and either returns true if there exists a 2-chromatic cycle that contains vertex v for k-vertex coloring h of H or returns false otherwise.*

Proof. By Lemma 3, to show that no cycle containing v is 2-chromatic it is enough to check that none of the induced cycles that contain v are 2-chromatic. We also know that the length of the longest induced cycle is $\Gamma(H)$. The for loop in line 10 considers

each neighbor w of the vertex at the front of the array A and recursively calls procedure TwoChromaticCycleRecurse with a copy of A appended with w. From line 12, the depth of the recursion of TwoChromaticCycleRecurse is $\Gamma(H)$ (we disregard calls to TwoChromaticCycleRecurse at depth $\Gamma(H)+1$). It follows that the procedure considers all possible paths of length at most $\Gamma(H)$ from v. In line 12, the algorithm checks if the front of array A is v, and if so, checks if the cycle in A is 2-chromatic. By Lemma 3, this is sufficient to check for the existence of a 2-chromatic cycle containing v. Since the number of cycles of length at most $\Gamma(H)$ that contain v is in $O(\Delta(H)^{\Gamma(H)})$, the running time of the algorithm is also in $O(\Delta(H)^{\Gamma(H)})$. □

Lemma 5. *Algorithm 1 returns YES if and only if there exists a reconfiguration sequence between h_s and h_e in graph H.*

Proof. All possible adjacent k-vertex colorings are enumerated in line 14 of Algorithm 1. Line 15 checks if the k-vertex coloring is acyclic by checking the neighbors of the recolored vertex, say v, to see if they share the same color and then calls Algorithm 2 to check for the existence of 2-chromatic cycles containing v. Since the color of only one vertex is changed, it is enough to check only the cycles containing v. Since Algorithm 2 is proved to be correct in Lemma 4, it is clear that an instance of k-ACR REACH is a YES instance if and only if Algorithm 1 returns YES. □

Lemma 6. *Algorithm 1 has running time $O((kn)^{\ell+\Gamma(G)})$.*

Proof. Algorithm 2 has running time in $O(\Delta(G)^{\Gamma(G)})$ (Lemma 4), and since $\Delta(G) \leq n$ the running time of Algorithm 2 is also in $n^{\Gamma(G)}$. Assuming that the graph is stored as an adjacency matrix, checking for the neighborhood of a vertex in line 15 takes $O(n)$ time. The for loop in line 14 iterates through kn possibilities. The depth of the recursion in line 17 is at most ℓ due to line 9. Therefore, the total time complexity of Algorithm 1 is $O((kn)^{\ell}n^{\Gamma(G)})$, which can be rewritten as $O((kn)^{\ell+\Gamma(G)})$. □

Theorem 5 then follows from Algorithm 1, Lemma 5, and Lemma 6.

Theorem 5. k-ACR REACH *is in XP when parameterized by $\ell+\Gamma(G)$ where ℓ is the length of its reconfiguration sequence and $\Gamma(G)$ the length of the longest induced cycle in G.*

Chapter 5

Equitable Coloring Reconfiguration

In this chapter, we consider reconfiguration problems in equitable colorings, specifically k-ECR REACH. We first investigate some general properties of k-ECR REACH in Section 5.1. Next, in Section 5.2, we prove that k-ECR REACH is PSPACE-hard. We then prove that k-ECR REACH can be solved in polynomial time when $k = 2$, in Section 5.3. Furthermore, in Section 5.4, we show that for paths, when $k = 3$ and the number of vertices is greater than 15, if two k-equitable colorings are viable to each other then there is a reconfiguration sequence between them.

5.1 General properties of k-ECR REACH

In this section, we investigate some general properties of k-ECR REACH. We prove some results that follow from the definitions provided in Section 2.3.2, and also some preliminary results.

The next observation follows from the definition of swapping.

Observation 7. *For a graph G and a k-equitable coloring f, the k-vertex coloring we obtain by swapping the colors of any two vertices is a k-equitable coloring if it is a k-coloring.*

As a consequence of Observation 7, when swapping the colors of two vertices we only need to consider whether the obtained k-vertex coloring is proper.

Before stating the following lemma, we remind the reader that the set of candidate vertices, given two k-equitable colorings, is defined in Definition 14.

Lemma 7. *Given a graph G and two k-equitable colorings f and h which are viable to each other, for every vertex v in $V(G)$ where $f(v) \neq h(v)$, the set $candidate(f, h, v)$ is non-empty.*

Proof. We prove the lemma by way of contradiction. If $candidate(f, h, v)$ is empty, then each vertex in $colorclass(h(v), f)$ has color $h(v)$ in h. In other words, $colorclass(h(v), f) \subseteq colorclass(h(v), h)$. But, we know that $v \in colorclass(h(v), h)$ and $v \notin colorclass(h(v), f)$, implying that $|colorclass(h(v), f)| < |colorclass(h(v), h)|$ (since $colorclass(h(v), f) \cup \{v\} \subseteq colorclass(h(v), h)$). This contradicts f being viable to h. □

Lemma 8. *Given a graph G and a k-equitable coloring f, for any color c used in f, $\lfloor \frac{|V(G)|}{k} \rfloor \leq |colorclass(c, f)| \leq \lceil \frac{|V(G)|}{k} \rceil$.*

Proof. We first show that for any color c, $|colorclass(c, f)| \geq \lfloor \frac{n}{k} \rfloor$, where $n = |V(G)|$. Towards contradiction, for a color c_1 used in f, suppose $|colorclass(c_1, f)| \leq \lfloor \frac{n}{k} \rfloor - 1$. Then for any other color c_i, $i \neq 1$, used in f, $|colorclass(c_i, f)| \leq \lfloor \frac{n}{k} \rfloor$, as otherwise f would not be equitable. But, $\sum_c |colorclass(c, f)| \leq (\lfloor \frac{n}{k} \rfloor - 1) + (k-1)\lfloor \frac{n}{k} \rfloor = k\lfloor \frac{n}{k} \rfloor - 1 < n$.

We next show that for any color c, $|colorclass(c, f)| \leq \lceil \frac{n}{k} \rceil$. Similar to the previous paragraph, towards contradiction, for a color c_2 used in f, suppose $|colorclass(c_2, f)| \geq \lceil \frac{n}{k} \rceil + 1$. Then for any other color c_i, $i \neq 2$, used in f, $|colorclass(c_i, f)| \geq \lceil \frac{n}{k} \rceil$. This contradicts f being equitable as $\sum_c |colorclass(c, f)| \geq (\lceil \frac{n}{k} \rceil + 1) + (k-1)\lceil \frac{n}{k} \rceil \geq k\lceil \frac{n}{k} \rceil + 1 > n$. □

Lemma 9. *Given a graph G and a k-equitable coloring f, if $k \geq 2\Delta(G) + 2$ then for any vertex v, $|valid(v, f)| \geq (k - 2\Delta(G) - 1)m$ where $m = \min_{1 \leq i \leq k} |colorclass(i, f)|$.*

Proof. We prove the lemma by showing the existence of a subset of $valid(v, f)$ with size at least $(k - 2\Delta(G) - 1)m$. We denote as *good colors* the colors that are not in the set $colors(N_G(v)) \cup \{f(v)\}$. There are at least $k - \Delta(G) - 1$ good colors. Let F be the set of all the vertices with good colors, or equivalently the union of the color classes of all the good colors. We prove the lemma by showing a lower bound on the size of F and then an upper bound on the number of vertices in F that can not be valid for v.

We first prove a lower bound on the size of F. Since the size of a color class of f can either be m or $m + 1$, it follows that $|F| \geq (k - \Delta(G) - 1)m$.

We next prove an upper bound on the number of vertices in F that can not be valid for v. A vertex in F is not valid for v if and only if the neighborhood of that vertex contains a vertex with color $f(v)$ (by the definition of F, there can be no vertex in the

neighborhood of v with the same color as a vertex in F). In the worst case, each vertex in $colorclass(f(v), f) \setminus \{v\}$ is adjacent to only vertices in F and the neighborhood of each vertex in $colorclass(f(v), f) \setminus \{v\}$ is disjoint. So, for each vertex in $colorclass(f(v), f) \setminus \{v\}$ there are at most $\Delta(G)$ vertices in F that are not valid for v. Since v is not adjacent to any vertex in F, there are at most $(|colorclass(f(v), f)| - 1)\Delta(G)$ vertices in F that are not valid for v. Since $|colorclass(f(v), f)| \leq m + 1$, there are at least $(k - 2\Delta(G) - 1)m$ vertices u in F for which $swap(u, v)$ is valid. □

Theorem 6. *If $k \geq n$, an instance (H, f_s, f_e) of k-*ECR REACH *can be solved in polynomial time.*

Proof. We prove the theorem by showing that if f_s and f_e are viable to each other then it is a yes-instance and a no-instance otherwise. It is immediate that if f_s and f_e are not viable to each other then (H, f_s, f_e) is a no-instance. What is left to prove is that (H, f_s, f_e) is a YES instance if f_s and f_e are viable to each other.

We show constructively that (H, f_s, f_e) is a yes-instance if f_s and f_e are viable to each other. Note that all vertices in $V(H)$ have distinct colors since $k \geq n$. So, a k-vertex coloring that we obtain by swapping the color of any two vertices is always equitable. Consequently, any two vertices in H are valid for each other. For each vertex v such that $f_s(v) \neq f_e(v)$, we can swap the colors of v and a vertex in $candidate(f_s, f_e, v)$ (which is guaranteed to be non-empty by Lemma 7). By repeating this for each vertex in H in the current k-equitable coloring whose color is different from its color in f_e, we can obtain a reconfiguration sequence between f_s and f_e.

Checking if two k-equitable colorings are viable to each other can be done by a linear scan of all the vertices in a graph, so our theorem is proved. □

5.2 k-ECR REACH is PSPACE-hard

In this section, we prove that k-ECR REACH is PSPACE-hard by reducing an instance (G, I_s, I_e) of TJ-IS REACH, which was defined in Section 2.4, to an instance (H, f_s, f_e) of k-ECR REACH[1]. In our reduction, we require k in k-ECR REACH to be $k = |V(G)| - |I_s| + 1$. It follows from the description of our reduction that a k-coloring of H

[1]Our reduction was developed in collaboration with Tatsuhiko Hatanaka and Haruka Mizuta, who are both affiliated with the Graduate School of Information Sciences, Tohoku University, Aoba-yama 6-6-05, Sendai 980-8579, Japan, at the Combinatorial Reconfiguration Workshop in Banff International Research Station, Alberta.

exists.

We copy to a graph H the vertices and edges in G and add $(k-1)\,|I_s|$ isolated vertices that we call *buffer vertices* and, additionally, a *frozen structure* that is a complete graph on $2n$ vertices. We call the vertices of the frozen structure *frozen vertices*. We call the copies of vertices of G that are in H *original vertices* and for each vertex v in G, we denote as v_H the corresponding original vertex in H. Similarly, for each original vertex u in $V(H)$, we denote as u_G its corresponding vertex in G. Note that, in addition to the edges we copy from G to H, we have added $k(k-1)$ edges to the frozen structure.

We first remove edges from the frozen structure and then add edges between the frozen and buffer vertices so that in any k-equitable coloring of H, valid swaps are possible only between original vertices. Let $M = \{u_1v_1, u_2v_2, \ldots, u_nv_n\}$ be a perfect matching of the frozen structure. We remove the edges corresponding to M from the frozen structure. Note that we have added $2k(k-1)$ edges. We can see that in any k-coloring of H, the frozen structure is colored by k colors and there is no valid swap between a frozen vertex and another vertex.

We group the $(k-1)\,|I_s|$ buffer vertices in $k-1$ groups of size $|I_s|$: $B_1, B_2, \ldots, B_{k-1}$. We draw edges from each vertex in a B_i, $1 \leq i \leq k-1$, to all the vertices in the set $V(M)\setminus\{u_i, v_i\}$. Since there are $(k-1)\,|I_s|$ buffer vertices, we have added $(k-1)\,|I_s|\,2(k-1)$ edges between frozen and buffer vertices. Since the frozen structure is colored by k colors by any k-coloring and there is no valid swap between a frozen vertex and another vertex, there is also no valid between a buffer vertex and another vertex.

As a consequence of our reduction, the following observation follows.

Observation 8. *For any k-equitable coloring of H, valid swaps are only possible between original vertices.*

The following observation follows as no original vertex is adjacent to a buffer vertex or a frozen vertex.

Observation 9. *For any original vertex u of H, each vertex in $N_H(u)$ is an original vertex.*

Claim 8. *Our reduction of an instance of* TJ-IS REACH *to an instance of* k-ECR REACH *can be executed in $O(n^3)$ time and space, where $n = |V(G)|$.*

Proof. We prove the observation by noting that we add $(k-1)|I_s|+2k$ vertices and $2k(k-1) + (k-1)|I_s|2(k-1)$ edges. Since I_s is a subset of $V(G)$, $|I_s| \leq n$; we also know that $k = |V(G)|-|I_s|+1$. It follows from this that the number of vertices and edges added is in $O(n^3)$, where $n = |V(G)|$. □

Below, we make two definitions that rely on the frozen vertices having specific colors. It follows from the frozen vertices having specific colors that a buffer vertex also has a specific color as it is adjacent to every frozen vertex apart from some u_i and v_i of M ($1 \leq i \leq k$). Since the frozen vertices can be assigned k colors and the buffer vertices can be assigned $k-1$ colors, the existence of a k-coloring of H where frozen vertices have specific colors follows from the existence of a k-vertex coloring of H that, when restricted to original vertices, is proper. That there exists a proper k-vertex coloring restricted to original vertices follows as, since $k = |V(G)|-|I_s|+1$, we can assign to the original vertices corresponding to the vertices of I_s color 1, and assign to the rest of the $|V(G)|-|I_s|$ original vertices distinct colors which are not 1.

Definition 22. We say that a k-equitable coloring f of H is *precolored* if the frozen vertices u_i and v_i, $1 \leq i \leq k$, have color i.

Definition 23. For an independent set I in G of size $|I_s|$, we define a *corresponding k-equitable coloring of I* as any k-equitable coloring f such that: f is precolored; for each vertex u in I, u_H has color 1 in f; and the remaining $|V(G)|-|I|$ original vertices have distinct colors from the set $\{2, \ldots, k\}$.

The following observations follow from our reduction and the definition of corresponding k-equitable colorings.

Observation 10. *Given an independent set I in G of size $|I_s|$ and a corresponding k-equitable coloring f of I, each original vertex u in H has color 1 in f if and only if $u_G \in I$.*

Observation 11. *Given an independent set I of G of size $|I_s|$ and a corresponding k-equitable coloring f of I, each original vertex u such that $u_G \notin I$ has a distinct color that is not 1.*

Observation 12. *For every independent set I in G of size $|I_s|$, there exists a corresponding k-equitable coloring.*

Proof. We prove this by constructing such a corresponding k-equitable coloring f of I. Let f be precolored. For each vertex u in I, we let u_H have color 1 in f. Since I is an independent set, no two adjacent vertices have the same color. We color the remaining original vertices with distinct colors from the set $\{2, \ldots, k\}$.

What is left to prove is that f is equitable. The frozen structure has two vertices with color i in f for $1 \leq i \leq k$, the buffer and original vertices have $|I_s|+1$ vertices with color j in f for $2 \leq j \leq k$, and the original vertices have $|I_s|$ vertices with color 1 in f. It follows from this that $|colorclass(1, f)| = |I_s|+2$ and $|colorclass(j, f)| = |I_s|+3$ for $2 \leq j \leq k$. $\square$

Claim 9. *Given an independent set I in G of size $|I_s|$ and a corresponding k-equitable coloring f of I, any swap between two original vertices u and v is valid if $u_G, v_G \notin I$.*

Proof. We prove the claim by noting that each vertex in the set of original vertices not in I has a distinct color in f that is not 1 (Observation 11). Also, the neighborhood of any original vertex contains only original vertices (Observation 9). It follows from this that u is the only vertex in $N_H(v)$ with color $f(u)$ in f and v is the only vertex in $N_H(u)$ with color $f(v)$ in f. Clearly $swap(u, v)$ is valid in f. □

Claim 10. *Given an independent set I in G of size $|I_s|$ and any corresponding k-equitable coloring f of I, if there is a valid swap between two original vertices u and v in f where $u_G \in I, v_G \notin I$, then $(I \setminus \{u_G\}) \cup \{v_G\}$ is an independent set of G.*

Proof. We prove the claim by showing that $(N_G(v_G) \setminus \{u_G\}) \cap I = \emptyset$. Since $swap(u, v)$ is valid in f, there is no vertex in $N_H(v) \setminus \{u\}$ with color $f(u)$ in f. From Observation 9, each vertex in $N_H(v)$ is an original vertex. We can also say from Observation 10 that $f(u) = 1$ and for each vertex w in $N_H(v) \setminus \{u\}$, $w_G \notin I$ (as $f(w) \neq 1$). It follows that $(N_G(v_G) \setminus \{u_G\}) \cap I = \emptyset$. □

Claim 11. *For a graph G and an independent set I, the nodes of any two corresponding k-equitable colorings of I are connected in $R_{k-ECR}(H)$ by a path of length at most $|V(G)| - |I_s|$.*

Proof. We prove the claim for any two corresponding k-equitable colorings f and h of I. We first show that f and h are viable to each other by noting that, for O being the induced subgraph on the original vertices, since f and h are both precolored, $|colorclass(i, f|_O)| = |colorclass(i, h|_O)|$ for $1 \leq i \leq k$.

Next, we show that only original vertices v such that $v_G \notin I$ have different colors in f and h. As both f and h are precolored, only original vertices can have different colors in f and h. Furthermore, since for each vertex w in I, $f(w_H) = h(w_H) = 1$ (from Observation 10), only original vertices v such that $v_G \notin I$ have different colors in f and h.

Now, we show that the color of each original vertex u such that $u_G \notin I$ and $f(u) \neq h(u)$ can be swapped with the color of a vertex x in $candidate(f, h, u)$. Such an x in $candidate(f, h, u)$ exists as f and h are viable to each other (Lemma 7). Since x is a candidate vertex, $f(x) \neq h(x)$ and x is an original vertex such that $x_G \notin I$. From Claim 9, as $x_G, v_G \notin I$ and x, v are original vertices, $swap(u, x)$ is valid in f. The k-equitable coloring that we obtain from $swap(u, x)$ is also a corresponding k-equitable coloring of I, so we can repeat this for all vertices in H that have different colors in f and h. Since there

are at most $|V(G)|-|I_s|$ vertices that have different colors in f and h, there is a path of length at most $|V(G)|-|I_s|$ between the nodes of f and h in $R_{k-ECR}(H)$. □

Claim 12. *For every two independent sets I_a and I_b whose nodes are connected by a path of length ℓ in $R_{IS-TJ}(G)$, the nodes of any two corresponding k-equitable colorings of I_a and I_b, respectively, in $R_{k-ECR}(H)$ are also connected by a path of length at most $\ell + (|V(H)|-|I_a|)$.*

Proof. We prove the claim by showing that for every two adjacent independent sets I_x and I_y in G, each corresponding k-equitable coloring of I_x is adjacent to some corresponding k-equitable coloring of I_y. Consequently, the node of each corresponding k-equitable coloring of I_a is connected to the node of some corresponding k-equitable coloring of I_b by a path of length at most ℓ in $R_{k-ECR}(H)$. Since, from Claim 11, the node of each corresponding k-equitable coloring of I_b is connected to the node of any other corresponding k-equitable coloring of I_b by a path of length at most $|V(H)|-|I_a|$, our claim follows.

We show that the statement that for every two adjacent independent sets I_x and I_y each corresponding k-equitable coloring of I_x is adjacent to some corresponding k-equitable coloring of I_y is equivalent to the statement that any token jump in G can simulated by a swap in H. Before stating our second statement formally, let f be any corresponding k-equitable coloring of I_x, $\{x, y\} = I_x \triangle I_y$ ($\{x, y\}$ is the symmetric difference of I_x and I_y), and assume without loss of generality that $x \in I_x$ and $y \in I_y$. Assuming that there is a token on each vertex of I_x in G, a token jump is moving a token from x to y to form I_y from I_x, and an equivalent swap is $swap(x_H, y_H)$ in f. Formally, our second statement states that for the adjacent independent sets I_x and I_y, $swap(x_H, y_H)$ in f is valid. If $swap(x_H, y_H)$ in f is valid, it can be seen that the k-equitable coloring that we obtain is a corresponding k-equitable coloring of I_y. To see why, notice that for each original vertex w such that $w_G \in (I_x \setminus \{x\}) \cup \{y\} = I_y$ has color 1 in the k-equitable coloring that is formed. It follows from this that our two statements are equivalent.

We now show that $swap(x_H, y_H)$ is valid in f. It is sufficient to show that there is no vertex in $N_H(x_H) \setminus \{y_H\}$ with color $f(y_H)$ in f and that there is no vertex in $N_H(y_H) \setminus \{x_H\}$ with color $f(x_H) = 1$ in f. We first show that there is no vertex in $N_H(y_H) \setminus \{x_H\}$ with color 1 in f. Since I_x and I_y are adjacent, y is not adjacent to any vertex in $I_x \setminus \{x\}$, and as only original vertices u such that $u_G \in I_x$ have color 1 in f (Observation 10), there is no vertex in $N_H(y_H) \setminus \{x_H\}$ that has color 1 in f. From Observation 11, each original vertex v such that $v_G \notin I_x$ has a color that is not 1 in f. Since for each vertex z in $N_H(x_H)$ $z_G \notin I_x$, there is no vertex in $N_H(x_H)$ that has color 1 in f. Consequently, $swap(x_H, y_H)$ in f is valid. □

Claim 13. *For every two k-equitable colorings f_x and f_y whose nodes in $R_{k-ECR}(H)$ are connected by a path of length ℓ, if f_x and f_y are the corresponding k-equitable colorings of independent sets I_x and I_y of G, respectively, then the nodes of I_x and I_y are connected by a path of length at most ℓ in $R_{IS-TJ}(G)$.*

Proof. We prove the claim by showing that each swap in the reconfiguration sequence between f_x and f_y can be simulated by a token jump in G. For a swap between an original vertex with color 1 and another original vertex, by a token jump, we specifically mean moving a token from the corresponding vertex in G of the original vertex with color 1 to the corresponding vertex in G of the other original vertex. Before formally making our statement, we let the reconfiguration sequence between f_x and f_y be $\gamma_1, \ldots, \gamma_{\ell-1}$ and also assume that $\gamma_0 = f_x$ and $\gamma_l = f_y$.

We now show by induction on the length of the reconfiguration sequence that for each i, $0 \leq i \leq \ell - 1$, γ_i and γ_{i+1} are corresponding k-equitable colorings of some two independent sets I_a and I_b, respectively, and either $I_a = I_b$ or I_a and I_b are adjacent. In the base case, when the length of the reconfiguration sequence is zero, $f_x = f_y$ and both of them are corresponding k-equitable colorings of $I_x (= I_y)$. In the inductive step, we can assume that the statement holds for each j, $0 \leq j \leq p$, and prove that it also holds for $p + 1$ $(1 \leq p \leq \ell - 1)$. Let the swap between γ_p and γ_{p+1} be between vertices u and v. We can also assume that γ_p is a corresponding k-equitable coloring of an independent set I_p. Since valid swaps are only possible between original vertices (Observation 8), u and v are original vertices. Consequently, the cases to consider are the following.

1. $u_G, v_G \notin I_p$.
2. $u_G \in I_p$, $v_G \notin I_p$ or $u_G \notin I_p$, $v_G \in I_p$.
3. $u_G, v_G \in I_p$.

In Case (1), since each vertex x such that $x_G \in I_p$ has color 1 in γ_{p+1}, γ_{p+1} is also a corresponding k-equitable coloring of I_p. So effectively, there is no token jump.

In Case (2), we can assume without loss of generality that $u_G \in I_p$, $v_G \notin I_p$. Next, from Claim 10, since $swap(u, v)$ is valid in γ_p and γ_p is a corresponding k-equitable coloring of I_p, the set $(I_p \setminus \{u_G\}) \cup \{v_G\}$ is an independent set. Since each vertex y such that $y_G \in (I_p \setminus \{u_G\}) \cup \{v_G\}$ has color 1 in γ_{p+1}, γ_{p+1} is a corresponding k-equitable coloring of $(I_p \setminus \{u_G\}) \cup \{v_G\}$. And $(I_p \setminus \{u_G\}) \cup \{v_G\}$ is adjacent to I_p. So the token jump is from u_G to v_G.

In Case (3), since both u and v have color 1 in γ_p, it is not a valid swap, a contradiction. □

We are now ready to prove Theorem 7.

Theorem 7. *k-ECR REACH is PSPACE-hard.*

Proof. We prove that k-ECR REACH is PSPACE-hard by reducing an instance (G, I_s, I_e) of TJ-IS REACH to an instance (H, f_s, f_e) of k-ECR REACH, and showing that an instance (G, I_s, I_e) is a yes-instance if and only if (H, f_s, f_e) is a yes-instance. The reduction we have described can be executed using polynomial time and space from Claim 8. From Observation 12, there exist two k-equitable colorings f_s and f_e which are corresponding k-equitable colorings of I_s and I_e, respectively. So each instance (G, I_s, I_e) of TJ-IS REACH has a corresponding instance (H, f_s, f_e) in k-ECR REACH. If (G, I_s, I_e) is a yes-instance, then the nodes of I_s and I_e are connected in $R_{IS-TJ}(G)$ and from Claim 12, the nodes of f_s and f_e are also connected in $R_{k-ECR}(H)$. If (H, f_s, f_e) is a yes-instance, then the nodes of f_s and f_e are connected in $R_{k-ECR}(H)$ and from Claim 13, the nodes of I_s and I_e are also connected. □

The following lemma follows since TJ-IS REACH is W[1]-hard when parameterized by the number of tokens [30], and for an instance (G, I_s, I_s) of TJ-IS REACH and a corresponding instance (H, f_s, f_e) of k-ECR REACH from our reduction, $|I_s| = colorclass(1, f_s) - 1$.

Lemma 10. *k-ECR REACH is W[1]-hard when parameterized by the size of a color class.*

5.3 2-equitable colorings

We prove that k-ECR REACH is solvable in polynomial time when $k = 2$. Note that if a 2-coloring exists for a graph, then the graph is bipartite. We also assume without loss of generality that the colors used in any 2-equitable coloring are 1 and 2.

Lemma 11. *Given a graph G and 2-equitable colorings f and h, if a vertex v in $V(G)$ has $f(v) \neq h(v)$ and $d(v) \geq 2$, then the nodes of f and h are not connected in $R_{2-ECR}(G)$.*

Proof. We consider two vertices adjacent to v, say w and x, and assume without loss of

generality that $f(v) = 1$. Since $f(w) \neq f(v)$ and $f(x) \neq f(v)$, $f(w) = f(x) = 2$. For any swap to change the color of v, the colors of w and x must also be changed to 1. But, to change the colors of w or x, the color of v must be changed to 2. Therefore, to change the color of at least one of the vertices u, v, or w, the colors of all three of them must be changed. Since a swap can change the colors of only two vertices, there is no sequence of swaps such that the colors of at least one of the vertices u, v, or w is changed. □

Observation 13. *Given a graph G and 2-equitable colorings f and h, a vertex v such that $f(v) \neq h(v)$, and any vertex u in candidate(f, h, v), then v is also in candidate(f, h, u).*

Proof. We can assume without loss of generality that $f(v) = 1$, and so $h(v) = 2$. Since u is in *candidate*(f, h, v), $f(u) = 2$ and $f(u) \neq h(u)$. It follows that v is in *candidate*(f, h, u). □

Notice that Observation 13 holds only for 2-equitable colorings. If more than two colors are used, then given k-equitable colorings f and h and a vertex v such that $f(v) \neq h(v)$, for a vertex u in *candidate*(f, h, v), it can be that $h(u) \neq f(v)$.

We describe, in Algorithm 3, an algorithm that, given as input a graph G and 2-equitable colorings f_s and f_e, returns a reconfiguration sequence if (G, f_s, f_e) is a yes-instance and an empty set otherwise. Before describing our algorithm we give a high-level overview.

The three possible cases that Algorithm 3 handles are: f_s and f_e are not viable to each other; there exists a vertex with degree greater than or equal to 2 having different colors in f_s and f_e; and (G, f_s, f_e) is a yes-instance. In the first case, it is clear that (G, f_s, f_e) is a no-instance. In the second case, from Lemma 11, (G, f_s, f_e) is a no-instance.

In the third case, Algorithm 3 groups the vertices with different colors in f_s and f_e in set S, changes the color of vertices in S with degree 1, and then changes the color of vertices with degree 0. Since there is no vertex in S with degree greater than or equal to 2, this suffices.

In the while loop in line 12, Algorithm 3 iteratively selects vertices u in S with degree 1. Since $f_i(u) \neq f_e(u)$, where f_i is the 2-equitable coloring under consideration in iteration i, and both f_i and f_e are 2-equitable colorings, u's neighbor, say w, is also in S and has degree exactly 1. It follows from this that *swap*(u, w) is valid in f_i. Since w has color $f_e(u)$, it is in *candidate*(f_i, f_e, u), and so u is in *candidate*(f_i, f_e, u) from Observation 13. Consequently, after *swap*(u, w), both u and w are colored with their target colors ($f_e(u)$ and $f_e(w)$, respectively).

Next, in the while loop in line 18, Algorithm 3 selects vertices x with degree 0. We can then select a candidate vertex y for x. Since y is also in S, y has degree 0 and so $swap(x, y)$ is valid in f_i. As x is in $candidate(f_i, f_e, y)$, y is also in $candidate(f_i, f_e, x)$ from Observation 13. So after $swap(x, y)$, both x and y are colored with their target colors ($f_e(x)$ and $f_e(y)$, respectively).

Algorithm 3 TwoEquitableRecolor

```
1: procedure TwoEquitableRecolor(G, f_s, f_e)
2:     if IsViable(f_s, f_e) = false then
3:         ▷ IsViable returns true if f_s and f_e are viable to each other and false otherwise
4:         return ()
5:     end if
6:     S ← {v | f_s(v) ≠ f_e(v)}
7:     if there exists a v in S such that d(v) ≥ 2 then
8:         return ()
9:     end if
10:    f_0 ← f_s
11:    i ← 0
12:    while there exists a u in S with d(u) = 1 do
13:        w ← the single vertex in N_G(u)
14:        f_{i+1} ← SwapVertices(f_i, u, w)
15:        S ← S \ {u, w}
16:        i ← i + 1
17:    end while
18:    while S ≠ ∅ do
19:        x ← some vertex in S
20:        y ← CandidateVertex(f_i, f_e, x)
21:        ▷ CandidateVertex returns a vertex in the set candidate(f_i, f_e, x)
22:        f_{i+1} ← SwapVertices(f_i, x, y)
23:        S ← S \ {x, y}
24:        i ← i + 1
25:    end while
       return (f_1, f_2, ..., f_{i-1})
26: end procedure
```

Lemma 12. *Algorithm 3 is correct and runs in $O(n^2)$ time, where $n = |V(G)|$.*

Proof. We first show that Algorithm 3 is correct. The three cases that Algorithm 3 handles

are: f_s and f_e are not viable to each other; there is a vertex with degree greater than 2 that has different colors in f_s and f_e; and all other cases. Algorithm 3 handles the first case in line 4, where it is immediate that if f_s and f_e are not viable to each other, then (G, f_s, f_e) is a no-instance. Algorithm 3 handles the second case in line 8, where it follows from Lemma 11 that if a vertex with degree greater than 2 has different colors in f_s and f_e, then (G, f_s, f_e) is a no-instance. In line 6, the set S contains all the vertices that have different colors in f_s and f_e. In the third case, Algorithm 3 iteratively changes the color of vertices in S to their respective color in f_e, in two steps.

The two steps in which Algorithm 3 changes the color of vertices in S are to first change the color of vertices in S that have degree 1, and then to change the color of vertices in S that have degree 0. This suffices as we know from the second case of our algorithm that there is no vertex in S with degree greater than 2. Whenever the color of a vertex in S is changed, the obtained k-equitable coloring is stored in f_{i+1} and vertices are removed from S so that it stores the vertices that have different colors in f_{i+1} and f_e.

The while loop in line 12 changes the color of vertices u in S with degree 1. Since f_i is proper, for the vertex w adjacent to u, $f_i(w) \neq f_i(u)$. As f_i is a 2-equitable coloring, $f_i(w) = f_e(u)$, and as f_e is proper, $f_e(w) \neq f_i(w)$. It follows from this that w is in $candidate(f_i, f_e, u)$ and $d(w) = 1$. Since $N_G(w) \setminus \{u\} = \emptyset$ and $N_G(u) \setminus \{w\} = \emptyset$, in line 14, $swap(u, w)$ is valid in f_i. We can also say that u is in $candidate(f_i, f_e, w)$ from Observation 13, as w is in $candidate(f_i, f_e, u)$. So after $swap(u, w)$, u and w have their target colors ($f_e(u)$ and $f_e(w)$, respectively), and after line 15 the set S only has vertices whose colors are different in f_{i+1} and f_e.

The while loop in line 18 changes the color of vertices x in S with degree 0. The set $candidate(f_i, f_e, x)$ is guaranteed to be non-empty by Lemma 7. In line 21, some vertex y in $candidate(f_i, f_e, x)$ is returned by the procedure CandidateVertex. Since y is a candidate vertex, $f_i(y) \neq f_e(y)$, and so $y \in S$. As $y \in S$, $d(y) = 0$. Clearly $swap(x, y)$ is valid. And from Observation 13, as y is in $candidate(f_i, f_e, x)$, x is in $candidate(f_i, f_e, y)$. So after line 23 the set S only has vertices whose colors are different in f_{i+1} and f_e.

We now show that Algorithm 3 runs in $O(n^2)$ time, where $n = |V(G)|$. A call to IsViable can be executed in $O(n)$ time as two k-equitable colorings can be checked if they are viable to each other by a linear scan of all the vertices. A call to CandidateVertex can also be executed in $O(n)$ time as a candidate vertex can be found by a linear scan of all the vertices. Since the number of iterations of the while loops in Algorithm 3 is linear in terms of the size of S, which is less than or equal to n, our algorithm runs in $O(n^2)$ time. □

The following corollary is immediate from the correctness of Algorithm 3.

Corollary 3. *We can characterize yes-instances (G, f_s, f_e) of 2-ECR by the following conditions.*

1. *f_s and f_e are viable to each other.*
2. *There is no vertex in G with degree greater than two that has different colors in f_s and f_e.*

5.4 Paths

In this section, we prove that for paths when the number of colors is 3, k-ECR REACH can be solved in polynomial time. We prove this by showing that, for $n > 15$, any 3-equitable coloring can be reconfigured to its canonical form, in four steps. For this, we define three properties: dense ones, alternating ones, and matching ones. We first show, in Corollary 4, that any 3-equitable coloring can be reconfigured to a 3-equitable coloring that satisfies the dense ones property. Next, in Corollary 5, we show that a 3-equitable coloring that satisfies the dense ones property can be reconfigured to a 3-equitable coloring that satisfies the alternating ones property. Third, in Corollary 6, we show that any 3-equitable coloring that satisfies the alternating ones property can be reconfigured to a 3-equitable coloring that satisfies the matching ones property. Finally, in Corollary 7, we show that a 3-equitable coloring that satisfies the matching ones property can be reconfigured to its canonical 3-equitable coloring.

We make the following note before going further into our proof.

Note 2. Since the reconfiguration step in k-ECR REACH is to swap the colors of two vertices, the length of a reconfiguration sequence between two 3-equitable colorings is the same as the number of swaps between the two 3-equitable colorings.

We now define the dense ones property and describe an algorithm, Algorithm 4, to reconfigure any 3-equitable coloring of a path to a 3-equitable coloring that satisfies the dense ones property.

Definition 24. We say that a 3-equitable coloring of a path $P = v_1, v_2, \ldots, v_n$ satisfies the *dense ones* property if for any two vertices v_i, v_j with color 1, $1 \leq i < j \leq n$, and $j - i \geq 4$, there exists a vertex v_t, $i < t < j$, with color 1, and the rightmost 1 is either v_n or v_{n-1}.

Algorithm 4, at each iteration, selects the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$ and "shifts" its color to a vertex v_t, which is at a distance of at most 1 from $v_{n_{sub}}$, such that the 3-equitable coloring restricted to the subpath of v_t and all vertices right of v_t satisfies the dense ones property. We are also assured that at each iteration, $v_{n_{sub}}$'s successor (if it exists) does not have color 1. Then, Algorithm 4 relies on the fact that if the rightmost 1 is neither $v_{n_{sub}}$ nor $v_{n_{sub}-1}$, then the color of the rightmost 1 can be swapped with the color of either the successor of it or the vertex that is at distance 2 from it and right of it. To this end, an auxiliary algorithm, Algorithm 5, repeatedly makes one of these swaps until the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$ is either $v_{n_{sub}}$ or $v_{n_{sub}-1}$. The value of n_{sub} is then set to the index of the vertex left of and at distance 2 from the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$. Even after the value of n_{sub} is changed, there is still no vertex successor of $v_{n_{sub}}$ with color 1. Initially, $n_{sub} = n$, and Algorithm 5 is called until there is no vertex with color 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$. Since the value of n_{sub} decreases after each call to Algorithm 5, in some iteration, there will be no vertex with color 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$.

An example of an input and output 3-equitable coloring of Algorithm 4 is given in Figure 5.1.

Algorithm 4 ShiftOnes

Input: A path $P = v_1, v_2, \ldots, v_n$ and a 3-equitable coloring f of the path that uses colors 1, 2, and 3.

Output: A 3-equitable coloring that satisfies the dense ones property.

1: **procedure** ShiftOnes($P = v_1, v_2, \ldots, v_n$, f)
2: $n_{sub} \leftarrow n$
3: **while** ExistsOne(P, n_{sub}, f) **do**
4: ▷ ExistsOne checks if there exists a vertex in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$ with color 1 in f
5: $r \leftarrow$ RightmostOne(P,n_{sub}, f)
6: ▷ RightmostOne returns the index of the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$
7: $(f, n'_{sub}) \leftarrow$ ShiftOnesAux(P, f, n_{sub}, r)
8: $n_{sub} \leftarrow n'_{sub}$
9: **end while**
10: **return** f
11: **end procedure**

Algorithm 5 makes swaps so that the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$ is either

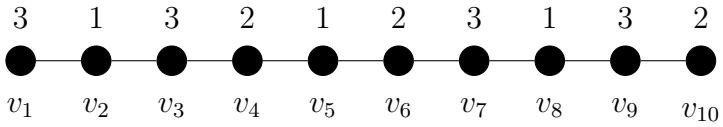
$v_{n_{sub}}$ or $v_{n_{sub-1}}$. Before an iteration of the while loop, v_t is the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$. In an iteration of the while loop, we consider three cases: $t < n_{sub}$; $t = 1$ or v_{t+1} and v_{t-1} have different colors; and all other cases. In the first two cases swaps are made between v_t and v_{t+1} or between v_t and v_{t+2}, respectively, and increment the value of t to $t + 1$ or $t + 2$, respectively. In the third case, we show that $t = n_{sub} - 1$ and so the while loop breaks. As the value of t is always increased or the while loop breaks, it follows that the while loop ends and that either $v_{n_{sub}}$ or $v_{n_{sub-1}}$ has color 1.

Algorithm 5 ShiftOnesAux

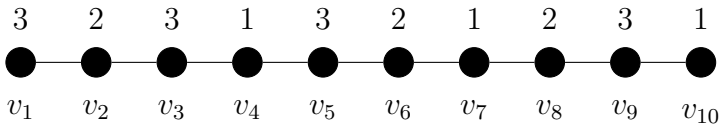
Input: A path $P = v_1, v_2, \ldots, v_n$; a 3-equitable coloring f of the path that uses colors 1, 2, and 3; the index n_{sub} of the vertex $v_{n_{sub}}$ such that we do not consider vertices right of it and the successor of $v_{n_{sub}}$ (if it exists) does not have color 1; and the index r of the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$.

Output: A 3-equitable coloring f such that the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$ is either $v_{n_{sub}}$ or $v_{n_{sub-1}}$ and the color of any vertex right of $v_{n_{sub}}$ is unchanged, and the index $t - 2$ of the vertex at distance 2 and left of the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$.

```
1: procedure ShiftOnesAux(P, f, n_sub, r)
2:     t ← r
3:     while t < n_sub do
4:         if t = 1 or f(v_{t-1}) ≠ f(v_{t+1}) then
5:             f ← SwapVertices(f, v_t, v_{t+1})
6:             t ← t + 1
7:         else if t ≤ n_sub − 2 then
8:             f ← SwapVertices(f, v_t, v_{t+2})
9:             t ← t + 2
10:        else
11:            break
12:        end if
13:    end while
14:    return (f, t − 2)
15: end procedure
```



(a) An input 3-equitable coloring to Algorithm 4.



(b) The output 3-equitable coloring, for the input in Figure 5.1a, to Algorithm 4.

Figure 5.1: The figure shows a sample input and output 3-equitable coloring of a path for Algorithm 4.

Lemma 13. *Algorithm 4 is correct and uses at most* $\lceil \frac{n^2}{3} \rceil$ *swaps, where* $n = |V(P)|$.

Proof. We first show the correctness of Algorithm 4 by assuming that Algorithm 5 is correct, and then proving the correctness of Algorithm 5. We first show that, at each iteration, the algorithm can call Algorithm 5. When $n_{sub} = n$, as there is no successor of $v_{n_{sub}}$ the algorithm can call Algorithm 5. In line 7, the value of n'_{sub} is updated to the value of the index of the vertex left of and at distance 2 from the rightmost 1 in $v_1, v_2, \ldots, v_{n_{sub}}$. So after line 7, the successor of $v_{n_{sub}}$ (if it exists) does not have color 1, and so it follows that at each iteration, the algorithm can call Algorithm 5. After the first iteration of the while loop in Algorithm 4, $f(v_n) = 1$ or $f(v_{n-1}) = 1$, so f restricted to the subpath of the rightmost 1 and all vertices right of it satisfies the dense ones property. In iteration i, after line 7, let v_{t_i} be the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$, and after line 8, the value of n_{sub} is $t_i - 2$. Since t_i is at most the value of n_{sub} before line 8, the value of n_{sub} is always strictly decreasing. In iteration $i + 1$, before line 7, the value of n_{sub} is $t_i - 2$, and after line 7, the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$, say $v_{t_{i+1}}$, is either $v_{n_{sub}}$ or $v_{n_{sub}-1}$. The distance between $v_{t_{i+1}}$ and v_{t_i} is at most 3, and so f restricted to $v_{t_{i+1}}$ and all the vertices right of it satisfies the dense ones property. It follows from this and the fact that the value of n_{sub} is always decreasing that Algorithm 4 outputs a 3-equitable coloring of the path that satisfies the dense ones property.

We now prove the correctness of Algorithm 5. We show that the swaps at each iteration of the while loop in Algorithm 5 are valid, and that after the end of the while loop the color of either $v_{n_{sub}}$ or $v_{n_{sub}-1}$ is 1. In the first iteration, before line 4, v_t has color 1, and since v_t is the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$ no vertex in $\{v_{t+1}, v_{t+2}, \ldots, v_{n_{sub}}\}$ has color 1.

The cases that we consider in an iteration of the while loop are: $t = 1$ or $f(v_{t-1}) \neq f(v_{t+1})$; $f(v_{t-1}) = f(v_{t+1})$, $t \leq n_{sub} - 2$; and all other cases. In the first case, as v_{t-1} and v_{t+1} have different colors $swap(v_t, v_{t+1})$ is valid. In the second case, $f(v_{t-1}) = f(v_{t+1})$, $t + 2 \leq n_{sub}$, and since f is proper $f(v_{t+1}) \neq f(v_{t+2})$; so $f(v_{t+2}) \neq f(v_{t-1})$. We also proved in the previous paragraph that $v_{n_{sub}}$'s successor (if it exists) does not have color 1. Consequently, $swap(v_t, v_{t+2})$ is valid. In the third case, we know that $t = n_{sub} - 1$, so the color of $v_{n_{sub}-1}$ is 1. In the first two cases, the value of t is modified in line 6 or 9 so that v_t is still the rightmost 1 in the subpath $v_1, v_2, \ldots, v_{n_{sub}}$, and it can also be seen that after the modification of t, $t \leq n_{sub}$ holds. At each iteration of the while loop, the value of t is either increased in the first two cases or the while loop exits in the third case. The correctness of Algorithm 5 follows.

We show the maximum number of swaps that Algorithm 4 may use by observing that each call to Algorithm 5 (ShiftOnesAux) makes at most $n_{sub} - r$ swaps. Notice that after line 7, either $v_{n_{sub}}$ or $v_{n_{sub}-1}$ has color 1, the color of no vertex right of $v_{n_{sub}}$ is changed in the call to Algorithm 5, and $n'_{sub} < n_{sub}$, so $\left|colorclass(1, f|_{v_{n_{sub}}, v_{n_{sub}+1}, \ldots, v_n})\right| < \left|colorclass(1, f|_{v_{n'_{sub}}, v_{n'_{sub}+1}, \ldots, v_n})\right|$. Hence, there can be at maximum $|colorclass(1, f)|$ calls to Algorithm 5. Thus, as $|colorclass(1, f)| \leq \lceil \frac{n}{3} \rceil$ (Lemma 8) and $n_{sub} - t \leq n$, there are at most $\lceil \frac{n^2}{3} \rceil$ swaps. □

As a consequence of the correctness of Algorithm 4, the following corollary is immediate.

Corollary 4. *The node of any 3-equitable coloring of a path P and a 3-equitable coloring of P that satisfies the dense ones property are connected by a path of length at most $\lceil \frac{n^2}{3} \rceil$ in $R_{3-ECR}(P)$.*

We now define the alternating ones property and an algorithm, Algorithm 8, to reconfigure any 3-equitable coloring of a path that satisfies the dense ones property to a 3-equitable coloring that satisfies the alternating ones property. We remind the reader that a 3-equitable coloring restricted to the subpath $v_1, v_2, \ldots, v_n$ is c-alternating if the colors of the vertices alternate between c and any other color (Definition 25).

Definition 25. We say that a 3-equitable coloring of a path $P = v_1, v_2, \ldots, v_n$ satisfies the *alternating ones* property if either v_n or v_{n-1} has color 1 and the subpath of P that contains the leftmost 1 and all vertices right of it is 1-alternating.

Before describing the algorithm (Algorithm 8) to convert a 3-equitable coloring that satisfies the dense ones property to a 3-equitable coloring that satisfies the alternating

ones property we describe two symmetric operations, Algorithms 6 and 7, that we use in Algorithm 8 and throughout the rest of the section. In Algorithm 8, we use Algorithm 7 to obtain a 3-equitable coloring where a subpath of P is 1-alternating.

Algorithm 6 takes as input a subpath $S = v_1, v_2, \ldots, v_n$ and a 3-equitable coloring where $S \setminus \{v_1\}$ is c-alternating and v_1 and v_2 do not have color c. At an iteration i, neither v_i nor v_{i+1} has color c and v_{i+2} has color c. We refer to v_i and v_{i+1} as the *buffer of iteration* i. At each iteration, the successor of v_{i+1} and the predecessor of v_i either have color c or do not exist. Since the vertices of the buffer of the iteration do not have color c, we can, optionally, swap the colors of the two vertices of the buffer. We refer to this as an *optional swap*. At each iteration, $swap(v_{i+1}, v_{i+2})$ is performed, and this is referred to as a *required swap*. If $f(v_{i+1}) = f(v_{i+3})$, the required swap is not valid, and so the algorithm first performs the optional swap. Since f is proper, after the optional swap, $f(v_{i+1}) \neq f(v_{i+3})$ and so $swap(v_{i+1}, v_{i+2})$ is valid. After $swap(v_{i+1}, v_{i+2})$, v_{i+2} does not have color c. Since before the first iteration $S \setminus \{v_1\}$ was c-alternating, if $n - i - 4 \geq 0$, v_{i+4} has color c. On incrementing i to $i+2$, either $i > n-2$ or neither v_i nor v_{i+1} has color c and v_{i+2} has color c. So at the end of the while loop $v_1, v_2, \ldots, v_{n-1}$ is c-alternating.

An example of these operations is given in Figure 5.2.

Algorithm 6 LeftShift

Input: A subpath $S = v_1, v_2, \ldots, v_n$ of some path and a 3-equitable coloring f of the path such that: $v_2, v_3, \ldots, v_n$ is c-alternating for some color c; $f(v_1), f(v_2) \neq c$ (i.e., each vertex with an odd index, apart from v_1, has color c); and if there is a predecessor of v_1 then it has color c in f.

Output: A 3-equitable coloring such that $v_1, v_2, \ldots, v_{n-1}$ is c-alternating, v_2 has color c, and the colors of vertices not in S are unchanged.

```
1: procedure LeftShift(S = v_1, ..., v_n, f)
2:     i ← 1
3:     while i ≤ n − 2 do
4:         if f(v_{i+1}) = f(v_{i+3}) then
5:             f ← SwapVertices(f, v_i, v_{i+1})
6:             ▷ Optional swap
7:         end if
8:         f ← SwapVertices(f, v_{i+1}, v_{i+2})
9:         ▷ Required swap
10:        i ← i + 2
11:    end while
12:    return f
13: end procedure
```

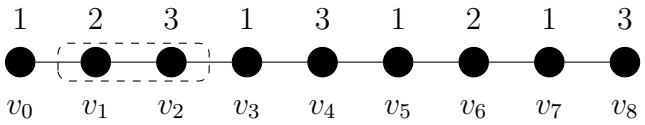
Algorithm 7 takes as input a subpath $S = v_1, v_2, \ldots, v_n$ and a 3-equitable coloring where $S \setminus \{v_n\}$ is c-alternating and v_n and v_{n-1} do not have color c. The algorithm reverses the order of vertices in S and passes S and f as input to Algorithm 6. After reversing the the order of vertices in S to its original order, the algorithm returns the produced 3-equitable coloring.

Algorithm 7 RightShift

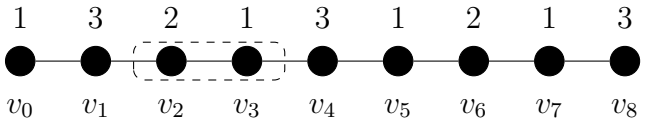
Input: A subpath $S = v_1, v_2, \ldots, v_n$ of some path and a 3-equitable coloring f of the path such that: $v_1, v_2, \ldots, v_{n-1}$ is c-alternating for some color c; $f(v_{n-1}), f(v_n) \neq c$ (i.e., each vertex, apart from v_n, whose index has the same parity as n has color c); and if there is a successor of v_n then it has color c in f.

Output: A 3-equitable coloring such that $v_2, v_3, \ldots, v_n$ is c-alternating, v_{n-1} has color c, and the color of vertices not in S are unchanged.

1: **procedure** RightShift($S = v_1, \ldots, v_n,\ f$)
2: ReversePath(S)
3: $f \leftarrow$ LeftShift($S = v_n, v_{n-1}, \ldots, v_1,\ f$)
4: ReversePath(S)
5: **return** f
6: **end procedure**

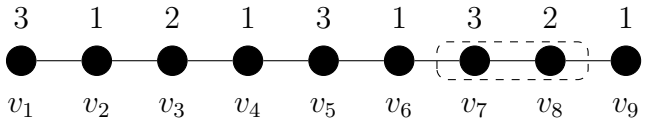


(a) An optional swap at iteration $i = 1$ between v_1 and v_2 because v_2 and v_4 have the same color.

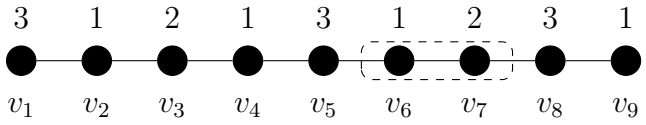


(b) Required swap at iteration $i = 1$ between v_2 and v_3.

Figure 5.2: One iteration of Algorithm 6



(a) Optional swap at iteration $i = 8$ between v_7 and v_8 because v_7 and v_5 have the same color.



(b) Required swap at iteration $i = 8$ between v_6 and v_7.

Figure 5.3: One iteration of Algorithm 7.

Since Algorithms 6 and 7 are symmetric, it suffices to prove the correctness of Algorithm 6.

Lemma 14. *Algorithm 6 is correct and uses at most $n - 1$ swaps.*

Proof. Before proceeding with our proof, we consider that there is a predecessor of v_1, v_0, and a successor of v_n, v_{n+1}. In Algorithm 6, it is optional for such a v_0 and v_{n+1} to exist, and our proof holds even when such a v_0 and v_{n+1} do not exist.

We prove by induction on the value of i, which can take values in the set $\{1, 3, \ldots, X\}$ ($X = n - 2$ when $n - 2$ is odd; $X = n - 3$ when $n - 2$ is even), that after an iteration i, both v_{i+1} and v_{i+4} have color c, neither v_{i+2} nor v_{i+3} has color c, and that f is proper. We also show that the swaps made in the algorithm are valid. From our input, this statement holds before the first iteration, which we assume to be after iteration $i = -1$. In the base case, after the iteration $i = -1$, our statement holds.

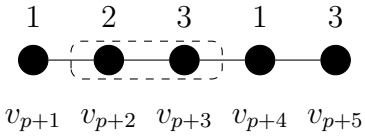
In the inductive step, we assume that our statement holds after the iteration $i = p$ and prove that it holds after the iteration $i = p + 2$. So, we can assume that $f(v_{p+1}) = f(v_{p+4}) = c$, $f(v_{p+2}) \neq c$, $f(v_{p+3}) \neq c$, and that f is proper.

We prove that the optional swap in line 6 for iteration $p + 2$ is valid. Since $f(v_{p+2}) \neq c$ and $f(v_{p+3}) \neq c$, neither of the two buffer vertices (v_{p+2} and $v_{(p+2)+1}$) has color c, and as $f(v_{p+1}) = f(v_{p+4}) = c$, both the predecessor of v_{p+2} and the successor of v_{p+3} have color c. It follows from this that $swap(v_{p+2}, v_{(p+2)+1})$ is valid (see Figure 5.4a).

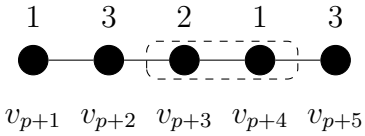
We prove that the required swap in line 7 for iteration $p+2$ is valid. We can say that $f(v_{(p+2)+1}) \neq f(v_{(p+2)+3})$ as otherwise the optional swap in line 6 would have been performed. Also, since $f(v_{(p+2)}) \neq c$, $swap(v_{(p+2)+1}, v_{(p+2)+2})$ is valid (see Figure 5.4b).

We now show that after iteration $p+2$, both $v_{(p+2)+1}$ and $v_{(p+2)+4}$ have color c, neither $v_{(p+2)+2}$ nor $v_{(p+2)+3}$ has color c, and that f is proper. It follows from the required swap in line 7 that $v_{(p+2)+1}$ has color c. As both the required swap and the optional swap are valid, f is proper. Since f is proper, $v_{(p+2)+2}$ does not have color c. Before the first iteration $v_2, v_3, \ldots, v_n$ is c-alternating, and after an iteration i, as only the colors of vertices in the set $\{v_i, v_{i+1}, v_{i+2}\}$ are changed in iteration i, the color of a vertex v_j, $i+3 \leq j \leq n$, is unchanged. So after an iteration i, $v_{i+3}, v_{i+4}, \ldots, v_n$ is c-alternating. Since after iteration p $v_{(p+2)+2}$ had color c and $v_{(p+2)+2}, v_{(p+2)+3}, \ldots, v_n$ was c-alternating, after iteration p, $v_{(p+2)+3}$ did not have color c and $v_{(p+2)+4}$ had color c. As the colors of $v_{(p+2)+3}$ and $v_{(p+2)+4}$ are unchanged in iteration $p+2$, after iteration $p+2$, $f(v_{(p+2)+3}) \neq c$ and $f(v_{(p+2)+4}) = c$. This also means that after iteration $i = 1$, v_2 has color 1.

As there are at most $\lceil \frac{n-2}{2} \rceil$ iterations and each iteration uses at most two swaps, the algorithm uses at most $n-1$ swaps. □



(a) The optional swap for iteration $p+2$, between v_{p+2} and v_{p+3}, is always valid.



(b) The required swap for iteration $p+2$, between v_{p+3} and v_{p+4}, is now valid.

Figure 5.4: Operations of Algorithm 6 for iteration $p+2$.

Definition 26. Given a 3-equitable coloring f of a path $v_1, v_2, \ldots, v_n$, for some i, $1 \leq i \leq n-3$, the vertices $v_i, v_{i+1}, v_{i+2}, v_{i+3}$ *form a dense block* if $f(v_i) = f(v_{i+3}) = 1$, $f(v_{i+1}) \neq 1$, and $f(v_{i+2}) \neq 1$.

Observation 14. *Given a 3-equitable coloring f of a path that satisfies the dense ones property, f does not satisfy the alternating ones property if and only if there is a dense block.*

Proof. It is can be seen that if there is a dense block in f, then f does not satisfy the alternating ones property. We next show that if f does not satisfy the alternating ones property, it implies that there is a dense block in f. If f satisfies the dense ones property and does not satisfy the alternating ones property but has no dense block, then the two possibilities are: the rightmost 1 is neither v_n nor v_{n-1}; and there are two vertices with color 1 and at distance greater than 3 from each other. But both the cases contradict the dense ones property. □

We now describe Algorithm 8. Algorithm 8 enumerates vertices $v_s, v_{s+1}, v_{s+2}, v_{s+3}$ from $s = n-3$ to $s = 1$ and checks, at each iteration, if the four vertices form a dense block. If in an iteration the vertices $v_s, v_{s+1}, v_{s+2}, v_{s+3}$ form a dense block, notice that f restricted to v_s, v_{s+1} is 1-alternating, $f(v_{s+1}) \neq 1$, $f(v_{s+2}) \neq 1$, and $f(v_{s+3}) = 1$. So we can call Algorithm 7 (RightShift) on the subpath v_s, v_{s+1}, v_{s+2}. After the call to Algorithm 7, f restricted to $v_s, v_{s+1}, v_{s+2}, v_{s+3}$ satisfies the alternating ones property. Also, as the color of no vertex right of v_{s+2} is changed and $f(v_{s+1}) = 1$, f restricted to $v_{s+1}, v_{s+2}, \ldots, v_n$ satisfies the alternating ones property. But, f restricted to $v_1, v_2, \ldots, v_{s+1}$ might not satisfy the dense ones property, as a vertex left of v_{s+1} with color 1 may be at distance 4 from v_{s+1}. To fix the dense ones property in f without changing the color of vertices right of v_s, Algorithm 9 is called, which we describe below.

Algorithm 9 takes as input a path P, a 3-equitable coloring f of the path, and vertices v_ℓ and v_r such that f restricted to the two connected subpaths in $P \setminus \{v_{\ell+1}, v_{\ell+2}, \ldots, v_{r-1}\}$, respectively, satisfy the dense ones property, both v_ℓ and v_r have color 1, and there is no vertex right of v_ℓ and left of v_r with color 1. Algorithm 9 outputs a 3-equitable coloring that satisfies the dense ones property such that the color of each vertex right of v_{r-1} is unchanged. Since f restricted to the two connected subpaths in $P \setminus \{v_{\ell+1}, v_{\ell+2} \ldots, v_{r-1}\}$ satisfies the dense ones property and both v_ℓ and v_r have color 1, the subpath of P where the dense ones property may not be satisfied is $v_\ell, v_{\ell+1}, \ldots, v_r$. If f restricted to $v_\ell, v_{\ell+1}, \ldots, v_r$ satisfies the dense ones property then it also follows that f satisfies the dense ones property. Otherwise, we can see that v_ℓ is at a distance of greater than or equal to 4 from v_r. In each iteration of a while loop, Algorithm 9 swaps the colors of only vertices left of v_r so that the distance between v_r and the closest vertex left of v_r with color 1 is reduced. Since no vertex in the set $\{v_{\ell+1}, \ldots, v_{r-1}\} \cup \{v_{\ell-1}\}$ has color 1 and f uses three colors, if $\ell > 1$, either $f(v_{\ell-1}) \neq f(v_{\ell+1})$ or $f(v_{\ell-1}) \neq f(v_{\ell+2})$. So either $swap(v_\ell, v_{\ell+1})$ is valid or $swap(v_\ell, v_{\ell+2})$

is valid. After either of these swaps ℓ is changed to $\ell + 1$ or $\ell + 2$, respectively, and the distance between v_ℓ and v_r is reduced. If the distance between v_ℓ and v_r is still greater than or equal to 4, then the while loop continues. Otherwise, if there is no vertex left of v_ℓ with color 1, then as f restricted to $v_\ell, v_{\ell+1}, \ldots, v_n$ satisfies the dense ones property f also satisfies the dense ones property, and f is returned. If there is a vertex left of v_ℓ with color 1, f restricted to $v_1, v_2, \ldots, v_\ell$ may not satisfy the dense ones property as the closest vertex left of v_ℓ with color 1, say w may be at a distance greater than or equal to 4 from v_ℓ, so ℓ is updated to the index of w, and r to the previous value of ℓ and the while loop continues. Since at each iteration either the distance between v_ℓ and v_r keeps decreasing (until the distance is less than 4) or ℓ is decremented, the while loop ends.

An example of one iteration of Algorithm 8 is given in Figure 5.5.

Algorithm 8 ShiftOnesCloser

Input: P and a 3-equitable coloring that satisfies the dense ones property.
Output: A 3-equitable coloring that satisfies the alternating ones property.

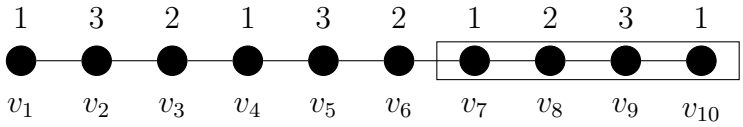
```
1: procedure ShiftOnesCloser(P, f)
2:     if n ≤ 3 then
3:         return f
4:     end if
5:     for s := n − 3 to 1 do
6:         if f(v_s), f(v_{s+3}) = 1 and f(v_{s+1}), f(v_{s+2}) ≠ 1 then
7:             f ← RightShift(S = v_s, v_{s+1}, v_{s+2}, f)
8:             if f restricted to v_{s−3}, v_{s−2}, v_{s−1}, v_s, v_{s+1} does not satisfy the dense ones property then
9:                 f ← FixDenseOnes(P, f, s − 3, s + 1)
10:            end if
11:        end if
12:    end for
13:    return f
14: end procedure
```

Algorithm 9 FixDenseOnes

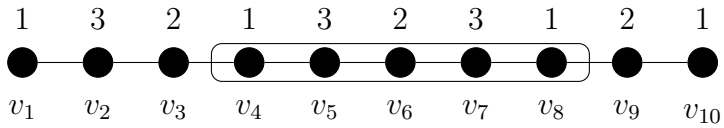
Input: P, a 3-equitable coloring f, the index of a vertex v_ℓ and the index of a vertex v_r such that f restricted to either of the two subpaths $v_1, v_2, \ldots, v_\ell$ or $v_r, v_{r+1}, \ldots, v_n$ satisfies the dense ones property, and there is no vertex right of v_ℓ and left of v_r with color 1.

Output: A 3-equitable coloring where the dense ones property is satisfied and the colors of vertices in the set $\{v_r, v_{r+1}, \ldots, v_n\}$ are unchanged.

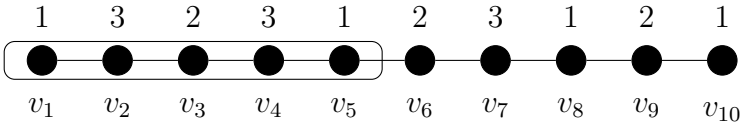
1: **procedure** FixDenseOnes(P, f, ℓ, r)
2: **while** Distance(P, v_ℓ, v_r) ≥ 4 or ExistsOne($P, \ell - 1, f$) = true **do**
3: ▷ ExistsOne checks if there exists a vertex in the subpath $v_1, v_2, \ldots, v_{\ell-1}$ with color 1 in f
4: **if** Distance(P, v_ℓ, v_r) ≥ 4 **then**
5: ▷ Distance returns the distance between v_ℓ and v_r in P
6: **if** $\ell > 1$ and $f(v_{\ell-1}) = f(v_{\ell+2})$ **then**
7: $f \leftarrow$ SwapVertices($f, v_\ell, v_{\ell+1}$)
8: $\ell \leftarrow \ell + 1$
9: **else**
10: $f \leftarrow$ SwapVertices($f, v_\ell, v_{\ell+2}$)
11: $\ell \leftarrow \ell + 2$
12: **end if**
13: **else if** ExistsOne($P, \ell - 1, f$) = true **then**
14: $r \leftarrow \ell$
15: $\ell \leftarrow$ the index of the vertex at minimum distance from v_r that has color 1 and is left of v_r
16: **end if**
17: **end while**
18: **return** f
19: **end procedure**



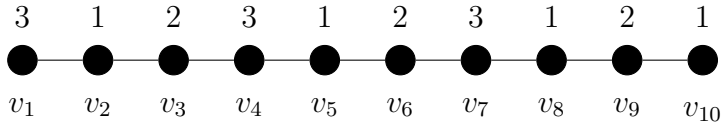
(a) In the first iteration of Algorithm 8, where $s = 7$, swaps are made to the dense block v_7, v_8, v_9, v_{10} so that it satisfies the alternating ones property.



(b) After line 7 in Algorithm 8, the dense ones property is not satisfied by the vertices enclosed in the rectangle with rounded corners. In the first iteration of Algorithm 9, in line 7, a swap is made between v_4 and v_5.



(c) After one iteration of Algorithm 9, the dense ones property is not satisfied by the vertices enclosed in the rectangle with rounded corners. In line 7, a swap is made between v_1 and v_2.



(d) The dense ones property is satisfied by the entire path, and the subpath v_7, v_8, v_9, v_{10} is 1-alternating.

Figure 5.5: Example of one iteration of Algorithm 8.

We give the proof of correctness of Algorithms 8 and 9 in Lemma 15.

Lemma 15. *Algorithm 8 is correct and the number of swaps it uses is in* $O(n^3)$.

Proof. We first show that Algorithm 8 is correct if Algorithm 9 is correct, and then prove the correctness of Algorithm 9. From Observation 14, if there is no dense block in f and f satisfies the dense ones property then f also satisfies the alternating ones property. Algorithm 8 enumerates, in a for loop, each four adjacent vertices in P $v_s, v_{s+1}, v_{s+2}, v_{s+3}$

from $s = n - 3$ to $s = 1$. At each iteration, the algorithm first checks if $v_s, v_{s+1}, v_{s+2}, v_{s+3}$ forms a dense block, and if so, as both v_s and v_{s+3} have color 1 and neither v_{s+1} nor v_{s+2} has color 1, Algorithm 7 (RightShift) can be called on this subpath in line 7. After Algorithm 7 (RightShift) is called, f restricted to $v_{s+1}, v_{s+2}, v_{s+3}$ is 1-alternating and so $v_s, v_{s+1}, v_{s+2}, v_{s+3}$ no longer forms a dense block (though $v_s, v_{s+1}, v_{s+2}, v_{s+3}$ satisfies the dense ones property). Due to previous iterations of the for loop, f restricted to $v_{s+1}, v_{s+2}, \ldots, v_n$ satisfies the dense ones property and there is no dense block in this subpath. However, f restricted to $v_1, v_2, \ldots, v_{s+1}$ need not satisfy the dense ones property as there might be a vertex at distance 4 from v_{s+1} with color 1 and left of v_{s+1}. To fix this, Algorithm 9 is called in line 9. As Algorithm 9 does not change the color of vertices to the right of v_{s+1}, the correctness of Algorithm 8 follows.

We prove the correctness of Algorithm 9. We show that after each iteration of the while loop, f restricted to $v_r, v_{r+1}, \ldots, v_n$ satisfies the dense ones property, there is no vertex with color 1 that is both right of v_ℓ and left of v_r, and that after each iteration either the values of ℓ and r are decremented or the distance between v_ℓ and v_r is reduced. In an iteration, we first consider the case when the distance between v_ℓ and v_r is greater than or equal to 4. In line 7, since $f(v_{\ell-1}) = f(v_{\ell+2})$ and f is proper, $f(v_{\ell+1}) \neq f(v_{\ell-1})$. Also, $f(v_{\ell+2}) \neq 1$ as there is no vertex right of v_ℓ and left of v_r with color 1. Consequently, $swap(v_\ell, v_{\ell+1})$ is valid in line 7 and ℓ is incremented to $\ell+1$ in the next line. In line 10, $v_{\ell+3}$ does not have color 1 (no vertex right of v_ℓ and left of v_r has color 1) and either $\ell = 1$ or $f(v_{\ell-1}) \neq f(v_{\ell+2})$. Consequently, $swap(v_\ell, v_{\ell+2})$ is valid in line 10 and ℓ is incremented to $\ell+2$ in the next line. It is immediate that after ℓ has been incremented, there is no vertex right of v_ℓ and left of v_r with color 1 and f restricted to $v_r, v_{r+1}, \ldots, v_n$ satisfies the dense ones property. We consider the case when the distance between v_ℓ and v_r is less than 4 and there exists a vertex left of v_ℓ with color 1. Since f restricted to $v_r, v_{r+1}, \ldots, v_n$ satisfies the dense ones property and the distance between v_ℓ and v_r is less than 4, f restricted to $v_\ell, v_{\ell+1}, \ldots, v_n$ satisfies the dense ones property. In line 14, r is changed to ℓ. In line 15, ℓ is changed to the index of the vertex at minimum distance and left of v_r with color 1. The swaps made in the algorithm are valid and at no point is the color of a vertex right of v_r changed. The correctness of Algorithm 9 follows.

We now prove that the number of swaps that Algorithm 8 uses is in $O(n^3)$. In an iteration of the for loop in Algorithm 8, the call to Algorithm 7 (RightShift) in line 7 uses at most 2 swaps. In line 9, for a value of $s+1$, the number of swaps that a call to Algorithm 9 uses is in $O(s^2)$. This is because, for a specific value of r, to decrease the distance between v_ℓ and v_r, Algorithm 9 uses at most r swaps, and r can be decremented until it has the value 1. Since in Algorithm 8 s can take values from $n-3$ to 1, the number of swaps used is in $O(n^3)$. □

The following corollary is immediate from the proof of correctness of Algorithm 8.

Corollary 5. *The node of a 3-equitable coloring of a path that satisfies the dense ones property is connected to the node of a 3-equitable coloring that satisfies the alternating ones property by a path of length in $O(n^3)$.*

We define the matching ones property, and algorithms to show that we can reconfigure from a 3-equitable coloring that satisfies the alternating ones property to a 3-equitable coloring that satisfies the matching ones property.

Definition 27. We say that a 3-equitable coloring of a path $P = v_1, v_2, \ldots, v_n$ satisfies the *matching ones* property if each vertex in P that has color 1 has the same color in its canonical 3-equitable coloring.

To reconfigure a 3-equitable coloring that satisfies the alternating ones property to a 3-equitable coloring that satisfies the matching ones property, we consider when n is odd and when n is even and describe two algorithms, Algorithm 10 and Algorithm 11, respectively. This is because, in Definition 18, we defined two configurations of canonical 3-equitable colorings based on whether n is odd or even.

Before describing Algorithms 10 and 11, we make the following observation. As a consequence of this observation, given a path and a 3-equitable coloring that satisfies the alternating ones property, if the number of vertices of the path is greater than 15, a vertex v that does not have color 1, such that both its predecessor and successor (if it exists) have color 1 is guaranteed to be returned by a procedure, GetIsolatedVertex, that we call in Algorithms 10 and 11.

Observation 15. *Given a path $P = v_1, v_2, \ldots, v_n$, where $n > 15$, a 3-equitable coloring that satisfies the alternating ones property, and color $c \neq 1$, there exists a vertex v in the path with color c such that either both the successor and predecessor of v have color 1 or the predecessor of v has color 1 and $v = v_n$.*

Proof. We assume without loss of generality that $c = 2$. We prove the observation by way of contradiction. Suppose that there is no such v. Let v_ℓ be the leftmost 1 in P. Notice that $v_\ell, v_{\ell+1}, \ldots, v_n$ is 1-alternating and as no vertex left of v_ℓ has color 1 and f is proper, there are at least $\lfloor \frac{\ell-1}{2} \rfloor$ vertices with color 3 in the subpath $v_1, v_2, \ldots, v_{\ell-1}$. We now consider the following two cases.

1. the rightmost 1 is v_{n-1}.

2. the rightmost 1 is v_n.

In Case (1), as the rightmost 1 is v_{n-1} and f satisfies the alternating ones property, the leftmost 1 is at a distance of $2\,|colorclass(1,f)| - 2$ from v_{n-1}, and so $\ell = n - 2\,|colorclass(1,f)| + 1$. Also, as there is no such v in P, no vertex in the subpath $v_\ell, v_{\ell+1}, \ldots, v_n$ can have color 2, and so, there are $|colorclass(1,f)|$ vertices in the subpath $v_\ell, v_{\ell+1}, \ldots, v_n$ with color 3. Since there are at least $\lfloor \frac{\ell-1}{2} \rfloor$ vertices with color 3 in the subpath $v_1, v_2, \ldots, v_{\ell-1}$, this implies Equation 5.4.

$$|colorclass(3,f)| \geq \left\lfloor \frac{\ell-1}{2} \right\rfloor + |colorclass(1,f)| \tag{5.1}$$

$$= \left\lfloor \frac{(n - 2\,|colorclass(1,f)| + 1) - 1}{2} \right\rfloor + |colorclass(1,f)| \tag{5.2}$$

$$= \left\lfloor \frac{n}{2} - \frac{2\,|colorclass(1,f)|}{2} \right\rfloor + |colorclass(1,f)| \tag{5.3}$$

$$= \left\lfloor \frac{n}{2} \right\rfloor \tag{5.4}$$

The functions $\frac{n}{2}$ and $\frac{n}{3}$ are non-decreasing and the growth rate of $\frac{n}{2}$ is higher than the growth rate of $\frac{n}{3}$. When $n = 15$, $\frac{n}{2} > 7$ and $\frac{n}{3} = 5$. For $n > 15$, as the difference between $\frac{n}{2}$ and $\frac{n}{3}$ is greater than or equal to two, the difference between $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{3} \rceil$ is greater than zero. But, this means that Equation 5.4 contradicts Lemma 8 ($|colorclass(3,f)| \leq \lceil \frac{n}{3} \rceil$) and the fact that $n > 15$.

In Case (2), as the rightmost 1 is v_n and f satisfies the alternating ones property, the leftmost 1 is at a distance of $2\,|colorclass(1,f)| - 2$ from v_n, and so, $\ell = n - 2(|colorclass(1,f)| - 1)$. And, as there is no such v in P, no vertex in the subpath $v_\ell, v_{\ell+1}, \ldots, v_n$ can have color 2, and so, there are $|colorclass(1,f)| - 1$ vertices in the subpath $v_\ell, v_{\ell+1}, \ldots, v_n$ with color 3. Since there are at least $\lfloor \frac{\ell-1}{2} \rfloor$ vertices with color 3 in the subpath $v_1, v_2, \ldots, v_{\ell-1}$, this implies Equation 5.8.

$$|colorclass(3,f)| \geq \left\lfloor \frac{\ell-1}{2} \right\rfloor + |colorclass(1,f)| - 1 \tag{5.5}$$

$$= \left\lfloor \frac{n-2(|colorclass(1,f)|-1)-1}{2} \right\rfloor + |colorclass(1,f)| - 1 \tag{5.6}$$

$$= \left\lfloor \frac{n+1}{2} - \frac{2\,|colorclass(1,f)|}{2} \right\rfloor + |colorclass(1,f)| - 1 \tag{5.7}$$

$$= \left\lfloor \frac{n+1}{2} \right\rfloor - 1 \tag{5.8}$$

The functions $\frac{n+1}{2}$ and $\frac{n}{3}$ are non-decreasing and the growth rate of $\frac{n+1}{2}$ is higher than the growth rate of $\frac{n}{3}$. When $n = 15$, $\frac{n+1}{2} = 8$ and $\frac{n}{3} = 5$. For $n > 15$, as the difference between $\frac{n+1}{2}$ and $\frac{n}{3}$ is greater than or equal to three, the difference between $\lfloor \frac{n+1}{2} \rfloor - 1$ and $\lceil \frac{n}{3} \rceil$ is greater than zero. But, this means that Equation 5.8 contradicts Lemma 8 ($|colorclass(3,f)| \leq \lceil \frac{n}{3} \rceil$) and the fact that $n > 15$. □

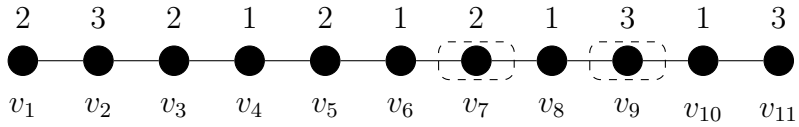
We first describe Algorithm 10, which handles the case when n is odd. The algorithm takes as input a path $v_1, v_2, \ldots, v_n$ of odd length and a 3-equitable coloring that satisfies the alternating ones property, and outputs a 3-equitable coloring that satisfies the matching ones property. Initially, it checks if v_n has color 1, and if so, then since f satisfies the alternating ones property it must also satisfy the matching ones property and so returns f. To eventually call Algorithm 7 (RightShift) on the subpath $v_\ell, v_{\ell+1}, \ldots, v_{n-1}$, the algorithm makes swaps so that v_{n-2} and v_{n-1} have different colors that are not 1. To make sure that v_{n-2} and v_{n-1} have different colors that are not 1, the algorithm first makes swaps so that v_{n-2} and v_n have different colors that are not 1.

Algorithm 10 MatchingOnesOdd

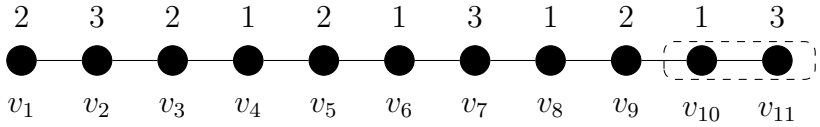
Input: $P = v_1, v_2, \ldots, v_n$ such that n is odd, $n > 15$, and a 3-equitable coloring that satisfies the alternating ones property.

Output: A 3-equitable coloring that satisfies the matching ones property.

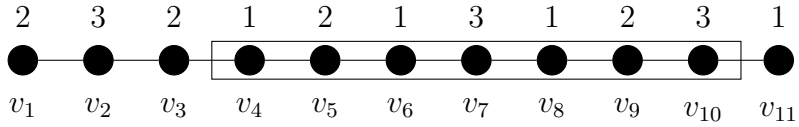
procedure MatchingOnesOdd(P, f)
    if $f(v_n) = 1$ **then**
        return f
    end if
    if $f(v_{n-2}) = f(v_n)$ **then**
        $c \leftarrow \{2, 3\} \setminus \{f(v_n)\}$
        $v \leftarrow$ GetIsolatedVertex(P, f, c)
        ▷ GetIsolatedVertex returns a vertex v with color c such that its predecessor and successor (if it exists) have color 1
        $f \leftarrow$ SwapVertices(f, v, v_{n-2})
    end if
    $f \leftarrow$ SwapVertices(f, v_{n-1}, v_n)
    $\ell \leftarrow$ LeftmostOne(P, n, f)
    $f \leftarrow$ RightShift($S = v_\ell, v_{\ell+1}, \ldots, v_{n-1}, f$)
    return f
end procedure



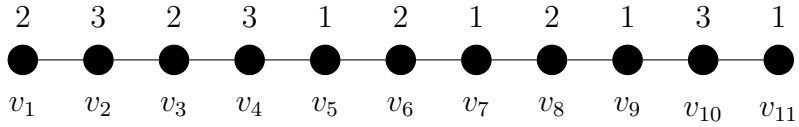
(a) To obtain a 3-equitable coloring where v_9 (v_{n-2}) and v_{11} (v_n) have different colors that are not 1, in line 9, the algorithm can perform $swap(v_7, v_9)$, where $v = v_7$.



(b) Since v_{11} (v_n) and v_9 (v_{n-2}) have different colors that are not 1, in line 11, the algorithm can perform $swap(v_{10}, v_{11})$, where $v_{n-1} = v_{10}$.



(c) In line 13, Algorithm 7 (RightShift) is called on input the subpath $v_4, v_5, \ldots, v_{10}$ and f.



(d) The 3-equitable coloring satisfies the matching ones property.

Figure 5.6: An example of the operations in Algorithm 10.

Lemma 16. *Algorithm 10 is correct and the number of swaps it uses is in $O(n)$.*

Proof. We show that the swaps used in the algorithm are valid and it returns a 3-equitable coloring that satisfies the matching ones property. In line 2, as v_n has color 1 and f satisfies the alternating ones property, $v_\ell, v_{\ell+1}, \ldots, v_n$ is 1-alternating, where v_ℓ is the leftmost 1 in the path, so since n is odd f satisfies the matching ones property.

We show that before line 11, $f(v_{n-2}) \neq f(v_n)$. In line 8, GetIsolatedVertex returns vertex v with color not in $\{1, f(v_{n-2})\}$ such that both its predecessor and successor (if $v \neq v_n$) have color 1; the vertex v is guaranteed by Observation 15, so a linear scan of the vertices of P suffices to return v. As v_{n-1} has color 1 and f is proper, v_{n-2} does not have color 1 before line 9 and as f satisfies the alternating ones property and, as $n > 15$,

$|colorclass(1, f)| \geq 2$ (Lemma 8) v_{n-3} has color 1. Hence, in line 9 $swap(v, v_{n-2})$ is valid, and moreover v_{n-2} still does not have color 1. As a consequence, we can assume before line 11 that $f(v_{n-2}) \neq f(v_n)$ as either $f(v_{n-2}) \neq f(v_n)$ originally or $swap(v, v_{n-2})$ is performed in line 9.

We now show that $swap(v_{n-1}, v_n)$ is valid in line 11 and a call to Algorithm 7 (RightShift) can be made in line 13. As $f(v_{n-2}) \neq f(v_n)$, $swap(v_{n-1}, v_n)$ is valid. After performing $swap(v_{n-1}, v_n)$ in line 11, both v_{n-2} and v_{n-1} have different colors that are not 1 and v_n has color 1. Since $v_\ell, v_{\ell+1}, \ldots, v_{n-2}$ is 1-alternating (as v_{n-3} has color 1), both v_{n-2} and v_{n-1} have different colors that are not 1, and v_n has color 1, Algorithm 7 (RightShift) can be called on the subpath $v_\ell, v_{\ell+1}, \ldots, v_{n-1}$. So $v_{\ell+1}, v_{\ell+2}, \ldots, v_{n-2}$ is 1-alternating and v_{n-2} has color 1.

Since the number of swaps made in lines 9 and 11 is in $O(1)$ and the number of swaps made by Algorithm 7 (RightShift) in line 13 is in $O(n)$ (Lemma 14), the number of swaps made by Algorithm 10 is also in $O(n)$. □

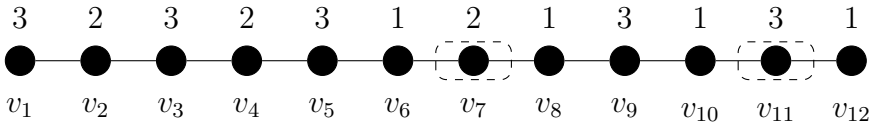
Algorithm 11 takes as input a path $v_1, v_2, \ldots, v_n$ of even length and a 3-equitable coloring that satisfies the alternating ones property, and outputs a 3-equitable coloring that satisfies the matching ones property. Initially, it checks if v_{n-1} has color 1, and if so, as f satisfies the alternating ones property f must also satisfy the matching ones property. Otherwise, to be able to call Algorithm 6 (LeftShift) on the subpath $v_{\ell-2}, v_{\ell-1}, \ldots, v_{n-2}$, the algorithm makes swaps so that $v_{\ell-3}$ has color 1, and for this purpose, the algorithm first makes swaps so that $v_{\ell-3}$ and v_{n-1} have different colors. After calling Algorithm 6 (LeftShift), the algorithm considers two cases: $f(v_{\ell-2}) \neq f(v_{n-2})$ and $f(v_{\ell-2}) = f(v_{n-2})$. In both the cases, the algorithm makes swaps so that v_{n-1} has color 1 and $v_{\ell-3}$ no longer has color 1.

Algorithm 11 MatchingOnesEven

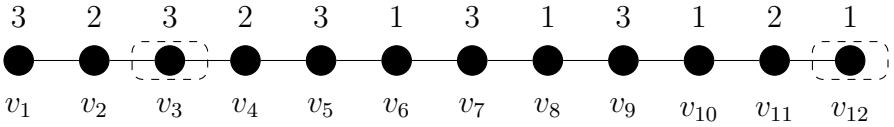
Input: $P = v_1, v_2, \ldots, v_n$ such that n is even, $n > 15$, and a 3-equitable coloring that satisfies the alternating ones property.

Output: A 3-equitable coloring that satisfies the matching ones property.

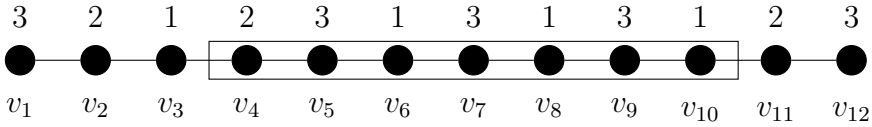
```
1: procedure MatchingOnesEven(P, f)
2:     if f(v_{n-1}) = 1 then
3:         return f
4:     end if
5:     ℓ ← LeftmostOne(P, n, f)
6:     if f(v_{ℓ-3}) = f(v_{n-1}) then
7:         c ← {2, 3} \ {f(v_{n-1})}
8:         v ← GetIsolatedVertex(P, f, c)
9:         ▷ GetIsolatedVertex returns a vertex v with color c such that its predecessor
   and successor (if it exists) has color 1
10:        f ← SwapVertices(f, v, v_{n-1})
11:    end if
12:    f ← SwapVertices(f, v_{ℓ-3}, v_n)
13:    f ← LeftShift(S = v_{ℓ-2}, v_{ℓ-1}, ..., v_{n-2}, f)
14:    if f(v_{ℓ-4}) ≠ f(v_{ℓ-2}) then
15:        v ← GetIsolatedVertex(P, f, f(v_{ℓ-4}))
16:        f ← SwapVertices(f, v, v_{ℓ-2})
17:    end if
18:    if f(v_{ℓ-2}) ≠ f(v_{n-2}) then
19:        f ← SwapVertices(f, v_{ℓ-3}, v_n)
20:        f ← SwapVertices(f, v_{n-1}, v_n)
21:    else
22:        f ← SwapVertices(f, v_{ℓ-3}, v_{n-1})
23:    end if
24:    return f
25: end procedure
```



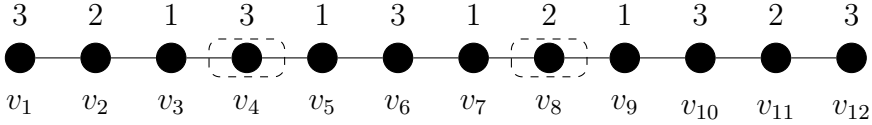
(a) To obtain a 3-equitable coloring where v_3 ($v_{\ell-3}$) and v_{11} (v_{n-1}) have different colors that are not 1, in line 10, the algorithm performs $swap(v_7, v_{11})$, where $v = v_7$.



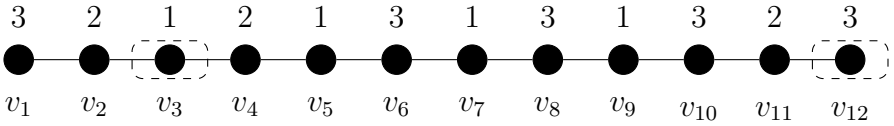
(b) Before calling Algorithm 6 (LeftShift), in line 12, the algorithm performs $swap(v_3, v_{12})$, where $v_{\ell-3} = v_3$ and $v_n = v_{12}$.



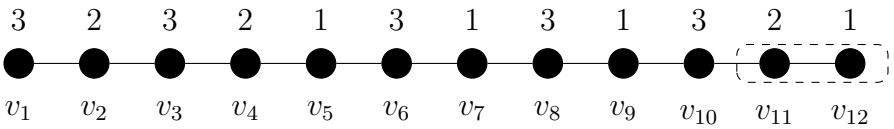
(c) In line 13, the algorithm calls Algorithm 6 (LeftShift) on the subpath $v_4, v_5, \ldots, v_{10}$.



(d) In line 16, to obtain a 3-equitable coloring where v_2 ($v_{\ell-4}$) and v_4 ($v_{\ell-2}$) have the same color, the algorithm performs $swap(v_4, v_8)$ ($v = v_8$).



(e) If $f(v_{\ell-2}) \neq f(v_{n-2})$, to make the rightmost 1 v_{11}, in line 19, the algorithm performs $swap(v_3, v_{12})$.



(f) In line 20, the algorithm performs $swap(v_{11}, v_{12})$, obtaining a 3-equitable coloring that satisfies the matching ones property.

Figure 5.7: An example of the operations in Algorithm 11.

Lemma 17. *Algorithm 11 is correct and the number of swaps it uses is in $O(n)$.*

Proof. We show that the swaps used in the algorithm are valid and that it returns a 3-equitable coloring that satisfies the matching ones property. In line 3, as v_{n-1} has color 1 and f is a 3-equitable coloring that satisfies the alternating ones property, $v_\ell, v_{\ell+1}, \ldots, v_{n-1}$ is 1-alternating where $f(v_n) \neq 1$, so f satisfies the matching ones property.

We now show that, before line 12, $f(v_{\ell-3}) \neq f(v_{n-1})$ (5.9). If $f(v_{\ell-3}) = f(v_{n-1})$, in line 8, GetIsolatedVertex returns a vertex v with color not in $\{1, f(v_{n-1})\}$ such that both its predecessor and successor (if $v \neq v_n$) have color 1; as in Algorithm 10, the vertex v is guaranteed by Observation 15 and so a linear scan of the vertices of P suffices to return v. As $n > 15$, $|colorclass(1, f)| \geq 2$ (Lemma 8), and as f satisfies the alternating ones property, both v_{n-2} and v_n have color 1. It follows that $swap(v, v_{n-1})$ is valid in line 10. If $f(v_{\ell-3}) \neq f(v_{n-1})$ originally, then the swap in line 10 is not performed.

We show that $swap(v_{\ell-3}, v_n)$ is valid in line 12 and Algorithm 6 (LeftShift) can be called in line 13. As v_ℓ is the leftmost 1 neither $v_{\ell-4}$ nor $v_{\ell-2}$ has color 1 and $f(v_{\ell-3}) \neq f(v_{n-1})$ (5.9), so $swap(v_{\ell-3}, v_n)$ is valid in line 12. Before $swap(v_{\ell-3}, v_n)$, v_ℓ was the leftmost 1 and v_n had color 1. Consequently, after $swap(v_{\ell-3}, v_n)$ neither $v_{\ell-2}$ nor $v_{\ell-1}$ has color 1 and $v_{\ell-3}$ has color 1. As $v_{\ell-3}$ has color 1, $v_{\ell-2}$ and $v_{\ell-1}$ have different colors that are not 1, and $v_\ell, v_{\ell+1}, \ldots, v_{n-2}$ is 1-alternating, Algorithm 6 (LeftShift) can be called on the subpath $v_{\ell-2}, v_{\ell-1}, \ldots, v_{n-2}$ in line 13. And after this $v_{\ell-2}, v_{\ell-1}, \ldots, v_{n-3}$ is 1-alternating, where $v_{\ell-1}$ has color 1.

We show that, before line 18, $f(v_{\ell-4}) = f(v_{\ell-2})$ (5.10). If $f(v_{\ell-4}) \neq f(v_{\ell-2})$, in line 15, GetIsolatedVertex returns a vertex v with color $f(v_{\ell-4})$ such that both its predecessor and successor (if $v \neq v_n$) have color 1. As $v_{\ell-3}$ has color 1 and, as Algorithm 6 (LeftShift) has been called in line 13, $v_{\ell-1}$ also has color 1, $swap(v, v_{\ell-2})$ is valid in line 16.

We next show that before line 18, $f(v_{n-2}) = f(v_n)$ (5.11). After line 3, as $v_\ell, v_{\ell+1}, \ldots, v_n$ is 1-alternating and f does not satisfy the matching ones property, both v_n and v_{n-2} have color 1 and v_{n-1} does not have color 1. Since the color of v_{n-1} is unchanged, v_{n-1} does not have color 1 before line 18. Since the colors of v_n and $v_{\ell-3}$ were swapped in line 12 and the color of v_n is unchanged, v_n does not have color 1. As Algorithm 6 (LeftShift) was called in line 13, v_{n-2} does not have color 1. Since none of the vertices in $\{v_{n-2}, v_{n-1}, v_n\}$ have color 1 (5.12) and f is proper and uses only three colors, $f(v_{n-2}) = f(v_n)$.

The algorithm now considers two cases: $f(v_{\ell-2}) \neq f(v_{n-2})$ and $f(v_{\ell-2}) = f(v_{n-2})$. In the case when $f(v_{\ell-2}) \neq f(v_{n-2})$, before line 19 (where $swap(v_{\ell-3}, v_{n-2})$ is performed), the following holds: $f(v_{\ell-4}) \neq f(v_n)$ (as $f(v_{\ell-4}) = f(v_{\ell-2})$ (5.10) and $f(v_n) = f(v_{n-2})$ (5.11)),

$f(v_{\ell-2}) \neq f(v_n)$ (as $f(v_{\ell-2}) \neq f(v_{n-2})$ and $f(v_n) = f(v_{n-2})$ (5.11)), $f(v_{\ell-3}) \neq f(v_{n-1})$ (as $f(v_{\ell-3}) = 1$), and there is no vertex right of v_n. Consequently, $swap(v_{\ell-3}, v_n)$ is valid in line 19. As v_{n-2} does not have color 1 (5.12), there is no vertex right of v_n, and v_n has color 1 after $swap(v_{\ell-3}, v_n)$, $swap(v_{n-1}, v_n)$ is valid in line 20. In the case when $f(v_{\ell-2}) = f(v_{n-2})$, before line 22, as $f(v_{\ell-2}) = f(v_{\ell-4})$ (5.10) and f is proper, $f(v_{n-1}) \neq f(v_{\ell-4})$ and $f(v_{n-1}) \neq f(v_{\ell-2})$. Also, neither v_n nor v_{n-2} has color 1 (5.12). So, since $v_{\ell-3}$ has color 1, $swap(v_{\ell-3}, v_{n-1})$ is valid in line 22.

After line 13, $v_{\ell-1}, v_\ell, \ldots, v_{n-3}$ is 1-alternating where v_{n-3} has color 1, and after line 22, v_{n-1} has color 1. It follows that $v_{\ell-1}, v_\ell, \ldots, v_{n-1}$ is 1-alternating where $f(v_{n-1}) = 1$.

The number of swaps used in the algorithm, other than in line 13, is in $O(1)$. As the number of swaps used by a call to Algorithm 6 (LeftShift) in line 13 is in $O(n)$ (Lemma 14), the number of swaps used in Algorithm 11 is also in $O(n)$. □

As a consequence of Lemmas 16 and 17, the following corollary is immediate.

Corollary 6. *The node of a 3-equitable coloring of a path P, such that $|V(P)| > 15$, that satisfies the alternating ones property is connected to the node of a 3-equitable coloring that satisfies the matching ones property by a path of length in $O(n)$.*

Before proceeding to Algorithm 14, we describe an operation, Algorithm 13. We use Algorithm 13 when obtaining the canonical 3-equitable coloring; specifically, we use it to obtain a 3-equitable coloring where each vertex right of the leftmost 1 that does not have color 1 has the same color in its canonical 3-equitable coloring.

Algorithm 13 takes as input a subpath $S = v_1, v_2, \ldots, v_n$ and a 3-equitable coloring that uses colors c_1, c_2, and c_3 where S is c_1-alternating and outputs a 3-equitable coloring where S is still c_1-alternating (and the color of each vertex with color c_1 is unchanged), and all vertices in S that have color c_2 (c_3) are grouped "close" to each other. (Here, by close we mean that each vertex with color c_2 (c_3) is at a distance of 2 from another vertex with color c_2 (c_3).) The algorithm iteratively selects a vertex from the set $\{v_2, v_3, \ldots, v_{n-1}\}$ with color c_3 and checks if there is a vertex right of it with color c_2, and if so, swaps the colors of these two vertices. If there is no vertex right of the current vertex with color c_3, then the algorithm exits as this means that S is still c_1-alternating and there is no vertex with color c_2 right of the leftmost c_3.

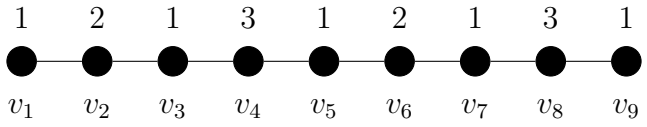
An example of an input and output of Algorithm 13 is given in Figure 5.8.

Algorithm 13 REARRANGE

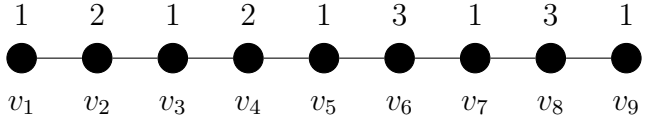
Input: A subpath $S = v_1, v_2, \ldots, v_n$ of some path and a 3-equitable coloring f of the path that uses colors c_1, c_2, c_3 such that $f(v_1) = c_1$ and $v_1, v_2, \ldots, v_n$ is c_1-alternating.

Output: A 3-equitable coloring such that $v_1, v_2, \ldots, v_n$ is c_1-alternating with $f(v_1) = c_1$, there is no vertex left of the rightmost c_2 with color c_3, and the color of a vertex not in S is unchanged.

```
1: procedure REARRANGE(S = v_1, ..., v_n, f, c_2, c_3)
2:     for i := 2 to n − 1 do
3:         if f(v_i) = c_3 then
4:             for j := i + 1 to n do
5:                 if f(v_j) = c_2 then
6:                     f ← SWAPVERTICES(f, v_i, v_j)
7:                     break
8:                 end if
9:             end for
10:            if j > n then
11:                break
12:            end if
13:        end if
14:    end for
15: end procedure
```



(a)



(b)

Figure 5.8: An example for Algorithm 13, where Figure 5.8a and Figure 5.8b show a subpath and a 3-equitable colorings before and after, respectively.

Lemma 18. *Algorithm 13 is correct and the number of swaps it uses is in $O(n)$, where $n = |V(S)|$.*

Proof. The proof of correctness of the algorithm follows from the observation that, as long as S is c_1-alternating, a swap of the colors of any two vertices in S that do not have color c_1 is valid.

We show by induction on i that, in line 3, there is no vertex left of v_i in S with color c_3. In the base case, when $i = 2$, as it is a condition on the input that $f(v_1) = c_1$, the statement is trivially true. In the inductive step, we assume that the statement is true for $i = p$ and prove the statement when $i = p + 1$. If $f(v_p) \neq c_3$ our statement is true, so we consider $f(v_p) = c_3$. In the case when there is no vertex right of v_p in S with color c_2, the algorithm terminates after iteration $i = p$ and i would not be incremented to $p + 1$. In the case when there is such a vertex v_j (where j is minimum) in iteration $i = p$, the colors of v_i and v_j are swapped in line 6. Since $j > i = p$ (v_j is right of v_i), v_p does not have color c_3 and no vertex left of v_{p+1} has color c_3.

As we have shown that, in line 3, there is no vertex left of v_i with color c_3, if $f(v_i) = c_3$, v_i is the leftmost c_3. Thus the algorithm terminates when there is no vertex right of the leftmost c_3 with color c_2.

We show that the number of swaps that the algorithm uses is in $O(n)$ by noting that the number of swaps is at most $|colorclass(c_2, f|_S)|$. This follows as, in line 5, when a vertex v_j with color c_2 is encountered, after $swap(v_i, v_j)$ is made in line 6, the vertex v_i, which now has color c_2, is not encountered again by the algorithm. As the swap in line 6 is made between a vertex with color c_3 and a vertex with color c_2, the number of swaps that the algorithm makes is at most $|colorclass(c_2, f|_S)|$, and so is in $O(n)$. □

We are now ready to describe Algorithm 14. The algorithm first calls Algorithm 13 (Rearrange) on the subpath $v_\ell, v_{\ell+1}, \ldots, v_n$ (where v_ℓ is the leftmost 1), so that each vertex in this subpath has the same color as in the canonical 3-equitable coloring. If v_1 has color 2, as the vertices with color 1 have the same color in the canonical 3-equitable coloring, it can be seen that f is the canonical 3-equitable coloring and is returned. Otherwise, to change the color of v_1 to 2 and to make $v_1, v_2, \ldots, v_{\ell-1}$ 3-alternating, $swap(v_1, v_\ell)$ (where v_ℓ is the leftmost 1) is performed so that Algorithm 6 (LeftShift) can be called on the subpath $v_1, v_2, \ldots, v_{\ell-1}$.

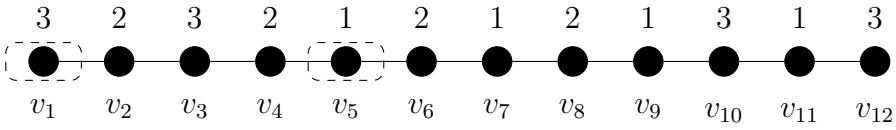
An example of the operations is given in Figure 5.9.

Algorithm 14 CanonicalReconf

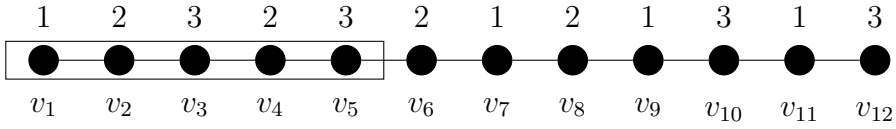
Input: A path $P = v_1, v_2, \ldots, v_n$ such that $n > 15$ and a 3-equitable coloring f that satisfies the matching ones property.

Output: The canonical 3-equitable coloring of P.

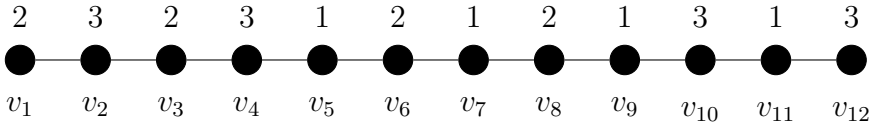
```
1: procedure CanonicalReconf(P,f)
2:     ℓ ← LeftmostOne(P, n, f)
3:     f ← Rearrange(S = v_ℓ, v_{ℓ+1}, ..., v_n, f, 2, 3)
4:     if f(v_1) = 2 then
5:         return f
6:     end if
7:     f ← SwapVertices(f, v_1, v_ℓ)
8:     f ← LeftShift(S = v_1, v_2, ..., v_ℓ, f)
9:     return f
10: end procedure
```



(a) In line 7, after calling Algorithm 13 and before calling Algorithm 6, the algorithm performs $swap(v_1, v_5)$ where $v_\ell = v_5$.



(b) In line 8, the algorithm calls Algorithm 6 (LeftShift) on the subpath $v_1, v_2, \ldots, v_5$ so that v_1 will have color 2.



(c) We obtain the canonical 3-equitable coloring!

Figure 5.9: An example of the operations in Algorithm 14.

Lemma 19. *Algorithm 14 is correct and the number of swaps it uses is in $O(n)$.*

Proof. We show that the swaps made in the algorithm are valid and the 3-equitable coloring produced is the canonical 3-equitable coloring. After the call to Algorithm 13 (REARRANGE) in line 3, from Lemma 18, we obtain a 3-equitable coloring where, in the subpath $v_\ell, v_{\ell+1}, \ldots, v_n$, there is no vertex left of the rightmost 2 with color 3. As v_ℓ is the leftmost 1, each vertex left of v_ℓ does not have color 1. Consequently, the colors of vertices in the subpath $v_1, v_2, \ldots, v_{\ell-1}$ alternate between 2 and 3. From Observation 1, $\ell - 1$ is even. So, in line 5, if $f(v_1) = 2$, $f(v_{\ell-1}) = 3$ and f is a canonical 3-equitable coloring (5.13).

We show that $swap(v_1, v_\ell)$ is valid in line 7. The colors of vertices v_1 and v_ℓ are swapped so that Algorithm 6 (LEFTSHIFT) can be called in line 8. If v_1 had color 2 f would have been returned in line 5 (5.13), so v_1 has color 3. As f is proper and $\ell > 2$ (as $n > 15$), v_2 has color 2. We now show that both $v_{\ell-1}$ and $v_{\ell+1}$ have color 2. From Observation 15, there is at least one vertex v with color 2 whose predecessor and successor (if the vertex is not v_n) have color 1. As Algorithm 13 (REARRANGE) had been called in line 3, there is no vertex left of the rightmost 2, in the subpath $v_\ell, v_{\ell+1}, \ldots, v_n$, with color 3. If $v_{\ell+1}$ does not have color 2 then no vertex right of $v_{\ell+1}$ can have color 2 and so no such v can exist, contradicting Observation 15. And, as $\ell - 1$ is even, v_1 has color 3, and the colors of the vertices in the subpath $v_1, v_2, \ldots, v_{\ell-1}$ alternate between 2 and 3, $v_{\ell-1}$ has color 2. So in line 7, $swap(v_1, v_\ell)$ is valid. Note that after $swap(v_1, v_\ell)$, v_1 has color 1.

We now show that Algorithm 6 (LEFTSHIFT) can be called on the subpath $v_1, v_2, \ldots, v_\ell$. We showed that v_2 has color 2, and after $swap(v_1, v_\ell)$ in line 7 v_1 has color 1. Also, since $v_2, v_3, \ldots, v_\ell$ is 3-alternating, Algorithm 6 (LEFTSHIFT) can be called in line 8.

After Algorithm 6 (LEFTSHIFT) has been called $v_1, v_2, \ldots, v_{\ell-1}$ is 3-alternating with $f(v_2) = 3$, and as $\ell - 1$ is even, $v_{\ell-1}$ has color 3. Since $v_{\ell+1}$ has color 2 and $v_{\ell-1}$ has color 3, v_ℓ has color 1. As the color of no vertex right of v_ℓ is changed by Algorithm 6 (LEFTSHIFT) and before line 8 only one vertex in the subpath $v_1, v_2, \ldots, v_\ell$ had color 1, since v_ℓ has color 1 after line 8, no other vertex left of v_ℓ can have color 1. Consequently, as v_2 has color 3, v_1 has color 2. Thus, as the colors of the vertices in the subpath $v_1, v_2, \ldots, v_{\ell-1}$ alternate between 2 and 3, v_1 has color 2, $v_{\ell-1}$ has color 3, and the color of each vertex right of $v_{\ell-1}$ in f is the same as the color of the vertex in the canonical 3-equitable coloring of f (as f satisfies the matching ones property), f is a canonical 3-equitable coloring.

As the number of swaps that Algorithm 13 (REARRANGE) and Algorithm 6 (LEFTSHIFT) use is in $O(n)$, the number of swaps that Algorithm 14 uses is also in $O(n)$. □

The following corollary is immediate from Lemma 19.

Corollary 7. *The node of a 3-equitable coloring of a path P, such that $|V(P)| > 15$, that satisfies the matching ones property is connected to the node of the canonical 3-equitable coloring by a path of length in $O(n)$.*

As a consequence of Corollaries 4, 5, 6, and 7, the following lemma is immediate.

Lemma 20. *The node of any 3-equitable coloring of a path P with $|V(P)| \geq 15$ is connected to its canonical 3-equitable coloring by a path of length in $O(n^3)$ in $R_{3-ECR}(P)$.*

We are now ready to prove Theorem 8.

Theorem 8. *Given a path P and two 3-equitable colorings f_s and f_e of P, we can decide if their nodes are connected in $R_{k-ECR}(P)$ in $O(|V(P)|)$ time.*

Proof. If $n \leq 15$, we can employ a brute-force algorithm to construct $R_{3-ECR}(P)$ and check if the nodes of f_s and f_e are connected. If $n > 15$, we can check if f_s and f_e are viable to each other in linear time. If f_s and f_e are viable to each other, then from Lemma 20, both f_s and f_e are connected to their canonical 3-equitable coloring, and their canonical 3-equitable colorings are the same. It follows that the nodes of f_s and f_e are connected in $R_{3-ECR}(P)$. □

Chapter 6

Conclusions and Future Work

We have shown that k-ACR REACH is PSPACE-hard for $k \geq 4$, and that k-ACR BOUND is in XP for the parameter $\ell + \Gamma(G)$. As we have shown that k-ACR BOUND is in XP, the next question is whether an XP algorithm is the best that we can hope for with respect to parameter $\ell + \Gamma(G)$. This raises the question of whether the W-hardness of k-ACR BOUND with respect to $\ell + \Gamma(G)$ can be proved. Additionally, since we only have weak PSPACE-hardness of k-ACR BOUND, the complexity of k-ACR BOUND is also a question for future study.

Since we have shown that for non-bipartite graphs of acyclic chromatic number three the reconfiguration graph for k-ACR REACH is not connected, an interesting question is whether other results about the connectivity of the reconfiguration graph for k-CR REACH can be extended to k-ACR REACH.

We have shown that k-ECR REACH is PSPACE-hard and W[1]-hard with respect to the size of a color class. It was also proved independently that k-ECR REACH is PSPACE-hard for fixed k even for the class of planar graphs when $k = 4$ [27]. Since it is unlikely for there to exist a fixed-parameter tractable algorithm with respect to either the size of a color class or k, whether there exists an FPT algorithm for k-ACR BOUND with respect to some parameter is a question of future study.

In Section 5.4, we show that k-ECR REACH can be solved in polynomial time for paths when $k = 3$. We conjecture that k-ECR REACH can also be solved in polynomial time for paths when $k > 3$. A possible algorithm could be to reconfigure a k-equitable coloring of a path to its canonical k-equitable coloring, which might look something like Figure 6.1.

Conjecture 1. k-ECR REACH can be solved in polynomial time for paths.

Conjecture 1 also raises the question of whether there exist polynomial-time algorithms for k-ECR REACH in other classes of graphs, like trees, stars, bounded treewidth graphs, and so on.

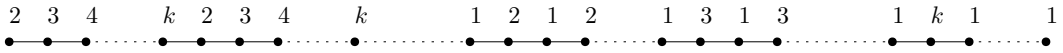


Figure 6.1: A possible canonical form for paths when $k > 3$.

References

[1] Noga Alon. Restricted colorings of graphs. *Surveys in Combinatorics*, 187:1–33, 1993.

[2] Noga Alon, Colin McDiarmid, and Bruce Reed. Acyclic coloring of graphs. *Random Structures and Algorithms*, 2(3):277–288, 1991.

[3] Kenneth Appel and Wolfgang Haken. Proof of 4-color theorem. *Discrete Mathematics*, 16(2):179–180, 1976.

[4] John Billingham, Robert Leese, and Hannu Rajaniemi. Frequency reassignment in cellular phone networks. *Smith Institute Study Group Report*, 2005.

[5] Hans L Bodlaender and Fedor V Fomin. Equitable colorings of bounded treewidth graphs. *Theoretical Computer Science*, 349(1):22–30, 2005.

[6] Béla Bollobás and Andrew J Harris. List-colourings of graphs. *Graphs and Combinatorics*, 1(1):115–127, 1985.

[7] Marthe Bonamy and Nicolas Bousquet. Recoloring bounded treewidth graphs. *Electronic Notes in Discrete Mathematics*, 44:257–262, 2013.

[8] Marthe Bonamy, Nicolas Bousquet, Carl Feghali, and Matthew Johnson. On a conjecture of Mohar concerning Kempe equivalence of regular graphs. *arXiv preprint arXiv:1510.06964*, 2015.

[9] Marthe Bonamy, Matthew Johnson, Ioannis Lignos, Viresh Patel, and Daniël Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 27(1):132–143, 2014.

[10] Paul Bonsma and Luis Cereceda. Finding paths between graph colourings: Pspace-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009.

[11] Paul Bonsma, Amer E Mouawad, Naomi Nishimura, and Venkatesh Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In *Proceedings of the 9^{th} International Symposium on Parameterized and Exact Computation*, pages 110–121. Springer, 2014.

[12] Oleg V Borodin. On acyclic colorings of planar graphs. *Discrete Mathematics*, 25(3):211–236, 1979.

[13] Rowland Leonard Brooks. On colouring the nodes of a network. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 37, pages 194–197. Cambridge University Press, 1941.

[14] Luis Cereceda. *Mixing graph colourings*. PhD thesis, The London School of Economics and Political Science (LSE), 2007.

[15] Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Connectedness of the graph of vertex-colourings. *Discrete Mathematics*, 308(5):913–919, 2008.

[16] Luis Cereceda, Jan Van den Heuvel, and Matthew Johnson. Mixing 3-colourings in bipartite graphs. *European Journal of Combinatorics*, 30(7):1593–1606, 2009.

[17] Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011.

[18] Reinhard Diestel. *Graph theory*. Springer-Verlag, Electronic Edition, 2005.

[19] Rodney G Downey and Michael R Fellows. *Fundamentals of Parameterized Complexity*, volume 4. Springer, 2013.

[20] Paul Erdős, Arthur L Rubin, and Herbert Taylor. Choosability in graphs. In *Processing West Coast Conference on Combinatorics, Graph Theory and Computing, Congressus Numerantium*, volume 26, pages 125–157, 1979.

[21] Carl Feghali, Matthew Johnson, and Daniël Paulusma. Kempe equivalence of colourings of cubic graphs. *Electronic Notes in Discrete Mathematics*, 49:243–249, 2015.

[22] Michael R Fellows, Fedor V Fomin, Daniel Lokshtanov, Frances Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Information and Computation*, 209(2):143–153, 2011.

[23] Steve Fisk. Geometric coloring theory. *Advances in Mathematics*, 24(3):298–340, 1977.

[24] Hanna Furmańczyk. Equitable coloring of graph products. *Opuscula Mathematica*, 26(1):31–44, 2006.

[25] Branko Grünbaum. Acyclic colorings of planar graphs. *Israel Journal of Mathematics*, 14(4):390–408, 1973.

[26] András Hajnal and Endre Szemerédi. Proof of a conjecture of P. Erdős. *Combinatorial Theory and its Applications*, 2:601–623, 1970.

[27] Tatsuhiko Hatanaka. Personal communication, Feb 2017. Corresponded through email.

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

[29] Takehiro Ito, Marcin Kamiński, and Erik Demaine. Reconfiguration of list edge-colorings in a graph. *Discrete Applied Mathematics*, 160(15):2199–2207, 2012.

[30] Takehiro Ito, Marcin Kamiński, Hirotaka Ono, Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka. On the parameterized complexity for token jumping on graphs. In *Proceedings of the International Conference on Theory and Applications of Models of Computation*, pages 341–351. Springer, 2014.

[31] Takehiro Ito, Kazuto Kawamura, Hirotaka Ono, and Xiao Zhou. Reconfiguration of list $L(2, 1)$-labelings in a graph. In *International Symposium on Algorithms and Computation*, pages 34–43. Springer, 2012.

[32] Takehiro Ito, Kazuto Kawamura, and Xiao Zhou. An improved sufficient condition for reconfiguration of list edge-colorings in a tree. In *Proceedings of the International Conference on Theory and Applications of Models of Computation*, pages 94–105. Springer, 2011.

[33] Svante Janson and Andrzej Rucinski. The infamous upper tail. *Random Structures and Algorithms*, 20(3):317–342, 2002.

[34] Tommy R Jensen and Bjarne Toft. *Graph Coloring Problems*, volume 39. John Wiley & Sons, 2011.

[35] Mark Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Structures and Algorithms*, 7(2):157–166, 1995.

[36] Matthew Johnson, Dieter Kratsch, Stefan Kratsch, Viresh Patel, and Daniël Paulusma. Colouring reconfiguration is fixed-parameter tractable. *arXiv preprint arXiv:1403.6347*, 2014.

[37] Richard M Karp. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[38] Alfred B Kempe. On the Geographical Problem of the Four Colours. *American Journal of Mathematics*, 2(3):193–200, 1879.

[39] Hal A Kierstead and Alexandr V Kostochka. A short proof of the Hajnal–Szemerédi Theorem on equitable colouring. *Combinatorics, Probability and Computing*, 17(02):265–270, 2008.

[40] Alexandr V Kostochka. *Upper Bounds of Chromatic Functions of Graphs (in Russian)*. PhD thesis, Novosibirsk, 1978.

[41] Alexandr V Kostochka. Personal communication, August 2016. Corresponded through email.

[42] Alexandr V Kostochka, Kittikorn Nakprasit, and Sriram V Pemmaraju. On equitable coloring of d-degenerate graphs. *SIAM Journal on Discrete Mathematics*, 19(1):83–95, 2005.

[43] Marek Kubale. *Graph Colorings*, volume 352. American Mathematical Society, 2004.

[44] Michel Las Vergnas and Henri Meyniel. Kempe classes and the Hadwiger conjecture. *Journal of Combinatorial Theory, Series B*, 31(1):95–104, 1981.

[45] R M R Lewis. *A Guide to Graph Colouring*. Springer International Publishing, Switzerland, 2016.

[46] Dániel Marx. Graph colouring problems and their applications in scheduling. *Periodica Polytechnica, Electrical Engineering*, 48(1):11–16, 2004.

[47] L S Melnikov and V G Vizing. New proof of Brooks' theorem. *Journal of Combinatorial Theory*, 7(4):289–290, 1969.

[48] Walter Meyer. Equitable coloring. *The American Mathematical Monthly*, 80(8):920–922, 1973.

[49] Bojan Mohar. Kempe equivalence of colorings. In *Graph Theory in Paris, Proceedings*

of a Conference in Memory of Claude Berge, pages 287–297. Birkhäuser Basel, 2007.

[50] Amer E Mouawad. *On Reconfiguration Problems: Structure and Tractability*. PhD thesis, University of Waterloo, 2015.

[51] Moritz Mühlenthaler and Rolf Wanka. On the connectedness of clash-free timetables. *arXiv preprint arXiv:1507.02805*, 2015.

[52] Sriram V Pemmaraju. Equitable coloring extends Chernoff-Hoeffding bounds. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, pages 285–296. Springer, 2001.

[53] Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas. The four-colour theorem. *Journal of Combinatorial Theory, Series B*, 70(1):2–44, 1997.

[54] Paul Seymour. Hadwiger's conjecture. In *Open Problems in Mathematics*, pages 417–437. Springer, 2016.

[55] Jan van den Heuvel. The complexity of change. *Surveys in Combinatorics*, 409:127–160, 2013.

[56] Eric Vigoda. Improved bounds for sampling colorings. *Journal of Mathematical Physics*, 41(3):1555–1569, 2000.

[57] Vadim G Vizing. Coloring the vertices of a graph in prescribed colors. *Diskret. Analiz*, 29(3):10, 1976.

[58] Jian-Sheng Wang, Robert H Swendsen, and Roman Kotecký. Antiferromagnetic Potts models. *Physical Review Letters*, 63(2):109, 1989.