Reconfiguration of List Edge-Colorings in a Graph

Takehiro Ito[1], Marcin Kamiński[2], and Erik D. Demaine[3]

[1] Graduate School of Information Sciences, Tohoku University, Aoba-yama 6-6-05, Sendai, 980-8579, Japan.
[2] Department of Computer Science, Université Libre de Bruxelles, CP 212, Bvd. du Triomphe, 1050 Bruxelles, Belgium.
[3] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA.
takehiro@ecei.tohoku.ac.jp, marcin.kaminski@ulb.ac.be, edemaine@mit.edu

Abstract. We study the problem of reconfiguring one list edge-coloring of a graph into another list edge-coloring by changing one edge color at a time, while at all times maintaining a list edge-coloring, given a list of allowed colors for each edge. First we show that this problem is PSPACE-complete, even for planar graphs of maximum degree 3 and just six colors. Then we consider the problem restricted to trees. We show that any list edge-coloring can be transformed into any other under the sufficient condition that the number of allowed colors for each edge is strictly larger than the degrees of both its endpoints. This sufficient condition is best possible in some sense. Our proof yields a polynomial-time algorithm that finds a transformation between two given list edge-colorings of a tree with n vertices using $O(n^2)$ recolor steps. This worst-case bound is tight: we give an infinite family of instances on paths that satisfy our sufficient condition and whose reconfiguration requires $\Omega(n^2)$ recolor steps.

1 Introduction

Reconfiguration problems arise when we wish to find a step-by-step transformation between two feasible solutions of a problem such that all intermediate results are also feasible. Recently, Ito *et al.* [8] proposed a framework of reconfiguration problems, and gave complexity and approximability results for reconfiguration problems derived from several well-known problems, such as INDEPENDENT SET, CLIQUE, MATCHING, etc. In this paper, we study a reconfiguration problem for list edge-colorings of a graph.

An (ordinary) *edge-coloring* of a graph G is an assignment of colors from a color set C to each edge of G so that every two adjacent edges receive different colors. In *list edge-coloring*, each edge e of G has a set $L(e)$ of colors, called the *list* of e. Then, an edge-coloring f of G is called an L*-edge-coloring* of G if $f(e) \in L(e)$ for each edge e, where $f(e)$ denotes the color assigned to e by f. Fig. 1 illustrates three L-edge-colorings of the same graph with the same list L;

Fig. 1. A sequence of L-edge-colorings of a graph.

the color assigned to each edge is surrounded by a box in the list. Clearly, an edge-coloring is an L-edge-coloring for which $L(e)$ is the same color set C for every edge e of G, and hence list edge-coloring is a generalization of edge-coloring.

Suppose now that we are given *two* L-edge-colorings of a graph G (e.g., the leftmost and rightmost ones in Fig. 1), and we are asked whether we can transform one into the other via L-edge-colorings of G such that each differs from the previous one in only one edge color assignment. We call this problem the LIST EDGE-COLORING RECONFIGURATION problem. For the particular instance of Fig. 1, the answer is "yes," as illustrated in Fig. 1, where the edge whose color assignment was changed from the previous one is depicted by a thick line. One can imagine a variety of practical scenarios where an edge-coloring (e.g., representing a feasible schedule) needs to be changed (to use a newly found better solution or to satisfy new side constraints) by individual color changes (preventing the need for any coordination) while maintaining feasibility (so that nothing breaks during the transformation). Reconfiguration problems are also interesting in general because they provide a new perspective and deeper understanding of the solution space and of heuristics that navigate that space.

Reconfiguration problems have been studied extensively in recent literature [1, 3, 4, 6–8], in particular for (ordinary) vertex-colorings. For a positive integer k, a *k-vertex-coloring* of a graph is an assignment of colors from $\{c_1, c_2, \dots, c_k\}$ to each vertex so that every two adjacent vertices receive different colors. Then, the k-VERTEX-COLORING RECONFIGURATION problem is defined analogously. Bonsma and Cereceda [1] proved that k-VERTEX-COLORING RECONFIGURATION is PSPACE-complete for $k \geq 4$, while Cereceda *et al.* [4] proved that k-VERTEX-COLORING RECONFIGURATION is solvable in polynomial time for $1 \leq k \leq 3$. Edge-coloring in a graph G can be reduced to vertex-coloring in the "line graph" of G. However, by this reduction, we can solve only a few instances of LIST EDGE-COLORING RECONFIGURATION; all edges e of G must have the same list $L(e) = C$ of size $|C| \leq 3$ although any edge-coloring of G requires at least $\Delta(G)$ colors, where $\Delta(G)$ is the maximum degree of G. Furthermore, the reduction does not work the other way, so we do not obtain any complexity results.

In this paper, we give three results for LIST EDGE-COLORING RECONFIGURATION. The first is to show that the problem is PSPACE-complete, even for planar graphs of maximum degree 3 and just six colors. The second is to give a sufficient condition for which there exists a transformation between any two L-edge-colorings of a tree. Specifically, for a tree T, we prove that any two L-edge-colorings of T can be transformed into each other if $|L(e)| \geq \max\{d(v), d(w)\} + 1$

for each edge $e = vw$ of T, where $d(v)$ and $d(w)$ are the degrees of the endpoints v and w of e, respectively. Our proof for the sufficient condition yields a polynomial-time algorithm that finds a transformation between given two L-edge-colorings of T via $O(n^2)$ intermediate L-edge-colorings, where n is the number of vertices in T. On the other hand, as the third result, we give an infinite family of instances on paths that satisfy our sufficient condition and whose transformation requires $\Omega(n^2)$ intermediate L-edge-colorings.

Our sufficient condition for trees was motivated by several results on the well-known "list coloring conjecture" [9]: it is conjectured that any graph G has an L-edge-coloring if $|L(e)| \geq \chi'(G)$ for each edge e, where $\chi'(G)$ is the chromatic index of G, that is, the minimum number of colors required for an ordinary edge-coloring of G. This conjecture has not been proved yet, but some results are known for restricted classes of graphs [2, 5, 9]. In particular, Borodin *et al.* [2] proved that any bipartite graph G has an L-edge-coloring if $|L(e)| \geq \max\{d(v), d(w)\}$ for each edge $e = vw$. Because any tree is a bipartite graph, one might think that it would be straightforward to extend their result [2] to our sufficient condition. However, this is not the case, because the focus of reconfiguration problems is not the *existence* (as in the previous work) but the *reachability* between two feasible solutions; there must exist a transformation between any two L-edge-colorings if our sufficient condition holds.

Finally we remark that our sufficient condition is best possible in some sense. Consider a star $K_{1,n-1}$ in which each edge e has the same list $L(e) = C$ of size $|C| = n - 1$. Then, $|L(e)| = \max\{d(v), d(w)\}$ for all edges $e = vw$, and it is easy to see that there is no transformation between any two L-edge-colorings of the star.

2 PSPACE-completeness

Before proving PSPACE-completeness, we introduce some terms and define the problem more formally. In Section 1, we have defined an L-edge-coloring of a graph $G = (V, E)$ with a list L. We say that two L-edge-colorings f and f' of G are *adjacent* if $|\{e \in E : f(e) \neq f'(e)\}| = 1$, that is, f' can be obtained from f by changing the color assignment of a single edge e; the edge e is said to be *recolored* between f and f'. A *reconfiguration sequence* between two L-edge-colorings f_0 and f_t of G is a sequence of L-edge-colorings $f_0, f_1, \ldots, f_t$ of G such that f_{i-1} and f_i are adjacent for $i = 1, 2, \ldots, t$. We also say that two L-edge-colorings f and f' are *connected* if there exists a reconfiguration sequence between f and f'. Clearly, any two adjacent L-edge-colorings are connected. Then, the LIST EDGE-COLORING RECONFIGURATION problem is to determine whether given two L-edge-colorings of a graph G are connected. The *length* of a reconfiguration sequence is the number of L-edge-colorings in the sequence, and hence the length of the reconfiguration sequence in Fig. 1 is 3.

The main result of this section is the following theorem.

Theorem 1. LIST EDGE-COLORING RECONFIGURATION *is PSPACE-complete for planar graphs of maximum degree* 3 *whose lists are chosen from six colors.*

Fig. 2. (a) A configuration of an NCL machine, (b) NCL And vertex, and (c) NCL Or vertex.

In order to prove Theorem 1, we give a reduction from Nondeterministic Constraint Logic (NCL) [7]. An NCL "machine" is specified by a *constraint graph*: an undirected graph together with an assignment of weights from $\{1, 2\}$ to each edge of the graph. A *configuration* of this machine is an orientation (direction) of the edges such that the sum of weights of incoming edges at each vertex is at least 2. Fig. 2(a) illustrates a configuration of an NCL machine, where each weight-2 edge is depicted by a thick line and each weight-1 edge by a thin line. A *move* from one configuration is simply the reversal of a single edge which results in another (feasible) configuration. Given an NCL machine and its two configurations, it is PSPACE-complete to determine whether there exists a sequence of moves which transforms one configuration into the other [7].

In fact, the problem remains PSPACE-complete even for And/Or *constraint graphs*, which consist only of two types of vertices, called "NCL And vertices" and "NCL Or vertices." A vertex of degree 3 is called an *NCL* And *vertex* if its three incident edges have weights 1, 1 and 2. (See Fig. 2(b).) An NCL And vertex behaves as a logical And, in the following sense: the weight-2 edge can be directed outward if and only if both weight-1 edges are directed inward. Note that, however, the weight-2 edge is not necessarily directed outward even when both weight-1 edges are directed inward. A vertex of degree 3 is called an *NCL* Or *vertex* if its three incident edges have weights 2, 2 and 2. (See Fig. 2(c).) An NCL Or vertex behaves as a logical Or: one of the three edges can be directed outward if and only if at least one of the other two edges is directed inward. It should be noted that, although it is natural to think of NCL And and Or vertices as having inputs and outputs, there is nothing enforcing this interpretation; especially for NCL Or vertices, the choice of input and output is entirely arbitrary since NCL Or vertices are symmetric. From now on, we call an And/Or constraint graph simply an NCL machine.

Proof of Theorem 1.

It is easy to see that list edge-coloring reconfiguration can be solved in (most conveniently, nondeterministic [10]) polynomial space. Therefore, in the remainder of this section, we show that the problem is PSPACE-hard by giving a reduction from NCL. This reduction involves constructing two types of gadgets which correspond to NCL And and Or vertices. We call an edge of an NCL machine an *NCL edge*, and say simply an *edge* of a graph for list edge-coloring reconfiguration.

Assume in our reduction that the color c_1 corresponds to "directed inward," and that both colors c_2 and c_3 correspond to "directed outward." Consider an

Fig. 3. (a) an NCL edge uv and (b) its corresponding edges ux and xv of a graph with lists $L(ux) = \{c_1, c_2\}$ and $L(xv) = \{c_1, c_3\}$.

NCL edge uv directed from u to v. (See Fig. 3(a).) Then, the NCL edge is directed outward for u, but is directed inward for v. However, in list edge-coloring, each edge can receive only one color, of course. Therefore, we need to split one NCL edge uv into two edges ux and xv of a graph with lists $L(ux) = \{c_1, c_2\}$ and $L(xv) = \{c_1, c_3\}$, as illustrated in Fig. 3(b). It is easy to see that one of ux and xv can be colored with c_1 if and only if the other edge is colored with either c_2 or c_3. This property represents that an NCL half-edge can be directed inward if and only if the other half is directed outward.

Fig. 4 illustrates three kinds of "And gadgets," each of which corresponds to an NCL And vertex; two edges $u_x x$ and $u_y y$ correspond to two weight-1 NCL half-edges, and the edge $u_z z$ corresponds to a weight-2 NCL half-edge. Since NCL And and Or vertices are connected together into an arbitrary NCL machine, there should be eight kinds of And gadgets according to the choice of lists $\{c_1, c_2\}$ and $\{c_1, c_3\}$ for three edges $u_x x$, $u_y y$ and $u_z z$. However, since the two weight-1 NCL edges are symmetric, it suffices to consider these three kinds: all the three edges have the same list, as in Fig. 4(a); $u_z z$ has a different list from the other two edges, as in Fig. 4(b); and one of $u_x x$ and $u_y y$ has a different list from the other two edges, as in Fig. 4(c).

We denote by $\mathcal{F}(A; c_x, c_y, c_z)$ the set of all L-edge-colorings f of an And gadget A such that $f(u_x x) = c_x$, $f(u_y y) = c_y$ and $f(u_z z) = c_z$. Since a triple (c_x, c_y, c_z) defines the direction of the three corresponding NCL half-edges, all the L-edge-colorings in $\mathcal{F}(A; c_x, c_y, c_z)$ correspond to the same configuration of the NCL And vertex. We now check that the three kinds of And gadgets satisfy the same constraints as an NCL And vertex; we check this property by enumerating all possible L-edge-colorings of the And gadgets. For example, in the And gadget A of Fig. 4(a), $u_z z$ can be colored with c_2 (directed outward) if and only if both $u_x x$ and $u_y y$ are colored with c_1 (directed inward); in other

Fig. 4. Three kinds of And gadgets.

words, $|\mathcal{F}(A; c_x, c_y, c_2)| \geq 1$ if and only if $c_x = c_y = c_1$. In addition, every AND gadget A satisfies the following two properties:

(i) For a triple (c_x, c_y, c_z), if $|\mathcal{F}(A; c_x, c_y, c_z)| \geq 2$, then any two L-edge-colorings f and f' in $\mathcal{F}(A; c_x, c_y, c_z)$ are "internally connected," that is, there exists a reconfiguration sequence between f and f' such that all L-edge-colorings in the sequence belong to $\mathcal{F}(A; c_x, c_y, c_z)$; and

(ii) For every two triples (c_x, c_y, c_z) and (c'_x, c'_y, c'_z) which differ in a single coordinate, if $|\mathcal{F}(A; c_x, c_y, c_z)| \geq 1$ and $|\mathcal{F}(A; c'_x, c'_y, c'_z)| \geq 1$, then there exist two L-edge-colorings f and f' such that f and f' are adjacent, $f \in \mathcal{F}(A; c_x, c_y, c_z)$ and $f' \in \mathcal{F}(A; c'_x, c'_y, c'_z)$.

Then, it is easy to see that the reversal of a single NCL half-edge in an NCL AND vertex can be simulated by a reconfiguration sequence between two L-edge-colorings each of which is chosen arbitrarily from the set $\mathcal{F}(A; c_x, c_y, c_z)$, where (c_x, c_y, c_z) corresponds to the direction of the three NCL half-edges.

Fig. 5 illustrates two kinds of "OR gadgets," each of which corresponds to an NCL OR vertex; three edges $u_x x$, $u_y y$ and $u_z z$ correspond to three weight-2 NCL half-edges. Since an NCL OR vertex is entirely symmetric, it suffices to consider these two kinds: all the three edges have the same list, as in Fig. 5(a); and one edge has a different list from the other two edges, as in Fig. 5(b). Then, similarly as AND gadgets, it is easy to see that both kinds of OR gadgets satisfy the same constraints as an NCL OR vertex, and that the reversal of a single NCL half-edge in an NCL OR vertex can be simulated by a reconfiguration sequence between corresponding two L-edge-colorings.

Given NCL machine, we construct a corresponding graph G with a list L by connecting the vertices x, y and z of AND or OR gadgets. Then, an L-edge-coloring of G corresponds to a configuration of the NCL machine. On the other hand, every configuration of the NCL machine can be mapped to at least one (in general, to exponentially many) L-edge-colorings of G. We can choose an arbitrary one for each of given two configurations, because each AND gadget satisfies Property (i) above and each OR gadget does the counterpart. It is now easy to see that there is a sequence of moves which transforms one configuration into the other if and only if there is a reconfiguration sequence between the two L-edge-colorings of G. Since NCL remains PSPACE-complete even if an NCL machine is planar [7], G is a planar graph of maximum degree 3. Furthermore, each list $L(e)$ is a subset of $\{c_1, c_2, \ldots, c_6\}$. $\square$

Fig. 5. Two kinds of OR gadgets.

3 Trees

Since LIST EDGE-COLORING RECONFIGURATION is PSPACE-complete, it is rather unlikely that the problem can be solved in polynomial time for general graphs. However, in Section 3.1, we give a sufficient condition for which any two L-edge-colorings of a tree T are connected; our sufficient condition can be checked in polynomial time. Moreover, our proof yields a polynomial-time algorithm that finds a reconfiguration sequence of length $O(n^2)$ between given two L-edge-colorings, where n is the number of vertices in T. In Section 3.2, we then give an infinite family of instances on paths that satisfy our sufficient condition and whose reconfiguration sequence requires length $\Omega(n^2)$.

3.1 Sufficient condition and algorithm

The main result of this subsection is the following theorem, whose sufficient condition is best possible in some sense as we mentioned in Section 1.

Theorem 2. *For a tree T with n vertices, any two L-edge-colorings f and f' of T are connected if $|L(e)| \geq \max\{d(v), d(w)\} + 1$ for each edge $e = vw$ of T. Moreover, there is a reconfiguration sequence of length $O(n^2)$ between f and f'.*

Since $\Delta(T) \geq \max\{d(v), d(w)\}$ for all edges vw of a tree T, Theorem 2 immediately implies the following sufficient condition for which any two (ordinary) edge-colorings of T are connected. Note that, for a positive integer k, a *k-edge-coloring* of a tree T is an L-edge-coloring of T for which all edges e have the same list $L(e) = \{c_1, c_2, \ldots, c_k\}$.

Corollary 1. *For a tree T with n vertices, any two k-edge-colorings f and f' of T are connected if $k \geq \Delta(T) + 1$. Moreover, there is a reconfiguration sequence of length $O(n^2)$ between f and f'.*

It is obvious that the sufficient condition of Corollary 1 is also best possible in some sense; consider a star $K_{1,n-1}$ in Section 1.

In the remainder of this subsection, as a proof of Theorem 2, we give a polynomial-time algorithm that finds a reconfiguration sequence of length $O(n^2)$ between given two L-edge-colorings f_0 and f_t of a tree T if our condition holds.

We first give an outline of our algorithm. By the breadth-first search starting from an arbitrary vertex r of degree 1, we order all edges $e_1, e_2, \ldots, e_{n-1}$ of a tree T. At the ith step, $1 \leq i \leq n-1$, the algorithm recolors e_i from the current color to its target color $f_t(e_i)$, as follows. From the current L-edge-coloring f, we first obtain an L-edge-coloring f' of T such that

(i) there is no edge which is adjacent with e_i and is colored with $f_t(e_i)$; and
(ii) there exists a reconfiguration sequence between f and f' in which any of the edges $e_1, e_2, \ldots, e_{i-1}$ is not recolored.

Then, we recolor e_i to $f_t(e_i)$. Therefore, e_i is never recolored after the ith step, while e_i may be recolored before the ith step even if e_i is colored with $f_t(e_i)$. We will show later that every edge of T can be recolored in such a way, and hence

Fig. 6. (a) Subtree T_u in the whole tree T and (b) inside of T_u.

we eventually obtain the target L-edge-coloring f_t. We will also show later that the algorithm recolors each edge e_j with $j \geq i$ at most once in the ith step, and hence we can recolor e_i by recoloring at most $n - i$ edges. Our algorithm thus finds a reconfiguration sequence of total length $\sum_{i=1}^{n-1}(n-i) = O(n^2)$.

Suppose that we are given a tree T with a list L such that

$$|L(e)| \geq \max\{d(v), d(w)\} + 1 \tag{1}$$

for each edge $e = vw$ in $E(T)$. We choose an arbitrary vertex r of degree 1 as the root of T, and regard T as a rooted tree. For a vertex u in $V(T) \setminus \{r\}$, let p be the parent of u in T. We denote by T_u the subtree of T which is rooted at p and is induced by p, u and all descendants of u in T. (See Fig. 6(a).) It should be noted that T_u includes the edge $e_u = pu$, but does not include the other edges incident to p. Therefore, T_u consists of a single edge if u is a leaf of T. We always denote by e_u the edge which joins u and its parent p. For an internal vertex u of T, let $u_1, u_2, \cdots, u_l$ be the children of u ordered arbitrarily, as illustrated in Fig. 6(b). Then, the subtree T_u consists of e_u and the subtrees T_{u_i}, $1 \leq i \leq l$.

For a vertex u of T, we denote by $L_u = L|T_u$ the *restriction* of the list L of T to the subtree T_u, that is, $L_u(e) = L(e)$ for each edge $e \in E(T_u)$. Clearly, $d(v,T) \geq d(v,T_u)$ for each vertex $v \in V(T_u)$, where $d(v,T)$ and $d(v,T_u)$ denote the degrees of v in T and T_u, respectively. Therefore, for each edge $e = vw$ in $E(T_u)$, by Eq. (1) we have

$$\begin{aligned}|L_u(e)| &= |L(e)| \\ &\geq \max\{d(v,T), d(w,T)\} + 1 \\ &\geq \max\{d(v,T_u), d(w,T_u)\} + 1.\end{aligned}$$

The list L_u of T_u thus satisfies Eq. (1). For an L-edge-coloring f of T, we denote by $g = f|T_u$ the *restriction* of f to T_u, that is, g is an L_u-edge-coloring of T_u such that $g(e) = f(e)$ for each edge e in $E(T_u)$. For an L_u-edge-coloring g of T_u, an edge vw of T_u and its endpoint v, we define a subset $C_{\text{av}}(g, vw, v)$ of $L_u(vw)$, as follows:

$$C_{\text{av}}(g, vw, v) = L_u(vw) \setminus \{g(vx) : vx \in E(T_u)\}. \tag{2}$$

Then, $C_{\text{av}}(g, vw, v)$ is the set of all colors in $L_u(vw)$ available on v for vw. Therefore, $C_{\text{av}}(g, vw, v) \cap C_{\text{av}}(g, vw, w)$ is the set of all colors in $L_u(vw)$ available for vw when we wish to recolor vw from g.

Algorithm.

We are now ready to describe our algorithm. Assume that all edges $e_1, e_2, \ldots, e_{n-1}$ of a tree T are ordered by the breadth-first search starting from the root r of T. At the ith step, $1 \le i \le n-1$, the algorithm recolors e_i to its target color $f_t(e_i)$. Consider the ith step of the algorithm, and let f be the current L-edge-coloring of T obtained by $i-1$ steps of the algorithm; let $f = f_0$ for the first step, that is, $e_i = e_1$. Then, we wish to recolor $e_i = pp'$ from $f(e_i)$ to $f_t(e_i)$. There are the following two cases to consider.

Case (a): $f_t(e_i) \in C_{\rm av}(f, e_i, p) \cap C_{\rm av}(f, e_i, p')$

In this case, $f_t(e_i)$ is available for e_i, that is, there is no edge which is adjacent with e_i and is colored with $f_t(e_i)$. We thus simply recolor e_i from $f(e_i)$ to $f_t(e_i)$, and obtain an L-edge-coloring f' of T: for each edge e in $E(T)$,

$$f'(e) = \begin{cases} f(e) & \text{if } e \in E(T) \setminus \{e_i\}; \\ f_t(e_i) & \text{if } e = e_i. \end{cases}$$

Case (b): $f_t(e_i) \notin C_{\rm av}(f, e_i, p) \cap C_{\rm av}(f, e_i, p')$

In this case, there are at most two edges pu and $p'u'$ which are colored with $f_t(e_i)$ and are sharing the endpoints p and p' with e_i, respectively.

If there is an edge pu which is colored with $f_t(e_i)$ and is sharing p with e_i, then we recolor $e_u = pu$ to a different available color c, as follows. By Eqs. (1) and (2) we have

$$\begin{aligned} |C_{\rm av}(f, e_u, p)| &\ge \big|L(e_u)\big| - \big|\{f(px) : px \in E(T)\}\big| \\ &\ge \max\{d(p), d(u)\} + 1 - d(p) \\ &\ge 1. \end{aligned}$$

Therefore, there exists at least one color c in $L(e_u)$ which is available on p for e_u. Clearly, $c \ne f(e_i)$ and $c \ne f_t(e_i)$ since both colors are in $\{f(px) : px \in E(T)\}$. It should be noted that $c \in C_{\rm av}(f, e_u, u)$ does not necessarily hold: c is not always available for e_u. Let $g = f|T_u$ be the restriction of f to T_u. Then, by using the procedure Recolor (which is described in the next page), we recolor e_u from $g(e_u)$ $(= f(e_u))$ to c without recoloring any edge in $E(T) \setminus E(T_u)$. More precisely, we have the following lemma, whose proof is omitted due to the page limitation.

Lemma 1. *Let T be a tree with a list L satisfying Eq.* (1), *and let f be an L-edge-coloring of T. For a vertex u of T, let $L_u = L|T_u$ and $g = f|T_u$ be the restrictions of L and f to the subtree T_u, respectively. Then, for an arbitrary color c in $L_u(e_u) \setminus \{g(e_u)\}$,* Recolor$(T_u, g, c)$ *returns a sequence $\mathcal{RS}$ of L_u-edge-colorings $g_1, g_2, \ldots, g_q$ of T_u which satisfies the following three properties:*

(i) *g and g_1 are adjacent;*
(ii) *g_{k-1} and g_k are adjacent for each k, $2 \le k \le q$; and*
(iii) *$g_k(e_u) = g(e_u)$ for each k, $1 \le k \le q-1$, and $g_q(e_u) = c$.*

Since c has been chosen from $C_{\rm av}(f, e_u, p)$ and g is the restriction of f to T_u, by Property (iii) of Lemma 1 we can easily extend each L_u-edge-coloring g_k of T_u

Procedure 1 RECOLOR(T_u, g, c)

```
1: $\mathcal{RS} \Leftarrow \emptyset$ {$\mathcal{RS}$ does not contain $g$}
2: if $c \in C_{\text{av}}(g, e_u, u)$ then {See also Fig. 6(b)}
3:    {The color $c$ is not assigned to any of the edges $uu_1, uu_2, \ldots, uu_l$}
4:    Recolor $e_u$ from $g(e_u)$ to $c$, and obtain an $L_u$-edge-coloring $g'$ of $T_u$
5:    return {$g'$}
6: else {The color $c$ is assigned to one of the edges $uu_1, uu_2, \ldots, uu_l$}
7:    Let $e_j = uu_j$ be the edge such that $g(e_j) = c$
8:    Choose an arbitrary color $c' \in C_{\text{av}}(g, e_j, u)$
9:    {Recolor $e_j$ to $c'$ via $L_j$-edge-colorings of $T_{u_j}$, where $L_j = L_u|T_{u_j}$}
10:   $\mathcal{RS}' \Leftarrow$ RECOLOR($T_{u_j}, g|T_{u_j}, c'$)
11:   for each $L_j$-edge-coloring $h_k$ in $\mathcal{RS}'$ (in the same order) do
12:      {Extend an $L_j$-edge-coloring $h_k$ of $T_{u_j}$ to an $L_u$-edge-coloring $g_k$ of $T_u$}
13:      Let $g_k(e) = \begin{cases} g(e) & \text{if } e \in E(T_u) \setminus E(T_{u_j}); \\ h_k(e) & \text{if } e \in E(T_{u_j}) \end{cases}$
14:      $\mathcal{RS} \Leftarrow \mathcal{RS} \cup \{g_k\}$
15:   end for
16:   {$e_j$ is now colored with $c'$, and hence $c$ is available for $e_u$}
17:   Recolor $e_u$ from $g(e_u)$ to $c$, and obtain an $L_u$-edge-coloring $g'$ of $T_u$
18:   return $\mathcal{RS} \Leftarrow \mathcal{RS} \cup \{g'\}$
19: end if
```

in $\mathcal{RS}$ to an L-edge-coloring f_k of T, as follows: for each edge e in $E(T)$,

$$f_k(e) = \begin{cases} f(e) & \text{if } e \in E(T) \setminus E(T_u); \\ g_k(e) & \text{if } e \in E(T_u). \end{cases}$$

Clearly, the sequence $f, f_1, f_2, \ldots, f_q$ of L-edge-colorings of T is a reconfiguration sequence which recolors e_u from $f(e_u)$ $\big(= f_t(e_i)\big)$ to c. Moreover, in the reconfiguration sequence, any of the edges in $E(T) \setminus E(T_u)$ is not recolored.

Similarly, if there is an edge $p'u'$ which is colored with $f_t(e_i)$ and is sharing the other endpoint p' with e_i, then we recolor $p'u'$ to a different color which is available on p' for $p'u'$ without recoloring any edge in $E(T) \setminus E(T_{u'})$.

Then, in the current L-edge-coloring of T, $f_t(e_i)$ is available for e_i. Therefore, we can finally recolor e_i from $f(e_i)$ to $f_t(e_i)$.

Proof of Theorem 2.

Remember that all edges $e_1, e_2, \ldots, e_{n-1}$ of a tree T are ordered by the breadth-first search starting from the root r of T, and that the algorithm recolors e_i to its target color $f_t(e_i)$ at the ith step, $1 \le i \le n-1$.

We first show that e_i is never recolored after the ith step of the algorithm, as in the following lemma. (The proof is omitted due to the page limitation.)

Lemma 2. *The algorithm does not recolor any edge e_j with $j < i$ in the ith step.*

Using Lemma 1 we have shown that the algorithm can recolor e_i to $f_t(e_i)$ at the ith step, and hence Lemma 2 implies that the algorithm terminates with the target L-edge-coloring f_t. Therefore, the algorithm finds a reconfiguration sequence between given two L-edge-colorings f_0 and f_t of T if L satisfies Eq. (1).

We now estimate the length of a reconfiguration sequence found by our algorithm. Clearly, the algorithm recolors an edge at most once in each step. Therefore, by Lemma 2, at most $n - i$ edges are recolored in the ith step. The total length of the reconfiguration sequence is thus $\sum_{i=1}^{n-1}(n-i) = O(n^2)$. $\square$

3.2 Length of reconfiguration sequence

We showed in Section 3.1 that any two L-edge-colorings of a tree T are connected via a reconfiguration sequence of length $O(n^2)$ if our sufficient condition holds. In this subsection, we show that this worst-case bound on the length is tight: we give an infinite family of instances on paths that satisfy our sufficient condition and whose reconfiguration sequence requires length $\Omega(n^2)$.

Consider a path $P = \{v_0, v_1, \ldots, v_{3m+1}\}$ of $3m+1$ edges in which every edge e has the same list $L(e) = \{c_1, c_2, c_3\}$. Clearly, the list L satisfies Eq. (1), and hence any two L-edge-colorings of P are connected. We construct two L-edge-colorings f_0 and f_t of P, as follows:

$$f_0(v_i v_{i+1}) = \begin{cases} c_3 & \text{if } i \equiv 0 \mod 3; \\ c_2 & \text{if } i \equiv 1 \mod 3; \\ c_1 & \text{if } i \equiv 2 \mod 3 \end{cases} \tag{3}$$

for each edge $v_i v_{i+1}$, $0 \leq i \leq 3m$, and

$$f_t(v_i v_{i+1}) = \begin{cases} c_3 & \text{if } i \equiv 0 \mod 3; \\ c_1 & \text{if } i \equiv 1 \mod 3; \\ c_2 & \text{if } i \equiv 2 \mod 3 \end{cases} \tag{4}$$

for each edge $v_i v_{i+1}$, $0 \leq i \leq 3m$. Then, we have the following theorem, whose proof is omitted from this extended abstract.

Theorem 3. *For a path P and its two L-edge-colorings f_0 and f_t defined above, every reconfiguration sequence between f_0 and f_t requires length $\Omega(n^2)$, where n is the number of vertices in P.*

4 Concluding Remarks

A reconfiguration sequence can be represented by a sequence of "recolor steps" (e, c), where a pair (e, c) denotes one recolor step which recolors an edge e to some color $c \in L(e)$. Then, the algorithm in Section 3.1 can be easily implemented so that it runs in time $O(n^2)$: we store and compute a sequence of recolor steps (e, c) together with only the current L-edge-coloring of a tree T. On the other hand, Theorem 3 suggests that it is difficult to improve the time-complexity $O(n^2)$ of the algorithm if we wish to find an actual reconfiguration sequence explicitly.

One may expect that our sufficient condition for trees holds also for some larger classes of graphs, such as bipartite graphs, bounded treewidth graphs, etc. However, consider the following even-length cycle, which is bipartite and whose treewidth is 2. For an even integer m, let C be the cycle of $3m$ edges obtained by identifying the edge v_0v_1 with the edge $v_{3m}v_{3m+1}$ of P in Section 3.2, and let f_0 and f_t be L-edge-colorings of C defined similarly as in Eqs. (3) and (4), respectively. Then, we cannot recolor any edge in the cycle, and hence there is no reconfiguration sequence between f_0 and f_t even though $|L(e)| = \max\{d(v), d(w)\} + 1$ holds for each edge $e = vw$.

Acknowledgments

We thank the Algorithms Research Group of Université Libre de Bruxelles, especially Jean Cardinal, Martin Demaine and Raphaël Jungers, for fruitful discussions.

References

1. Bonsma, P., Cereceda, L.: Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 738–749. Springer, Heidelberg (2007)
2. Borodin, O.V., Kostochka, A.V., Woodall, D.R.: List edge and list total colourings of multigraphs. J. Combinatorial Theory, Series B 71, 184–204 (1997)
3. Călinescu, G., Dumitrescu, A., Pach, J.: Reconfigurations in graphs and grids. SIAM J. Discrete Mathematics 22, 124–138 (2008)
4. Cereceda, L., van den Heuvel, J., Johnson, M.: Finding paths between 3-colourings. In: Proc. of IWOCA 2008, pp. 182–196 (2008)
5. Fujino, T., Zhou, X., Nishizeki, T.: List edge-colorings of series-parallel graphs. IEICE Trans. Fundamentals E86-A, 1034–1045 (2003)
6. Gopalan, P., Kolaitis, P.G., Maneva, E.N., Papadimitriou, C.H.: The connectivity of Boolean satisfiability: computational and structural dichotomies. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 346–357. Springer, Heidelberg (2006)
7. Hearn, R.A., Demaine, E.D.: PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. Theoretical Computer Science 343, 72–96 (2005)
8. Ito, T., Demaine, E.D., Harvey, N.J.A., Papadimitriou, C.H., Sideri, M., Uehara, R., Uno, Y.: On the complexity of reconfiguration problems. In: Hong, S., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 28–39. Springer, Heidelberg (2008)
9. Jensen, T.R., Toft, B.: Graph Coloring Problems. Wiley-Interscience, New York (1995)
10. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Computer and System Sciences 4, 177–192 (1970)