Improved Approximation Algorithm for Convex Recoloring of Trees

Reuven Bar-Yehuda* Ido Feldman* Dror Rawitz†

November 21, 2005

Abstract

A pair (T, C) of a tree T and a coloring C is called a *colored tree*. Given a colored tree (T, C) any coloring C' of T is called a *recoloring* of T. Given a weight function on the vertices of the tree the *recoloring distance* of a recoloring is the total weight of recolored vertices. A coloring of a tree is *convex* if for any two vertices u and v that are colored by the same color c, every vertex on the path from u to v is also colored by c. In the *minimum convex recoloring problem* we are given a colored tree and a weight function and our goal is to find a convex recoloring of minimum recoloring distance.

The minimum convex recoloring problem naturally arises in the context of *phylogenetic trees.* Given a set of related species the goal of phylogenetic reconstruction is to construct a tree that would best describe the evolution of this set of species. In this context a convex coloring correspond to *perfect phylogeny.* Since perfect phylogeny is not always possible the next best thing is to find a tree which is as close to convex as possible, or, in other words, a tree with minimum recoloring distance.

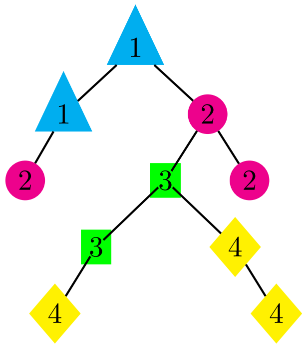
We present a $(2 + \varepsilon)$-approximation algorithm for the minimum convex recoloring problem, whose running time is $O(n^2 + n(1/\varepsilon)^2 4^{1/\varepsilon})$. This result improves the previously known 3-approximation algorithm for this NP-hard problem.

1 Introduction

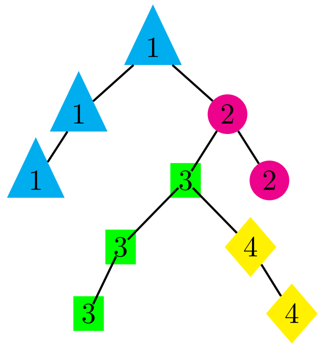
Problem statement and motivation. Given a tree $T = (V, E)$ with n vertices, a *coloring* of the tree is a function $C : V \to \mathcal{C}$, where $\mathcal{C}$ is a set of colors. A pair (T, C) of a tree and a coloring is called a *colored tree.* A coloring C of a tree is *convex* if for every two vertices u and v such that $C(u) = C(v) = c$ the color of every vertex on the path from u to v is also c. That is, a coloring is convex if the set of vertices colored by c induces a (possibly empty) subtree for every color $c \in \mathcal{C}$. Examples of a non-convex coloring and a convex coloring are given in Figure 1. Given a colored tree (T, C) any coloring C' of T is called a *recoloring* of T. A vertex u is *recolored* if $C(v) \neq C'(v)$. Given a non-negative weight function w on the vertices of T the *recoloring distance* of C' is the total weight of recolored vertices. For example, given the coloring in Figure 1(a) and assuming unit weights, the recoloring cost of the coloring in Figure 1(b) is 2. In the *minimum convex recoloring problem* we are given a colored tree (T, C) and a non-negative weight function w and our goal is to find a convex recoloring C' of minimum recoloring distance.

The minimum convex recoloring problem was first introduced by Moran and Snir [12], who showed that this problem arises in the context of *phylogenetic trees.* Given a set of related species

*Department of Computer Science, Technion, Haifa 32000, Israel. E-mail: {reuven,idofeld}@cs.technion.ac.il.
†Caesarea Rothschild Institute, University of Haifa, Haifa 31905, Israel. E-mail: rawitz@cri.haifa.ac.il.



(a) Non-convex coloring



(b) Convex coloring

Figure 1: Transforming a non convex coloring into a convex coloring.

the goal of phylogenetic reconstruction is to construct a tree that would best describe the evolution of this set of species. In such a phylogenetic tree the leaves represent the species, while internal vertices represent extinct species. A *character* is an attribute shared by the entire set of species represented by the tree. Such a character has different *states*, and each species is associated with one of these states. For example, the character may be as simple as the existence of wings, and the states are *wings*, and *no wings*. It is not hard to see that the states of a given character correspond to a set of colors, and that the states associated with the species correspond to a coloring of the tree. A natural biological constraint is that the tree does not contain *reverse* or *convergent* transitions with respect to every character. A reverse transition occurs when some species has a common character state with an old ancestor while its direct ancestor is associated with a different character state. In a convergent transition two species share a character state which is different from the character state of their least common ancestor. The absence of reverse and convergent transitions implies that the path in the tree connecting two species with some character state must contain only species with an identical character state. In other words, a character with respect to which there are no convergent and reverse transitions is a convex coloring of the tree. Hence, our goal is to construct a tree in which every character is a convex coloring. This problem is known as the *perfect phylogeny problem* (see, e.g., [6, 9, 10, 11]).

Since perfect phylogeny is not always possible the next best thing is to find a tree which is as close to convex as possible with respect to each character. However, the meaning of *close to convex* must be defined first. One possible measure of closeness is the *parsimony score* which is the number of mutated edges summed over all characters [15, 14]. Another measure is the *phylogenetic number* [8] which is defined as the maximum number of connected components induced by a single state. In [12] Moran and Snir defined a natural distance from a phylogenetic tree to a convex one—the *recoloring distance*. We note that the parsimony score and the phylogenetic number do not specify a distance to an actual convex coloring of the given tree. Moreover, there are trees with large phylogenetic numbers and parsimony scores that can be made convex only by changing the color of one vertex, while other trees with small phylogenetic numbers that can become convex only by changing the color of a large number of vertices.

For more details about phylogenetic trees and other applications of the minimum convex recoloring problem we refer the reader to [12, 13].

Previous Results. The minimum convex recoloring problem was first defined by Moran and Snir [12]. They showed that the problem is NP-Hard even on strings (trees with two leaves) with unit weights. In addition, they presented dynamic programming based algorithm for computing an optimal convex recoloring of trees. The running time of this algorithm is $O(n \cdot n^* \cdot \Delta^{n^*+2})$, where n^* is the number of colors that violate convexity in the input tree, and Δ is the maximum degree of vertices in the tree. Moran and Snir also show that a modification of their algorithm has a running time of $O(n^4 \cdot n^* \cdot \mathrm{Bell}(n^*))$, where $\mathrm{Bell}(m)$ is the number of unordered partitions of m elements to any number of non empty subsets and is asimptotically smaller than $(\frac{m}{\log m})^m$.

In a followup paper [13] Moran and Snir presented a 3-approximation algorithm based on the *local ratio technique* [5, 1, 3, 2, 4] whose running time is $O(n_c n^2)$, where n_c is the number of colors, and an $O(n_c n)$ time 2-approximation algorithm for strings.

Our Results. We obtain a polynomial time $(2 + \varepsilon)$-approximation algorithm for the minimum convex recoloring problem. Our algorithm depends on an accuracy parameter $k \geq 2$, and consists of two phases. The first phase is a local ratio algorithm in which we manipulate the weights such that the original weighted colored tree is transformed into a weighted colored tree we call k-simple. The approximation ratio of this phase is $2 + \frac{2}{k-1}$ and the running time is $O(n^2)$. In the second phase of the algorithm we use dynamic programming to compute an optimal solution. The running time of this phase is $O(n^2 + nk^2 2^k)$. For example, if we set $k = \log n/2 + 1$ we get a $(2 + 4/\log n)$-approximation algorithm whose time complexity is $O(n^2)$.

Our dynamic programming algorithm for computing an optimal convex recoloring (for general colored trees) is faster than the best previously known algorithm presented in [12], since the running time of our algorithm (in terms of n^* and Δ) is $O(n^2 + n \cdot n^* \cdot \Delta^{n^*+1})$. We also show that the parametrized complexity of this algorithm is $O(n^2 + n \cdot k \cdot 2^k)$.

Finally, we present an elegant local ratio 2-approximation algorithm for strings whose running time is $O(n^2)$. As opposed to the 3-approximation algorithm (for trees) from [13] and our $(2+\frac{2}{k-1})$-approximation algorithm, it relies on the notion of minimality with respect to set inclusion (see [4] for more details on minimality). We believe that this direction may be useful in improving the approximation ratio in the case of trees.

Overview. The remainder of the paper is organized as follows. Section 2 contains most of our definitions and notation. Our dynamic programming algorithm is given in Section 3, and in Section 4 we study its parametrized complexity. In Section 5 we define k-simple trees and analyze the dynamic programming algorithm for the special case of k-simple trees. In Section 6 we present our $(2 + \frac{2}{k-1})$-approximation algorithm. Finally, our 2-approximation algorithm for strings is given in Section 7.

2 Preliminaries

In this section we focus on two main issues. First, we define the notion of convex partial colorings, and show that it is sufficient look for a convex partial recoloring of a given colored tree. Next, we examine the form of an optimal solution.

2.1 Partial Colorings

A *partial coloring* of a tree T is function $C : V \to \mathcal{C} \cup \{\varnothing\}$, where $\varnothing$ stands for *no color*. That is, if $C(v) = \varnothing$ then v is assumed to be *uncolored*. A pair (T, C) of a tree and a partial coloring is called a *partially colored tree*. A partial coloring C is called convex if it can be extended to a (total) convex coloring.

Observation 1. *A convex partial coloring can be completed in $O(n^2)$ time.*

We consider an extended version of the minimum convex recoloring problem in which both the input and output colorings may be partial. That is, we are given a tree $T = (V, E)$, and a partial coloring C of the tree and our goal is to compute a convex partial recoloring C'. We say that a recoloring C' of C *discolors* a vertex u if $C(u) \neq \varnothing$ and $C'(u) \neq C(u)$. That is, C' discolors u if it changes or removes its original color. Given a non-negative weight function w on the vertices of T the *recoloring distance* of C' is the total weight of discolored vertices. (Informally, this means that we pay for removing a color, but not for applying a color.) Hence, we may assume without loss of generality that if $w(u) = 0$ then $C(u) = \varnothing$, and vice versa. Observe that since coloring uncolored vertices cost nothing, turning a convex partial coloring into a convex coloring incurs no cost.

Observation 2. *The weight of an optimal convex partial recoloring is equal to the weight of an optimal convex recoloring.*

Given a partially colored tree (T, C), a vertex set $X \subseteq V$ is called a *cover* if there is a convex partial recoloring C' such that X is the set of vertices that are discolored by C'. For a set of vertices $U \subseteq V$ and a weight function w we define $w(U) = \sum_{u \in U} w(u)$. Hence, the cost of a cover X is defined as $w(X)$, and the *recoloring distance* of a corresponding convex partial coloring C' is $w(C') = w(X)$.

2.2 Form of an Optimal Solution

Given a subset of vertices $U \subseteq V$ we denote the set of colors that are used to color U by $C(U)$, i.e., $C(U) = \{c \in \mathcal{C} : C(u) = c \text{ and } u \in U\}$. Notice that $C(u)$ does not include $\varnothing$. Given a subtree T' of T we denote the set of colors used in T' by $C(T')$, i.e., $C(T') = C(V(T'))$.

Given colored tree (T, C), a *color block* in T is a maximal set of vertices which induces a monochromatic subtree. A c-block is a color block colored by c. (For example, in Figure 1(a) the tree contains two 2-blocks, and one 3-block.) If C is a convex coloring then for every color c there exists only one c-block. Moran and Snir [13] referred to a coloring C' as an *expanding* recoloring of C if in each block of C' at least one vertex v is not recolored, i.e., $C'(v) = C(v)$.

Observation 3 ([13]). *Let (T, C, w) be a weighted colored tree. Then there exists an expanding optimal convex recoloring of the tree.*

It follows that there exists an optimal convex (partial) recoloring C' that uses only colors that were originally used by C. Next, we show that there exists an optimal partial recoloring in which each vertex has a limited choice of colors, and a vertex colored by $\varnothing$ is not located on the path between two c-blocks for some color c.

Given a tree T we denote by $T \setminus v$ the set of subtree obtained when v is removed from T. Given a colored tree (T, C), we say that v *separates* c (with respect to C) if there are at least two subtrees in $T \setminus v$ that contain a vertex u such that $C(u) = c$. The *separation number* $\textsc{sep}_v(c)$ of a vertex v

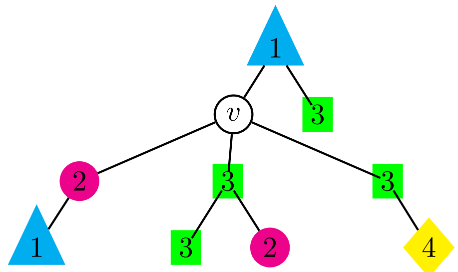


Figure 2: $S(v) = \{1, 2, 3\}$, $\Sigma_v = 3$, and $\Pi_v = \text{SEP}_v(1) \cdot \text{SEP}_v(2) \cdot \text{SEP}_v(3) = 2 \cdot 2 \cdot 3 = 12$.

with respect to a color $c \neq \varnothing$ is defined as the number of subtrees in $T \setminus v$ that contain a vertex u such that $C(u) = c$. Let $S(v)$ be the set of colors that are separated by v, i.e., let $S(v)$ be the set of colors for which the separation number is greater than 1. That is, $S(v) = \{c : \text{SEP}_v(c) > 1\}$. We define $\Sigma_v = |S(v)|$ and $\Pi_v = \prod_{c \in S(v)} \text{SEP}_v(c)$. (If $\Sigma_v = 0$ then $\Pi_v = 1$.) An example is given in Figure 2.

Definition 1. *Given a colored tree (T, C) we define the* color set *of v by $G(v) \triangleq S(v) \cup \{C(v), \varnothing\}$. (Recall that $C(v)$ may be $\varnothing$.) A partial recoloring C' is called* good *if (1) $C'(v) \in G(v)$ for every $v \in V$, and (2) if $C'(v) = \varnothing$, then v does not separate any color c with respect to C'.*

In the next lemma we show that there exists a good optimal convex partial recoloring. Hence, we can concentrate on finding good partial recolorings, and, in particular, it is enough to design an algorithm that computes an optimal good partial recoloring in order to solve the problem.

Lemma 1. *Let (T, C, w) be a weighted colored tree. Then there exists a good optimal convex partial recoloring C'.*

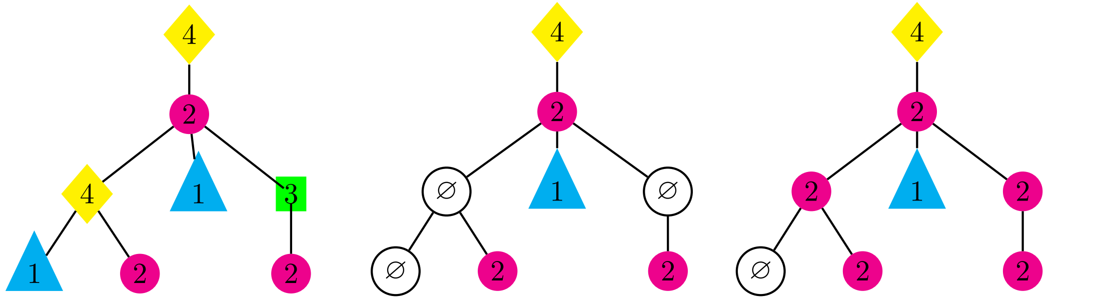
Proof. Let C' be an optimal convex recoloring, and let X be the corresponding cover. We construct a good optimal partial recoloring C'' that correspond to the same cover X. First, we set $C''(v) = C(v)$ for every $v \notin X$. Next, we discolor the vertices in X. Observe that, with respect to C', every $v \in X$ separates at most 1 color (since X is a cover). Hence, for every $v \in X$ that separates a color c, we define $C''(v) = C'(v) = c$, and otherwise, we define $C''(v) = \varnothing$. (See Figure 3 for an illustration). Clearly, C'' is good. Moreover, since we only changed vertices in X, we get that $w(C'') = w(C')$. Also, C'' is convex, since it can be extended to C'. ■

Lemma 2. *Let $v \in V$ be a vertex and $T' \in T \setminus v$ be a subtree. If C' is a good partial recoloring, then $C'(T') \subseteq C(T') \cup \{\varnothing\}$.*

Proof. Consider a vertex u in T'. If $C'(u) = C(u)$ or $C'(u) = \varnothing$ we are done. Otherwise, since $C'(u) \neq C(u)$ we know that $u \in X$. It follows that u separates $C'(u)$, and thus $C'(u) \in C(T')$. ■

3 Dynamic Programming Algorithm

In this section we describe a dynamic programming algorithm for computing an optimal convex partial recoloring whose running time is $O(n^2 + \sum_{v \in V} \Sigma_v \cdot (\deg(v) + \Pi_v))$, where $\deg(v)$ is the degree



(a) Non-convex coloring (b) Convex partial recoloring (c) Good convex partial recoloring

Figure 3: Example of a good convex partial recoloring.

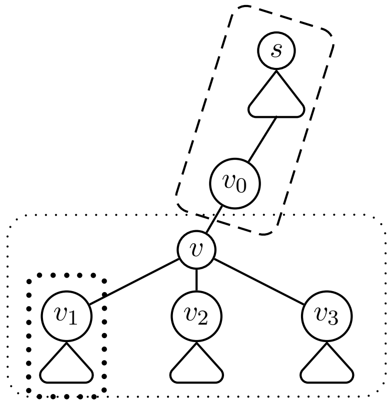


Figure 4: $T_0(v)$ is marked by the dashed line, $T_1(v)$ is marked by the thick dotted line, and $T(v)$ is marked by the thin dotted line.

of the vertex v. This expression becomes polynomial in n and exponential in k for the special case of k-simple trees (defined in the next section).

Throughout this section we treat the input tree T as a rooted tree. This is done by choosing an arbitrary root s. Let $v \in V$ be a vertex with r children (i.e., $\deg(v) = r$). We denote the ith child of v by v_i, and the parent of v by v_0. As before the set of subtrees obtained by the removal of V is denoted by $T \setminus v$. We denote the subtree of the ith child by $T_i(v)$, and by $T(v)$ the tree rooted at v. We also denote by $T_0(v)$ the subtree obtained by removing $T(v)$ from T. (If the tree was unrooted then $T_0(v)$ would be the subtree $T_0(v)$ of the parent v_0.) See Figure 4.

Let C' be a convex good partial recoloring of T, and consider a vertex v that is colored by c. If $c \neq \varnothing$ then C' induces a partition of $\mathcal{C} \setminus \{c\}$. A color $d \neq c$ that is used in $T_i(v)$ cannot be used in $T_j(v)$ where $i \neq j$. If $c = \varnothing$ then C' induces a partition on $\mathcal{C}$. In both cases, v partitions the color set into $r + 1$ mutually disjoint color sets. If $c \neq \varnothing$ then c may be used to color vertices in more than one subtree from $T \setminus v$. Obviously, the use of this color is possible only if the vertices colored by it form a subtree.

Definition 2. *Let (T, C) be a colored tree, let c be a color, and let $v \in V$ be a vertex with r children. We say that $(D_0, \dots, D_r)$ is a* good partition with respect to v and c *if $D_i \subseteq C(T_i(v))$ for $i \in \{0, \dots, r\}$, and $(D_0, \dots, D_r)$ is a partition of $\mathcal{C} \setminus \{c\}$ ($\mathcal{C}$ when $c = \varnothing$).*

$\textsc{good}(v,c)$ *denotes the set of all good partitions with respect to v and c.*

Observation 4. *Let v be a vertex, and let c be a color. Then, $|\textsc{good}(v,c)| \leq \Pi_v$.*

Let v be a vertex, and let c be a color. Also, let C be a coloring that is consitent with some good partition $(D_0,\ldots,D_r)$ with respect to v and c. Then, if v is colored by c, the colors in D_0 cannot be used in $T(v)$. We refer to these colors as *forbidden* with respect to v.

Definition 3. *Let $v \in V$ be a vertex. Define,*

$$\textsc{forb}(v) = \{D : \exists c \in G(v)\ \exists(D_1,\ldots,D_r),\ (D,D_1,\ldots,D_r) \in \textsc{good}(v,c)\}$$

Thus, $D \in \textsc{forb}(v)$ if there is a good partition, where the color set D is used only in $T_0(v)$.

Lemma 3. $|\textsc{forb}(v)| \leq 2^{\Sigma_v}$ *for every $v \in V$.*

Proof. Let v be a vertex with r children, let c be a color, and let i and j be indices such that $0 \leq i < j \leq r$. Observe that if $c \in C(T_i(v))$ and $c \in C(T_j(v))$, then v separates c. Thus, if v does not separate a color c' then either $c' \in D_0$ for every $(D_0,\ldots,D_r) \in \textsc{good}(v,c)$, or $c' \notin D_0$ for every $(D_0,\ldots,D_r) \in \textsc{good}(v,c)$. Therefore, $|\textsc{forb}(v)| \leq 2^{|\{c' : v \text{ separates } c'\}|} = 2^{\Sigma_v}$. ■

We now turn to design a dynamic programming algorithm for computing the optimal good convex partial recoloring of a given colored tree. We construct an optimal solution bottom-up, by storing intermediate values on the vertices of the tree.

Definition 4. *Let v be a vertex in T, let $c \in \mathcal{C} \cup \{\varnothing\}$, and let $D \in \textsc{forb}(v)$.*

- *A good convex recoloring C' of $T(v)$ is a (v,D)*-coloring *if it is a recoloring in which the colors in D are not used to color $T(v)$, i.e., it is a recoloring of $T(v)$ such that $C'(T(v)) \cap D = \emptyset$. $\textsc{opt}(v,D)$ denotes the weight of an optimal (v,D)-coloring.*
- *A good convex recoloring C' of $T(v)$ is a (v,c,D)*-coloring *if it is a recoloring in which the colors in D are not used to color $T(v)$ and v is colored by c, i.e., it is a (v,D)-coloring of $T(v)$ such that $C'(v) = c$. $\textsc{opt}(v,c,D)$ denotes the weight of an optimal (v,c,D)-coloring.*

Observe that, for a tree T with root s, the weight of an optimal recoloring of the whole tree is $\textsc{opt}(s,\emptyset)$.

We compute $\textsc{opt}$ recursively using the following rules:

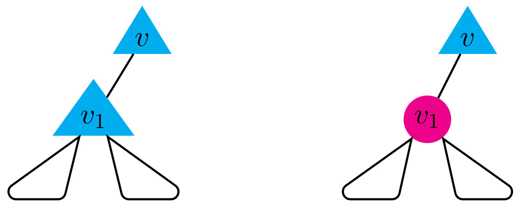
1. $R(v,D) = \min_{c \in G(v) \setminus D} R(v,c,D)$.
2. If $C(v) = c$ then

$$R(v,c,D) = \min_{(D,D_1,\ldots,D_r) \in \text{GOOD}(v,c)} \left\{ \sum_{i=1}^{r} R'(v_i,c,\mathcal{C} \setminus D_i) \right\}$$

otherwise

$$R(v,c,D) = w(v) + \min_{(D,D_1,\ldots,D_r) \in \text{GOOD}(v,c)} \left\{ \sum_{i=1}^{r} R'(v_i,c,\mathcal{C} \setminus D_i) \right\}$$

where $R'(v,c,D) \triangleq \min\{R(v,D \cup \{c\}), R(v,c,D)\}$.



(a) Case 1: $R(v_1, c, D)$ (b) Case 2: $R(v_1, d, D \cup \{d\})$

Figure 5: Combining a recoloring of $T_1(v)$ with a recoloring of v

where v is a vertex with r children, $D \in \textsc{forb}(v)$, and and $c \in G(v) \setminus D$. (Recall that v_i is the ith child of v for every $i \in \{1, \ldots, r\}$.)

In Rule 1 we go through all (v, c, D)-colorings and find the best one that colors the subtree $T(v)$. In Rule 2 we try to glue colorings of the subtrees $T_1(v), \ldots, T_r(v)$ to a coloring of v. Notice that, for every $i \in \{1, \ldots, r\}$, if v is colored by c then either c is not used in $T_i(v)$, or it is used to color v_i. (See Figure 5.)

Theorem 1. *Let v be a vertex with r children, let $D \in \textsc{forb}(v)$ be a set of colors, and let $c \in G(v) \setminus D$. Then, $R(v, c, D) = \textsc{opt}(v, c, D)$, and $R(v, D) = \textsc{opt}(v, D)$.*

Proof. We proof the theorem by induction on the tree. In the induction base v is a leaf, and in this case

$$R(v, c, D) = \begin{cases} w(v) & C(v) \neq c, \\ 0 & \text{otherwise}, \end{cases}$$

and

$$R(v, D) = \begin{cases} w(v) & C(v) \in D \\ 0 & \text{otherwise}, \end{cases}$$

as required.

Next, for the inductive step we assume that $R(v_i, D) = \textsc{opt}(v_i, D)$ and $R(v_i, c, D) = \textsc{opt}(v_i, c, D)$ for every $i \in \{1, \ldots, r\}$. We first prove that $R(v, c, D) \leq \textsc{opt}(v, c, D)$. Let C' be an optimal (v, c, D)-coloring. Let D_i be the set of colors which C' uses in coloring $T_i(v)$. That is, $D_i' = C'(T_i(v)) \setminus \{c\}$ for every $i \in \{0, \ldots, r\}$. Since C' is convex and $C'(v) = c$, we must have that for $i \neq j$, $D_i \cap D_j = \emptyset$. By Lemma 2, it follows that $D_i \subseteq C(T_i(v)) \setminus \{c\}$ for every i. Thus, $\bigcup_{i=0}^{r} D_i \subseteq \bigcup_{i=0}^{r} C'(T_i(v)) \setminus \{c\}$. If $\bigcup_{i=0}^{r} D_i \subsetneq \bigcup_{i=0}^{r} C'(T_i(v)) \setminus \{c\}$ we include each missing color d in an arbitrary set D_i satisfying $d \in C'(T_i(v))$. Hence, $(D, D_1, \ldots, D_r) \in \textsc{good}(v, c)$. Furthermore, the recoloring of $T_i(v)$ that is induced by C' is a good recoloring of $T_i(v)$ for every $i \in \{1, \ldots, r\}$. In addition this recoloring of $T_i(v)$ does not use colors from $\mathcal{C} \setminus (D_i \cup \{c\})$, and if it uses c then $C'(v_i) = c$. Therefore, $R'(v_i, c, \mathcal{C} \setminus D_i) \leq w(T_i(v))$ for every $i \in \{1, \ldots, r\}$, and it follows that $R(v, c, D) \leq \textsc{opt}(v, c, D)$.

Next we show that $\textsc{opt}(v, c, D) \leq R(v, c, D)$. Let $(D, D_1, \ldots, D_r)$ be a good partition which minimizes the RHS of Rule 2. Let C_i' be the recoloring of $T_i(v)$ whose weight is $R'(v_i, c, \mathcal{C} \setminus D_i)$ for $i \in \{1, \ldots, r\}$. We obtain a recoloring of $T(v)$ as follows: $C'(v) = c$ and $C'(u) = C_i'(u)$ for every u that belong to $T_i(v)$. By its construction C' is a good convex recoloring of $T(v)$ that does not use colors from D and such that $C'(v) = c$. Hence, there exists a good convex recoloring C' of $T(v)$ whose weight is $R(v, c, D)$. Therefore, $\text{OPT}(C, c, v) \leq R(v, c, D)$.

It remains to show that:

$$\text{OPT}(v, D) = \min_{c \in G(v) \setminus D} \text{OPT}(v, c, D) = \min_{c \in G(v) \setminus D} R(v, c, D) = R(v, D)$$

and we are done. ∎

The number of entries computed by the dynamic programming algorithm is: $O(\sum_{v \in V} |G(v)| \cdot |\text{FORB}(v)|)$. An additional $O(nc)$ space is needed for storing $G(v)$ for every vertex v. Hence, the space complexity of the algorithm is $O(nc + \sum_{v \in V} \Sigma_v \cdot 2^{\Sigma_v})$.

For each entry of the form $R(v, D)$ the running time is $O(|G(v)|)$. For each entry of the form $R(v, c, D)$ we need to go through all possibilities of good partitions of the form $(D, D_1, \ldots, D_r)$, and for each such good partition we perform $O(\deg(v))$ operations. Observe that for every v and c we actually go through every good partition in $\text{GOOD}(v, c)$ for computing the values of $R(v, c, D)$ for all $D \in \text{FORB}(v)$. Since every good partition is visited exactly once, we invest $O(|\text{GOOD}(v, c)| \cdot \deg(v))$ operations for every pair of vertex v and color c. Hence, by Observation 4, this brings us to $O(\sum_{v \in V} \Sigma_v \cdot \Pi_v \cdot \deg(v))$. An additional $O(n^2)$ is needed to computing $G(v)$ for every v. Hence, the total running time is $O(n^2 + \sum_{v \in V} \Sigma_v \cdot \Pi_v \cdot \deg(v))$.

Note that the computation of $R(v, c, D)$ can be modified also to output a corresponding solution. This can be done by keeping track of which option was taken in both rules. Afterwards we can reconstruct the optimal recoloring in a top down manner.

Next, we explain how to improve the running time of the algorithm. Let v be a vertex v with r children, and c a color. Also, let $\mathcal{D} = (D_0, \ldots, D_r), \mathcal{D}' = (D'_0, \ldots, D'_r) \in \text{GOOD}(v, c)$. We define $\Delta(\mathcal{D}, \mathcal{D}') \triangleq \{i : D_i \neq D'_i\}$. We also define $R(v, c, \mathcal{D}) = \sum_i R'(v_i, c, \mathcal{C} \setminus D_i)$, and $w(v, c, \mathcal{D}, \mathcal{D}') \triangleq R(v, c, \mathcal{D}) - R(v, c, \mathcal{D}')$. If we know the value of $R(v, c, \mathcal{D})$ and $|\Delta(\mathcal{D}, \mathcal{D}')| = 2$ we can compute $R(v, c, \mathcal{D}')$ by using the equation $R(v, c, \mathcal{D}) = R(v, c, \mathcal{D}') + w(v, c, \mathcal{D}, \mathcal{D}')$ in $O(1)$ time.

For a vertex $v \in V$, and a color c we say that an ordering $\mathcal{D}_1, \ldots, \mathcal{D}_{|\text{GOOD}(v,c)|}$ of the set $\text{GOOD}(v, c)$ is a *close order* if $\Delta(\mathcal{D}_i, \mathcal{D}_{i+1}) = 2$ for every i. We now describe how to construct a close order of $\text{GOOD}(v, c)$ for any $v \in V$ with r children and a color c. A good partition $(D_0, ..., D_r) \in \text{GOOD}(v, c)$ can be described by a word $\sigma_1 \cdots \sigma_m$ such that $\sigma_i = j$ if $c_i \in D_j$. (Notice that $\sigma_i = j$ is possible only if $c_i \in C(T_j(v))$.) An order of this set of words in which the hamming distance between every two consecutive words is 1 defines a close order on $\text{GOOD}(v, c)$. Such a close order can be obtained using the description in [7].

We examine the time complexity for computing the value of $R(v, c, D)$ for every vertex v, $D \in \text{FORB}(v)$, and $c \in G(v) \setminus D$. Computing the value corresponding to the first member in the close order of $\text{GOOD}(v, c)$ takes $O(\deg(v))$ time. For any other member the computation takes $O(1)$. Therefore, the time complexity for a vertex v is

$$O(\sum_{c \in G(v)} (\deg(v) + |\text{GOOD}(v, c)|)) = O(\sum_{c \in G(v)} (\deg(v) + \Pi_v)) \ .$$

It follows that the total running time is:

$$O(n^2 + \sum_{v \in V} \Sigma_v \cdot (\deg(v) + \Pi_v)) \ .$$

Let n^* be the number of colors that violate convexity in the input tree. There are Σ_v colors that are separated by a vertex v. At least $\Sigma_v - 1$ of them must be colors that violate convexity in the input tree. Hence, $\Sigma_v - 1 \leq n^*$ for every v. It follows that the running time of the algorithm can be bounded by $O(n^2 + n \cdot n^* \cdot \Delta^{n^*+1})$, where Δ is the maximum degree of vertices in the tree.

4 Fixed Parameter Tractability

As in [12] we consider the parameterized complexity of the unweighted convex recoloring problem. Given a colored tree (T, C) we denote by k the recoloring distance of an optimal convex recoloring. A fixed-parameter algorithm for the convex recoloring problem is an algorithm that computes an optimal convex recoloring and whose running time is $n^{O(1)} \cdot f(k)$, where f is a function that depend on k (and not on n). In this section we show that our dynamic programming algorithm is in fact an $O(n^2 + n \cdot k \cdot 2^k)$ time fixed parameter tractability algorithm.

As we have seen, the total running time of the dynamic programming algorithm is: $O(n^2 + \sum_{v \in V} \Sigma_v \cdot (\deg(v) + \Pi_v))$. In the sequel we show that $\Sigma_v \leq k+1$ and $\Pi_v \leq \deg(v) \cdot 2^k$ for every vertex v. Hence, the running time is bounded by

$$O(n^2 + \sum_{v \in V} (k+1) \cdot (\deg(v) + \deg(v) \cdot 2^k)) = O(n^2 + n \cdot k \cdot 2^k) \ .$$

We show that $\Sigma_v \leq k+1$ for every vertex v. Consider a vertex v. Recall that $S(v)$ containes the Σ_v colors that are seperated by v. That is, $S(v)$ contains the colors that appear in more than one subtree in $T \setminus v$. Let C' be a convex recoloring and let $c \in S(v)$ be such a color. Also assume that $C'(v) = c_0 \neq c$. C' must recolor at least one vetrex colored by c in at least one subtree of $T \setminus v$. Hence, $\Sigma_v - 1 \leq k$.

Next, we show that $\Pi_v \leq \deg(v) \cdot 2^k$ for every vertex v. Using the notation of the previous paragraph, C' must recolor the vertices colored by c in every subtree of $T \setminus v$, but maybe one. Hence, c causes at least $\textsc{sep}_v(c) - 1$ recolorings. This argument holds for every color $c \in S(v) \setminus \{c_0\}$. Therefore,

$$\sum_{c \in S(v) \setminus \{c_0\}} (\textsc{sep}_v(c) - 1) \leq k \ .$$

By the following observation we get that $\prod_{c \in S(v) \setminus \{c_0\}} \textsc{sep}_v(c) \leq 2^k$

Observation 5. *Let $\alpha_1, \ldots, \alpha_m \in \mathbb{N}$ and $\sum_i \alpha_i \leq k$. Then, $\prod_i (\alpha_i + 1) \leq 2^k$.*

It follows that

$$\Pi_v = \prod_{c \in S(v)} \textsc{sep}_v(c) \leq \textsc{sep}_v(c_0) \cdot \prod_{c \in S(v) \setminus \{c_0\}} \textsc{sep}_v(c) \leq \deg(v) \cdot 2^k$$

as required.

5 Simple Trees

In this section we define the notion of a *k-simple* tree. Then, we show that the running time of the dynamic programming algorithm from Section 3 amounts to $O(n^2 + n \cdot k^2 2^k)$ in the special case of k-simple trees.

Let (T, C) be a colored tree and $u \in V$ be a vertex. We say that u is a (t, d)-*separator* if there are t different colors $c_1, \ldots, c_t$ such that for $1 \leq i \leq t$, $\textsc{sep}_u(c_i) \geq d$. Observe that in this case $\Pi_v \geq d^t$. Also, notice that if v is a (t, d)-separator with $r \geq t$ children, then for every $c_i \in \{c_1, \ldots, c_t\}$ there are d vertices $u_1^i, \ldots, u_d^i$ on d different components of $T \setminus v$ such that $C(u_j^i) = c_i$ and $w(u_j^i) > 0$ for every $1 \leq j \leq d$. We refer to such set of $t \cdot d$ vertices as a (t, d)-*separating witness of* v.

Definition 5. *Let* (T, C) *be a colored tree, and define*

$$SEP \triangleq \{(2, k), (3, 4), (4, 3), (k, 2)\}$$

where $k \geq 2$. *We say that the colored tree is* k-simple *if* v *is not a* (t, d)*-separator for every* $v \in V$ *and* $(t, d) \in SEP$.

Observation 6. *Let* (T, C) *be* k*-simple for* $k \geq 2$. *Then,* $\Sigma_v < k$ *for every* $v \in V$.

Consider a vertex v in a k-simple tree. Since $\Sigma_v < k$, v separates at most $k - 1$ colors, and therefore $|G(v)| \leq k + 1$.

Lemma 4. *Let* (T, C) *be* k*-simple for* $k \geq 2$. *Then,* $\Pi_v = O(\deg(v) \cdot k \cdot 2^k)$ *for every* $v \in V$.

Proof. Let $\mathcal{C} = \{c_1, \ldots, c_m\}$, and consider a vertex $v \in V$. Without loss of generality we assume that $\textsc{sep}_v(c_i) \geq \textsc{sep}_v(c_{i+1})$ for every i. We show that the following conditions hold: (1) $\textsc{sep}_v(c_1) \leq \deg(v)$, (2) $\textsc{sep}_v(c_2) \leq k - 1$, (3) $\textsc{sep}_v(c_3) \leq 3$, (4) $\textsc{sep}_v(c_4) \leq 2$, and (5) $\textsc{sep}_v(c_i) \leq 1$ for every $i \geq k$. Hence, $\Pi_v \leq \deg(v) \cdot (k - 1) \cdot 3 \cdot 2^{\Sigma_v - 2} = O(\deg(v) \cdot k \cdot 2^k)$.

First, $\textsc{sep}_u(c_1) \leq |T \setminus u| = \deg(v)$. Also, if $\textsc{sep}_v(c_2) \geq k$ then u is a $(2, k)$-separator; if $\textsc{sep}_v(c_3) \geq 4$ then v is a $(3, 4)$-separator; if $\textsc{sep}_v(c_4) \geq 3$ then v is a $(4, 3)$-separator; and if $\textsc{sep}_v(c_k) \geq 2$ then v is a $(k, 2)$-separator. All in contradiction to the fact that (T, C) is k-simple. ■

Now we analyze the running time of the dynamic programming algorithm for the special case of k-simple trees. Since $\Sigma_v < k$ and $\Pi_v = O(\deg(v) \cdot k2^k)$ for every v, the total running time is

$$O(n^2 + \sum_{v \in V} k \cdot \deg(v) \cdot k2^k) = O(n^2 + n \cdot k^2 2^k) \ .$$

The number of triplets of the form (v, c, D) computed by the dynamic programming algorithm is $O(nc + \sum_{v \in V} k \cdot 2^k) = O(nc + n \cdot k2^k)$.

6 $(2 + \varepsilon)$-approximation Algorithm

In this section we develop an algorithm that given a colored tree and an accuracy parameter k, computes a $(2 + \frac{2}{k-1})$-approximate convex partial recoloring. The running time of the algorithm is $O(n^2 + n \cdot k^2 \cdot 2^k)$.

The algorithm consists of two phases. In the first phase we use the local ratio technique. We manipulate the weights such that the original weighted colored tree is transformed into a k-simple tree. The approximation ratio of this phase is $2 + \frac{2}{k-1}$ and the running time is $O(n^2)$. In the second phase of the algorithm we use our dynamic programming algorithm to compute an optimal solution. We note that if we set $k = \frac{\log n}{2} + 1$ the approximation guarantee is $(2 + \frac{4}{\log n})$ and the time complexity is $O(n^2)$.

The local ratio technique [5, 1, 3, 2] is based on the Local Ratio Theorem, which applies to optimization problems of the following type. The input is a non-negative weight vector $w \in \mathbb{R}^n$ and a set of feasibility constraints $\mathcal{F}$. The problem is to find a solution vector $x \in \mathbb{R}^n$ that minimizes (or maximizes) the inner product $w \cdot x$ subject to the constraints $\mathcal{F}$.

Theorem 2 (Local Ratio [3])**.** *Let $\mathcal{F}$ be a set of constraints and let w, w_1, and w_2 be weight vectors such that $w = w_1 + w_2$. Then, if x is r-approximate both with respect to $(\mathcal{F}, w_1)$ and with respect to $(\mathcal{F}, w_2)$, for some r, then x is also an r-approximate solution with respect to $(\mathcal{F}, w)$.*

Algorithm **CR-LR** is our local ratio approximation algorithm. It uses our dynamic programming algorithm which is referred to as Algorithm **CR-DP**. Apart from a weighted colored tree, the input to our algorithm includes an accuracy parameter k. As we shall see this algorithm computes $(2 + \frac{2}{k-1})$-approximate solutions, and its running time is polynomial is n and exponential in k.

Algorithm 1 : **CR-LR**(T, C, w, k)

if (T, C) is k-simple **then**
  Return **CR-DP**(T, w)
else
  Find $v \in V$ and $(t, d) \in \text{SEP}$ such that v is a (t, d)-separator
  Find a (t, d)-separating witness U of v
  Let $\varepsilon = \min_{u \in U} w(u)$
  Define $w_1(u) = \begin{cases} \varepsilon & u \in U, \\ 0 & \text{otherwise} \end{cases}$
  Return **CR-LR**$(T, w - w_1)$
end if

We first analyze the time complexity of the algorithm. Observe that given a vertex v, checking whether v is a (t, d)-separator, where $(t, d) \in \text{SEP}$, can be done in linear time. Since in each weight subtraction the weight of at least one vertex becomes zero, after no more than n subtraction there are no (t, d)-separators left in the given tree. Hence, the local ratio phase of the algorithm can be implemented to run in $O(n^2)$ time. Moreover, since the input to the dynamic programming algorithm is a k-simple tree, the total running time is $O(n^2 + n \cdot k^2 \cdot 2^k)$.

The computed solution is feasible since we use our dynamic programming algorithm the solution returned is feasible. It remains to show that the solution returned is $(2 + \frac{2}{k-1})$-approximate. We prove this by induction on the recursion. At the recursive base the solution returned is optimal, since it is computed by the dynamic programming algorithm. For the inductive step, we assume that the solution returned by the recursive call is $(2 + \frac{2}{k-1})$-approximate with respect to $w - w_1$. We show that every solution is $(2 + \frac{2}{k-1})$-approximate with respect to w_1. Thus, by the Local Ratio Theorem the solution is $(2 + \frac{2}{k-1})$-approximate with respect to w as well.

Lemma 5. *Every convex partial recoloring is $(2 + \frac{2}{k-1})$-approximate with respect to w_1.*

Proof. Obviously, there are four possible types of w_1 that correspond to the four members of SEP. Let v be a (t, d)-separator, where $(t, d) \in \text{SEP}$, and let U be the (t, d)-separating witness. Consider a cover X that corresponds to a partial convex recoloring C'. It is not hard to see that $w_1(X) \leq \varepsilon \cdot td$.

On the other hand, in a convex partial recoloring, for every $v \in V$ there is at most one color $c \in \mathcal{C}$ such that v separates c. Therefore, at least $(t-1)(d-1)$ vertices in U must be recolored.

Thus, $w(X) \geq \varepsilon \cdot (t-1)(d-1)$. It follows that the weight of every cover X is within a factor of $\frac{td}{(t-1)(d-1)}$ from the optimum with respect to w_1. The lemma follows, since for $(2,k)$ and $(k,2)$ we get $\frac{td}{(t-1)(d-1)} = \frac{2k}{k-1} = 2 + \frac{2}{k-1}$ and for $(3,4)$ and $(4,3)$ we get $\frac{td}{(t-1)(d-1)} = 2$. ■

7 2-approximation Algorithm for Strings

In this section we present a local ratio 2-approximation algorithm for strings. We start the section with some simple properties of partially colored strings.

Let (S, C) be a partially colored string. Given a vertex u, we denote by S_u the string S after the removal of u from S.

Observation 7. *Let (S, C) be a partially colored string, and let u be a zero weight vertex. If X is a feasible cover with respect to S_u, then $X \cup \{u\}$ is a feasible cover with respect to S. Furthermore, the optimum with respect to S is equal to the optimum with respect to S_u.*

Let (S, C) be a partially colored string. Two vertices u and v are called *friendly* if $(u, v) \in E$ and $C(u) = C(v)$.

Observation 8. *Let (S, C) be a partially colored string, and let u and v be friendly vertices. If X is a cover then either $u, v \in X$, or $X \setminus \{u, v\}$ is a cover.*

It follows that given a pair of friendly vetices u and v we may assume that either both u and v are both in the cover or both are not in the cover. Hence, we can *fuse* u and v into a new single vertex y whose color is $C(y) = C(u)$ and weight is $w(y) = w(u) + w(v)$. We denote by S_{uv} and w_{uv} the resulting string and weight function, respectively. Now we can compute for a cover for S by computing a cover X for S_{uv} and replacing y with u and v, if $y \in X$. Henceforth, we abuse notation by treating a cover of S_{uv} as a cover for S.

Given a partially colored string (S, C) we assume that the string S it is horizontally aligned. Consider a colored string that does not contain friendly pairs. Let u_0 be the leftmost vertex in S. We refer to u_0 as the *pivot* of the colored string. Denote by $u_1, \ldots, u_\ell$ be the vertices in S that are colored by $C(u_0)$, where u_i is to the left of u_j if $i < j$. Let v_i be the vertex immediately to the left of u_i. We refer to the set $\{u_0, v_1, u_1, \ldots, v_\ell, u_\ell\}$ as the *comb* of u_0 and denote it by $\textsc{comb}(u_0)$. Note that if u_0 is the only vertex colored by $C(u_0)$ then $\ell = 0$ and $\textsc{comb}(u_0) = \{u_0\}$.

Algorithm **String** is our local-ratio 2-approximation algorithm.

We first analyze the time complexity of the algorithm. Finding a zero weight vertex (Line 2), a friendly pair (Line 3), or the pivot (Line 4) can all be done in linear time. The consturction of w_1 can also be done in linear time as well. Furthermore, in every recursive call the length of the string decreases or the weight of one of the vertices becomes zero. It follows that there are no more than $O(n)$ recursive calls, and therefore the total running time is $O(n^2)$.

It remains to show that the returned cover is 2-approximate. We prove this by induction on the number of recursive calls. At the recursive base the solution returned is optimal. For the inductive step, we assume that the solution returned by the recursive call is 2-approximate. We denote that solution returned by the recursive call by X' and the solution returned by the currect call by X. There are three cases. If the call is made in Line 2, $X = \{u\} \cup X'$ is 2-approximte with respect to S and C due to Observation 7. If the call is made in Line 3, then by Observation 8 $X = X'$ is 2-approximate. Finally, if the call is made in Line 7, by the inductive hypothesis X' is 2-approximate with respect to $w - w_1$. Since $w(u_0) = 0$, X is also 2-approximate with respect

Algorithm 2 : **String**(S, C, w)

1: **if** S is empty **then** return $\emptyset$
2: **if** there exists a zero weight vetrex u **then** return $\{u\} \cup$ **String**(S_u, C, w)
3: **if** there exists a friendly pair u, v **then** return **String**(S_{uv}, C, w_{uv})
4: Let u_0 be the pivot of (S, C)
5: Let $\varepsilon = \min_{u \in \text{COMB}(u_0)} w(u)$.
6: Define $w_1(u) = \begin{cases} \varepsilon & u \in \text{COMB}(u_0), \\ 0 & \text{otherwise} \end{cases}$
7: $X' \leftarrow$ **String**$(S, C, w - w_1)$
8: **if** $X' \setminus \{u_0\}$ is a cover **then**
9:    return $X' \setminus \{u_0\}$
10: **else**
11:    return X'
12: **end if**

to $w - w_1$. Next, we show that X is 2-approximate with respect to w_1. Thus, by the Local Ratio Theorem the solution is 2-approximate with respect to w as well.

We say that a cover X is u_0-*minimal* if either $u_0 \notin X$ or $u_0 \in X$, but $X \setminus \{u_0\}$ is not a cover. Lines 9-13 make sure that X is u_0-minimal. We prove that any u_0-minimal cover is 2-approximate with respect to w_1. First, observe that, for every i, if $v_i \notin Y$ then either $u_i \in Y$ or $u_{i-1} \in Y$. Hence, $w(Y) \geq \ell \cdot \varepsilon$. On the other hand, is Y is u_0-minimal $|Y \cap \text{COMB}(u_0)| \leq 2\ell$, which means that $w(Y) \leq 2\ell \cdot \varepsilon$.

Acknowledgments

We thank the anonymous referees for their helpful comments and suggestions. We also thank Benny Chor for asking about the parametrized complexity of the dynamic programming algorithm.

References

[1] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.

[2] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

[3] R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *Algorithmica*, 27(2):131–144, 2000.

[4] R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz. Local ratio: a unified framework for approximation algorithms. *ACM Computing Surveys*, 36(4):422–463, 2004.

[5] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.

[6] H. L. Bodlaender, M. R. Fellows, and T. J. Warnow. Two strikes against perfect phylogeny. In *19th International Colloquium on Automata, Languages, and Programming*, volume 623 of *LNCS*, pages 273–283. Springer, 1992.

[7] M. Er. On generating the n-ary reflected gray codes. *IEEE Transactions on Computers*, 33:739–741, 1984.

[8] L. A. Goldberg, P. W. Goldberg, C. A. Phillips, E. Sweedyk, and T. Warnow. Minimizing phylogenetic number to find good evolutionary trees. *Discrete Applied Mathematics*, 71(1-3):111–136, 1996.

[9] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991.

[10] S. Kannan and T. Warnow. Inferring evolutionary history from DNA sequences. *SIAM Journal on Computing*, 23(4):713–737, 1994.

[11] S. Kannan and T. Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing*, 26(6):1749–1763, 1997.

[12] S. Moran and S. Snir. Convex recoloring of trees and strings: definitions, hardness results, and algorithms. In *9th Workshop on Algorithms and Data Structures*, 2005. To appear.

[13] S. Moran and S. Snir. Efficient approximation of convex recoloring. In *8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, 2005. To appear.

[14] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28:35–42, 1975.

[15] C. Semple and M. Steel. *Phylogenetics*, volume 22 of *Mathematics and its Applications series*. Oxford University Press, 2003.