On Two Measures of Distance Between Fully-Labelled Trees

Giulia Bernardini

University of Milano - Bicocca, Milan, Italy
giulia.bernardini@unimib.it

Paola Bonizzoni

University of Milano - Bicocca, Milan, Italy
paola.bonizzoni@unimib.it

Paweł Gawrychowski

Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Abstract

The last decade brought a significant increase in the amount of data and a variety of new inference methods for reconstructing the detailed evolutionary history of various cancers. This brings the need of designing efficient procedures for comparing rooted trees representing the evolution of mutations in tumor phylogenies. Bernardini et al. [CPM 2019] recently introduced a notion of the rearrangement distance for fully-labelled trees motivated by this necessity. This notion originates from two operations: one that permutes the labels of the nodes, the other that affects the topology of the tree. Each operation alone defines a distance that can be computed in polynomial time, while the actual rearrangement distance, that combines the two, was proven to be NP-hard.

We answer two open question left unanswered by the previous work. First, what is the complexity of computing the permutation distance? Second, is there a constant-factor approximation algorithm for estimating the rearrangement distance between two arbitrary trees? We answer the first one by showing, via a two-way reduction, that calculating the permutation distance between two trees on n nodes is equivalent, up to polylogarithmic factors, to finding the largest cardinality matching in a sparse bipartite graph. In particular, by plugging in the algorithm of Liu and Sidford [ArXiv 2020], we obtain an $\tilde{\mathcal{O}}(n^{4/3+o(1)})$ time algorithm for computing the permutation distance between two trees on n nodes. Then we answer the second question positively, and design a linear-time constant-factor approximation algorithm that does not need any assumption on the trees.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Approximation algorithms analysis; Theory of computation → Problems, reductions and completeness

Keywords and phrases Tree distance, Cancer progression, Approximation algorithms, Fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.CPM.2020.6

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 872539.
Giulia Bernardini: Partially supported by a research internship at CWI.

1 Introduction

Phylogenetic trees represent a plausible evolutionary relationship between the most disparate objects: natural languages in linguistics [22, 36, 44], ancient manuscripts in archaeology [12], genes and species in biology [25, 26]. The leaves of such trees are labelled by the entities they represent, while the internal nodes are unlabelled and stand for unknown or extinct items. A great wealth of methods to infer phylogenies have been developed over the

© Giulia Bernardini, Paola Bonizzoni, and Paweł Gawrychowski;
licensed under Creative Commons License CC-BY

decades [19,42], together with various techniques to compare the output of different algorithms, e.g., by building a consensus tree that captures the similarity between a set of conflicting trees [11, 20, 27, 28] or by defining a metric between two trees [10, 16–18, 39, 40].

Fully-labelled trees, in opposition to classical phylogenies, may model an evolutionary history where the internal nodes, just like the leaves, correspond to extant entities. An important phenomenon that fits this model well is cancer progression [23, 37]. With the increasing amount of data and algorithms becoming available for inferring cancer evolution [6, 7, 29, 34, 46], there is a pressing need of methods to provide a meaningful comparison among the trees produced by different approaches. Besides the well-studied edit distance for fully-labelled trees [35, 38, 43, 47], a few recent papers proposed ad-hoc metrics for tumor phylogenies [13, 15, 21, 31]. Taking inspiration from the existing literature [4, 8, 14, 42] on phylogeny rearrangement, the study of an operational notion of distance for rearranging a fully-labelled tree is of great interest, and there are still many unexplored questions to be answered.

Following this line of research, we revisit the two notions of operational distance between fully-labelled trees recently introduced by Bernardini et al. [5]. We consider rooted trees on n nodes labelled with distinct labels from $[n] = \{1, 2, \ldots, n\}$, and identify nodes with their labels. We recall the following two basic operations on such trees:

- **link-and-cut operation**: given u, v and w such that v is a child of u and w is not a descendant of v, the link-and-cut operation $v \,|\, u \to w$ consists of two suboperations: cut the edge (v, u) and add the edge (v, w), effectively switching the parent of v from u to w.
- **permutation operation**: apply some permutation $\pi : [n] \to [n]$ to the nodes. If a node u was a child of v before the operation, then after the operation $\pi(u)$ is a child of $\pi(v)$.

The size $|\pi|$ of a permutation is the number of elements x s.t. $\pi(x) \neq x$. Two trees T_1 and T_2 are isomorphic if and only if one can reorder the children of every node so as to make the trees identical after disregarding the labels. The *permutation distance* $d_\pi(T_1, T_2)$ between two isomorphic trees is the smallest size $|\pi|$ of a permutation π that transforms T_1 into T_2. Bernardini et al. [5] designed a cubic time algorithm for computing the permutation distance.

The size of a sequence of link-and-cut and permutation operations is the sum of the number of link-and-cut operations and the total size of all permutations. The *rearrangement distance* $d(T_1, T_2)$ between two (not necessarily isomorphic) trees with identical roots is the smallest size of any sequence of link-and-cut and permutation operations that, without permuting the root, transform T_1 into T_2. Bernardini et al. [5] proved that computing the rearrangement distance is NP-hard, but for binary trees there exists a polynomial time 4-approximation algorithm.

We consider two natural open questions. First, what is the complexity of computing the permutation distance? Second, is there a constant-factor approximation algorithm for estimating the rearrangement distance between two arbitrary trees? For computing the permutation distance, in Section 3 we connect the complexity to that of calculating the largest cardinality matching in a sparse bipartite graph. By designing two-way reductions we show that these problems are equivalent, up to polylogarithmic factors. Due to the recent progress in the area of fine-grained complexity we now know, for many problems that can be solved in polynomial time, what is essentially the best possible exponent in the running time, conditioned on some plausible but yet unproven hypothesis [45]. For max-flow, and more specifically maximum matching, this is not the case yet, although we do have some understanding of the complexity of the related problem of computing the max-flow between all pairs of nodes [1, 2, 32]. So, even though our reductions don't tell us what is the best possible exponent in the running time, they do imply that it is the same

as for maximum matching in a sparse bipartite graph. In particular, by plugging in the asymptotically fastest known algorithm [33], we obtain an $\tilde{\mathcal{O}}(n^{4/3+o(1)})$ time algorithm for computing the permutation distance between two trees on n nodes. The main technical novelty in our reduction from permutation distance is that, even though the natural approach would result in multiple instances of weighted maximum bipartite matching, we manage to keep the graphs unweighted.

For the rearrangement distance, in Section 4 we design a linear-time constant-factor approximation algorithm that does not assume that the trees are binary. The algorithm consists of multiple phases, each of them introducing more and more structure into the currently considered instance, while making sure that we don't pay more than the optimal distance times some constant. To connect the number of steps used in every phase with the optimal distance, we introduce a new combinatorial object that can be used to lower bound the latter inspired by the well-known algorithm for computing the majority [9].

2 Preliminaries

Let $[n] = \{1, 2, \ldots, n\}$. We consider rooted trees and forests on nodes labelled with distinct labels from $[n]$, and identify nodes with their labels. The parent of u in F is denoted $p_F(u)$, and we use the convention that $p_F(u) = \bot$ when u is a root in F. $F|u$ denotes the subtree of F rooted at u, $\mathsf{children}_F(u)$ stands for the set of children of a node u in F, and $\mathsf{level}_F(u)$ is the level of u in F (with the roots being on level 0).

Two trees T_1 and T_2 are isomorphic, denoted $T_1 \equiv T_2$, if and only if there exists a bijection μ between their nodes such that, for every $u \in [n]$ with $p_{T_1}(u) \neq \bot$, it holds that $\mu(p_{T_1}(u)) = p_{T_2}(\mu(u))$, implying in particular that μ maps the root of T_1 to the root of T_2. Let $\mathcal{I}(T_1, T_2)$ denote the set of all such bijections μ. Given two isomorphic trees T_1 and T_2, we seek a permutation π with the smallest size that transforms T_1 into T_2. This is equivalent to finding $\mu \in \mathcal{I}(T_1, T_2)$ that maximises the number of conserved nodes $\mathsf{conserved}(\mu) = \{u : u = \mu(u)\}$, as these two values sum up to n.

When working on the rearrangement distance, for ease of presentation, instead of the link-and-cut operation we will work with the cut operation defined as follows:

- **cut operation**: let u, v be two nodes such that v is a child of u. The cut operation $(v \dagger u)$ removes the edge (v, u), effectively making v a root.

The size of a sequence of cut and permutation operations is defined similarly as for a sequence of link-and-cut and permutation operations. Since a permutation operation is essentially just renaming the nodes, we can assume that all permutation operations precede all link-and-cut (or cut) operations, or vice versa. Furthermore, multiple consecutive permutation operations can be replaced by a single permutation operation without increasing the total size.

This leads to the notion of rearrangement distance between two forests F_1 and F_2. We write $F_1 \sim F_2$ to denote that, for every $u \in [n]$, at least one of the following three conditions holds: (i) $p_{F_1}(u) = p_{F_2}(u)$, (ii) $p_{F_1}(u) = \bot$, or (iii) $p_{F_2}(u) = \bot$. The rearrangement distance $\tilde{d}(F_1, F_2)$ is the smallest size of any sequence of cut and permutation operations that transforms F_1 into F_1' such that $F_1' \sim F_2$. This is the same as the smallest size of any sequence of cut and permutation operations that transforms F_2 into F_2' such that $F_1 \sim F_2'$, as both sizes are equal to the minimum over all permutations π that fix the original root of the following expression

$$|\{u : \pi(u) \neq u\}| + |\{u : p_{F_1}(u) \neq p_{F_2}(\pi(u)) \ \wedge \ p_{F_1}(u) \neq \bot \ \wedge \ p_{F_2}(\pi(u)) \neq \bot\}|.$$

Consequently, $\tilde{d}$ defines a metric. The original notion of rearrangement distance d between two trees was similarly defined as the smallest size of any sequence of link-and-cut and permutation operations that transforms T_1 into T_2, under the additional assumption that

the roots of T_1 and T_2 are identical (so $d(T_1, T_2)$ is well-defined) and cannot participate in any permutation operation [5]. In Section 4 we connect $d(T_1, T_2)$ and $\tilde{d}(T_1, T_2)$, and then work with the latter.

A matching in a bipartite graph is a subset of edges with no two edges meeting at the same vertex. A maximum matching in an unweighted bipartite graph is a matching of maximum cardinality, whereas a maximum weight matching in a weighted bipartite graph is a matching in which the sum of weights is maximised. Given an unweighted bipartite graph with m edges, the well-known algorithm by Hopcroft and Karp [24] finds a maximum matching in $\mathcal{O}(m^{1.5})$ time. This has been recently improved by Liu and Sidford to $\tilde{\mathcal{O}}(m^{4/3+o(1)})$ [33].

A *heavy path decomposition* of a tree T is obtained by selecting, for every non-leaf node $u \in T$, its *heavy child* v such that $T|v$ is the largest: there will be some subtlety in how to resolve a tie in this definition that will be explained in detail later. This procedure decomposes the nodes of T into node-disjoint paths called *heavy paths*. Each heavy path p starts at some node, called its *head*, and ends at a leaf: $\mathsf{head}_T(u)$ denotes the head of the heavy path containing a node u in T. An important property of such a decomposition is that the number of distinct heavy paths above any leaf (that is, intersecting the path from a leaf to the root) is only logarithmic in the size of T [41].

3 A Fast Algorithm for the Permutation Distance

Our aim is to find $\mu \in \mathcal{I}(T_1, T_2)$ that maximises $\mathsf{conserved}(\mu)$, that is $\gamma(T_1, T_2) = \max\{\mathsf{conserved}(\mu) : \mu \in \mathcal{I}(T_1, T_2)\}$. To make the notation less cluttered, we define $\gamma(x, y) = \gamma(T_1|x, T_2|y)$. Let us start by describing a simple polynomial time algorithm which follows the construction of [5]. We will then show how to improve it to obtain a faster algorithm that uses unweighted bipartite maximum matching. Finally, we will show a reduction from bipartite maximum matching to computing the permutation distance, establishing that these two problems are in fact equivalent, up to polylogarithmic factors.

3.1 Polynomial Time Algorithm

We first run the folklore linear-time algorithm of [3] for determining if two rooted trees are isomorphic. Recall that this algorithm assigns a number from $\{1, 2, \ldots, 2n\}$ to every node of T_1 and T_2 so that the subtrees rooted at two nodes are isomorphic if and only if their numbers are equal. The high-level idea is then to consider a weighted bipartite graph $G(u, v)$ for each $u, v \in [n]$ such that $\mathsf{level}_{T_1}(u) = \mathsf{level}_{T_2}(v)$ and $T_1|u \equiv T_2|v$. The vertices of $G(u, v)$ are $\mathsf{children}_{T_1}(u)$ and $\mathsf{children}_{T_2}(v)$, and there is an edge of weight $\gamma(u', v')$ between $u' \in \mathsf{children}_{T_1}(u)$ and $v' \in \mathsf{children}_{T_2}(v)$ if and only if $T_1|u' \equiv T_2|v'$ and $\gamma(u', v') > 0$. We call such graphs the *distance graphs* for T_1 and T_2 and denote them collectively by $\mathcal{G}(T_1, T_2)$.

$\gamma(u, v)$ is computed as follows, with $\mathcal{M}(G(u, v))$ denoting the weight of a (not necessarily perfect) maximum weight matching in $G(u, v)$, $\Gamma(u, v) = 1$ if $u = v$ and $\Gamma(u, v) = 0$ otherwise.

$$\gamma(u, v) = \begin{cases} \mathcal{M}(G(u, v)) + \Gamma(u, v) & \text{if } T_1|u \equiv T_2|v, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The overall number of edges created in all graphs is $\mathcal{O}(n^2)$. Indeed, for each $u \in [n]$ such that $\mathsf{level}_{T_1}(u) = \mathsf{level}_{T_2}(u)$ and $T_1|u \equiv T_2|u$, and for each pair of ancestors z of u in T_1 and w of u in T_2 such that $\mathsf{level}_{T_1}(z) = \mathsf{level}_{T_2}(w)$ and $T_1|z \equiv T_2|w$, we possibly add an edge (z, w) to the graph $G(p_{T_1}(z), p_{T_2}(w))$. Since there are up to n pairs of ancestors on the same level for each label, and the labels are n, there are $\mathcal{O}(n^2)$ edges overall.

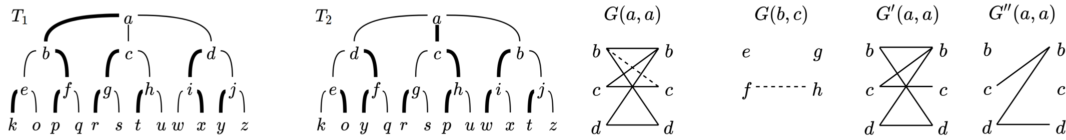


Figure 1 $G(a,a)$ (type 1), $G'(a,a)$, $G''(a,a)$ and $G(b,c)$ (type 3) for T_1 and T_2 on the left. The special edge in each graph is dashed.

We then start from the deepest level in both trees, and we move up level by level towards the roots in both trees simultaneously. For each level k, we consider all pairs of isomorphic subtrees rooted at level k, build the corresponding distance graphs, and use Equation (1) to weigh the edges. After having reached the roots, we return the value of $\gamma(T_1, T_2)$. The correctness of the algorithm follows directly from Lemma 13 of [5], stating that the permutation distance is equal to the minimum number of labels that are not conserved by any isomorphic mapping, i.e., $d_\pi(T_1, T_2) = n - \gamma(T_1, T_2)$. The running time is polynomial if we plug in any polynomial-time maximum weight matching algorithm.

In the next subsection we show how to obtain a better running time by constructing a different version of distance graphs, so that the total weight of their edges will be subquadratic, and replacing maximum weight matching with maximum matching.

3.2 Reduction to Bipartite Maximum Matching

We start by finding a heavy path decomposition of T_1 and T_2, with some extra care in resolving a tie if there are multiple children with subtrees of the same size, as follows. Recall that we already know which subtrees of T_1 and T_2 are isomorphic, as the algorithm of [3] assigns the same number from $\{1, 2, \ldots, 2n\}$ to nodes of T_1 and T_2 with isomorphic subtrees. For every $u, v \in [n]$ such that $T_1|u \equiv T_2|v$, we would like the heavy child u' of u in T_1 and v' of v in T_2 to be such that $T_1|u' \equiv T_2|v'$. This can be implemented in linear time: it suffices to group the nodes with isomorphic subtrees together, and then make the choice just once for every such group.

Consider a graph $G(u, v)$ for some $u, v \in [n]$: the edge corresponding to the heavy child u' of u in T_1 and the heavy child v' of v in T_2 is called *special* (note that this edge might not exist). The key observation is that the properties of heavy path decomposition allow us to bound the total weight of non-special edges by $\mathcal{O}(n \log n)$.

▶ **Lemma 1.** *The total weight of all non-special edges in $\mathcal{G}(T_1, T_2)$ is $\mathcal{O}(n \log n)$.*

Proof. Consider any $u \in [n]$ such that $\mathsf{level}_{T_1}(u) = \mathsf{level}_{T_2}(u)$ and $T_1|u \equiv T_2|u$. For each pair of ancestors z of u in T_1 and w of u in T_2 such that $\mathsf{level}_{T_1}(z) = \mathsf{level}_{T_2}(w)$, $T_1|z \equiv T_2|w$ and either $\mathsf{head}_{T_1}(z) = z$ or $\mathsf{head}_{T_2}(w) = w$, u will contribute 1 to the weight of an edge (z, w) in $G(p_{T_1}(z), p_{T_2}(w))$. Because there are at most $\log n$ heavy paths above any node of T_1 or T_2, each label $u \in [n]$ contributes 1 to the weight of at most $2 \log n$ non-special edges, making their total weight $\mathcal{O}(n \log n)$ overall. ◀

We divide the graphs in $\mathcal{G}(T_1, T_2)$ into three types: see Figure 1 for an example.

Type 1: graphs $G(u, v)$ with at least one non-special edge.

Type 2: graphs $G(u, v)$ with no non-special edges, and $\Gamma(u, v) = 1$.

Type 3: graphs $G(u, v)$ with no non-special edges, and $\Gamma(u, v) = 0$.

We will construct only the graphs of type 1 and 2, and extract from them the information that the graphs of type 3 would have captured. In what follows we show how to construct the graphs of type 1 and 2 in $\mathcal{O}(n \log^2 n)$ time.

Constructing the Graphs of Type 1 and 2. The first step is to find all pairs of nodes that correspond to graphs of type 1 or 2, and store them in a dictionary D implemented as a balanced search tree with $\mathcal{O}(\log n)$ access time. The second step is to find the non-special edges of these graphs, and store them in a separate dictionary, also implemented as a balanced search tree with $\mathcal{O}(\log n)$ access time. Note that the weights will be found at a later stage of the algorithm. We assume that both trees have been already decomposed into heavy paths, and we already know which subtrees are isomorphic. This can be preprocessed in $\mathcal{O}(n)$ time.

▶ **Lemma 2.** *All graphs of type 1 and 2 can be identified in $\mathcal{O}(n \log^2 n)$ time.*

Proof. We consider every $u \in [n]$ such that $\mathsf{level}_{T_1}(u) = \mathsf{level}_{T_2}(u)$ and $T_1|u \equiv T_2|u$ in two passes. In the first pass, we need to iterate over every ancestor z of u in T_1 and w of u in T_2 such that $\mathsf{level}_{T_1}(z) = \mathsf{level}_{T_2}(w)$, $T_1|z \equiv T_2|w$ and either $\mathsf{head}_{T_1}(z) = z$ or $\mathsf{head}_{T_2}(w) = w$, and if additionally $T_1|p_{T_1}(z) \equiv T_2|p_{T_2}(w)$ then designate $G(p_{T_1}(z), p_{T_2}(w))$ to be a graph of type 1 and insert it into D. As a non-special edge (z, w) of a graph $G(p_{T_1}(z), p_{T_2}(w))$ is such that either z or w are not on the same heavy path as their parents, this correctly determines all graphs of type 1.

To efficiently iterate over all such z and w given u, we assume that the nodes of every heavy path of a tree T are stored in an array, so that, given any node $u \in T$, we are able to access the node that belongs to the same heavy path as u and whose level is ℓ in constant time, if it exists. We denote such operation $\mathsf{access}_T(u, \ell)$. Given two nodes $u \in T_1$ and $v \in T_2$ on the same level, the procedure below shows how to iterate over every ancestor z of u and w of v such that $\mathsf{level}_{T_1}(z) = \mathsf{level}_{T_2}(w)$ and either $\mathsf{head}_{T_1}(z) = z$ or $\mathsf{head}_{T_2}(w) = w$, in $\mathcal{O}(\log n)$ time, implying that all graphs of type 1 can be identified in $\mathcal{O}(n \log^2 n)$ time.

```
1  while u ≠ ⊥ and v ≠ ⊥ do
2      if level_T1(head_T1(u)) < level_T2(head_T2(v)) then
3          output access_T1(u, level_T2(head_T2(v))) and head_T2(v)
4          v ← p_T2(head_T2(v))
5      if level_T1(head_T1(u)) > level_T2(head_T2(v)) then
6          output head_T1(u) and access_T2(v, level_T1(head_T1(u)))
7          u ← p_T1(head_T1(u))
8      else
9          output head_T1(u) and head_T2(v)
10         u ← p_T1(head_T1(u))
11         v ← p_T2(head_T2(v))
```

In the second pass, for each $u \in [n]$ such that $\mathsf{level}_{T_1}(u) = \mathsf{level}_{T_2}(u)$ and $T_1|u \equiv T_2|u$, we designate $G(u, u)$ to be a graph of type 2, unless it has been already designated to be a graph of type 1. ◀

▶ **Lemma 3.** *All graphs of type 1 and 2 can be populated with their edges in $\mathcal{O}(n \log^2 n)$ time.*

Proof. For each such graph $G(u, v)$ such that none of u, v is a leaf, let u' be the unique heavy child of u, and v' be the unique heavy child of v. We add the special edge (u', v') to $G(u, v)$. To find the non-special edges, we again consider every $u \in [n]$ such that $\mathsf{level}_{T_1}(u) = \mathsf{level}_{T_2}(u)$

and $T_1|u \equiv T_2|u$: we iterate over the ancestors z of u in T_1 and w of u in T_2 such that $\mathsf{level}_{T_1}(z) = \mathsf{level}_{T_2}(w)$, $T_1|z \equiv T_2|w$ and either $\mathsf{head}_{T_1}(z) = z$ or $\mathsf{head}_{T_2}(w) = w$, and if additionally $T_1|p_{T_1}(z) \equiv T_2|p_{T_2}(w)$ then add a non-special edge (z, w) to $G(p_{T_1}(z), p_{T_2}(w))$. This takes $\mathcal{O}(n \log^2 n)$ time overall. ◀

Processing the Graphs of Type 1 and 2. Having constructed the graphs of type 1 and 2 in $\mathcal{O}(n \log^2 n)$ time, we process them level by level. Consider $G(u, v)$: for each of its edges (u', v') corresponding to $u' \in \mathsf{children}_{T_1}(u)$ and $v' \in \mathsf{children}_{T_2}(v)$, we need to extract its weight $\gamma(u', v')$. If $G(u', v')$ is of type 1 or 2, the graph can be extracted from the dictionary in $\mathcal{O}(\log n)$ time. Otherwise, $G(u', v')$ is of type 3 and we need to make up for not having processed such graphs.

To this aim, we associate a sorted list of levels with each pair of heavy paths of T_1 and T_2. The lists are stored in a dictionary indexed by the heads of the heavy paths. For every $u, v \in [n]$ such that $G(u, v)$ is of type 1 or 2, we append the levels of u and v to the lists associated with the respective heavy paths. The lists can be constructed in $\mathcal{O}(n \log^2 n)$ time by processing the graphs level by level, and allow us to efficiently use the following lemma.

▶ **Lemma 4.** *Consider $u, v \in [n]$ such that $\mathsf{level}_{T_1}(u) = \mathsf{level}_{T_2}(v)$ and $T_1|u \equiv T_2|v$, but $G(u, v)$ is of type 3. Either both u and v are leaves and $\gamma(u, v) = 0$, or the heavy child of u is u', the heavy child of v is v', and $\gamma(u, v) = \gamma(u', v')$.*

Proof. First observe that $u \neq v$, as otherwise $G(u, v)$ would be of type 2. Becase $T_1|u \equiv T_2|v$, either both u and v are leaves or none of them is a leaf. In the former case, $G(u, v)$ is empty and $\gamma(u, v) = 0$. By how we resolve ties in the heavy path decomposition, in the latter case we have $T_1|u' \equiv T_2|v'$, where u' is the heavy child of u and v' is the heavy child of v. $G(u, v)$ consists of the unique special edge corresponding to the heavy child u' of u and v' of v, so $\mathcal{M}(G(u, v))$ is equal to the cost of the special edge, and by (1) we obtain that $\gamma(u, v) = \gamma(u', v')$. ◀

Given $u, v \in [n]$ such that $\mathsf{level}_{T_1}(u) = \mathsf{level}_{T_2}(v) = \ell$ and $T_1|u \equiv T_2|v$, we extract $\gamma(u, v)$ by accessing the sorted list associated with the heavy paths of u and v: we binary search for the smallest level $\ell' \geq \ell$ such that the heavy paths of u and v respectively contain a node u' and v', both on level ℓ', with $G(u', v')$ of type 1 or 2. Then Lemma 4, together with the fact that in our heavy path decomposition the subtrees rooted at the heavy children of two nodes with isomorphic subtrees are also isomorphic, implies that $\gamma(u, v) = \gamma(u', v')$.

It remains to describe how to compute $\mathcal{M}(G(u, v))$ for every graph $G(u, v)$ of type 1 and 2. We could have used any maximum weight matching algorithm, but this would result in a higher running time. Our goal is to plug in a maximum matching algorithm. This seems problematic as $G(u, v)$ is a weighted bipartite graph, but we will show that maximum weight matching can be reduced to multiple instances of maximum matching. However, bounding the overall running time will require bounding the total weight of all edges belonging to graphs of type 1 and 2. By Lemma 1 we already know that the total weight of all non-special edges is $\mathcal{O}(n \log n)$, but such bound doesn't hold for the special edges. Therefore, we proceed as follows. Let u' be the heavy child of u and v' be the heavy child of v. We construct $G'(u, v)$ by removing the special edge from $G(u, v)$. We also construct $G''(u, v)$ from $G(u, v)$ by removing all the edges incident to u' and v' (see Figure 1 for an example). Equation (1) can then be rewritten as follows:

$$\gamma(u, v) = \max\{\mathcal{M}(G'(u, v)), \mathcal{M}(G''(u, v)) + \gamma(u', v')\} + \Gamma(u, v). \tag{2}$$

This is because a maximum weight matching in $G(u,v)$ either includes the special edge (u',v'), implying that no other edges incident to u' and v' can be part of the matching and thus $\mathcal{M}(G(u,v)) = \mathcal{M}(G''(u,v)) + \gamma(u',v')$, or it does not include it, thus $\mathcal{M}(G(u,v)) = \mathcal{M}(G'(u,v))$. Since the graphs $G'(u,v)$ and $G''(u,v)$ contain only non-special edges, the overall weight of all edges in the obtained instances of maximum weight matching is $\mathcal{O}(n \log n)$.

We already know that constructing all the relevant graphs takes $\mathcal{O}(n \log^2 n)$ time. It remains to analyze the time to calculate the maximum weight matching in every $G'(u,v)$ and $G''(u,v)$. We first present a preliminary lemma that connects the complexity of calculating the maximum weight matching in a weighted bipartite graph to the complexity of calculating the maximum matching in an unweighted bipartite graph.

▶ **Lemma 5** ([30]). *Let G be a weighted bipartite graph, and let N be the total weight of all the edges of G. Calculating the maximum weight matching in G can be reduced in $\mathcal{O}(N)$ time to multiple instances of calculating the maximum matching in an unweighted bipartite graph, in such a way that the total number of edges in all such graphs is at most N.*

Proof. Using the decomposition theorem of Kao, Lam, Sung, and Ting [30], we can reduce computing the maximum weight matching in a weighted bipartite graph such that the total weight of all edges is N to multiple instances of calculating the largest cardinality matching in an unweighted bipartite graph. The total number of edges in all unweighted bipartite graphs is $\sum_i m_i = N$ and the reduction can be implemented in $\mathcal{O}(N)$ time by maintaining a list of edges with weight w, for every $w = 1, 2, \ldots, N$. ◀

▶ **Theorem 6.** *Let $f(m)$ be the complexity of calculating the maximum matching in an unweighted bipartite graph on m edges, and let $f(m)/m$ be nondecreasing. The permutation distance can be computed in $\tilde{\mathcal{O}}(f(n))$ time.*

Proof. The total number of edges in all constructed graphs is $\mathcal{O}(n \log n)$, and the total time to construct the relevant graphs and extract the costs of their edges is $\mathcal{O}(n \log^2 n)$. Thus, the total running time is $\mathcal{O}(n \log^2 n)$ plus the time to compute the maximum weight matching in every graph of type 1 and type 2. Let N_i be the total weight of all non-special edges in the i-th of these graphs. By Lemma 1, $\sum_i N_i = \mathcal{O}(n \log n)$. Additionally, $N_i \leq n$. Let $m_{i,j}$ be the number of edges in the j-th instance of unweighted bipartite matching for the i-th graph. By Lemma 5, the overall time is hence $\sum_{i,j} f(m_{i,j})$, where $\sum_{i,j} m_{i,j} \leq \sum_i N_i = \mathcal{O}(n \log n)$ and $m_{i,j} \leq N_i \leq n$. We upper bound $\sum_{i,j} f(m_{i,j})$ using the assumption that $f(m)/m$ is nondecreasing as follows:

$$\sum_{i,j} f(m_{i,j}) = \sum_{i,j} m_{i,j} \cdot f(m_{i,j})/m_{i,j} \leq \sum_{i,j} m_{i,j} \cdot f(n)/n = \mathcal{O}(f(n) \log n). \quad ◀$$

▶ **Corollary 7.** *The permutation distance can be computed in $\tilde{\mathcal{O}}(n^{4/3+o(1)})$ time.*

3.3 Reduction from Bipartite Maximum Matching

We complement the algorithm described in Subsection 3.2 with a reduction from bipartite maximum matching to computing the permutation distance: see Figure 2 for an example.

▶ **Theorem 8.** *Given an unweighted bipartite graph on m edges, we can construct in $\mathcal{O}(m)$ time two trees with permutation distance equal to the cardinality of the maximum matching.*

Proof. We first modify the graph so that the degree of every node is at most 3. This can be ensured in $\mathcal{O}(m)$ time by repeating the following transformation: take a node u with neighbours $v_1, v_2, \ldots, v_k$, $k \geq 4$. Replace u with u' and u'' both connected to a new node v,

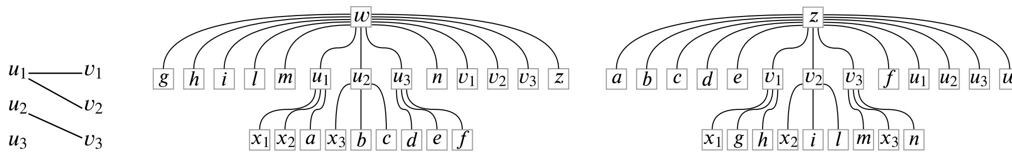


Figure 2 The two trees built for the graph on the left, according to Theorem 8.

connect u' to $v_1, v_2, \ldots, v_{k-2}$ and u'' to v_{k-1}, v_k. It can be verified that the cardinality of the maximum matching in the new graph is equal to that in the original graph increased by 1. By storing, for every node, the incident edges in a linked list, we can implement a single step of this transformation in constant time, and there are at most m steps.

We will now first construct two unlabelled trees and then explicitly assign appropriate labels to their nodes. Without loss of generality, let the nodes of the graph be $u_1, u_2, \ldots, u_m$ and $v_1, v_2, \ldots, v_m$. In the first tree we create m nodes, labelled with $u_1, u_2, \ldots, u_m$, connected to a common unlabelled root. In the second tree we do the same with nodes $v_1, v_2, \ldots, v_m$. Then, for every edge (u_i, v_j) of the graph, we attach a new leaf to u_i in the first tree and to v_j in the second tree, and assign the same label to both of them. Finally, we attach enough unlabelled leaves to every u_i and v_j to make their degrees all equal to 3. To make both trees fully-labelled on the same set of labels, we further attach $1 + m + 3m - m = 3m + 1$ *extra leaves* to the roots of both trees. For every unlabelled leaf attached to $u_1, u_2, \ldots, u_m$ of the first tree, we choose an unlabelled extra leaf of the second tree, and assign the same label to both of them. We then assign the same label to the root of the first tree and an extra leaf of the second tree, and label the last m extra leaves of the second tree with $u_1, u_2, \ldots, u_m$. We finally swap the trees and repeat the same procedure: see Figure 2 for an example.

The permutation distance between the two trees is equal to the cardinality of the maximum matching. Indeed, the trees are clearly isomorphic; moreover, any isomorphism must match extra leaves with extra leaves, and every u_i to a $v_{\pi(j)}$, for some permutation π on $[m]$. The extra leaves do not contribute to the number of conserved nodes, while u_i and $v_{\pi(j)}$ contribute 1 if and only if $(u_i, v_{\pi(j)})$ was an edge in the original graph. Thus, the distance is equal to the maximum over all permutations π of the number of edges $(u_i, v_{\pi(j)})$. This in turn is equal to the cardinality of the maximum matching in the original graph. ◀

4 A Constant-Factor Approximation Algorithm for the Rearrangement Distance

A linear-time algorithm that, given two trees T_1 and T_2, approximates $d(T_1, T_2)$ within a constant factor, was known for the case where at least one of the trees is binary [5]: here we do not make any assumptions on the degrees. Throughout this section, we actually consider $\tilde{d}(F_1, F_2)$, and show how to approximate it within a constant factor. This allows us to approximate $d(T_1, T_2)$ within a constant factor using the following procedure. First, we add n leaves $n+1, n+2, \ldots, n$ attached to the (identical) roots of T_1 and T_2 to obtain T'_1 and T'_2, respectively. We call the resulting trees *anchored*. Because T_1 and T_2 are assumed to have the same root that cannot be permuted, we have $d(T_1, T_2) = d(T'_1, T'_2)$. We claim that $\tilde{d}(T'_1, T'_2) = d(T'_1, T'_2)$. Intuitively, in one direction it suffices to replace every link-and-cut operation $v \,|\, u \to w$ with a cut operation $(v \dagger u)$; for the other direction, we argue that it does not make sense to permute the root, and every cut operation $(v \dagger u)$ can be replaced by $v \,|\, u \to w$, where $w = p_{T_2}(v)$, and such link-and-cut operations are reordered so as not to make w a descendant of v.

▶ **Lemma 9.** *For any two anchored trees T_1 and T_2, $\tilde{d}(T_1, T_2) = d(T_1, T_2)$.*

We can thus approximate $\tilde{d}(T'_1, T'_2)$ within a constant factor to obtain a constant factor approximation of $d(T_1, T_2)$. In the remaining part of this section we design an approximation algorithm for $\tilde{d}(F_1, F_2)$, where F_1 and F_2 are two arbitrary forests.

We start with describing the notation. Consider two forests F_1 and F_2. For every $i \in [n]$, let $a[i] \in [n]$ be the parent of a non-root node i in F_1, and $a[i] = 0$ if i is a root in F_1. Formally, $a[i] = p_{F_1}(i)$ when $p_{F_1}(i) \neq \perp$ and $a[i] = 0$ otherwise; $b[i]$ is defined similarly but for F_2. We think of a and b as vectors of length n.

The algorithm consists of four steps, with step j transforming forest F_1^{j-1} into F_1^j by performing $\mathrm{ALG}(j)$ operations, starting from $F_1^0 = F_1$. We will guarantee that $\mathrm{ALG}(j) = \mathcal{O}(\tilde{d}(F_1^{j-1}, F_2))$. Then, by triangle inequality and symmetry, $\tilde{d}(F_1^j, F_2) \leq \tilde{d}(F_1^{j-1}, F_1^j) + \tilde{d}(F_1^{j-1}, F_2) \leq \mathrm{ALG}(j) + \tilde{d}(F_1^{j-1}, F_2) = \mathcal{O}(\tilde{d}(F_1^{j-1}, F_2))$, so by induction $\tilde{d}(F_1^j, F_2) = \mathcal{O}(\tilde{d}(F_1, F_2))$. Consequently, $\mathrm{ALG}(j) = \mathcal{O}(\tilde{d}(F_1, F_2))$, making the overall cost $\sum_j \mathrm{ALG}(j) = \mathcal{O}(\tilde{d}(F_1, F_2))$ In the j-th step of the algorithm $a[i]$ refers to the parent of i in F_1^{j-1}. To analyse each step of the algorithm we will use the following two structures, the first of which is a streamlined version of family partitions defined in the previous paper [5].

▶ **Definition 10** (family partition)**.** *Given two forests F_1 and F_2, the* family partition *$P(F_1, F_2)$ is the set $\{(a[i], b[i]) : a[i], b[i] \neq 0 \;\wedge\; a[i] \neq b[i]\}$.*

▶ **Definition 11** (migrations graph)**.** *Given two forests F_1 and F_2, the* migrations graph *$MG(F_1, F_2)$ consists of edges $\{(i, j) : a[i], a[j], b[i], b[j] \neq 0 \;\wedge\; a[i] = a[j] \;\wedge\; b[i] \neq b[j]\}$.*

For a multiset S, let $|S|$ denote its cardinality, that is, the sum of multiplicities of all distinct elements of S. The mode of S, denoted $\mathsf{mode}(S)$, is any element $s \in S$ with the largest multiplicity $\mathsf{freq}_S(s)$. We will use the following combinatorial lemma.

▶ **Lemma 12.** *Given any multiset S, let $f = \min\{|S| - \mathsf{freq}_S(\mathsf{mode}(S)), \lfloor |S|/2 \rfloor\}$. All $|S|$ elements of S can be partitioned into f pairs $(x_1, y_1), \ldots, (x_f, y_f)$, $x_i \neq y_i$, for every $i \in [f]$, and the remaining $|S| - 2f$ elements.*

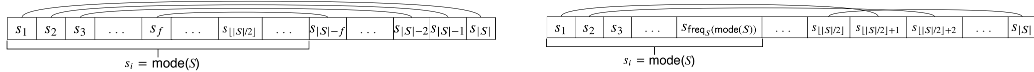


Figure 3 Pairing in the case $f = |S| - \mathsf{freq}_S(\mathsf{mode}(S))$ (left) and $f = \lfloor |S|/2 \rfloor$ (right).

Proof. Number the elements of S so that $s_1 = \ldots = s_{\mathsf{freq}_S(\mathsf{mode}(S))} = \mathsf{mode}(S)$ and all of the others are sorted and numbered from $\mathsf{freq}_S(\mathsf{mode}(S)) + 1$ to $|S|$ accordingly. Then, if $f = |S| - \mathsf{freq}_S(\mathsf{mode}(S))$, pairs $(s_i, s_{|S|-i+1})$, $i \in [f]$ are s.t. $s_i \neq s_{|S|-i+1}$ (Figure 3, left); if $f = \lfloor |S|/2 \rfloor$, pairs $(s_i, s_{\lfloor |S|/2 \rfloor + i})$, $i \in [\lfloor |S|/2 \rfloor]$ are s.t. $s_i \neq s_{\lfloor |S|/2 \rfloor + i}$ (Figure 3, right). ◀

4.1 Step 1

Roughly speaking, the aim of the first step is to ensure that all nodes that might be possibly involved in a permutation, i.e., the nodes with different children in F_1 and F_2, are roots. This is so that we do not need to worry about the relationship with their parents. For every $i \in [n]$ such that $a[i]$ and $b[i]$ are both defined and different, we cut the edges from $a[i]$ and

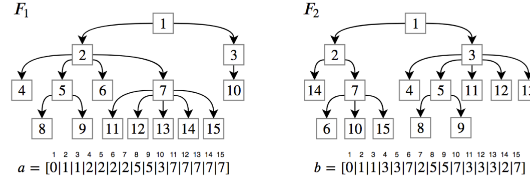


Figure 4 F_1 and F_2. The family partition is $P = \{(2,3),(2,7),(3,7),(7,3),(7,2)\}$.

$b[i]$ to their parents in F_1, thus making both of them roots. In other words, for every i such that $a[i], b[i] \neq 0$ and $a[i] \neq b[i]$, we cut edges $(a[i], a[a[i]])$ and $(b[i], a[b[i]])$. The resulting forest F_1^1 has the following property: for each $i \in [n]$ such that the parents of i in F_1^1 and in F_2 are both defined and different, $a[a[i]] = a[b[i]] = \bot$.

The number of cuts in this step is by definition at most twice the size of the family partition $P(F_1, F_2)$. Bernardini et al. [5] already showed that $|P(T_1, T_2)| \leq 2d(T_1, T_2)$ for two trees T_1 and T_2. We show that this still holds for forests and $\tilde{d}$: for completeness, we provide a self-contained proof (cf. Lemma 16 in [5]).

▶ **Lemma 13.** $|P(F_1, F_2)| \leq 2\tilde{d}(F_1, F_2)$, *implying* $ALG(1) \leq 4\tilde{d}(F_1, F_2)$.

Proof. It is enough to verify that applying a single cut operation might decrease the size of the family partition by at most one, while applying a permutation operation π might decrease the size of the family partition by at most $2s$, where $s = |\{u : u \neq \pi(u)\}|$.

Consider a cut operation $(v \dagger u)$. The only change to a is that $a[v]$ becomes 0, so indeed the size of the family partition might decrease by at most one.

Now consider a permutation π. After applying π, an edge $(i, a[i])$ becomes $(\pi(i), \pi(a[i]))$, making $\pi(a[\pi^{-1}(i)])$ the parent of i. This transforms the family partition P into

$$P' = \{(\pi(a[i]), b[\pi(i)]) : a[i] \neq 0 \ \wedge \ b[\pi(i)] \neq 0 \ \wedge \ \pi(a[i]) \neq b[\pi(i)]\}.$$

To lower bound the size of $|P'|$, we first focus on the subset of P corresponding to the nodes that are fixed by π. We therefore define

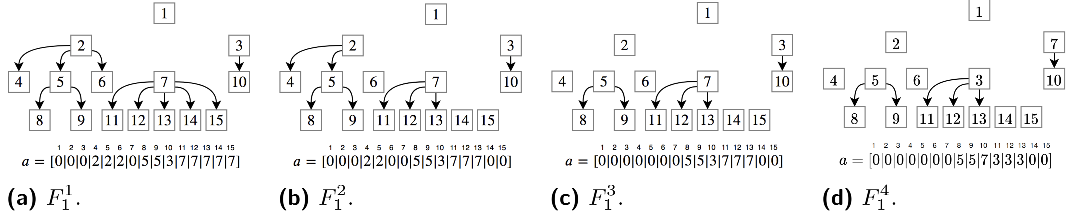
$$P_f = \{(a[i], b[i]) : a[i] \neq 0 \ \wedge \ b[i] \neq 0 \ \wedge \ a[i] \neq b[i] \ \wedge \ \pi(i) = i\}.$$

By definition, we can equivalently rewrite P_f as

$$P_f = \{(a[i], b[\pi(i)]) : a[i] \neq 0 \ \wedge \ b[\pi(i)] \neq 0 \ \wedge \ a[i] \neq b[\pi(i)] \ \wedge \ \pi(i) = i\}.$$

Now consider all pairs with the same second coordinate y in P_f: $(x_1, y), (x_2, y), \ldots, (x_k, y)$, where $x_i \neq y$ for every $i \in [k]$. P' contains all pairs $(\pi(x_i), y)$ such that $\pi(x_i) \neq y$. If $\pi(y) = y$ then $\pi(x_i) = y$ cannot happen and P' contains all pairs with the second coordinate y from P_f; otherwise, P' contains all such pairs except possibly one. Overall, $|P'| \geq |P_f| - s$, and $|P_f| \geq |P| - s$ so indeed $|P'| \geq |P| - 2s$. ◀

▶ **Example 14.** Consider F_1 and F_2 depicted in Figure 4. Step 1 consists of cut operations $(2 \dagger 1)$ (because, e.g., $a[4] \neq b[4]$ and $a[4] = 2$), $(3 \dagger 1)$ (because $b[4] = 3$) and $(7 \dagger 2)$ (because, e.g., $a[11] \neq b[11]$ and $a[11] = 7$). The resulting forest F_1^1 is shown in Figure 5a.



(a) F_1^1. **(b)** F_1^2. **(c)** F_1^3. **(d)** F_1^4.

Figure 5 The forests obtained after Step 1 (5a), Step 2 (5b), Step 3 (5c) and Step 4 (5d).

4.2 Step 2

Consider $u \in [n]$, and let $\mathsf{children}_{F_1^1}(u) = \{v_1, \ldots, v_k\}$. We define the multiset $B(u) = \{b[v_i] : b[v_i] \neq 0\}$ containing the parents in F_2 of the children of u in F_1^1. Recall that $\mathsf{mode}(B(u))$ is the most frequent element of $B(u)$ (ties are broken arbitrarily). We cut all edges (v_i, u) such that $b[v_i] \neq 0$ and $b[v_i] \neq \mathsf{mode}(B(u))$, and define, for each $u \in [n]$, its representative $\mathsf{rep}(u) = \mathsf{mode}(B(u))$. Intuitively, $\mathsf{rep}(u)$ is the node that might be convenient to replace u with using a permutation. Roughly speaking, in this step we get rid of all of the children of u that would be misplaced after permuting u and $\mathsf{rep}(u)$, for each $u \in [n]$. The resulting forest F_1^2 has the following property: for each $u \in [n]$, for any child v of u in F_1^2, either $b[v] = 0$ or $b[v] = \mathsf{rep}(u)$, i.e., the children of each node u of F_1^2 have all the same parent $\mathsf{rep}(u)$ in F_2.

To bound the number of cuts in this step we first need a technical lemma relating the rearrangement distance of two forests and the size of any matching in their migrations graph.

▶ **Lemma 15.** *Consider two forests F_1 and F_2 and their migrations graph $MG(F_1, F_2)$. For any matching M in $MG(F_1, F_2)$ it holds that $|M| \leq \tilde{d}(F_1, F_2)$.*

Proof. By definition, there is an edge between i and j in $MG(F_1, F_2)$ if and only if $a[i] = a[j]$, but $b[i] \neq b[j]$. Let M be any matching in $MG(F_1, F_2)$. If $|M| > 0$ then $\tilde{d}(F_1, F_2) \geq 1$, so it is enough to show that, for a single operation transforming F_1 into F_1', the graph $MG(F_1', F_2)$ contains a matching M' of size at least $|M| - s$, where $s = 1$ for a cut operation and $s = |\{u : u \neq \pi(u)\}|$ for a permutation operation π.

First, consider a cut operation $(v \dagger u)$. The only change in $MG(F_1', F_2)$ is removing all edges incident to v. M contains at most one edge incident to v, so we construct M' of size at least $|M| - 1$ from M by possibly removing a single edge. Second, consider a permutation operation π: we construct M' from M by removing every edge (v, w) such that $v \neq \pi(v)$ or $w \neq \pi(w)$. Because there is at most one edge incident to every u such that $u \neq \pi(u)$, M' contains at least $|M| - s$ edges. M' is a matching in $MG(F_1', F_2)$, as for every $(v, w) \in M'$ we have $p_{F_1'}(v) = p_{F_1}(v)$ and $p_{F_1'}(w) = p_{F_1}(w)$. ◀

▶ **Lemma 16.** *$ALG(2) \leq 2\tilde{d}(F_1^1, F_2)$.*

Proof. We consider each $u \in [n]$ separately. Let $m = \mathsf{freq}_{B_u}(\mathsf{mode}(B_u))$ and MG_u be the subgraph of $MG(F_1^1, F_2)$ induced by B_u. We will first construct a matching of appropriate size in every MG_u. We cut every (v_i, u) such that $b[v_i] \neq 0$ and $b[v_i] \neq \mathsf{mode}(B_u)$, making $|B_u| - m$ cuts. Let $f = \min(|B_u| - m, \lfloor |B_u|/2 \rfloor)$. By Lemma 12, we can partition a subset of B_u into f pairs $(b[v_i], b[v_j])$ such that $b[v_i] \neq b[v_j]$. We add every edge (v_i, v_j) to the constructed matching. We claim that $|B_u| - m \leq 2f$. This holds because $|B_u| - m \leq 2(|B_u| - m)$ and $|B_u| - m \leq |B_u| - 1 \leq 2\lfloor |B_u|/2 \rfloor$ for nonempty B_u.

We take the union of all such matchings to obtain a single matching M. As argued above, the total number of cuts is at most $2|M|$. Together with Lemma 15, this implies that $\mathrm{ALG}(2) \leq 2|M| \leq 2\tilde{d}(F_1^1, F_2)$. ◀

▶ **Example 17.** Consider again F_1 and F_2 of Figure 4. $B(7) = \{3, 3, 3, 2, 7\}$, thus we cut $(14 \dagger 7)$ and $(15 \dagger 7)$. $B(2) = \{3, 3, 7\}$, implying $(6 \dagger 2)$. The resulting F_1^2 is shown in Figure 5b.

4.3 Step 3

If after Step 2 all of the children of a node u of F_1 have the same parent $\mathsf{rep}(u)$ in F_2, it still may be the case where $\mathsf{rep}(u) = \mathsf{rep}(v)$ with $u \neq v$, i.e., all of the children of two distinct nodes of F_1 have the same parent in F_2. In this case, it is not clear how to choose whether to replace u or v with $\mathsf{rep}(u) = \mathsf{rep}(v)$ in a permutation. This step aims at resolving this situation by cutting the ambiguous edges.

Consider thus $u \in [n]$, and let $\mathsf{children}_{F_2}(u) = \{v_1, v_2, \ldots, v_k\}$. We define the multiset $B'(u) = \{a[v_i] : a[v_i] \neq 0\}$ containing the parents in F_1^2 of the children of u in F_2. We cut all edges $(v_i, a[v_i])$ such that $a[v_i] \neq 0$ and $a[v_i] \neq \mathsf{mode}(B'(u))$, breaking ties arbitrarily, and define $\mathsf{rep}'(u) = \mathsf{mode}(B'(u))$. The resulting forest F_1^3 has the following property: for each $u \in [n]$, for any child v of u in F_2, we have $a[v]) = \perp$ or $a[v] = \mathsf{rep}'(u)$.

We observe that the number of cuts performed by the above procedure is the same as if we had applied Step 2 on F_2 and F_1^2. Therefore, Lemma 16 implies the following.

▶ **Lemma 18.** $ALG(3) \leq 2\tilde{d}(F_1^2, F_2)$.

▶ **Example 19.** Consider again F_1 and F_2 of Figure 4. We have $B'(3) = \{2, 2, 7, 7, 7\}$, we thus cut $(4 \dagger 2)$ and $(5 \dagger 2)$. The resulting forest F_1^3 is shown in Figure 5c.

4.4 Step 4

We summarize the properties of F_1^3 and F_2:

1. For each $u \in [n]$ such that $a[u], b[u] \neq 0$ and $a[u] \neq b[u]$, $a[u]$ and $b[u]$ are roots in F_1^3.
2. For each $u \in [n]$ we can define $\mathsf{rep}(u) \in [n]$ in such a way that, for any child v of u in F_1^3, we have $b[v] = 0$ or $b[v] = \mathsf{rep}(u)$.
3. For each $u \in [n]$ we can define $\mathsf{rep}'(u) \in [n]$ in such a way that, for any child v of u in F_2, we have $a[v] = 0$ or $a[v] = \mathsf{rep}'(u)$.

To finish the description of the algorithm, we show how to find a permutation operation π of size $\mathcal{O}(\tilde{d}(F_1^3, F_2))$ that transforms F_1^3 into F_1^4 such that $F_1^4 \sim F_2$.

For every u such that $a[u], b[u] \neq 0$ and $a[u] \neq b[u]$, we require that $\pi(a[u]) = b[u]$. Due to Property 1, for every such u we have ensured that $a[u]$ and $b[u]$ are roots of F_1^3. So, if we can find a permutation π that satisfies all the requirements and does not perturb the non-roots of F_1^3, then it will transform F_1^3 into F_1^4 such that $F_1^3 \sim F_2$. Furthermore, if for every x perturbed by π there exists u such that $a[u], b[u] \neq 0$ and $a[u] \neq b[u]$ with $x = a[u]$ or $x = b[u]$ then by Lemma 13 $|\pi| \leq 2|P(F_1^3, F_2)| \leq 4\tilde{d}(F_1^3, F_2)$ as required.

To see that there indeed exists such π, observe that due to Property 2 there cannot be two requirements $\pi(x) = y$ and $\pi(x) = y'$ with $y \neq y'$. Similarly, due to Property 3 there cannot be two requirements $\pi(x) = y$ and $\pi(x') = y$ with $x \neq x'$. Thinking of the requirements as a graph, the in- and out-degree of every node is hence at most 1, so we can add extra edges to obtain a collection of cycles defining a permutation π that does not perturb the nodes not participating in any requirement.

▶ **Example 20.** Consider F_1 and F_2 of Figure 4. $\pi = (3\ 7)$ transforms F_1^3 into $F_1^4 \sim F_2$. The final F_1^4 is shown in Figure 5d.

References

1. Amir Abboud, Loukas Georgiadis, Giuseppe F. Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. In *46th ICALP*, pages 7:1–7:15, 2019.
2. Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. New algorithms and lower bounds for all-pairs max-flow in undirected graphs. In *31st SODA*, pages 48–61, 2020.
3. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
4. Benjamin L Allen and Mike Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001.
5. Giulia Bernardini, Paola Bonizzoni, Gianluca Della Vedova, and Murray Patterson. A rearrangement distance for fully-labelled trees. In *30th CPM*, pages 28:1–28:15, 2019.
6. Paola Bonizzoni, Simone Ciccolella, Gianluca Della Vedova, and Mauricio Soto. Beyond perfect phylogeny: Multisample phylogeny reconstruction via ilp. In *8th ACM-BCB*, pages 1–10, 2017.
7. Paola Bonizzoni, Simone Ciccolella, Gianluca Della Vedova, and Mauricio Soto. Does relaxing the infinite sites assumption give better tumor phylogenies? an ilp-based comparative approach. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 16(5):1410–1423, 2018.
8. Magnus Bordewich and Charles Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, 8(4):409–423, 2005.
9. Robert S. Boyer and J. Strother Moore. MJRTY: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 105–118. Kluwer Academic Publishers, 1991.
10. Gerth Stølting Brodal, Rolf Fagerberg, Thomas Mailund, Christian NS Pedersen, and Andreas Sand. Efficient algorithms for computing the triplet and quartet distance between trees of arbitrary degree. In *24th SODA*, pages 1814–1832, 2013.
11. David Bryant. A classification of consensus methods for phylogenetics. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 61:163–184, 2003.
12. Peter Buneman. The recovery of trees from measures of dissimilarity. *Mathematics in the Archaeological and Historical Sciences*, 1971.
13. Simone Ciccolella, Giulia Bernardini, Luca Denti, Paol Bonizzoni, Marco Previtali, and Gianluca Della Vedova. Triplet-based similarity score for fully multi-labeled trees with poly-occurring labels, 2020. `arXiv:https://www.biorxiv.org/content/early/2020/04/14/2020.04.14.040550.full.pdf`.
14. Bhaskar DasGupta, Xin He, Tao Jiang, Ming Li, John Tromp, and Louxin Zhang. On distances between phylogenetic trees. In *8th SODA*, pages 427–436, 1997.
15. Zach DiNardo, Kiran Tomlinson, Anna Ritz, and Layla Oesper. Distance measures for tumor evolutionary trees. *Bioinformatics*, November 2019.
16. Annette J Dobson. Comparing the shapes of trees. In *Combinatorial Mathematics III*, pages 95–100. Springer, 1975.
17. Bartłomiej Dudek and Paweł Gawrychowski. Computing quartet distance is equivalent to counting 4-cycles. In *51st STOC*, pages 733–743, 2019.
18. George F Estabrook, FR McMorris, and Christopher A Meacham. Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. *Systematic Zoology*, 34(2):193–200, 1985.
19. Joseph Felsenstein and Joseph Felenstein. *Inferring phylogenies*, volume 2. Sinauer Associates Sunderland, MA, 2004.
20. Pawel Gawrychowski, Gad M. Landau, Wing-Kin Sung, and Oren Weimann. A faster construction of greedy consensus trees. In *45th ICALP*, pages 63:1–63:14, 2018.
21. Kiya Govek, Camden Sikes, and Layla Oesper. A consensus approach to infer tumor evolutionary histories. In *9th BCB*, pages 63–72, 2018.
22. Russell D Gray, Alexei J Drummond, and Simon J Greenhill. Language phylogenies reveal expansion pulses and pauses in pacific settlement. *Science*, 323(5913):479–483, 2009.

23 Iman Hajirasouliha, Ahmad Mahmoody, and Benjamin J Raphael. A combinatorial approach for analyzing intra-tumor heterogeneity from high-throughput sequencing data. *Bioinformatics*, 30(12):i78–i86, 2014.

24 John E. Hopcroft and Richard M. Karp. An $n^{2.5}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

25 Katharina T. Huber and Vincent Moulton. Phylogenetic networks from multi-labelled trees. *Journal of Mathematical Biology*, 52(5):613–632, 2006.

26 Daniel H Huson and David Bryant. Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, 23(2):254–267, 2006.

27 Jesper Jansson, Ramesh Rajaby, Chuanqi Shen, and Wing-Kin Sung. Algorithms for the majority rule (+) consensus tree and the frequency difference consensus tree. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 15(1):15–26, 2016.

28 Jesper Jansson, Chuanqi Shen, and Wing-Kin Sung. Improved algorithms for constructing consensus trees. *Journal of the ACM*, 63(3):1–24, 2016.

29 Wei Jiao, Shankar Vembu, Amit G Deshwar, Lincoln Stein, and Quaid Morris. Inferring clonal evolution of tumors from single nucleotide somatic mutations. *BMC bioinformatics*, 15(1):35, 2014.

30 Ming-Yang Kao, Tak Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM J. Comput.*, 31(1):18–26, 2001.

31 Nikolai Karpov, Salem Malikic, Md Khaledur Rahman, and S Cenk Sahinalp. A multi-labeled tree dissimilarity measure for comparing "clonal trees" of tumor progression. *Algorithms for Molecular Biology*, 14(1):17, 2019.

32 Robert Krauthgamer and Ohad Trabelsi. Conditional lower bounds for all-pairs max-flow. *ACM Trans. Algorithms*, 14(4):42:1–42:15, 2018.

33 Yang P Liu and Aaron Sidford. Faster divergence maximization for faster maximum flow. *arXiv preprint arXiv:2003.08929*, 2020.

34 Salem Malikic, Farid Rashidi Mehrabadi, Simone Ciccolella, Md Khaledur Rahman, Camir Ricketts, Ehsan Haghshenas, Daniel Seidman, Faraz Hach, Iman Hajirasouliha, and S Cenk Sahinalp. Phiscs: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome Research*, 29(11):1860–1877, 2019.

35 Matt McVicar, Benjamin Sach, Cédric Mesnage, Jefrey Lijffijt, Eirini Spyropoulou, and Tijl De Bie. Sumoted: An intuitive edit distance between rooted unordered uniquely-labelled trees. *Pattern Recognition Letters*, 79:52–59, 2016.

36 Luay Nakhleh, Tandy Warnow, Don Ringe, and Steven N Evans. A comparison of phylogenetic reconstruction methods on an indo-european dataset. *Transactions of the Philological Society*, 103(2):171–192, 2005.

37 Peter C Nowell. The clonal evolution of tumor cell populations. *Science*, 194(4260):23–28, 1976.

38 Mateusz Pawlik and Nikolaus Augsten. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems*, 40(1):1–40, 2015.

39 David F Robinson and Leslie R Foulds. Comparison of weighted labelled trees. In *Combinatorial Mathematics VI*, pages 119–126. Springer, 1979.

40 David F Robinson and Leslie R Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, 1981.

41 D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

42 Mike Steel. *Phylogeny: discrete and random processes in evolution*. SIAM, 2016.

43 Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3):422–433, 1979.

44 Robert S Walker, Søren Wichmann, Thomas Mailund, and Curtis J Atkisson. Cultural phylogenetics of the tupi language family in lowland south america. *PLOS One*, 7(4), 2012.

45 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *International Congress of Mathematicians*, 2018.

46 Ke Yuan, Thomas Sakoparnig, Florian Markowetz, and Niko Beerenwinkel. Bitphylogeny: a probabilistic framework for reconstructing intra-tumor phylogenies. *Genome Biology*, 16(1):36, 2015.

47 Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.