Subcubic Algorithm for (Unweighted) Unrooted Tree Edit Distance

Krzysztof Pióro ✉
Jagiellonian University, Kraków, Poland

Abstract

The tree edit distance problem is a natural generalization of the classic string edit distance problem. Given two ordered, edge-labeled trees T_1 and T_2, the edit distance between T_1 and T_2 is defined as the minimum total cost of operations that transform T_1 into T_2. In one operation, we can contract an edge, split a vertex into two or change the label of an edge.

For the weighted version of the problem, where the cost of each operation depends on the type of the operation and the label on the edge involved, $\mathcal{O}(n^3)$ time algorithms are known for both rooted and unrooted trees. The existence of a truly subcubic $\mathcal{O}(n^{3-\epsilon})$ time algorithm is unlikely, as it would imply a truly subcubic algorithm for the APSP problem. However, recently Mao (FOCS'21) showed that if we assume that each operation has a unit cost, then the tree edit distance between two rooted trees can be computed in truly subcubic time.

In this paper, we show how to adapt Mao's algorithm to make it work for unrooted trees and we show an $\widetilde{\mathcal{O}}(n^{(7\omega+15)/(2\omega+6)}) \leq \mathcal{O}(n^{2.9417})$ time algorithm for the unweighted tree edit distance between two unrooted trees, where $\omega \leq 2.373$ is the matrix multiplication exponent. It is the first known subcubic algorithm for unrooted trees.

The main idea behind our algorithm is the fact that to compute the tree edit distance between two unrooted trees, it is enough to compute the tree edit distance between an arbitrary rooting of the first tree and every rooting of the second tree.

2012 ACM Subject Classification Theory of computation → Algorithm design techniques

Keywords and phrases tree edit distance, dynamic programming, matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.88

Related Version *Full Version*: `https://arxiv.org/abs/2304.08632` [16]

Acknowledgements I would like to thank Adam Polak for introducing me to the topic, numerous discussions regarding the problem and his great help in editing this paper.

1 Introduction

The tree edit distance problem is a natural generalization of the classic string edit distance problem. Given two trees T_1 and T_2, the edit distance between T_1 and T_2 is defined as the minimum total cost of operations that transform T_1 into T_2. The exact definition of the problem depends on whether we consider rooted or unrooted trees.

For the unrooted variant, which is the focus of this paper, the trees are edge-labeled and for each vertex its neighbors form a cyclic order. We can think of unrooted trees as if they were embedded in the plane. In one operation, we can contract an edge, split a vertex into two or change the label of any edge. The cost of each operation depends on the type of the operation and the label on the edge involved.

Computing the edit distance between two trees has found applications in many different areas, such as computational biology [18, 11], image processing [1, 12, 14, 17] and comparing XML data [3, 4, 10]. One of the most notable examples is comparing the secondary structures of RNA molecules, which can be represented as rooted trees [18].

The tree edit distance problem has been already studied for many years. In 1977, Tai [19] showed the first algorithm that computes the edit distance between two rooted trees of size n in $\mathcal{O}(n^6)$ time. Next, Zhang and Shasha [21] used a dynamic programming approach to improve the complexity to $\mathcal{O}(n^4)$ time. Their dynamic programming scheme was used as a basis for later algorithms. Klein [13] showed how to optimize their algorithm to $\mathcal{O}(n^3 \log n)$ time by better choosing the direction of transitions in their dynamic programming scheme. Furthermore, he showed that his algorithm can be extended to also work for the unrooted trees in the same time complexity. Later, Demaine, Mozes, Rossman and Weimann [7] further improved the time complexity for the rooted case to $\mathcal{O}(n^3)$. Finally, Dudek and Gawrychowski [8] showed that the tree edit distance between unrooted trees can also be solved in $\mathcal{O}(n^3)$ time.

From the lower bound side, Bringmann, Gawrychowski, Mozes and Weimann [2] showed that the existence of a truly subcubic $\mathcal{O}(n^{3-\epsilon})$ time algorithm for the tree edit distance is unlikely. They proved that the existence of such an algorithm implies a truly subcubic algorithm for the All Pairs Shortest Paths problem (assuming an alphabet of size $\Theta(n)$) and $\mathcal{O}(n^{k(1-\epsilon)})$ time algorithm for finding a maximum weight k-clique (assuming a sufficiently large constant size alphabet).

However, all of these previous algorithms and lower bounds work when the costs of the operations are arbitrary. Recently, Mao [15] showed that if we assume that each operation has unit cost, then the tree edit distance between two rooted trees can be computed in $\mathcal{O}(n^{2.9546})$ time. Since in the weighted setting it was possible to obtain the same time complexity for unrooted trees as for rooted trees, Mao posed the following open problem:

Is it possible to compute the unweighted tree edit distance between two unrooted trees in subcubic time?

We answer this question affirmatively.

1.1 Our contribution

We build on ideas of Mao [15] and Klein [13] and obtain the first ever known subcubic algorithm for the tree edit distance between unrooted trees in the unweighted setting. Our main result is the following:

▶ **Theorem 1.** *There is an $\widetilde{\mathcal{O}}(mn^{(5\omega+9)/(2\omega+6)}) \leq \mathcal{O}(mn^{1.9417})$ time algorithm that computes the (unweighted) tree edit distance between two unrooted trees of sizes n and m.*

We were not able to adapt one of the optimizations from Mao, thus our algorithm is slightly slower than the one for rooted trees. Note that, however the numerical value of the exponent in our result is actually better than in the original Mao's paper. It is only due to the fact that we replaced the algorithm for max-plus multiplication of two bounded-difference $n \times n$ matrices that Mao used (running in $\mathcal{O}(n^{2.8244})$ time) with a more recent result from Chi, Duan, Xie and Zhang ($\mathcal{O}(n^{2.687})$ time) [6]. Dürr [9] showed that with this new result, Mao's algorithm works in $\widetilde{\mathcal{O}}(mn^{1.915})$ time. Thus, our algorithm is in fact slower than Mao's algorithm.

1.2 Technical overview

Sketch of Mao's algorithm for rooted trees. First, let us note that in the unrooted tree edit distance, we consider edge-labeled trees, but Mao's algorithm works for node-labeled trees. However, the modification of Mao's algorithm to work for rooted edge-labeled trees is

simple. Indeed, for both trees we can introduce a virtual root that becomes a parent of the original root and then we can associate the label of every vertex with the label of edge to its parent.

Mao's algorithm is based on Chen's algorithm [5]. Chen showed a completely different approach than previous algorithms that were based on Zhang and Shasha's dynamic programming scheme. He reduced the problem to the $(\min, +)$ matrix multiplication and obtained an algorithm working in $\mathcal{O}(n^4)$ time[1].

Mao builds on Chen's approach but considers an equivalent maximization problem of the similarity of trees. Due to this, the matrices occurring in his algorithm satisfy additional properties: they are monotone and the difference between two adjacent cells is bounded by a constant. For such matrices, there exists a truly subcubic time algorithm that computes the $(\min, +)$ product [6], which is crucial for obtaining a subcubic complexity in the tree edit distance problem.

In summary, Mao's algorithm, given two trees T_1 and T_2, computes matrices $S(T_1')$ for some subtrees T_1' of tree T_1. Every such matrix $S(T_1')$ encodes the similarity between tree T_1' and all relevant fragments of tree T_2. These fragments are defined as segments of the Euler tour sequence[2] and are formally defined in Definition 8.

Mao first shows a dynamic programming scheme based on Chen's algorithm that computes $S(T_1)$ in $\mathcal{O}(n^4)$ time. Next, he optimizes it to $\mathcal{O}(n^3)$ time by exploiting the special properties of similarity matrices. Then, to achieve a subcubic complexity, he presents a special decomposition scheme, which allows him to skip some of the subproblems. For block size $\Delta = n^d$ for some d slightly smaller than 0.5, he decomposes the computation of $S(T)$ into $\mathcal{O}(n/\Delta)$ transitions of one of two types.

The first type is a concatenation of two trees. For that, he shows an $\mathcal{O}(t^{1-\epsilon}n^2)$ time algorithm that computes the $(\max, +)$ product of two bounded-difference matrices such that the entries in one of them are bounded by t. The second transition type involves expanding a subtree by adding a path going up from its root along with some additional subtrees attached to this path. For these transitions, Mao presents a special three-part combinatorial method.

Sketch of our algorithm for unrooted trees. To generalize Mao's algorithm to unrooted trees, we use the idea that Klein [13] used in his $\mathcal{O}(n^3 \log n)$ time algorithm. Klein used the fact that to compute the tree edit distance between two unrooted trees, it is enough to consider arbitrary rooting of the first tree and try all possible rootings of the second tree.

Direct application of this fact requires solving $\mathcal{O}(n)$ particular instances of the rooted tree edit distance. However, Klein showed that his $\mathcal{O}(n^3 \log n)$ time algorithm for the rooted tree edit distance can be modified to solve all of these instances at once in the same time complexity. To achieve this, he used the fact that his algorithm for rooted trees computes the edit distance between some fragments of the first tree and all subtrees of the second tree that correspond to some segment of the Euler tour sequence of that tree. Since the different rootings of the second tree correspond to different cyclic shifts of the Euler tour sequence, Klein modified his algorithm so that it computes the edit distance between some fragments of the first tree and all cyclic segments of the Euler tour sequence of the second tree.

We notice that Mao's algorithm has properties similar to those of Klein's algorithm. For some of the subtrees T_1' of the first tree, it computes a similarity matrix $S(T_1')$ that encodes the similarity between tree T_1' and all subtrees of T_2 that correspond to some segment of the

[1] By reduction we mean that he used $(\min, +)$ matrix multiplication as a sub-procedure in his algorithm.
[2] Actually, Mao used the so-called bi-order traversal sequence, but Euler tour sequence is its equivalent for edge-labeled trees.

Euler tour sequence of that tree. Thus, we need to modify the algorithm to handle cyclic segments of the Euler tour sequence. To do this, we build our new similarity matrices on a doubled Euler tour sequence (see Definitions 7–9 in Section 2). Now every cyclic segment of the original Euler tour occurs as a normal segment in our sequence.

The introduction of a doubled Euler tour requires us to make a few modifications to Mao's algorithm. The key technical challenge is handling of multiplication of new similarity matrices which we describe in Section 4.1.

2 Preliminaries

In the tree edit distance problem we consider ordered trees with labels on edges. We consider both rooted and unrooted trees. For rooted trees, ordered means that for each vertex we are given a left-to-right order of its children. For unrooted trees, ordered means that for each vertex its neighbors form a cyclic order.

Note that we can transform an unrooted tree into a rooted tree by choosing a root and the first edge to its children. This choice carries over to the other vertices and defines the order of their children. It uniquely determines the rooting of the tree, thus there are $2(n-1)$ possible rootings for an unrooted tree with n vertices.

For simplicity, we first assume that both input trees are of equal size n, but at the end we address the case when they have different sizes.

As we are interested in unrooted tree edit distance, we consider only trees that are edge labeled (for rooted tree edit distance it is more common to have labels on nodes). In this paper, we assume that every edge is labeled from some alphabet Σ of size $\mathcal{O}(n)$.

▶ **Definition 2** ((Unweighted) Tree Edit Distance). *Let T_1 and T_2 be rooted, ordered trees with labels on edges. We consider two types of operations:*

- *Label change of a selected edge in tree T_1 or T_2.*
- *Contraction of a selected edge in tree T_1 or T_2. When contracting an edge pu where p is parent of u, children of u become children of p, they replace u in the children's list of p and keep their order.*

The tree edit distance between T_1 and T_2, denoted by $\mathrm{ed}(T_1, T_2)$, is the minimum number of operations we have to perform on T_1 and T_2 to transform both trees into an identical tree.

▶ **Definition 3** ((Unweighted) Unrooted Tree Edit Distance). *Let T_1 and T_2 be unrooted, ordered trees with labels on edges. We define the tree edit distance between T_1 and T_2 as the minimum edit distance over all possible rootings of T_1 and T_2.*

Klein [13] mentioned, it is enough to consider arbitrary rooting of the first tree and try all possible rootings of the second tree to find an optimal solution for the unrooted tree edit distance.

▶ **Lemma 4.** *Let T_1 and T_2 be unrooted trees. For every rooting of tree T_1 there is at least one rooting of T_2 that admits minimum edit distance between T_1 and T_2.*

Same as Mao, we consider a maximization problem equivalent to the tree edit distance problem.

▶ **Definition 5** (Similarity). *The similarity between two rooted trees T_1 and T_2 is defined as* $\mathrm{sim}(T_1, T_2) = |E(T_1)| + |E(T_2)| - \mathrm{ed}(T_1, T_2)$.

Similarity can also be interpreted as the weight of the heaviest matching between the edges of the tree T_1 and T_2, where the cost of edge matching is 2 when edges have the same labels and 1 when they are different. In addition, the matching must respect the tree structure, which means:

- if edge $a \in T_1$ is matched to edge $b \in T_2$, then edges in the subtree of a can be matched only to edges in the subtree of b,
- if edge $a \in T_1$ is matched to edge $b \in T_2$ and $c \in T_1$ is matched to $d \in T_2$, then a is "to the left" of c if and only if b is "to the left" of d.

Note that $0 \leq \mathrm{sim}(T_1, T_2) \leq 2\min(|E(T_1)|, |E(T_2)|)$.

▶ **Definition 6.** *By $\eta(e, f)$ we denote the cost of matching edges e and f, that means $\eta(e, f) = 2$ if labels of these edges are equal and $\eta(e, f) = 1$ otherwise.*

Tree definitions. For a rooted tree T, by L_T we denote the subtree of the first (leftmost) child of the root of T along with the edge to the root of T. Similarly, by R_T we denote the subtree of the last (rightmost) child of the root of T along with the edge to the root of T. Given an edge e in a rooted tree, we use $sub(e)$ to represent the subtree rooted at edge e..

For two rooted trees T_1 and T_2, by $T_1 + T_2$ we denote the tree formed by merging the roots of tree T_1 and T_2 such that edges from the tree T_1 are "to the left" of the edges from the tree T_2.

For two trees (rooted/unrooted) T_1 and T_2 such that $T_1 \subseteq T_2$, by $T_2 - T_1$ we denote the tree formed from tree T_2 by contracting all edges that appear in tree T_1.

To describe the "fragments" of a tree that we will consider in our algorithm, we use segments of the Euler cycle of the tree. Due to technical reasons, instead of dealing with a cyclic sequence, we consider a doubled Euler tour sequence.

▶ **Definition 7.** *Let T be an unrooted tree. Consider the walk on this tree that starts at an arbitrary edge and goes twice through the Euler Tour, which visits neighbors according to their order.*

(a) *For $i \in \{1, \ldots, 4|E(T)|\}$ by $T(i)$ we denote the i-th edge of this walk.*
(b) *By $I(e)_i$ we denote the index of the i-th occurrence of the edge e in this walk.*
(c) *By $l_{i,j}(e)$ we denote the index of the first occurrence of the edge e in $T(i), \ldots, T(j-1)$.*
(d) *By $r_{i,j}(e)$ we denote the index of the second occurrence of the edge e in $T(i), \ldots, T(j-1)$.*

Note that each edge appears exactly 4 times in this walk.

▶ **Definition 8** (Segment). *Given an unrooted tree T and integers l, r such that $1 \leq l \leq r \leq 4|E(T)|$ and $r - l \leq 2|E(T)|$, by $T[l, r)$ we denote the tree formed from the tree T by contracting every edge that occurs less than 2 times in $T(l), \ldots, T(r-1)$.*

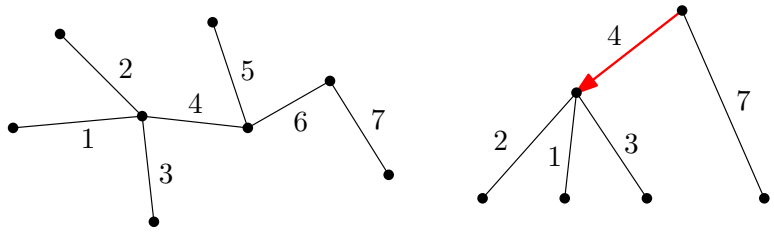
Let us note here that despite the fact that our input tree is unrooted, we can view the segments of this tree as rooted trees. Indeed, the first non-contracted edge that appears in the segment defines the rooting of this segment. Thus, the segments of length $2|E(T)|$ define all possible rootings of the tree T. See Figure 1.

Matrix definitions. Now we are ready to define similarity matrix – it encodes information about similarity of the whole rooted tree T and all "relevant fragments" of unrooted tree Q.

▶ **Definition 9** (Similarity matrix). *Given a rooted tree T and an unrooted tree Q similarity matrix $S(T, Q)$ is a matrix of size $4|E(Q)| \times 4|E(Q)|$, where:*

$$S(T,Q)_{i,j} = \begin{cases} \mathrm{sim}(T, Q[i,j)) & \text{if } i \leq j \text{ and } j - i \leq 2|E(Q)| \\ -\infty & \text{if } i > j \text{ or } j - i > 2|E(Q)| \end{cases}$$

For simplicity, we also introduce additional notation for naming of similarity matrix cells. We divide them into three types:



1 3 3 4 6 7 7 6 5 5 4 2 2 1 1 3 3 4 6 7 7 6 5 5 4 2 2 1

Figure 1 Example unrooted tree T with its doubled Euler tour sequence and segment $T[11, 22)$ of this tree. The edge marked as red defines the rooting of this segment.

- *valid cells* – cells $S(T, Q)_{i,j}$ for which $i \leq j$ and $j - i \leq 2|E(Q)|$,
- *invalid cells* – cells $S(T, Q)_{i,j}$ for which $j - i > 2|E(Q)|$,
- *cells under diagonal*.

During our algorithm we will allow invalid cells to store values other than $-\infty$. Because of that, we introduce a special equality relation for similarity matrices:

▶ **Definition 10.** *By $=_{valid}$ we denote the relation on matrices which is true if and only if they are equal on the set of valid cells.*

▶ **Definition 11.** *Let A and B be $n \times n$ matrices. By $A \star B$ we denote their* $(\max, +)$ *product i.e.* $(A \star B)_{i,j} = max_{1 \leq k \leq n} A_{i,k} + B_{k,j}$.

Now, we define a few properties of matrices. Given an $n \times n$ matrix A we call it:

(a) *finite-upper-triangular* matrix if all entries below diagonal are $-\infty$ and the rest are finite,
(b) *row-monotone* matrix if $A_{i,j} \leq A_{i,j+1}$ for all i, j,
(c) *column-monotone* matrix if $A_{i+1,j} \leq A_{i,j}$ for all i, j,
(d) *W-bounded-difference* matrix if for all i, j we have:

$$|A_{i,j} - A_{i-1,j}| \leq W$$
$$|A_{i,j} - A_{i,j+1}| \leq W$$

(e) *bounded-difference* matrix if it is a W-bounded-difference matrix for some $W = \mathcal{O}(1)$,
(f) *finite-upper-triangular-W-bounded-difference* matrix if it is a finite-upper-triangular matrix and property d holds for all $i \leq j$,
(g) *M-bounded similarity matrix* if it is a similarity matrix and all finite entries are integers between 0 and M.

It is easy to see that for similarity matrices $S(T, Q)$ the properties row-monotone and column monotone hold for all cells except invalid cells. The following lemma, which is an analogy to Lemma 4.1 from Mao [15], tells that the finite-upper-triangular-2-bounded-difference property also holds for all cells except invalid cells.

▶ **Lemma 12.** *Given a rooted tree T and an unrooted tree Q, we have:*

- $\text{sim}(T, Q[i, j + 1)) \leq \text{sim}(T, Q[i, j)) + 2$ *for all* $i \leq j, j - i + 1 \leq 2|E(Q)|$.
- $\text{sim}(T, Q[i - 1, j)) \leq \text{sim}(T, Q[i, j)) + 2$ *for all* $i \leq j, j - i + 1 \leq 2|E(Q)|$.

3 Cubic algorithm for Unrooted Tree Edit Distance

In this section, we introduce a basic dynamic programming scheme that computes the similarity of trees. It is the same scheme that Mao used, except that we use slightly different similarity matrices. We show how to efficiently compute this dynamic programming scheme in $\widetilde{\mathcal{O}}(n^3)$ time. This gives us a basic algorithm that will be later used in our subcubic algorithm.

As mentioned in Lemma 4, it is enough to consider an arbitrary rooting of the first tree and try all possible rootings of the second tree to find an optimal solution. Because of that, we can assume that we are given a rooted tree T, an unrooted tree Q and we want to compute the edit distances between T and every rooting of Q. To achieve that, we will compute the similarity matrix $S(T, Q)$. We will do this by computing recursively similarity matrices $S(T', Q)$ where T' is some fragment of tree T. As the second argument of $S(T', Q)$ will always be Q in our algorithm, we will use a shorthand $S(T') := S(T', Q)$. Additionally, we call $S(T')$ the similarity matrix of tree T'.

3.1 Dynamic programming scheme

To compute $S(T')$, we consider the following cases:

(a) T' has no edges (it is a single vertex). Then the values in all valid cells are 0.

(b) The root of T' has only one child. Let r be the root and u be its only child. We choose if we want to match ru to some edge e of $Q[i, j)$ or not:

$$S(T')_{i,j} = \max \begin{cases} S(T' - ru)_{i,j} \\ \max_{e \in Q[i;j)} \left\{ S(T' - ru)_{l_{i,j}(e)+1, r_{i,j}(e)} + \eta(ru, e) \right\} \end{cases}$$

(c) The root of T' has more than one child. Let $R_{T'}$ be the subtree of the last child of the root of T' along with the edge to the root of T'. Then:

$$S(T')_{i,j} = \max_{i \le k \le j} \{ S(T' - R_{T'})_{i,k} + S(R_{T'})_{k,j} \}$$

In other words: $S(T') = S(T' - R_{T'}) \star S(R_{T'})$.

Using this scheme, we can compute $S(T)$ in $O(n^4)$ time. Proof of the correctness of this algorithm is included in the full version of this paper [16].

3.2 Optimization to cubic

To optimize our algorithm to $\widetilde{\mathcal{O}}(n^3)$ time, we will exploit special properties of similarity matrices. We will rely on the fact that $S(T')$ is a $2|E(T')|$-bounded similarity matrix.

Furthermore, we will use the fact that these matrices are almost row-monotone and column-monotone. The only exception to this property are invalid cells, however their value is always $-\infty$. Because of that, we are able to use the same computation model that Mao used to store row-monotone, column-monotone matrices with slight modifications to handle invalid cells.

Let $l := 2n - 2$. We consider the following operations for $2l \times 2l$ matrices:

- create a new $2l \times 2l$ matrix $[-\infty]_{2l,2l}$,
- given a matrix A, create a copy of this matrix,

- given a matrix A, indexes i', j' and a value x, create a new $2l \times 2l$ matrix $B := \text{rangemax}(A, i', j', x)$, such that:

$$B_{ij} = \begin{cases} \max(A_{ij}, x) & \text{if } i \leq i' \text{ and } j \geq j' \text{ and } j - i \leq l \\ A_{ij} & \text{otherwise} \end{cases}$$

Furthermore, given matrix A we consider the following queries:

- get the value $A_{i,j}$,
- get the index $\text{mincol}(A, i, x) = \min\{j \mid A_{ij} \geq x\}$ or any index in $[1, 2l]$ if such an index does not exist,
- get the index $\text{maxrow}(A, j, x) = \max\{i \mid A_{ij} \geq x\}$ or any index in $[1, 2l]$ if such an index does not exist.

Note that the underlying data structure can keep a row-monotone, column-monotone matrix, and we can just modify queries to return $-\infty$ when reading invalid cells. This means we don't need to have the $j - i \leq l$ condition in the rangemax operation, thus we can use a simple 2D max operation (the same as Mao). All these operations can be performed with well-known data structures, such as persistent 2D segment trees in $\widetilde{\mathcal{O}}(1)$ time.

From now on, we assume that we store similarity matrices using the above computation model. We will go through all three cases of our dynamic programming scheme and we will show how to efficiently solve them using the above data structure:

(a) We can easily initialize all valid cells to 0 using $\mathcal{O}(n)$ rangemax queries. This gives us $\widetilde{\mathcal{O}}(n^2)$ time for the entire algorithm.

(b) Let us recall the equation for this case:

$$S(T')_{i,j} = \max \begin{cases} S(T' - ru)_{i,j} \\ \max_{e \in Q[i;j)} \left\{ S(T' - ru)_{l_{i,j}(e)+1, r_{i,j}(e)} + \eta(ru, e) \right\} \end{cases}$$

Naive computation of this equation relies on iterating through all cells we want to compute and for each $S(T')_{i,j}$ we iterate through each edge of $Q[i, j)$. Instead, we can do the opposite and iterate through each edge of Q and update for each of them all relevant cells of $S(T')$. Thus, we first initialize $S(T')$ with $S(T' - ru)$ and then for each edge e of Q we make three rangemax operations. See Algorithm 1 for exact values of the parameters.

Algorithm 1 Computation of case b.

Input $T, Q, S(T' - ru)$
Output $S(T')$

$S(T') \leftarrow S(T' - ru)$
for all $e \in E(Q)$ **do**
  $S(T') \leftarrow \text{rangemax}(S(T'), I(e)_1, I(e)_2 + 1, S(T' - ru)_{I(e)_1+1, I(e)_2} + \eta(ru, e))$
  $S(T') \leftarrow \text{rangemax}(S(T'), I(e)_2, I(e)_3 + 1, S(T' - ru)_{I(e)_2+1, I(e)_3} + \eta(ru, e))$
  $S(T') \leftarrow \text{rangemax}(S(T'), I(e)_3, I(e)_4 + 1, S(T' - ru)_{I(e)_3+1, I(e)_4} + \eta(ru, e))$
return $S(T')$

A single computation of this case takes $\widetilde{\mathcal{O}}(n)$ time, which gives us $\widetilde{\mathcal{O}}(n^2)$ time for the entire algorithm.

(c) This case is a matrix multiplication $S(T') = S(T' - R_{T'}) \star S(R_{T'})$. To speed up the computation, we will relay on the fact that $S(T')$ is a $2|E(T')|$ -bounded similarity matrix. The following lemma, which corresponds to Lemma 4.3 from Mao [15], shows that we can multiply similarity matrices in complexity dependent on the bounds of values of these matrices:

▶ **Lemma 13.** *Let A, B be $l \times l$ matrices. If A is a t_A-bounded similarity matrix and B is a t_B-bounded similarity matrix, then we can compute $C = A \star B$ in $\widetilde{\mathcal{O}}(t_A t_B l)$ time.*

The algorithm that computes $C = A \star B$ is the same as Algorithm 1 from Mao [15] and can be found in the full version of this paper [16] together with the proof of correctness. Thus, every time we multiply similarity matrices of trees $T' - R_{T'}$ and $R_{T'}$, we contribute $\widetilde{\mathcal{O}}(|T' - R_{T'}||R_{T'}|n)$ time to the total time complexity of this case. To compute the sum of these components, we can correlate $|T' - R_{T'}||R_{T'}|$ with the number of pairs of edges (e, f) where $e \in T' - R_{T'}$ and $f \in R_{T'}$. Let us sum these pairs over all multiplications. It is easy to see that each pair of edges from tree T' appears at most once in such a sum. Thus, the total time complexity of this case is equal to $\widetilde{\mathcal{O}}(n^3)$.

After considering all three cases, we can see that the whole algorithm works in $\widetilde{\mathcal{O}}(n^3)$ time. Note here that if the input trees are of different sizes n and m, then this algorithm works in $\widetilde{\mathcal{O}}(nm^2)$ time.

4 Subcubic algorithm for unrooted tree edit distance

In this section, we show how to get a subcubic algorithm for the unrooted tree edit distance problem. Our algorithm is based on the same decomposition scheme as Mao's algorithm. However, we will start by showing a different bottom-up view at $\widetilde{\mathcal{O}}(n^3)$ algorithm. This will give us a better intuition about Mao's decomposition scheme.

We can view the $\widetilde{\mathcal{O}}(n^3)$ algorithm as a decomposition scheme that computes $S(T)$ using the following two types of transitions:

- **Type I:** for trees T_1, T_2:
  compute $S(T_1 + T_2)$ from $S(T_1)$ and $S(T_2)$
- **Type II:** for trees T_1, T_2 such that T_2 is T_1 with one added edge going from the root up:
  compute $S(T_2)$ from $S(T_1)$

Note that the total time complexity of type I transitions is $\widetilde{\mathcal{O}}(n^3)$, while the total complexity of type II transitions is $\widetilde{\mathcal{O}}(n^2)$. This gives us an idea that to get a subcubic complexity we can try to balance these transitions. To reduce the number of type I transitions, we can generalize transitions of type II. Instead of adding a single edge going up from the root of T_1, we can add a "hat", that is, a path going up from the root along with some additional subtrees. This is the general idea of Mao's decomposition scheme, which we will now formally describe. See Figure 2 for an illustration of the "hat" structure.

First, we introduce additional definition that will help us define sub-problems occuring in the decomposition scheme. This definition corresponds to the synchronous subtree definition from Mao.

▶ **Definition 14** (Connected segment). *Given an unrooted tree T, segment $T[i, j)$ is called a connected segment if there is no edge that occurs in $T(i), \ldots, T(j-1)$ exactly once.*

A connected segment can be alternatively defined by a selection of a vertex $v \in T$ and a connected interval of children of the vertex v (we call v the root of the segment), so that subtrees of these children along with the edges from v to these children belong to this segment. Thus, a connected segment forms a connected subtree of tree T.

Now let Δ be the block size – a parameter, that we will set later. We decompose the computation of $S(T)$ into $\mathcal{O}(n/\Delta)$ transitions of two types:

- **Type I:** for connected segments T_1, T_2 such that $|T_1| \geq \Delta$ and $|T_2| \geq \Delta$: compute $S(T_1 + T_2)$ from $S(T_1)$ and $S(T_2)$
- **Type II:** for connected segments T_1, T_2 such that $T_1 \subset T_2$ and $|T_2| - |T_1| = \mathcal{O}(\Delta)$: compute $S(T_2)$ from $S(T_1)$

To compute this decomposition, we will use the same algorithm as Mao did. This algorithm is included in the full version of this paper [16].

The following theorem, which corresponds to Theorem 4.5 from Mao [15], tells us how we can perform a type I transition efficiently. We prove this in Section 4.1.

▶ **Theorem 15.** *Let A, B be $n \times n$ similarity matrices of unrooted trees. If A is t-bounded, then $C = A \star B$ can be computed in $\mathrm{MUL}(t, n) := \widetilde{\mathcal{O}}(t^{0.8145} n^2)$ time.*

For type II transitions, in Section 4.2 we prove the following theorem, which corresponds to Theorem 4.6 from Mao [15].

▶ **Theorem 16.** *Let T_1, T_2 be the connected segments of tree T such that $T_1 \subset T_2$ and $|T_2| - |T_1| = \mathcal{O}(\Delta)$. Given $S(T_1)$ we can compute $S(T_2)$ in $\widetilde{\mathcal{O}}(\mathrm{MUL}(\Delta, n) + n\Delta^4)$ time.*

Combining these two theorems, together with the decomposition scheme, gives us an algorithm from Theorem 1. Analysis of the time complexity of this algorithm can be found in the full version of this paper [16].

Note that in the Theorem 16 we have slightly worse time complexity for the type II transition than $\widetilde{\mathcal{O}}(\mathrm{MUL}(\Delta, n) + n\Delta^3)$ time from Mao's algorithm.

4.1 Type I transitions

In this section, we show how to efficiently multiply similarity matrices of unrooted trees and prove Theorem 15. Mao showed how to multiply similarity matrices of rooted trees in $\widetilde{\mathcal{O}}(t^{0.9038} n^2)$ time, where t is a bound on the entries of one of the matrices. For unrooted trees, we will use his algorithm with a slight modification.

Mao's algorithm is a recursive procedure that uses the $(\max, +)$ matrix multiplication of bounded-difference matrices as a sub-procedure. Lately, Chi et al. [6] showed a better algorithm for the $(\max, +)$ product of bounded-difference matrices. This algorithm allows us to get a better exponent in Mao's matrix multiplication.

▶ **Theorem 17** ([6])**.** *There is an $\widetilde{\mathcal{O}}(n^{(3+\omega)/2}) \leq \widetilde{\mathcal{O}}(n^{2.687})$ time randomized algorithm that computes the $(\min, +)$ product of any two $n \times n$ bounded-difference matrices.*

To analyze the improvement in the exponent, we will use the following lemma implicitly proven by Mao:

▶ **Lemma 18** ([15, Section 4.4])**.** *Assume there is an $\widetilde{\mathcal{O}}(n^c)$ time algorithm that computes the $(\min, +)$ product of any two $n \times n$ bounded-difference matrices. Let A, B be row-monotone, column-monotone and finite-upper-triangular-bounded-difference $n \times n$ matrices whose entries on the main diagonals are zero. If A is t-bounded-upper-triangular and δ is a positive value, then $C = A \star B$ can be computed in $\widetilde{\mathcal{O}}(n^2 t^2/\delta + n^2 \delta^{c-2})$ time.*

The above complexity reaches the optimum when $n^2 t^2/\delta = n^2 \delta^{c-2}$. By setting $\delta = t^{2/(c-1)}$ we get that we can multiply the similarity matrices of rooted trees in $\widetilde{\mathcal{O}}(t^{(2c-4)/(c-1)} n^2)$ time, if one of the matrices has values bounded by t.

Now we want to use this algorithm also for unrooted trees. However, we need to make sure that the values from invalid cells do not affect the values of valid cells. To easily handle this, before calling the algorithm from Lemma 18 we fix the values of invalid cells based on the values of valid cells.

Let A, B be similarity matrices of unrooted trees. We want to compute $C = A \star B$. For every invalid cell $A_{i,j}$ we set new value of $A_{i,j}$ as the maximum of valid cells $A_{i',j'}$, such that $i \leq i' \leq j' \leq j$. Let A' be the resulting matrix. We can easily compute this transformation in $\mathcal{O}(n^2)$ time by using simple recursive equation $A'_{i,j} = \max(A'_{i+1,j}, A'_{i,j-1})$. Similarly, we convert matrix B into matrix B'.

The obtained matrices A' and B' are obviously monotone and finite-upper-triangular. To prove that these new matrices are bounded-difference let us use induction on $i + j$. For a valid cell, our inductive thesis is satisfied. Now, take any invalid cell $A'_{i,j}$. Then, by inductive hypothesis we have that $A'_{i-1,j} \in [A'_{i-1,j-1}, A'_{i-1,j-1}+2]$ and $A'_{i,j-1} \in [A'_{i-1,j-1}, A'_{i-1,j-1}+2]$. Thus, $|A'_{i,j} - A'_{i-1,j}| \leq 2$ and $|A'_{i,j} - A'_{i,j-1}| \leq 2$, so matrices A' and B' are bounded-difference matrices.

Let $C' = A' \star B'$. It remains to show that $C' =_{valid} C$. Let us take any valid cell $C'_{i,j}$. There is a position k such that $C'_{i,j} = A'_{i,k} + B'_{k,j}$. If one of $A'_{i,k}$, $B'_{k,j}$ is an invalid cell, then the other would be equal to $-\infty$ and we would have $C'_{i,j} < 0$. But $C'_{i,j} \geq A'_{i,i} + B'_{i,j} \geq 0$, which gives us a contradiction. Thus, invalid cells of A' and B' have no effect on valid cells of C', so $C' =_{valid} C$.

This gives us that we can compute $C = A \star B$ in $\widetilde{\mathcal{O}}(t^{(2c-4)/(c-1)}n^2)$ time assuming we can compute the $(\min, +)$ product of any two $n \times n$ bounded-difference matrices in $\widetilde{\mathcal{O}}(n^c)$ time.

Combining this with Theorem 17 we get the proof of Theorem 15.

4.2 Type II transitions

In this section, we prove Theorem 16. Let T_1, T_2 be connected segments of tree T such that $T_1 \subset T_2$ and $|T_2| - |T_1| = \mathcal{O}(\Delta)$. We want to compute $S(T_2)$ given $S(T_1)$.

Let us consider the path $p = e_1 e_2 \dots e_k$ in tree T that goes from the root of T_2 to the root of T_1. For $1 \leq i \leq k$, let L_i be the subtree of tree T_2 consisting of siblings of edge e_i that are to the left of e_i and all descendants of these edges. Additionally, let L_{k+1} be a subtree of tree T_2 consisting of descendants of edge e_k that are to the left of tree T_1. Similarly, we denote subtrees that are to the right of path p by R_i for $1 \leq i \leq k+1$. See Figure 2.

For $1 \leq i \leq j \leq k+1$, by $L_{i,j}$ we denote the tree $L_i + L_{i+1} + \ldots + L_j$ and by $R_{i,j}$ we denote the tree $R_j + R_{j-1} + \ldots + R_i$.

First, for all trees $L_{i,j}$, $R_{i,j}$, we compute the similarity matrices $S(L_{i,j})$, $S(R_{i,j})$ using the $\widetilde{\mathcal{O}}(nm^2)$ time algorithm from Section 3.2. We notice that in one run of this algorithm, we can compute $S(L_{i,j})$ for a fixed i and all $j \geq i$. Thus, we only need to use that algorithm $\mathcal{O}(\Delta)$ times, which in total gives us $\widetilde{\mathcal{O}}(n\Delta^3)$ time for that step.

Now, we consider two cases:

(a) None of the edges $e_1, \dots, e_k$ is matched to some edge of tree Q.

(b) At least one of the edges $e_1, \dots, e_k$ is matched to some edge of tree Q.

For case (a), we can use Theorem 15 to compute $S(L_{1,k+1}) \star S(T_1) \star S(R_{1,k+1})$ in $\mathrm{MUL}(\Delta, n)$ time. For case (b), we define a restricted version of the similarity matrix for trees T', such that the root of T' has only one child.

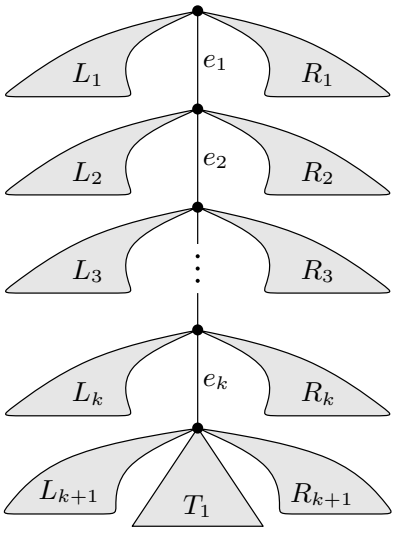


Figure 2 Tree T_2. In the type II transition we compute $S(T_2)$ based on $S(T_1)$.

▶ **Definition 19** (Restricted similarity matrix). *Let T' be a rooted tree such that the root u of T' has only one child v. The restricted similarity matrix $\hat{S}(T')$ is a $4|E(Q)| \times 4|E(Q)|$ matrix where:*

$$\hat{S}(T')_{i,j} = \begin{cases} \max_{e \in Q[i,j)}\{\mathrm{sim}(T' - uv, \mathrm{sub}(e) - e) + \eta(uv, e)\} & \text{if } i \leq j, j - i \leq 2|E(Q)| \\ -\infty & \text{otherwise} \end{cases}$$

We will compute $\hat{S}(\mathrm{sub}(e_i))$ for all edges e_i on path p. But first, let us assume that we have already computed these restricted similarity matrices. We show how to compute $S(T_2)$ using these matrices.

As mentioned before we can first initialize $S(T_2)$ with $S(L_{1,k+1}) \star S(T_1) \star S(R_{1,k+1})$. Now, let us consider a single valid cell $S(T_2)_{i,j}$ that we want to compute. To cover case (b), we can iterate through:

- First edge e_x from path p that is matched.
- Edge $e \in Q[i,j)$ such that e_x is matched to e.

Then, we have:

- Edges from $L_{1,x}$ are matched to edges from $Q[i, l_{i,j}(e))$ contributing $S(L_{1,x})_{i,l_{i,j}(e)}$.
- Edges from $\mathrm{sub}(e_x)$ are matched to edges from $Q[l_{i,j}(e), r_{i,j}(e)+1)$ contributing $\hat{S}(\mathrm{sub}(e_x))_{l_{i,j}(e), r_{i,j}(e)+1}$.
- Edges from $R_{1,x}$ are matched to edges from $Q[r_{i,j}(e)+1, j)$ contributing $S(R_{1,x})_{r_{i,j}(e)+1,j}$.

Note that we allow edge e_x to be matched to some other edge than e from $Q[l_{i,j}(e), r_{i,j}(e)+1)$. However, the cost of such matching will be at least as good as the cost of best matching that matches e_x to e, thus it does not affect the correctness of our algorithm.

This gives us that we can compute $S(T_2)$ from all $\hat{S}(\mathrm{sub}(e_i))$ in $\widetilde{\mathcal{O}}(n^3\Delta)$. To optimize this, we can use a similar idea to the one from Algorithm 1. Instead of calculating all the cells from $S(T_2)$ separately, we can do it globally. We iterate through:

- First edge e_x from path p that is matched.
- Edge $e \in Q$ such that e_x is matched to e.
- Occurrence $(I(e)_c, I(e)_{c+1})$ of the edge e in the doubled Euler tour.
- Cost a of matching edges from $L_{1,x}$.
- Cost b of matching edges from $R_{1,x}$.

Now we can find the shortest segment $[i, j)$, such that it contains $(I(e)_c, I(e)_{c+1})$, the cost of matching $L_{1,x}$ to $Q[i, I(e)_c)$ is at least a and the cost of matching $R_{1,x}$ to $Q[I(e)_{c+1} + 1, j)$ is at least b. Then we can use a single rangemax operation to update all valid cells $S(T_2)_{i',j'}$ for which $i' \leq i$ and $j \leq j'$. This gives us that we can compute $S(T_2)$ from $\hat{S}(\text{sub}(e_i))$'s in $\widetilde{\mathcal{O}}(n\Delta^3)$ time. The pseudocode for this part can be found in the full version of this paper [16].

Now, it remains to show how to compute all $\hat{S}(\text{sub}(e_i))$'s. We compute them starting with e_k and going up to e_1. Thus, assume that we have already computed $\hat{S}(\text{sub}(e_i))$'s for all i greater than some x. We want to compute $\hat{S}(\text{sub}(e_x))$. Let us consider two cases:

(a) None of the edges $e_{x+1}, \ldots, e_k$ is matched.

(b) At least one of the edges $e_{x+1}, \ldots, e_k$ is matched.

Both of these cases can be computed using similar ideas that we used to compute $S(T_2)$ from all $\hat{S}(\text{sub}(e_i))$. In the full version of this paper [16], we present how to compute case **(a)** in $\widetilde{\mathcal{O}}(n\Delta^2)$ time and case **(b)** in $\widetilde{\mathcal{O}}(n\Delta^3)$ time.

Thus, we can compute single $\hat{S}(\text{sub}(e_x))$ in $\widetilde{\mathcal{O}}(n\Delta^3)$ time and all of them in $\widetilde{\mathcal{O}}(n\Delta^4)$ time. Therefore, a whole single transition of type II can be computed in $\widetilde{\mathcal{O}}(\text{MUL}(\Delta, n) + n\Delta^4)$ time, which finishes the proof of Theorem 16.

5 Final remarks

We have presented the first truly subcubic algorithm for the unweighted variant of the unrooted tree edit distance. However, as mentioned before our algorithm has a slightly worse exponent than the best-known algorithm for rooted trees. The difference comes from Theorem 16 where we have time complexity $\widetilde{\mathcal{O}}(\text{MUL}(\Delta, n) + n\Delta^4)$, while Mao has $\widetilde{\mathcal{O}}(\text{MUL}(\Delta, n) + n\Delta^3)$ in corresponding theorem. It will be interesting to see if our algorithm could be optimized to match the time complexity of Mao's algorithm.

For the weighted tree edit distance probably the most interesting open problem is a question whether there exists a weakly subcubic algorithm for that version. One of the lower bounds [2] that claims this problem cannot be solved in a truly subcubic time is based on APSP conjecture. However, for APSP weakly subcubic time algorithms are already known. Williams [20] showed that the APSP problem can be solved in $\mathcal{O}(n^3/2^{\Omega(\sqrt{\log n})})$ time. Thus, we can try to obtain a weakly subcubic algorithm for the weighted tree edit distance by showing a reduction to the APSP problem.

References

1 J. Bellando and R. Kothari. Region-based modeling and tree edit distance as a basis for gesture recognition. In *Proceedings 10th International Conference on Image Analysis and Processing*, pages 698–703, 1999. doi:10.1109/ICIAP.1999.797676.

2 Karl Bringmann, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless apsp can). *ACM Trans. Algorithms*, 16(4), July 2020. doi:10.1145/3381878.

3 Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed xml. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 141–152. Morgan Kaufmann, 2003. doi:10.1016/B978-012722442-8/50021-5.

4 Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 90–101, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. doi:10.5555/645925.671669.

5 Weimin Chen. New algorithm for ordered tree-to-tree correction problem. *J. Algorithms*, 40(2):135–158, 2001. doi:10.1006/jagm.2001.1170.

6 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1529–1542, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520057.

7 Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, 6(1), December 2010. doi:10.1145/1644015.1644017.

8 Bartlomiej Dudek and Pawel Gawrychowski. Edit Distance between Unrooted Trees in Cubic Time. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.45.

9 Anita Dürr. Improved bounds for rectangular monotone min-plus product and applications. *Inf. Process. Lett.*, 181:106358, 2023. doi:10.1016/j.ipl.2023.106358.

10 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and Senthilmurugan Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57, November 2009. doi:10.1145/1613676.1613680.

11 Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in rna secondary structures. In *Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003*, pages 159–168, 2003. doi:10.1109/CSB.2003.1227315.

12 Philip Klein, Srikanta Tirthapura, Daniel Sharvit, and Ben Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 696–704, USA, 2000. Society for Industrial and Applied Mathematics. doi:10.5555/338219.338628.

13 Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms - ESA '98, 6th Annual European Symposium, Venice, Italy, August 24-26, 1998, Proceedings*, volume 1461 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 1998. doi:10.1007/3-540-68530-8_8.

14 Philip N. Klein, Thomas B. Sebastian, and Benjamin B. Kimia. Shape matching using edit-distance: An implementation. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 781–790, USA, 2001. Society for Industrial and Applied Mathematics. doi:10.5555/365411.365779.

15 Xiao Mao. Breaking the cubic barrier for (unweighted) tree edit distance. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 792–803. IEEE, 2021. doi:10.1109/FOCS52979.2021.00082.

16 Krzysztof Pióro. Subcubic algorithm for (unweighted) unrooted tree edit distance, 2023. arXiv:2304.08632.

17 Thomas Sebastian, Philip Klein, and Benjamin Kimia. Recognition of shapes by editing shock graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26:550–571, June 2004. doi:10.1109/TPAMI.2004.1273924.

18 Bruce A. Shapiro and Kaizhong Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Bioinformatics*, 6(4):309–318, October 1990. doi:10.1093/bioinformatics/6.4.309.

19 Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, July 1979. doi:10.1145/322139.322143.

20 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 664–673, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591811.

21 Kaizhong Zhang and Dennis E. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989. doi:10.1137/0218082.