Faster Weighted and Unweighted Tree Edit Distance and APSP Equivalence*

Jakob Nogler
ETH Zurich
Zurich, Switzerland
jnogler@ethz.ch

Adam Polak
Bocconi University
Milan, Italy
adam.polak@unibocconi.it

Barna Saha
University of California at San Diego
La Jolla, USA
bsaha@ucsd.edu

Virginia Vassilevska Williams
Massachusetts Institute of Technology
Cambridge, USA
virgi@mit.edu

Yinzhan Xu
University of California at San Diego
La Jolla, USA
xyzhan@ucsd.edu

Christopher Ye
University of California at San Diego
La Jolla, USA
czye@ucsd.edu

Abstract

The tree edit distance (TED) between two rooted ordered trees with n nodes labeled from an alphabet Σ is the minimum cost of transforming one tree into the other by a sequence of valid operations consisting of insertions, deletions and relabeling of nodes. The tree edit distance is a well-known generalization of string edit distance and has been studied since the 1970s. Its running time has seen steady improvements starting with an $O(n^6)$ algorithm [Tai, J.ACM 1979], improved to $O(n^4)$ [Shasha, Zhang, SICOMP 1989] and to $O(n^3 \log n)$ [Klein, ESA 1998], and culminating in an $O(n^3)$ algorithm [Demaine, Mozes, Rossman, Weimann, ACM TALG 2010]. The latter is known to be optimal for any dynamic programming based algorithm that falls under a certain decomposition framework that captures all known sub-n^4 time algorithms. Fine-grained complexity casts further light onto this hardness showing that a truly subcubic time algorithm for TED implies a truly subcubic time algorithm for All-Pairs Shortest Paths (APSP) [Bringmann, Gawrychowski, Mozes, Weimann, ACM TALG 2020]. Therefore, under the popular APSP hypothesis, a truly subcubic time algorithm for TED cannot exist. However, unlike many problems in fine-grained complexity for which conditional hardness based on APSP also comes with equivalence to APSP, whether TED can be reduced to APSP has remained unknown.

In this paper, we resolve this. Not only we show that TED is fine-grained *equivalent* to APSP, our reduction is tight enough, so that combined with the fastest APSP algorithm to-date [Williams, SICOMP 2018] it gives the first ever subcubic time algorithm for TED running in $n^3/2^{\Omega(\sqrt{\log n})}$ time.

We also consider the unweighted tree edit distance problem in which the cost of each edit (insertion, deletion, and relabeling) is one. For unweighted TED, a truly subcubic algorithm is known due to Mao [Mao, FOCS 2022], and later improved slightly by Dürr [Dürr, IPL 2023] to run in $O(n^{2.9148})$ time. Since their algorithm uses bounded monotone min-plus product as a crucial subroutine, and the best running time for this product is $\tilde{O}(n^{\frac{3+\omega}{2}}) \leq O(n^{2.6857})$ (where ω is the exponent of fast matrix multiplication), the much higher running time of unweighted TED remained unsatisfactory. In this work, we close this gap and give an algorithm for unweighted TED that runs in $\tilde{O}(n^{\frac{3+\omega}{2}})$ time.

CCS Concepts

• **Theory of computation** → *Divide and conquer*; **Graph algorithms analysis**; Dynamic programming.

Keywords

Tree Edit Distance, Fine-grained Complexity, Divide and conquer

ACM Reference Format:
Jakob Nogler, Adam Polak, Barna Saha, Virginia Vassilevska Williams, Yinzhan Xu, and Christopher Ye. 2025. Faster Weighted and Unweighted Tree Edit Distance and APSP Equivalence. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25), June 23–27, 2025, Prague, Czechia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3717823.3718116

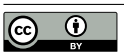
1 Introduction

First introduced by Selkow in the late 1970s [55] as a generalization of the more than classical (String) Edit Distance Problem (ED), the Tree Edit Distance Problem (TED) is a problem of significant interest with applications spanning computational biology [36, 38, 56, 67], structured data analysis [14, 17, 29], image processing [10, 41, 42, 53], compiler optimization [23] and more.

In the classical formulation of TED, two rooted trees, $\mathbf{T}$ and $\mathbf{T}'$, are given, with nodes arranged in a left-to-right order and labeled from a set Σ. The goal is to compute the *tree edit distance* between $\mathbf{T}$ and $\mathbf{T}'$, denoted by $\mathrm{ed}(\mathbf{T}, \mathbf{T}')$, defined as the minimum cost required to transform $\mathbf{T}$ into $\mathbf{T}'$ using a sequence of valid operations, which can be of three types: changing a label ℓ to ℓ' at a cost $\delta(\ell, \ell')$; removing a vertex with label ℓ at a cost $\delta(\ell, \varepsilon)$, while reattaching its children to its parent in the original order; or inserting a vertex with label ℓ at a cost $\delta(\varepsilon, \ell)$ between an existing node and a subsequence of consecutive children of that node. In the *unweighted tree edit distance* problem, we specify all operations to have cost 1.

Over the past three decades, the algorithms for TED have been progressively improved, culminating in the current best-known $O(n^3)$ time algorithm by Demaine, Mozes, Rossman, and Weinmann [23] for two trees on n nodes (see table 1 for a summary).

*The full version of the paper is available at https://arxiv.org/abs/2411.06502.



STOC '25, Prague, Czechia

ACM ISBN 979-8-4007-1510-5/25/06
https://doi.org/10.1145/3717823.3718116

For unweighted TED, a slightly subcubic running time $O(n^{2.9546})$ was recently shown by Mao [45] and later improved by Dürr [28] to $O(n^{2.9148})$. Nevertheless, even the $O(n^3)$ time algorithm for the more general weighted TED was not easy to obtain. The previous best [39] ran in $O(n^3 \log n)$ time, and it is unclear whether further logs can be shaved.

Question 1: Is there an $o(n^3)$ time algorithm for TED?

To address this, [12] use fine-grained complexity. They show that a truly subcubic time ($O(n^{3-\varepsilon})$ for constant $\varepsilon > 0$) algorithm for TED would imply a truly subcubic time algorithm for the All-Pairs Shortest Paths (APSP) problem, thus refuting the popular APSP hypothesis of fine-grained complexity (see the survey [63]) which states that $n^{3-o(1)}$ is needed for APSP on n-node graphs in the word-RAM model of computation.

This reduction explained why the exponent of the running time is stuck at 3 but did not address whether any tiny sub-polynomial improvement can be obtained over n^3. More frustratingly, no reduction *from* TED *to* APSP is known to exist. This makes TED seem *different* from the other problems for which APSP-based conditional lower bounds have been proven (e.g. graph radius, Wiener index, dynamic maximum matching etc., see [2, 37, 63, 64]) which are all either known to be fine-grained *equivalent* to APSP or whose hardness can be based on even harder problems, such as OMv [37], or the Exact Triangle problem (which is known to be at least as hard as both 3SUM and APSP, see [63]). A tantalizing open question is thus:

Question 2: Is TED fine-grained equivalent to APSP, or can its hardness be based on a harder problem such as Exact Triangle?

TED was originally conceived as a generalization of string Edit Distance, and the latter problem has been shown to be very hard within fine-grained complexity: Edit Distance requires $n^{2-o(1)}$ not only under the Orthogonal Vectors conjecture and the Strong Exponential Time Hypothesis (SETH) [9] but also under even more believable hypotheses such as NC-SETH, and even $O(n^2/\log^c(n))$ time algorithms for Edit Distance for large enough c would imply so-far unattainable Circuit Lower Bounds [1].

Because of all this, it is conceivable that TED is similarly difficult and that (w.r.t. Question 1) only small polylogarithmic improvements are potentially attainable and (w.r.t. Question 2) it is truly harder than APSP.

A third question concerns the unweighted TED problem. Mao [45] showed that a truly subcubic running time is possible for the problem. The current record for the running time is by Dürr [28] and is $\tilde{O}(n^{(4\omega+12)/(\omega+5)}) = O(n^{2.9148})$.[1] Here, ω is the $n \times n$ matrix multiplication exponent, where one can multiply two $n \times n$ matrices in $O(n^{\omega+\varepsilon})$ time for all $\varepsilon > 0$.[2] The current bound on ω, due to [6], is $\omega \leq 2.371339$.

There are many structured variants of problems related to APSP and for a very large number of them, shortly after a truly subcubic time algorithm was found, an $\tilde{O}(n^{(3+\omega)/2})$ time (or better) algorithm was also found. There are many such examples, we present a few:

- The All Pairs Bottleneck Paths [61] and All pairs nondecreasing paths [62] problems were first shown to have truly subcubic time algorithms in the first decade of the century and are now both known to be solvable in $\tilde{O}(n^{(3+\omega)/2})$ time [24, 25].
- The Min-Plus product of $n \times n$ matrices (given A, B, compute C with $C[i,j] = \min_k(A[i,k] + B[k,j])$ is known to be equivalent to APSP in n-node graphs [30] so that under the APSP Hypothesis it requires $n^{3-o(1)}$ time. Various structured variants of the Min-Plus matrix product have been shown to have truly subcubic time algorithms, e.g. when the matrices have "bounded differences" [13] or whose entries are bounded by $O(n)$ and are monotone (non-decreasing in the rows or columns) [35, 65]. Later, [19] showed that these variants can be solved in $\tilde{O}(n^{(3+\omega)/2})$ time. These structured Min-Plus products have many applications for a variety of fundamental problems, e.g. to many sequence similarity problems such as Language Edit Distance (aka Scored Parsing [3, 11, 49]), RNA Folding [5, 47, 49, 66, 69] and Dyck Edit Distance [11, 22, 31, 48]. All of these problems now have $\tilde{O}(n^{(3+\omega)/2})$ time algorithms.

As TED seems related to APSP and since its unweighted version is now known to have a truly subcubic algorithm, a natural question is:

Question 3: Can unweighted TED be solved in $\tilde{O}(n^{(3+\omega)/2})$ time?

Mao's truly subcubic time algorithm for unweighted TED [45] can be viewed as a reduction to Bounded Monotone Min-Plus product which can be solved in $\tilde{O}(n^{(3+\omega)/2})$ time. The reduction, however, is not efficient enough to achieve the same running time for unweighted TED. One way to resolve Question 3 is by presenting a tight reduction. Is such a reduction possible?

Our Results. We resolve all three questions above. Similarly to the prior work on TED, we focus on solving the more general edit distance problem on ordered *forests* (rather than just single trees): collections of rooted trees with a left-to-right ordering. Our first main theorem is a fine-grained reduction from (forest) TED to the Min-Plus product problem mentioned earlier:

Min-Plus Matrix Multiplication (MUL)
Input: Two $m \times m$ matrices $A = (a_{i,j})$ and $B = (b_{i,j})$.
Output: The distance matrix $C = A \star B$, where $C = (c_{i,j})$ is defined as $c_{i,j} = \min_{k \in [1\,..\,m]}\{a_{i,k} + b_{k,j}\}$.

The formal theorem for our reduction is as follows:

Theorem 1.1. *Let $\mathbf{F}, \mathbf{F}'$ be forests of size $n = |\mathbf{F}|, m = |\mathbf{F}'|$ with $n \geq m$. Then, there is an algorithm computing Tree Edit Distance between $\mathbf{F}, \mathbf{F}'$ in time $\tilde{O}\left((n/m)^{1+o(1)} \cdot \left(\mathsf{T}_{MUL}(m) + m^{2+o(1)}\right)\right)$.*

If $m = n$, the theorem states that TED on n-node forests can be solved in $\tilde{O}(\mathsf{T}_{\mathsf{MUL}}(n))$. We thus complete the missing direction in establishing the equivalence between TED and APSP, resolving Question 2. In fact, because our reduction is very efficient and only

[1] $\tilde{O}$ hides poly-logarithmic factors.
[2] Slightly abusing notation, we omit this ε from the rest of the paper, which is in line with the literature. The reader should be aware that in many running times in the literature that one states in terms of ω, a secret ε is always hiding. The running times with a numerical-valued exponent use a strict upper bound on ω (such as the $O(n^{2.9546})$ running time for unweighted TED of [45]), so this ε does not appear.

Table 1: Computational bounds of TED across different works. In the unweighted setting, complexities are computed using the best known bound on the matrix multiplication exponent $\omega \leq 2.371339$ from [6]. The stated bound for [45] additionally uses results on Rectangular Bounded Monotone Min-plus Product by Dürr [28].

Work	Setting	Complexity
Tai [58]	weighted	$O(n^6)$
Shasha, Zhang [70]	weighted	$O(n^4)$
Klein [39]	weighted	$O(n^3 \log n)$
Demaine, Mozes, Rossman, Weimann [23]	weighted	$O(n^3)$
Bringmann, Gawrychowski, Mozes, Weinmann [12]	weighted	no $O(n^{3-\varepsilon})$ algorithm under APSP
Mao [45]	unweighted	$\tilde{O}(n^{(4\omega+12)/(\omega+5)}) = O(n^{2.9148})$
This work	**weighted**	$\mathbf{n^3/2^{\Omega(\sqrt{\log n})}}$
This work	**unweighted**	$\tilde{O}(\mathbf{n^{(3+\omega)/2}}) = O(\mathbf{n^{2.6857}})$

adds polylogarithmic factors over the APSP running time, we are also able to resolve Question 1 using Williams' [68] $m^3/2^{\Omega(\sqrt{\log m})}$ running time for APSP.

Corollary 1.2. *Let $\mathbf{F}, \mathbf{F}'$ be forests. Then, there is an algorithm for TED running in time $n^3/2^{\Omega(\sqrt{\log n})}$, where $n = \max(|\mathbf{F}|, |\mathbf{F}'|)$.*

Beyond providing a faster algorithm for TED, corollary 1.2 underscores once again the difference in nature between ED and TED by demonstrating that, while we (probably) cannot eliminate an arbitrary number of logarithmic factors for the former, we can do so for the latter.

The last several TED algorithms (prior to ours) all use the same dynamic programming approach which was formalized as a decomposition framework for solving TED [26, 27]. Formalizing the framework allowed for proving lower bounds on the running time of any algorithm that falls into that framework. The first such lower bound was by [26] who showed such algorithms must take $\Omega(n^2 \log^2 n)$ time. The approach culminated in obtaining an $\Omega(n^3)$ time lower bound [23] for any algorithm that falls within the framework. Since our algorithm runs faster than cubic time, it falls outside the framework.

We also resolve Question 3 by providing an $\tilde{O}(n^{(3+\omega)/2})$ time algorithm for unweighted TED. This is a significant improvement over Mao's result [45]. We achieve our result via a tight reduction to the Bounded Monotone Min-Plus product problem.

Bounded Monotone MUL. (MonMUL)
Input: An $m \times n$ matrix $A = (a_{i,j})$ and an $n \times \ell$ matrix $B = (b_{i,j})$ s.t. either for all $i \in [m]$, $j \in [n]$, $a_{i,j} \leq a_{i,j+1}$ ("row-monotone") or for all $i \in [m]$, $j \in [n]$, $a_{i,j} \leq a_{i+1,j}$ ("column-monotone").
Output: The distance matrix $C = A \star B$, where $C = (c_{i,j})$ is defined as $c_{i,j} = \min_{k \in [1..n]}\{a_{i,k} + b_{k,j}\}$.

When $m = n = \ell$ and $D = O(n)$, MonMUL can be solved in $\tilde{O}(n^{(3+\omega)/2})$ [19].

We denote by $\mathsf{T}_{\mathsf{MonMUL}}(m, n, \ell, D)$ the running time for computing MonMUL. When $m = n = \ell = D$, we simply write $\mathsf{T}_{\mathsf{MonMUL}}(m)$ for the running time of MonMUL.

Theorem 1.3. *Let $\mathbf{F}, \mathbf{F}'$ be forests of size $n = |\mathbf{F}|, m = |\mathbf{F}'|$ with $n \geq m$. Suppose that $\mathsf{T}_{MonMUL}(N, N, N, \mathrm{D}) = O(f(N)g(\mathrm{D}))$. Then, there is an algorithm computing Unweighted Tree Edit Distance between $\mathbf{F}, \mathbf{F}'$ in time $\tilde{O}\left((n/m)^{1+o(1)} \cdot \left(\mathsf{T}_{MonMUL}(m) + m^{2+o(1)} g(m)\right)\right)$.*

The best known bound for the MonMUL running time [19, 28] is $\mathsf{T}_{\mathsf{MonMUL}}(N, N, N, D) = \tilde{O}(N^{(2+\omega)/2} D^{1/2})$. Hence, theorem 1.3 implies that given two forests of equal size, we can compute Unweighted Tree Edit Distance in $\tilde{O}(m^{(3+\omega)/2})$ time, matching the complexity of bounded monotone min-plus product [19] and resolving Question 3.

More generally, we obtain the following result.

Corollary 1.4. *There is an algorithm for Unweighted TED running in time $n^{1+o(1)} m^{(1+\omega)/2}$, where $n = \max(|\mathbf{F}|, |\mathbf{F}'|)$ and $m = \min(|\mathbf{F}|, |\mathbf{F}'|)$.*

At the Core of Our Results: TED Alignment Graphs. The edit distance between two length-n strings can be represented by examining the classical dynamic computation on an $[1..(n+1)] \times [1..(n+1)]$ grid. The solution is then traced by following the shortest path from $(1, 1)$ to $(n+1, n+1)$. Such graph-based representation is commonly referred to as an *alignment graph*.

The power of this representation is evident in the numerous results for string ED that have either emerged directly from this visualization or can be clearly illustrated through it (including but not limited to [15, 15, 16, 20, 32–34, 43, 44, 59]). Consequently, it is not surprising that specific properties of alignment graphs, such as border-to-border distances, play a central role. For string ED, these distances can be computed in $O(n^2)$ time [7, 40, 51, 59].

Alignment graphs for TED were introduced as a counterpart to those for ED and serve as a visualization tool for TED solutions. However, they appear predominantly in less recent works [8, 46, 60] and have played a less central role than in ED.

In this work, we reestablish the alignment graph for TED as a central tool and demonstrate that it provides a more powerful language and visualization framework for capturing the combinatorial structure of solutions than previously recognized. As a technical contribution, we show that in the alignment graph for TED, border-to-border distances can be computed in APSP time, while for unweighted TED, they can be determined in $\tilde{O}(n^{(3+\omega)/2})$.

Prior Attempts to Reduce TED to APSP and MUL. At least two prior papers have attempted to leverage min-plus products for computing

TED. The first of these works was by Chen [18], who introduced a dynamic programming scheme formulated using applications of MUL. However, this did not result in a true reduction, as the algorithm ran in $O(n^4)$ time (Chen initially claimed a running time of $O(n^{3.5})$, but this was later corrected in [52]). The second work, by [45], uses bounded difference min-plus products (a special case of bounded monotone min-plus multiplication) to achieve truly subcubic time for the unweighted (and very small weight) tree edit distance problem. However, Mao's scheme was not powerful enough to achieve a fine-grained reduction from the general weight case of TED to MUL.

Other Related Works. Recently, there has been significant interest in TED approximation algorithms [11, 54] as well as TED when the distance is bounded [4, 20, 21].

2 Techical Overview

2.1 Similarity and Alignment Graphs

We examine TED through its *mapping formulation* (as in [45]). This means that rather than thinking of TED as finding a least-cost transformation via insertions, deletions, and matches, we shift our perspective towards seeking a maximum weight mapping between two forests.

String Similarity. To ease into this formulation, we first discuss the mapping formulation of (weighted) string edit distance. In the (String) Edit Distance Problem (ED), we are given two strings $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_m$, and a cost function δ. The goal is to find the least cost needed to transform A into B, by substituting, inserting or deleting characters. The costs $\delta(a_i, b_j)$, $\delta(a_i, \varepsilon)$, and $\delta(\varepsilon, b_j)$ describe the cost of substituting a_i with b_j, deleting a_i, and inserting b_j, respectively.

We define $\eta(a_i, b_j) \coloneqq \delta(a_i, \varepsilon) + \delta(\varepsilon, b_j) - \delta(a_i, b_j)$ to be *the weight of matching a_i with b_j*. Determining the string edit distance between A and B translates into calculating $\mathsf{sim}(A, B)$, called the *similarity between A and B*, and defined as

$$\max_{\substack{i_1<\cdots<i_k\in[\,1\,..\,n\,]\\ j_1<\cdots<j_k\in[\,1\,..\,m\,]}} \left\{ \eta(a_{i_1}, b_{j_1}) + \eta(a_{i_2}, b_{j_2}) + \cdots + \eta(a_{i_k}, b_{j_k}) \right\}.$$

Thereby, we obtain $\mathsf{sim}(A, B) = \sum_i \delta(a_i, \varepsilon) + \sum_j \delta(\varepsilon, b_j) - \mathsf{ed}(A, B)$.

String Alignment Graphs. The value $\mathsf{sim}(A, B)$ is 0 if $|A| = 0$ or $|B| = 0$, otherwise it can be computed using the formula

$$\max\begin{Bmatrix} \mathsf{sim}(A[\,2\,..\,n\,], B),\\ \mathsf{sim}(A, B[\,2\,..\,m\,]),\\ \mathsf{sim}(A[\,2\,..\,n\,], B[\,2\,..\,m\,]) + \eta(a_1, b_1) \end{Bmatrix}.$$

These computations can be reformulated as finding a longest path on a directed weighted acyclic graph, commonly referred to as the *alignment graph*. This graph has as the vertex set the grid $[\,1\,..\,(n+1)\,] \times [\,1\,..\,(m+1)\,]$, where each node (i, j) is connected with an edge of weight zero to its right and upper neighbors, i.e., $(i+1, j)$ and $(i, j+1)$, if they are within the borders. Additionally, from each node $(i, j) \in [\,1\,..\,n\,] \times [\,1\,..\,m\,]$, there is an edge to $(i+1, j+1)$ with weight $\eta(a_i, b_j)$. Computing $\mathsf{sim}(A, B)$ translates to finding a longest path between $(1, 1)$ and $(n+1, m+1)$ in this graph. Each time the longest path traverses an edge from (i, j) to $(i+1, j+1)$ we map a_i to b_j.

It is worth noting that in the literature several studies [7, 40, 51, 59] have focused on computing all longest distances from the lower-left border to the upper-right border in an alignment graph. For both weighted and unweighted edit distance, this task can be accomplished in time $O((n+m)^2)$.

Tree Similarity. Similarly, given two forests $\mathbf{F}$ and $\mathbf{F}'$, define $\eta(v, v') \coloneqq \delta(v, \varepsilon) + \delta(\varepsilon, v') - \delta(v, v')$. The mapping we consider for TED corresponds to two sequences of distinct nodes $v_1, \ldots v_k \in \mathbf{F}$ and $v'_1, \ldots v'_k \in \mathbf{F}'$ such that for all $1 \le i < j \le k$:

- v_i is an ancestor of v_j in $\mathbf{F}$ iff v'_i is an ancestor of v'_j in $\mathbf{F}'$,
- v_j is an ancestor of v_i in $\mathbf{F}$ iff v'_j is an ancestor of v'_i in $\mathbf{F}'$, and
- if neither v_i nor v_j is the ancestor of the other, v_i comes before v_j in the pre-order traversal of $\mathbf{F}$ iff v'_i comes before v'_j in the pre-order traversal of $\mathbf{F}$.

The *similarity between $\mathbf{F}$ and $\mathbf{F}'$*, denoted as $\mathsf{sim}(\mathbf{F}, \mathbf{F}')$, maximizes $\sum_{1\le i\le k} \eta(v_i, v'_i)$, where the maximum is taken over all such mappings.

As before, $\mathsf{sim}(\mathbf{F}, \mathbf{F}') = \sum_{v\in\mathbf{F}} \delta(v, \varepsilon) + \sum_{v'\in\mathbf{F}'} \delta(\varepsilon, v') - \mathsf{ed}(\mathbf{F}, \mathbf{F}')$.

Forest Alignment Graphs. Suppose we are given the similarities $\mathsf{sim}(\mathsf{sub}(v), \mathsf{sub}(v'))$ for all $v \in \mathbf{F}$ and $v' \in \mathbf{F}'$, where $\mathsf{sub}(v)$ indicates the subtree rooted at node v. Then, the value $\mathsf{sim}(\mathbf{F}, \mathbf{F}')$ can be computed using Shasha and Zhang's recurrence scheme [57]. Given forests $\mathbf{F}$ and $\mathbf{F}'$ with pre-order $v_1, \ldots, v_{|\mathbf{F}|}$ and $v'_1, \ldots, v'_{|\mathbf{F}'|}$, they set $\mathsf{sim}(\mathbf{F}, \mathbf{F}') = 0$ if $\mathbf{F} = \varnothing$ or $\mathbf{F}' = \varnothing$, otherwise, they compute $\mathsf{sim}(\mathbf{F}, \mathbf{F}')$ using the formula

$$\max\begin{Bmatrix} \mathsf{sim}(\mathbf{F} \setminus v_1, \mathbf{F}'),\\ \mathsf{sim}(\mathbf{F}, \mathbf{F}' \setminus v'_1),\\ \mathsf{sim}(\mathbf{F} \setminus \mathsf{sub}(v_1), \mathbf{F}' \setminus \mathsf{sub}(v'_1)) + \mathsf{sim}(\mathsf{sub}(v_1), \mathsf{sub}(v'_1)) \end{Bmatrix}.$$

Once again, these computations can be rephrased as finding the longest path in a directed acyclic graph with a grid as the vertex set, but only under the condition that we have the values $\mathsf{sim}(\mathsf{sub}(v), \mathsf{sub}(v'))$ for all $v \in \mathbf{F}$ and $v' \in \mathbf{F}'$. With this condition in place, we can construct a grid $[\,1\,..\,(|\mathbf{F}|+1)\,] \times [\,1\,..\,(|\mathbf{F}'|+1)\,]$. Similar to the case of strings, each node (i, i') is connected to its right and upper neighbors with an edge of weight zero, i.e., $(i+1, i')$ and $(i, i'+1)$ (if they are within the borders). Additionally, from each node $(i, i') \in [\,1\,..\,|\mathbf{F}|\,] \times [\,1\,..\,|\mathbf{F}'|\,]$, there is an edge to (j, j') with weight $\mathsf{sim}(\mathsf{sub}(v_i), \mathsf{sub}(v'_{i'}))$. Here, j and j' are the smallest integers $j \ge i$ and $j' \ge i'$ such that $v_j \notin \mathsf{sub}(v_i)$ and $v'_{j'} \notin \mathsf{sub}(v'_{i'})$.

2.2 Warm-up: TED on Caterpillar Trees

To better understand our algorithm for TED, let us first examine a more restricted yet significant case.

Caterpillar Trees. Caterpillar trees consist of a central path with nodes labeled $c_1, c_2, \ldots, c_n$, where c_1 is the root. Additional, each central node c_i has both a left child l_i and a right child r_i.[3]

Let us examine the similarity mapping for two such caterpillar trees $\mathbf{T}$ and $\mathbf{T}'$, of size $|\mathbf{T}| = 3n, |\mathbf{T}'| = 3n'$ with node sets

[3] In standard literature, caterpillar trees can take a more general form. Here, we focus on a specific type of caterpillar tree, though we continue to refer to them simply as caterpillar trees.

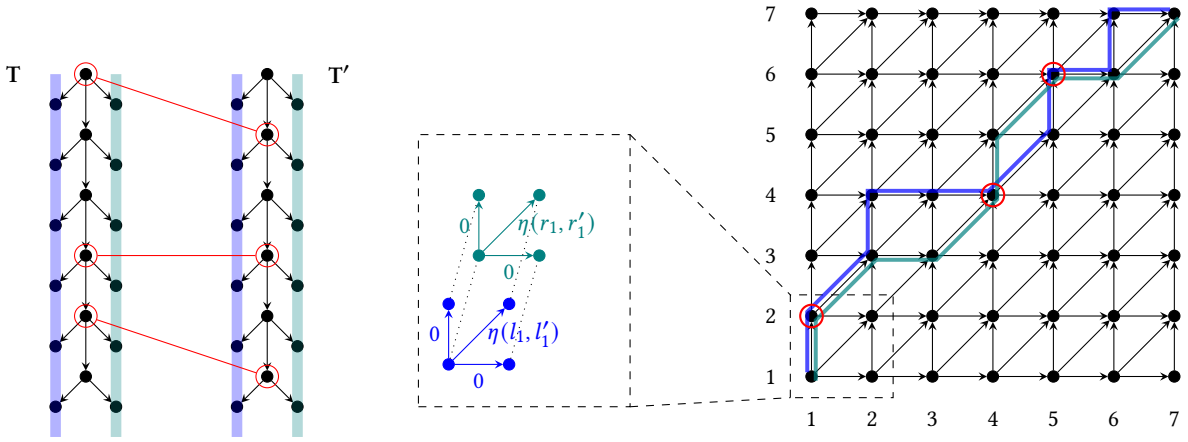


Figure 1: Overlaying the alignment graphs for string edit distance for (L, L') and (R, R') leads to an intuitive visualization of TED on caterpillar trees under the assumption that central nodes are only mapped to central nodes, left children only to left ones, and right children only to right ones (assumption A). The problem can be visualized as two paths in the two graphs, which, whenever they intersect, allow mapping of central nodes to central nodes.

$\{l_i, c_i, r_i\}_{i \in [\,1\,..\,n\,]}$ and $\{l'_i, c'_i, r'_i\}_{i \in [\,1\,..\,n'\,]}$. We examine such a mapping under an additional simplifying assumption: nodes from one side of the caterpillar $\mathbf{T}$ are always mapped to nodes of the same side of the caterpillar $\mathbf{T}'$.

Assumption A. *The nodes $\{c_i\}_i$, $\{l_i\}_i$ and $\{r_i\}_i$ are always only mapped by $\mathsf{sim}(\mathbf{T}, \mathbf{T}')$ to nodes in $\{c'_i\}_i$, $\{l'_i\}_i$ and $\{r'_i\}_i$, respectively.*

Under assumption A, suppose $\mathsf{sim}(\mathbf{T}, \mathbf{T}')$ maps c_i to $c'_{i'}$ and c_j to $c'_{j'}$ for $i < j$, and no further node $c_{i+1}, \ldots, c_{j-1}$ is mapped. Then, the nodes from $l_i, \ldots, l_{j-1}$ are mapped to nodes from $l'_{i'}, \ldots, l_{j'-1'}$ as in $\mathsf{sim}(L[\,i\,..\,j\,), L'[\,i'\,..\,j'\,))$, where $L = l_1 \cdots l_n, L' = l'_1, \cdots l'_{n'}$ are strings built from the left children of the central nodes. Similarly, the nodes from $r_i, \ldots, r_{j-1}$ are mapped to nodes from $r'_{i'}, \ldots, r_{j'-1'}$ as in $\mathsf{sim}(R[\,i\,..\,j\,), R'[\,i'\,..\,j'\,))$, where $R = r_1 \cdots r_n, R' = r'_1, \cdots r'_{n'}$ are strings built from the right children of the central nodes.

This brings us to the visualization of $\mathsf{sim}(\mathbf{T}, \mathbf{T}')$ (under assumption A) illustrated in fig. 1. Consider a $[\,1\,..\,(n+1)\,] \times [\,1\,..\,(n'+1)\,]$ grid, and overlay on it the alignment graphs for string edit distance for (L, L') and (R, R'). Then, $\mathsf{sim}(\mathbf{T}, \mathbf{T}')$ can be visualized as two paths from $(1, 1)$ to $(n+1, n'+1)$ in the two respective alignment graphs. Whenever these paths intersect at a grid point (i, i'), $\mathsf{sim}(\mathbf{T}, \mathbf{T}')$ has the opportunity to map c_i to $c'_{i'}$, provided neither c_i nor $c'_{i'}$ has been mapped previously.

Henceforth, we denote by $\mathsf{sim}((x, x'), (y, y'))$ the maximum value achievable by a sum of three terms:

(1) the weight of a path from (x, x') to $(n+1, n'+1)$ in the alignment graph of $\mathsf{sim}(L, L')$;
(2) the weight of a path from (y, y') to $(n+1, n'+1)$ in the alignment graph of $\mathsf{sim}(R, R')$; and
(3) a sum of values of the form $\eta(c_i, c'_{i'})$ for (i, i') where the two paths intersect, provided each c_i and $c'_{i'}$ appears at most once.

By the previous discussion, $\mathsf{sim}(\mathbf{T}, \mathbf{T}') = \mathsf{sim}((1, 1), (1, 1))$ under assumption A.

Reduction of Caterpillar TED (under Assumption A) to APSP. To compute $\mathsf{sim}((1, 1), (1, 1))$, we utilize a divide-et-impera scheme. Given a rectangle in the grid defined by the lower-left corner (a, a') and the upper-right corner (b, b'), along with $\mathsf{sim}((x, x'), (y, y'))$ for all $(x, x'), (y, y')$ on the upper-right border of the rectangle, our task is to determine $\mathsf{sim}((x, x'), (y, y'))$ for all $(x, x'), (y, y')$ on the lower-left border of the rectangle. Note that computing the inputs becomes trivial when the rectangle we are considering is the whole grid.

Let $a < r < b$. We split the rectangle vertically into two smaller rectangles (see fig. 2): one with corners (a, a') and (r, b') and the other with corners (r, a') and (b, b'). We aim to recurse on those two subrectangles.

For the right subrectangle, we can directly recurse since its inputs are a subset of those for the full rectangle.

For the left subrectangle, we must compute its input before recursing on it. First, let us consider $\mathsf{sim}((x, x'), (y, y'))$ for all $(x, x'), (y, y') \in [\,a\,..\,r\,] \times b'$. These values corresponds to the case where both paths cross the upper border $[\,a\,..\,r\,] \times b'$ in fig. 2. Note that these values are part of the input for the entire rectangle.

Similarly, when both paths cross the right border of the left subrectangle, i.e. $r \times [\,a'\,..\,b'\,]$, no work needs to be done, as we can obtain the inputs from the right subrectangle's output.

Next, suppose the path in $\mathsf{sim}(L, L')$ crosses the upper border at (x, b') for some $x \in [\,a\,..\,r\,]$, and the path in $\mathsf{sim}(R, R')$ crosses the right border at (r, y') for some $y' \in [\,a'\,..\,b'\,]$. Then, we can decompose the path in $\mathsf{sim}(R, R')$ into two parts (see fig. 2), and compute $\mathsf{sim}((x, b'), (r, y'))$ as

$$\max_{(w, w') \in ([\,r\,..\,b\,] \times b') \cup (b \times [\,a'\,..\,b'\,])} \Big\{ \mathsf{sim}(R[\,r\,..\,w\,), R'[\,y'\,..\,w'\,)) + \mathsf{sim}((x, b'), (w, w')) \Big\}. \quad (1)$$

In this maximization, the latter summands are part of the input for the full rectangle, and the former summands are border-to-border paths in an alignment graph which can be computed in time

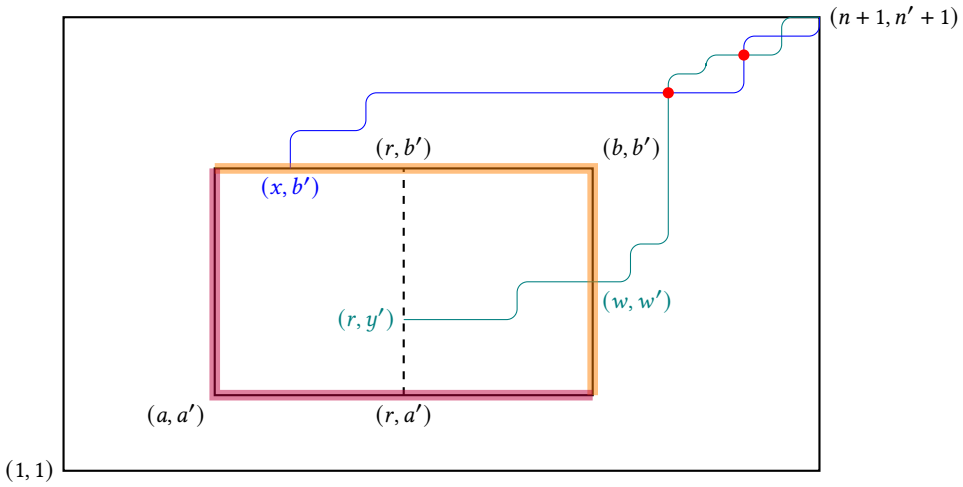


Figure 2: The figure illustrates an instance of the recursive scheme used to solve TED on caterpillars. The inputs can be visualized as fixing the starting points of the two paths on the upper right border (in orange) of the rectangle, and we are determining the maximum value achievable from there onwards. We are required to compute the same values for the lower left border (in purple). The divide-et-impera scheme divides the rectangle vertically into two smaller ones, parameterized by the corners (a, a'), (r, b') and (r, a'), (b, b'). The figure also demonstrates how to compute the inputs for the scheme on the left subrectangle for one specific case: one path leaves the upper border, and the other exits from the right border.

$O(m^2)$, where $m = \max(b-a, b'-a')$. Equation (1) can be computed concurrently for all such x and y' via a max-plus product[4] in time $O(\mathsf{T}_{\mathsf{MUL}}(m))$.

The case where the path in $\mathsf{sim}(L, L')$ crosses the right border of the left subrectangle, and the the path in $\mathsf{sim}(R, R')$ crosses the upper border of the left subrectangle can be handled symmetrically. This allows us now to recurse on the left subrectangle as well.

It remains to discuss how to patch together the outputs of the two subrectangles to get the outputs of the full rectangle. For the sake of brevity, we omit discussing this here, but it is not difficult to see that by employing similar calculations to those before (border-to-border paths in an alignment graph combined with the outputs of the two subrectangles via max-plus products), we can calculate $\mathsf{sim}((x, x'), (y, y'))$ for all (x, x'), (y, y') on the lower-left border of the rectangle, ignoring the contribution arising from c_a or $c'_{a'}$ being mapped to other central nodes. With some additional computations (which requires redefining $\mathsf{sim}((x, x'), (y, y'))$ to compute 2^4 values instead of one, depending on whether central nodes $c_a, c_b, c'_{a'}, c'_{b'}$ were already mapped or not), we can factor in these contributions and compute the desired output for the entire rectangle.

This shows how the divide-et-impera scheme can handle vertical cuts. By symmetry of the problem, the scheme can handle horizontal cuts as well, allowing us to split a single instance in four roughly equal parts to recurse on.

For a rough analysis, let us focus on the case when $n = n' = 2^k$ for which the scheme always recurses on squares. This results in the recurrence relation $\mathsf{T}(n) = 4 \cdot \mathsf{T}(n/2) + O(\mathsf{T}_{\mathsf{MUL}}(n))$, which implies $\mathsf{T}(n) = O(\mathsf{T}_{\mathsf{MUL}}(n))$.[5]

Getting Rid of Assumption A. Up to this point, we argued how to determine the similarity between two caterpillar trees, $\mathbf{T}$ and $\mathbf{T}'$, under assumption A. In the general case, the mapping of $\mathsf{sim}(\mathbf{T}, \mathbf{T}')$ (when observed from the root downwards) preserves this assumption up to some point, i.e., there exist a, b, a', b' such that nodes in $l_1, \ldots, l_{a-1}$ are only mapped to nodes in $l'_1, \ldots, l'_{a'-1}$, nodes in $r_1, \ldots, r_{b-1}$ are only mapped to nodes in $r'_1, \ldots, r'_{b'-1}$, nodes in $c_1, \ldots, c_{\min(a,b)-1}$ are only mapped to nodes in $c'_1, \ldots, c'_{\min(a',b')-1}$. Then, one of two cases can occur:

(1) l_a is mapped to $c'_{a'}$, nodes in $l_{a+1}, \ldots, l_{b-1}$ are exclusively mapped to nodes $r'_{a'-1}, \ldots, r'_{b'+1}$, and c_b is mapped to $r'_{b'}$. The first and third conditions are optional; if they do not hold, we include $l_a, r'_{a'}$, and $l_b, r'_{b'}$, respectively, in the middle condition.
(2) r_a is mapped to $c'_{a'}$, nodes in $r_{a-1}, \ldots, r_{b+1}$ are mapped to nodes $l'_{a'+1}, \ldots, l'_{b'-1}$, and c_b is mapped to $l'_{b'}$. As before, the first and third conditions are optional; if they do not hold, we include r_a, r_b and $l'_{a'}, l'_{b'}$, respectively, in the middle condition.

Without loss of generality, we focus on the former case, as the two cases are symmetric.

This scenario is depicted in fig. 3 and can be visualized by overlaying an additional string similarity graph onto the existing ones in the grid (specifically, the one involved in $\mathsf{sim}(L, \mathsf{rev}(R'))$, where $\mathsf{rev}(R')$ denotes the reversed string R'). When overlaying this similarity graph, we ensure that the indices of $\mathsf{rev}(R')$ align with the

[4] In max-plus products, the minimum operator is replaced by a maximum. Note that min and max products are equivalent, as one can be transformed into the other by appropriately reversing the signs of the matrices. For the remainder of this paper, we treat them as equivalent.

[5] Here, we assume $\mathsf{T}_{\mathsf{MUL}}(n) = \Omega(n^{2+\varepsilon})$ for some small $\varepsilon > 0$, a reasonable assumption given the widely accepted conjecture that no truly subcubic algorithms exist for MUL.

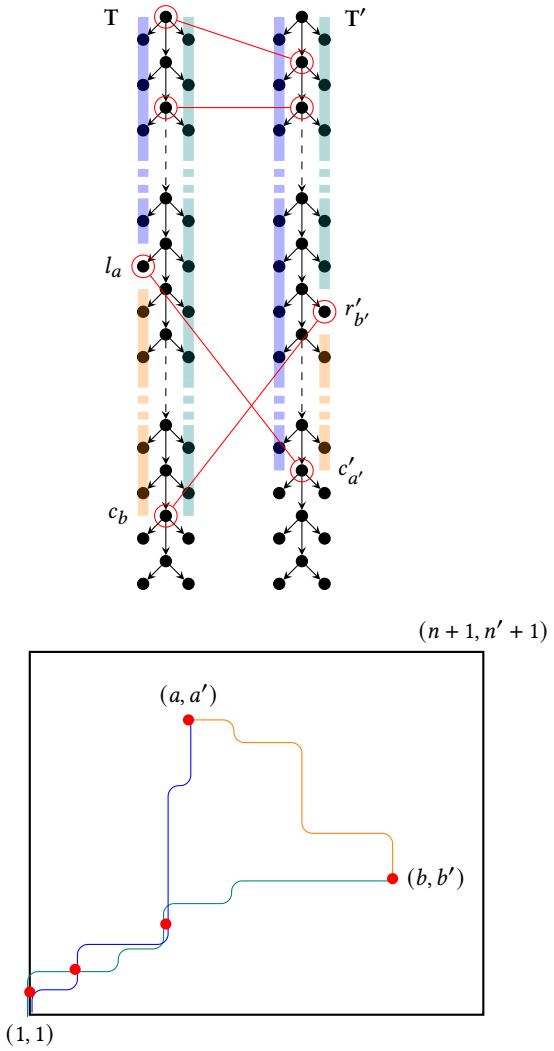


Figure 3: The general case for similarity mappings on caterpillar trees.

indexing of the grid, and the paths in this string alignment graph travel from the upper-left border to the lower-right border. The two paths that were previously observed now extend from $(1, 1)$ to (a, a') and (b, b'), where l_a is mapped to $c'_{a'}$ and c_b is mapped to $l'_{b'}$. Within this new alignment graph, we search for a path connecting $(a+1, a')$ with $(b, b'+1)$. If l_a is not mapped to $c'_{a'}$ or c_b is not mapped to $l'_{b'}$, then this path starts/ends at (a, a') and (b, b'), respectively.

The details necessary for considering this final path can be incorporated into the earlier divide-et-impera framework, although it necessitates additional inputs and outputs for the scheme. For the sake of brevity, we omit details here.

2.3 From TED on Caterpillar Trees to SED

Spine Edit Distance. Our algorithm for TED on caterpillar trees generalizes to a problem known as *Spine Edit Distance* (SED), initially proposed in [11].

In SED, besides two forests $\mathbf{F}$ and $\mathbf{F}'$, we are provided with a spine for each forest. A spine $\mathbf{S} \subseteq \mathbf{F}$ is any root-to-leaf path within a forest (in cases where the forest contains multiple trees, the spine can start from any root of a tree contained in the forest). In SED we are given the similarity between all pairs of subtrees of $\mathbf{F}$ and $\mathbf{F}'$, provided at least one of the two roots does not lie on a spine. The task is to compute the similarity for all missing pairs of subtrees. Put more formally:

Spine Edit Distance (SED)
Input: Two forests $\mathbf{F}, \mathbf{F}'$, two spines $\mathbf{S} \subseteq \mathbf{F}, \mathbf{S}' \subseteq \mathbf{F}'$, and $\mathsf{sim}(\mathsf{sub}(v), \mathsf{sub}(v'))$ for all $(v, v') \in (\mathbf{F} \times \mathbf{F}') \setminus (\mathbf{S} \times \mathbf{S}')$.
Output: $\mathsf{sim}(\mathsf{sub}(v), \mathsf{sub}(v'))$ for all $(v, v') \in \mathbf{S} \times \mathbf{S}'$.

As already noticed in [11], TED can be reduced to SED by employing appropriate tree decompositions. We prove the following:

Lemma 2.1. *Suppose there exists an algorithm for SED on two forests $\mathbf{H}, \mathbf{H}'$ running in time $\mathsf{T}_{SED}(m, m') = O(f(m)g(m'))$, where $m = |\mathbf{H}|, m' = |\mathbf{H}'|$ and $f(m) = \Omega(m), g(m') = \Omega(m')$ are some functions.*

Then, there is an algorithm for TED on two forests $\mathbf{F}, \mathbf{F}'$ running in time $O(f(n)g(n') \log^2 \max(n', n))$, where $n = |\mathbf{F}|, n' = |\mathbf{F}'|$.

Reducing spine edit distance to APSP. In the remaining part of this (sub)section, we describe the main ingredients we need to reduce SED to APSP.

We can assume, without loss of generality, that $\mathbf{F}$ and $\mathbf{F}'$ are trees (by adding a virtual root and defining weights accordingly to enforce their deletion).

Let us further decompose $\mathbf{F}$ and $\mathbf{F}'$ into $\mathbf{L}, \mathbf{S}, \mathbf{R}$ and $\mathbf{L}', \mathbf{S}', \mathbf{R}'$, where $\mathbf{R}$ is obtained from $\mathbf{F}$ by removing all nodes in $\mathbf{S}$ and those to the left of it. Similarly, $\mathbf{L}$ is obtained from $\mathbf{F}$ by removing all nodes in $\mathbf{S}$ and those to the right of it. We define $\mathbf{L}'$ and $\mathbf{R}'$ symmetrically. For simplicity, let us make a similar assumption as in the caterpillar case.

Assumption B. *For all $(v, v') \in \mathbf{S} \times \mathbf{S}'$, nodes from $\mathbf{L}, \mathbf{S}$, and $\mathbf{R}$ are always only mapped by $\mathsf{sim}(\mathsf{sub}(v), \mathsf{sub}(v'))$ to nodes of $\mathbf{L}', \mathbf{S}'$, and $\mathbf{R}'$, respectively.*

At this point, we can attempt to approach the problem similarly to how we did for caterpillar trees. Provided with the values $\mathsf{sim}(\mathsf{sub}(v), \mathsf{sub}(v'))$ for all $(v, v') \in (\mathbf{F} \times \mathbf{F}') \setminus (\mathbf{S} \times \mathbf{S}')$ in SED, we can construct the forest alignment graph of $\mathsf{sim}(\mathbf{L}, \mathbf{L}')$ and $\mathsf{sim}(\mathbf{R}, \mathbf{R}')$, aiming to overlay them on top of each other. Refer to fig. 4 for a visualization and for some more (informal) discussion.

However, this approach presents two challenges:

- The two forest alignment graphs might have different grid sizes.
- Identifying meeting points of paths in the two alignment graphs where nodes from $\mathbf{S}$ can be mapped to nodes of $\mathbf{S}'$ is not as straightforward as for TED on caterpillar trees.

Despite these challenges, we prove the following result:

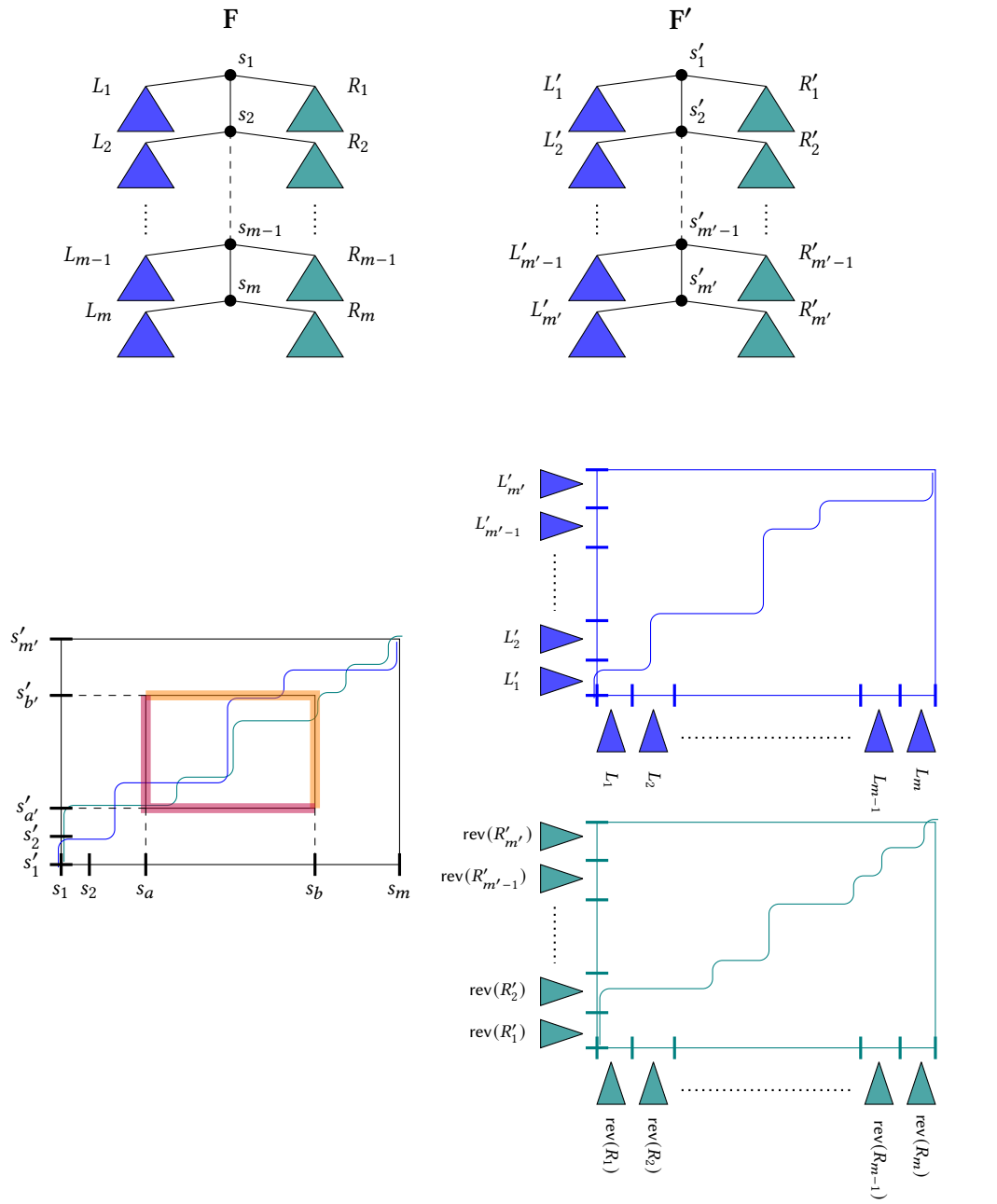


Figure 4: To compute $\mathrm{sim}(\mathrm{F}, \mathrm{F}')$ for the two depicted trees T and T′ under assumption B, we can use the same visualization approach as before. Specifically, we overlay two tree alignment graphs corresponding to the concatenated left and right subtrees, tracing two paths that, whenever they intersect at two spine nodes, provide the possibility to map them. These graphs correspond to $L_1L_2\cdots L_m$ vs. $L'_1L'_2\cdots L'_{m'}$ and $\mathrm{rev}(R_1R_2\cdots R_m)$ vs. $\mathrm{rev}(R'_1R'_2\cdots R'_{m'})$, shown in blue and teal, respectively. Given the similarity values $\mathrm{sim}(\mathrm{sub}(v), \mathrm{sub}(v'))$ for all $(v, v') \in (\mathrm{F}\times\mathrm{F}')\setminus(\mathrm{S}\times\mathrm{S}')$, we can construct these trees.
In Spine Edit Distance (SED), instead of computing only $\mathrm{sim}(\mathrm{F}, \mathrm{F}')$, we also need to determine $\mathrm{sim}(\mathrm{sub}(s), \mathrm{sub}(s'))$ for all $(s, s') \in (\mathrm{S}\times\mathrm{S}')$. Fortunately, by employing a similar divide-and-conquer approach as used for caterpillars, computing $\mathrm{sim}(\mathrm{F}, \mathrm{F}')$ naturally leads to obtaining these values as well.
When designing such a divide-and-conquer scheme, the subproblems must be indexed by rectangles whose edges align with coordinates corresponding to spine nodes. This ensures that no diagonal edges in the tree alignment graphs "jump over" the sides of the rectangle, allowing for a "clean" partitioning into subproblems.
As with caterpillar trees, removing assumption B would introduce a third path between the two existing ones.

Theorem 2.2. *Suppose there exists an algorithm computing the min-plus product of two $m \times m$ matrices in time $\mathsf{T}_{MUL}(m)$.*

Then, there is an algorithm for SED running in time $O(\mathsf{T}_{MUL}(n) + n^{2+o(1)})$, where $n = \max(|\mathbf{F}|, |\mathbf{F}'|)$.

Observe that theorem 2.2 together with lemma 2.1 yields theorem 1.1. Moreover, note that we establish not only the equivalence between TED and APSP, but also between SED and APSP.

Central to overcome the aforementioned challenges is the notation introduced by Mao in [45]. This notation allows us to more directly formalize the problem in terms of forests, without relying on paths as an intermediary. Additionally, it leads to a more concise description of the divide-et-impera strategy that we employ to solve SED. However, this comes at the cost of obscuring the intuitive understanding provided by the two crossing paths, which remain essential for visualizing the structure of the problem.

Forest Edit Distance. Generalizing TED on caterpillar trees on SED involves a shift from alignment graphs on strings to alignment graphs on forests. While there are algorithms finding border-to-border paths in the former in quadratic time, we still need to come up with an efficient way to compute such distances in the latter. The objective here is to develop either a truly subcubic algorithm or a reduction to APSP. We call the problem of finding border-to-border distances in a forest alignment graph the *Forest Edit Distance* problem (FED), and we present it here formulated using Mao's notation (to be introduced in the next section).[6]

Before introducing FED, let us briefly introduce some notation and the bi-order traversal of a forest, already used in [45]. The bi-order traversal of a forest $\mathbf{F}$ is a sequence of length $2|\mathbf{F}|$ generated by starting a depth-first traversal from the virtual root, and adding a node to the sequence whenever we enter or leave a node. We then use $\mathbf{F}[\,x\,..\,y\,)$ to denote the induced sub-forest of $\mathbf{F}$ consisting of nodes that appear twice in the segment $[\,x\,..\,y-1\,]$ in the bi-order traversal of $\mathbf{F}$. For example $\mathbf{F} = \mathbf{F}[\,1\,..\,2|\mathbf{F}|+1\,)$. For a node $v \in \mathbf{F}$, we let $\mathsf{l}(v)$ denote the first occurrence of v in the bi-order traversal of $\mathbf{F}$ and $\mathsf{r}(v)$ denote one *plus* the last occurrence of v.

Forest Edit Distance (FED)
Input: Two forests $\mathbf{F}$ and $\mathbf{F}'$ and $\mathsf{sim}(\mathsf{sub}(v), \mathsf{sub}(v'))$ for all $(v, v') \in \mathbf{F} \times \mathbf{F}'$.
Output: The following values:
- $\mathsf{sim}(\mathbf{F}[\,x\,..\,y\,), \mathbf{F}')$ for all $x, y \in [\,1\,..\,(2|\mathbf{F}|+1)\,]$,
- $\mathsf{sim}(\mathbf{F}[\,x\,..\,(2|\mathbf{F}|+1)\,), \mathbf{F}'[\,1\,..\,y'\,))$ for all $x \in [\,1\,..\,(2|\mathbf{F}|+1)\,], y' \in [\,1\,..\,(2|\mathbf{F}'|+1)\,]$,
- $\mathsf{sim}(\mathbf{F}, \mathbf{F}'[\,x'\,..\,y'\,))$ for all $x', y' \in [\,1\,..\,(2|\mathbf{F}'|+1)\,]$, and
- $\mathsf{sim}(\mathbf{F}[\,1\,..\,y\,), \mathbf{F}'[\,x'\,..\,(2|\mathbf{F}|+1)\,))$ for all $y \in [\,1\,..\,(2|\mathbf{F}|+1)\,], x' \in [\,1\,..\,(2|\mathbf{F}'|+1)\,]$.

We demonstrate that FED can indeed be reduced to APSP, and we prove:

Theorem 2.3. *Suppose there exists an algorithm computing the min-plus product of two $m \times m$ matrices in time $\mathsf{T}_{MUL}(m)$.*

Then, there is an algorithm for FED running in time $O(\mathsf{T}_{MUL}(n) + n^{2+o(1)})$, where $n = \max(|\mathbf{F}|, |\mathbf{F}'|)$.

[6] We remark that a different variant of FED was introduced already in [11]. There, it is defined with the same input, but the only required output is $\mathsf{sim}(\mathbf{F}, \mathbf{F}')$.

To address FED, we adopt a divide-et-impera approach. Our recursive scheme presented in theorem 2.3 builds upon Mao's decomposition scheme for forests introduced in [45].

2.4 Unweighted Tree Edit Distance

We now discuss our algorithm for Unweighted Tree Edit Distance. Consider two forests $\mathbf{F}, \mathbf{F}'$ with n, n' nodes respectively. We can make two simple observations with respect to the similarity matrices involved in max-plus products in our reduction:

- They are row-monotone and column-monotone.
- Their entries are bounded by $O(\min(n, n'))$.

Note that the first observation holds even in the weighted setting. For example, in Equation (1), for a fixed x, $\mathsf{sim}((x, b'), (r, y'))$ must be non-decreasing in y'. In the unweighted setting, the second observation additionally holds, since the value of any alignment is at most twice the number of nodes. In particular, in computing FED (Theorem 2.3) and SED (Theorem 2.2), we can instead use Bounded Monotone Min-Plus Matrix Multiplication.

Recall that the min-plus (or max-plus) product of an $n \times n$ arbitrary integer matrix and an $n \times n$ bounded monotone matrix can be computed in time $\tilde{O}(n^{(3+\omega)/2})$ [19], with subsequent generalizations to arbitrary rectangular matrices [28, 50].

Directly applying the observation, we have that SED between two forests of size m and m', respectively, can be computed in time $\tilde{O}(\max(m, m')^{(3+\omega)/2})$ [19]. This running may be improved to $\tilde{O}(\sqrt{\min(m, m')} \cdot \max(m, m')^{(2+\omega)/2})$ by our bound on entries [28].

Let us see the result we can obtain via Lemma 2.1.

Recall that Lemma 2.1 states that given an $O(f(m)g(m'))$ algorithm for SED, there is an $\tilde{O}(f(n)g(n'))$ algorithm for TED on two forests of size n, n'. Suppose $f(x) = O(x^a), g(x) = O(x^b)$. Setting $m' = 1$, we have $m^{(2+\omega)/2} = O(m^a m'^b) = O(m^a)$, so that $a \geq (2+\omega)/2$. Similarly, we obtain $b \geq (2+\omega)/2$, so that we only obtain a TED algorithm running in time $O(n^{2+\omega})$ for two forests of size n. Even if $\omega = 2$, the running time $O(n^4)$ is prohibitively expensive.

In general, a running time $\tilde{O}(\min(m, m')^c \max(m, m')^d)$ can be upper bounded by $\tilde{O}(m^{\max(c,d)} m'^d)$. Hence, in order to keep the total exponents of the two formulations the same, we hope to obtain a running time with $c \geq d$ without increasing $c+d$, i.e., we hope for an algorithm for SED with running time $\tilde{O}(\min(m, m')^c \max(m, m')^d)$ for $c \geq d$ and $c + d = (3+\omega)/2$. This is achieved by the following theorem:

Theorem 2.4. *There is an algorithm for unweighted SED running in time*

$$(n/n')^{1+o(1)} \cdot \left(\mathsf{T}_{MonMUL}(n') + n'^{2+o(1)} g(n')\right),$$

where $n = |\mathbf{F}|, n' = |\mathbf{F}'|$, $n \geq n'$, and $\mathsf{T}_{MonMUL}(n', n', n', \mathrm{D}) = O(f(n')g(\mathrm{D}))$ for some functions f, g.

We now illustrate the techniques required to obtain the above result. Suppose we have two trees $\mathbf{F}, \mathbf{F}'$ of size n, n' with spines $\mathbf{S} = \{r, \cdots, q\}, \mathbf{S}' = \{r', \cdots, q'\}$. Assume without loss of generality that $n \gg n'$. For simplicity, we again assume Assumption B.

We can formulate the SED problem using a divide-et-impera scheme. Since $n \gg n'$, we only decompose the larger tree $\mathbf{F}$. For a

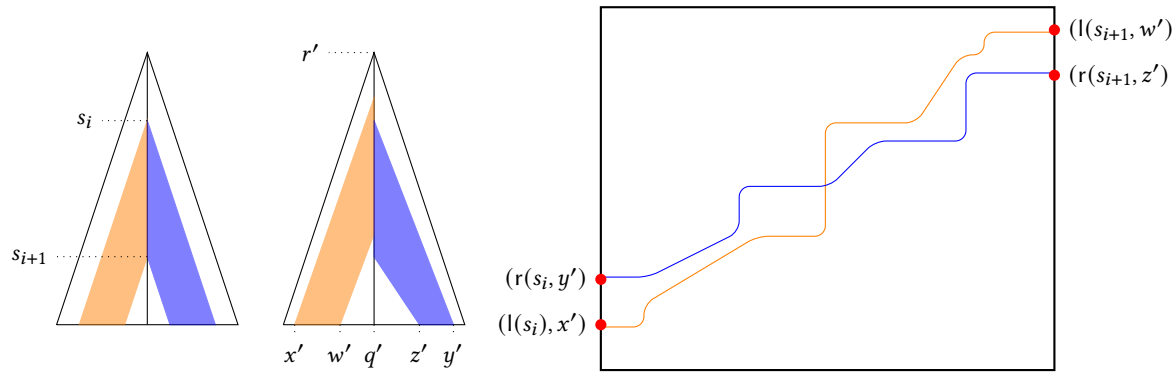


Figure 5: The alignment whose value is computed in Equation (2) visualized as a Border-to-Border (BBD) distance computation. The value of the orange path corresponds to the first summand, or the alignment between $\mathbf{F}[\,\mathsf{l}(s_i)\,..\,\mathsf{l}(s_{i+1})\,]$ and $\mathbf{F}'[\,x'\,..\,w'\,]$. The value of the blue path corresponds to the last summand, or the alignment between $\mathbf{F}[\,\mathsf{r}(s_{i+1})\,..\,\mathsf{r}(s_i\,)$ and $\mathbf{F}'[\,z'\,..\,y'\,]$. For every pair of points on the right border, we have the optimal alignment between $\mathrm{sub}(s_{i+1})$ and $\mathbf{F}'[\,w'\,..\,z'\,]$.

given threshold γ, we can efficiently find a subsequence of spine vertices $I = [\,s_1\,..\,s_d\,]$ with $r = s_1, q = s_d$ such that for all $i < d$, either $|\mathrm{sub}(s_i) \setminus \mathrm{sub}(s_{i+1})| \le \gamma$ or s_i immediately precedes s_{i+1} in $\mathbf{S}$.

We can then compute SED recursively in a bottom-up manner. We begin by computing $\mathsf{sim}(\mathrm{sub}(s_d), \mathbf{F}'[\,x'\,..\,y'\,])$ for all $x', y' \in [\,1\,..\,2|\mathbf{F}'| + 1\,]$. Since $\mathrm{sub}(s_d)$ is a single node, we can compute this efficiently. Next, for $i < d$, given $\mathsf{sim}(\mathrm{sub}(s_{i+1}), \mathbf{F}'[\,x'\,..\,y'\,])$ for all x', y', our hope is to compute $\mathsf{sim}(\mathrm{sub}(s_i), \mathbf{F}'[\,x'\,..\,y'\,])$ for all x', y'. First, suppose $|\mathrm{sub}(s_i) \setminus \mathrm{sub}(s_{i+1})| \le \gamma$. In this case we recurse on the smaller problem of size (γ, n'). Since there are at most $3n/\gamma$ sub-problems in I, the total time on the first case is

$$\mathsf{T}(n, n') = (3n/\gamma) \cdot \mathsf{T}(\gamma, n')$$

which yields total time $(n/n')^{1+o(1)} \cdot n'^{(3+\omega)/2}$, since when $\gamma = O(n')$, we apply the recursive scheme employed in Theorem 2.2 in time $n'^{(3+\omega)/2}$.

Thus, it remains to handle the case where s_i immediately precedes s_{i+1}. Under Assumption B, we can decompose the similarity when s_i is not aligned, by writing $\mathsf{sim}(\mathrm{sub}(s_i), \mathbf{F}'[\,x'\,..\,y'\,])$ as

$$\max_{\substack{w' \in [\,\mathsf{l}(r')\,..\,\mathsf{l}(q')\,] \\ z' \in [\,\mathsf{r}(q')\,..\,\mathsf{r}(r')\,]}} \Big\{ \mathsf{sim}(\mathbf{F}[\,\mathsf{l}(s_i)\,..\,\mathsf{l}(s_{i+1})\,], \mathbf{F}'[\,x'\,..\,w'\,]) + \mathsf{sim}(\mathrm{sub}(s_{i+1}), \mathbf{F}'[\,w'\,..\,z'\,]) + \mathsf{sim}(\mathbf{F}[\,\mathsf{r}(s_{i+1})\,..\,\mathsf{r}(s_i)\,], \mathbf{F}'[\,z'\,..\,y'\,]) \Big\}. \tag{2}$$

To handle the case where s_i is aligned, say to vertex v, we can add $\eta(s_i, v)$ to the value of $\mathsf{sim}(\mathrm{sub}(s_i), \mathbf{F}'[\,\mathsf{l}(v) + 1\,..\,\mathsf{r}(v)\,])$. For completeness, we also ensure monotonicity after updating the values in a post-processing step (i.e. $\mathsf{sim}(\mathrm{sub}(s_i), \mathbf{F}'[\,x'\,..\,y'\,]) \ge \mathrm{sub}(s_i, \mathbf{F}'[\,\mathsf{l}(v)\,..\,\mathsf{r}(v)\,])$ if $x' \preceq \mathsf{l}(v) \prec \mathsf{r}(v) \preceq y'$). The computation can be written as a max-plus product of three $O(n') \times O(n')$ matrices with entries bounded by $O(n')$ since each entry is a similarity measure between two forests, one of size at most $O(n')$. Since the matrices are also monotone, the summands can be combined in time $\mathsf{T}_{\mathsf{MonMUL}}(n') = \tilde{O}(n'^{(3+\omega)/2})$.

The second summand is available to us as part of the bottom-up recursion. It remains to compute the first and last summand. Consider the first summand. Observe that it is exactly the third output of FED between $\mathbf{F}[\,\mathsf{l}(s_i)\,..\,\mathsf{l}(s_{i+1})\,]$ and $\mathbf{F}'[\,\mathsf{l}(r')\,..\,\mathsf{l}(q')\,]$. Similarly, the third summand is exactly the third output of FED between $\mathbf{F}[\,\mathsf{r}(s_{i+1})\,..\,\mathsf{r}(s_i)\,]$ and $\mathbf{F}'[\,\mathsf{r}(q')\,..\,\mathsf{r}(r')\,]$. Since none of the forests contain any spine nodes $\mathbf{S}, \mathbf{S}'$, all inputs to the FED instances are given by the inputs of the SED instance.

Our goal is now to solve these two FED instances. We can bound the size of the two forests $\mathbf{F}[\,\mathsf{l}(s_i)\,..\,\mathsf{l}(s_{i+1})\,]$ and $\mathbf{F}[\,\mathsf{r}(s_{i+1})\,..\,\mathsf{r}(s_i)\,]$ by $|\mathrm{sub}(s_i) \setminus \mathrm{sub}(s_{i+1})|$. However, it is possible that for this last term $|\mathrm{sub}(s_i) \setminus \mathrm{sub}(s_{i+1})| = \Theta(n)$ holds, in which case directly applying Theorem 2.3 on the FED instance already takes $O(\mathsf{T}_{\mathsf{MonMUL}}(n))$ time in this one step alone. Even if the entries are bounded by $O(n')$, the output of FED instance is already size $\Omega(n^2)$. How can we reduce the dependence on n? In our SED application, we have only used the third output of FED. In fact, we show that all outputs except the first output (which has size $\Omega(n^2)$) can be computed efficiently. Formally, we define the Unbalanced Forest Edit Distance (UFED) problem as computing all but the first output of FED and show the following theorem.

Theorem 2.5. *There is an algorithm for unweighted UFED running in time*

$$(n/n')^{1+o(1)} \cdot \left(\mathsf{T}_{MonMUL}(n') + n'^{2+o(1)} g(n')\right),$$

where $n = |\mathbf{F}|$, $n' = |\mathbf{F}'|$, $n \ge n'$ and $\mathsf{T}_{MonMUL}(n', n', n', \mathrm{D}) = O(f(n')g(\mathrm{D}))$ for some functions f, g.

In the remainder of the technical overview of the unweighted algorithm, we give an overview of the proof of Theorem 2.5. Further simplifying under Assumption B, our previous discussion shows that it is enough to only compute the third FED output. To this end, consider the alignment graph $\bar{\mathbf{G}}$ of two forests $\mathbf{F}, \mathbf{F}'$ with vertices $\mathbf{F} \times \mathbf{F}'$ arranged in a grid where both trees are ordered in pre-order traversal.

The grid contains horizontal and vertical edges of weight 0 (corresponding to vertex deletions) and edges from (v, v') to (w, w')

of weight $\mathrm{sim}(\mathrm{sub}(v), \mathrm{sub}(v'))$ where w (resp. w') is the first vertex after $\mathrm{sub}(v)$ (resp. $\mathrm{sub}(v')$) in pre-order traversal i.e. the first vertices that can be aligned after aligning $\mathrm{sub}(v)$ and $\mathrm{sub}(v')$. The desired output of FED then corresponds to computing all distances from the left border to the right border of the grid. In contrast, the general FED problem asks to compute all distances from the left and bottom borders to the right and top borders.

The alignment graph $\mathbf{G}$ has dimension $n \times n'$, so ideally we hope to be able to breaking the it into $O(n/n')$ subgraphs, each of size $O(n') \times O(n')$. Then, we can compute FED on each subgraph and combine the results using $O(n/n')$ bounded monotone max-plus products, thus computing the desired FED output in total time $O((n/n') \cdot \mathsf{T}_{\mathsf{MonMUL}}(n'))$. However, in reality, it is not simple to decompose $\mathbf{G}$ to $O(n/n')$ subgraphs and combine the results, as there can be edges connecting two nodes that are far away in the alignment graph. Hence, in our actual algorithm, we apply a divide-et-impera scheme utilizing Mao's tree decomposition [45], which incurs an additional $(n/n')^{o(1)}$ factor. Still, we obtain the desired almost-linear dependence on the size of the larger forest $n = |\mathbf{F}|$. In particular, at least under Assumption B, both FED and SED can be computed in time $O(\min(m, m')^{(1+\omega)/2} \max(m, m')^{1+o(1)})$.

To remove Assumption B, we must be able to additionally compute distances from the left border to the top border in BBD instances, corresponding to the second and fourth outputs of the FED problem. In the SED instance we then proceed by careful case analysis to ensure that we consider all possible alignments between the two forests. Applying Lemma 2.1 then yields an $n^{1+o(1)} n'^{(1+\omega)/2}$-time algorithm for unweighted tree edit distance between forests of size n, n' and $n \geq n'$, proving Corollary 1.4.

References

[1] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. 2016. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*. 375–388. doi:10.1145/2897518.2897653

[2] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th Annual Symposium on Foundations of Computer Science (FOCS)*.

[3] Alfred V. Aho and Thomas G Peterson. 1972. A minimum distance error-correcting parser for context-free languages. *SIAM J. Comput.* 1, 4 (1972), 305–312.

[4] Shyan Akmal and Ce Jin. 2021. Faster Algorithms for Bounded Tree Edit Distance. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*. 12:1–12:15. doi:10.4230/LIPIcs.ICALP.2021.12

[5] Tatsuya Akutsu. 1999. Approximation and exact algorithms for RNA secondary structure prediction and recognition of stochastic context-free languages. *J. Comb. Optim.* 3 (1999), 321–336.

[6] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. 2025. More Asymmetry Yields Faster Matrix Multiplication. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. to appear.

[7] Carlos Eduardo Rodrigues Alves, E. N. Cáceres, and Siang Wun Song. 2008. An all-substrings common subsequence algorithm. *Discret. Appl. Math.* 156, 7 (2008), 1025–1035. doi:10.1016/J.DAM.2007.05.056

[8] Rolf Backofen, Shihyen Chen, Danny Hermelin, Gad M. Landau, Mikhail A. Roytberg, Oren Weimann, and Kaizhong Zhang. 2007. Locality and Gaps in RNA Comparison. *Journal of Computational Biology* 14, 8 (2007), 1074–1087. doi:10.1089/cmb.2007.0062 arXiv:https://doi.org/10.1089/cmb.2007.0062 PMID: 17985988.

[9] Arturs Backurs and Piotr Indyk. 2015. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*. 51–58. doi:10.1145/2746539.2746612

[10] John Bellando and Ravi Kothari. 1999. Region-Based Modeling and Tree Edit Distance as a Basis for Gesture Recognition. In *Proceedings of the 10th International Conference on Image Analysis and Processing (ICIAP)*. 698–703. doi:10.1109/ICIAP.1999.797676

[11] Mahdi Boroujeni, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. 2019. $1+\varepsilon$ approximation of tree edit distance in quadratic time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 709–720. doi:10.1145/3313276.3316388

[12] Karl Bringmann, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. 2020. Tree Edit Distance Cannot be Computed in Strongly Subcubic Time (Unless APSP Can). *ACM Trans. Algorithms* 16, 4, Article 48 (jul 2020), 22 pages. doi:10.1145/3381878

[13] Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. 2019. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM J. Comput.* 48, 2 (2019), 481–512.

[14] Peter Buneman, Martin Grohe, and Christoph Koch. 2003. Path Queries on Compressed XML. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*. 141–152. doi:10.1016/B978-012722442-8/50021-5

[15] Alejandro Cassis, Tomasz Kociumaka, and Philip Wellnitz. 2023. Optimal Algorithms for Bounded Weighted Edit Distance. In *Proceedings of the 2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. 2177–2187. doi:10.1109/FOCS57990.2023.00135

[16] Panagiotis Charalampopoulos, Tomasz Kociumaka, and Shay Mozes. 2020. Dynamic String Alignment. In *Proceedings of the 31st Annual Symposium on Combinatorial Pattern Matching (CPM)*. 9:1–9:13. doi:10.4230/LIPIcs.CPM.2020.9

[17] Sudarshan S. Chawathe. 1999. Comparing Hierarchical Data in External Memory. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*. 90–101. http://www.vldb.org/conf/1999/P8.pdf

[18] Weimin Chen. 2001. New Algorithm for Ordered Tree-to-Tree Correction Problem. *J. Algorithms* 40, 2 (2001), 135–158. doi:10.1006/jagm.2001.1170

[19] Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. 2022. Faster min-plus product for monotone instances. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 1529–1542.

[20] Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, and Barna Saha. 2023. Weighted Edit Distance Computation: Strings, Trees, and Dyck. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*. 377–390. doi:10.1145/3564246.3585178

[21] Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, Barna Saha, and Hamed Saleh. 2022. Õ(n+poly(k))-time Algorithm for Bounded Tree Edit Distance. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. 686–697. doi:10.1109/FOCS54457.2022.00071

[22] Debarati Das, Tomasz Kociumaka, and Barna Saha. 2022. Improved Approximation Algorithms for Dyck Edit Distance and RNA Folding. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP)*. 49:1–49:20. doi:10.4230/LIPICS.ICALP.2022.49

[23] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. 2010. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms* 6, 1, Article 2 (dec 2010), 19 pages. doi:10.1145/1644015.1644017

[24] Ran Duan, Ce Jin, and Hongxun Wu. 2019. Faster Algorithms for All Pairs Non-Decreasing Paths Problem. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*.

[25] Ran Duan and Seth Pettie. 2009. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

[26] Serge Dulucq and Hélène Touzet. 2003. Analysis of Tree Edit Distance Algorithms. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM)*. 83–95.

[27] Serge Dulucq and Hélène Touzet. 2005. Decomposition algorithms for the tree edit distance problem. *J. Discrete Algorithms* 3, 2 (2005), 448–471. doi:10.1016/j.jda.2004.08.018

[28] Anita Dürr. 2023. Improved bounds for rectangular monotone Min-Plus Product and applications. *Inf. Process. Lett.* 181, C (March 2023), 9 pages. doi:10.1016/j.ipl.2023.106358

[29] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. 2009. Compressing and indexing labeled trees, with applications. *J. ACM* 57, 1 (2009), 4:1–4:33. doi:10.1145/1613676.1613680

[30] Michael J. Fischer and Albert R. Meyer. 1971. Boolean Matrix Multiplication and Transitive Closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT)*. 129–131.

[31] Dvir Fried, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, Ely Porat, and Tatiana Starikovskaya. 2024. An improved algorithm for the k-Dyck edit distance problem. *ACM Trans. Algorithms* 20, 3 (2024), 1–25.

[32] Daniel Gibney, Ce Jin, Tomasz Kociumaka, and Sharma V. Thankachan. 2024. Near-Optimal Quantum Algorithms for Bounded Edit Distance and Lempel-Ziv Factorization. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 3302–3332. doi:10.1137/1.9781611977912.118

[33] Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. 2019. Sublinear Algorithms for Gap Edit Distance. In *Proceedings of the 2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. 1101–1120. doi:10.1109/FOCS.2019.00070

[34] Egor Gorbachev and Tomasz Kociumaka. 2024. Bounded Edit Distance: Optimal Static and Dynamic Algorithms for Small Integer Weights. arXiv:2404.06401 [cs.DS] https://arxiv.org/abs/2404.06401

[35] Yuzhou Gu, Adam Polak, Virginia Vassilevska Williams, and Yinzhan Xu. 2021. Faster Monotone Min-Plus Product, Range Mode, and Single Source Replacement Paths. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP)*. 75:1–75:20. doi:10.4230/LIPICS.ICALP.2021.75

[36] Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press. doi:10.1017/CBO9780511574931

[37] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing (STOC)*.

[38] Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. 2003. Local Similarity in RNA Secondary Structures. In *Proceedings of 2nd IEEE Computer Society Bioinformatics Conference (CSB)*. 159–168. doi:10.1109/CSB.2003.1227315

[39] Philip N. Klein. 1998. Computing the Edit-Distance between Unrooted Ordered Trees. In *Proceedings of the 6th Annual European Symposium on Algorithms (ESA)*. 91–102. doi:10.1007/3-540-68530-8_8

[40] Philip N. Klein. 2005. Multiple-source shortest paths in planar graphs. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 146–155.

[41] Philip N. Klein, Thomas B. Sebastian, and Benjamin B. Kimia. 2001. Shape matching using edit-distance: an implementation. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA)*. 781–790. http://dl.acm.org/citation.cfm?id=365411.365779

[42] Philip N. Klein, Srikanta Tirthapura, Daniel Sharvit, and Benjamin B. Kimia. 2000. A tree-edit-distance algorithm for comparing simple, closed shapes. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 696–704. http://dl.acm.org/citation.cfm?id=338219.338628

[43] Tomasz Kociumaka, Jakob Nogler, and Philip Wellnitz. 2024. On the Communication Complexity of Approximate Pattern Matching. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*. 1758–1768. doi:10.1145/3618260.3649604

[44] Gad M. Landau and Uzi Vishkin. 1988. Fast string matching with k-differences. *J. Comput. Syst. Sci.* 37, 1 (Aug. 1988), 63–78. doi:10.1016/0022-0000(88)90045-1

[45] Xiao Mao. 2022. Breaking the Cubic Barrier for (Unweighted) Tree Edit Distance. In *Proceedings of the 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. 792–803. doi:10.1109/FOCS52979.2021.00082

[46] Shay Mozes, Dekel Tsur, Oren Weimann, and Michal Ziv-Ukelson. 2009. Fast algorithms for computing tree LCS. *Theor. Comput. Sci.* 410, 43 (Oct. 2009), 4303–4314. doi:10.1016/j.tcs.2009.07.011

[47] Ruth Nussinov and Ann B. Jacobson. 1980. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proc. Natl. Acad. Sci. U.S.A.* 77, 11 (1980), 6309–6313.

[48] Barna Saha. 2014. The Dyck language edit distance problem in near-linear time. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*. 611–620.

[49] Barna Saha. 2017. Fast & space-efficient approximations of language edit distance and RNA folding: An amnesic dynamic programming approach. In *Proceedings of the 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 295–306.

[50] Barna Saha and Christopher Ye. 2024. Faster Approximate All Pairs Shortest Paths. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 4758–4827.

[51] J.P. Schmidt. 1995. All shortest paths in weighted grid graphs and its application to finding all approximate repeats in strings. In *Proceedings Third Israel Symposium on the Theory of Computing and Systems*. 67–77. doi:10.1109/ISTCS.1995.377044

[52] Stefan Schwarz, Mateusz Pawlik, and Nikolaus Augsten. 2017. A New Perspective on the Tree Edit Distance. In *Proceedings of the 10th International Conference on Similarity Search and Applications (SISAP)*. 156–170.

[53] Thomas B. Sebastian, Philip N. Klein, and Benjamin B. Kimia. 2004. Recognition of Shapes by Editing Their Shock Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 5 (2004), 550–571. doi:10.1109/TPAMI.2004.1273924

[54] Masoud Seddighin and Saeed Seddighin. 2022. $3+\varepsilon$ Approximation of Tree Edit Distance in Truly Subquadratic Time. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*. 115:1–115:22. doi:10.4230/LIPIcs.ITCS.2022.115

[55] Stanley M. Selkow. 1977. The tree-to-tree editing problem. *Inf. Process. Lett.* 6, 6 (1977), 184–186. doi:10.1016/0020-0190(77)90064-3

[56] Bruce A. Shapiro and Kaizhong Zhang. 1990. Comparing multiple RNA secondary structures using tree comparisons. *Bioinformatics* 6, 4 (10 1990), 309–318. doi:10.1093/bioinformatics/6.4.309

[57] Dennis Shasha and Kaizhong Zhang. 1989. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM J. Comput.* 18, 6 (1989), 1245–1262. doi:10.1137/0218082 arXiv:https://doi.org/10.1137/0218082

[58] Kuo-Chung Tai. 1979. The Tree-to-Tree Correction Problem. *J. ACM* 26, 3 (jul 1979), 42–433. doi:10.1145/322139.322143

[59] Alexander Tiskin. 2006. All semi-local longest common subsequences in subquadratic time. In *Proceedings of the 1st International Computer Science Conference on Theory and Applications (CSR)*. 352–363. doi:10.1007/11753728_36

[60] Hélène Touzet. 2005. A Linear Tree Edit Distance Algorithm for Similar Ordered Trees. In *Combinatorial Pattern Matching*, Alberto Apostolico, Maxime Crochemore, and Kunsoo Park (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 334–345.

[61] Virginia Vassilevska, Ryan Williams, and Raphael Yuster. 2009. All Pairs Bottleneck Paths and Max-Min Matrix Products in Truly Subcubic Time. *Theory Comput.* 5, 1 (2009), 173–189.

[62] Virginia Vassilevska Williams. 2010. Nondecreasing paths in a weighted graph or: How to optimally read a train schedule. *ACM Trans. Algorithms* 6, 4 (2010).

[63] Virginia Vassilevska Williams. 2018. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, Vol. 3. World Scientific, 3431–3472.

[64] Virginia Vassilevska Williams and R. Ryan Williams. 2018. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM* 65, 5 (2018), 1–38.

[65] Virginia Vassilevska Williams and Yinzhan Xu. 2020. Truly Subcubic Min-Plus Product for Less Structured Matrices, with Applications. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 12–29. doi:10.1137/1.9781611975994.2

[66] Balaji Venkatachalam, Dan Gusfield, and Yelena Frid. 2014. Faster algorithms for RNA-folding using the Four-Russians method. *Algorithms Mol. Biol.* 9 (2014), 1–12.

[67] Michael S. Waterman. 1995. *Introduction to computational biology: maps, sequences and genomes*. CRC Press.

[68] R. Ryan Williams. 2018. Faster All-Pairs Shortest Paths via Circuit Complexity. *SIAM J. Comput.* 47, 5 (2018), 1965–1985. doi:10.1137/15M1024524

[69] Shay Zakov, Dekel Tsur, and Michal Ziv-Ukelson. 2011. Reducing the worst case running times of a family of RNA and CFG problems, using Valiant's approach. *Algorithms Mol. Biol.* 6 (2011), 1–22.

[70] Kaizhong Zhang and Dennis Shasha. 1989. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM J. Comput.* 18, 6 (1989), 1245–1262. doi:10.1137/0218082 arXiv:https://doi.org/10.1137/0218082

Received 2024-11-04; accepted 2025-02-01