On the Hardness of Computing the Edit Distance of Shallow Trees

Panagiotis Charalampopoulos[1][0000−0002−6024−1557], Paweł Gawrychowski[2][0000−0002−6993−5440], Shay Mozes[3,⋆][0000−0001−9262−1821], and Oren Weimann[4,⋆][0000−0002−4510−7552]

[1] Birkbeck, University of London, London, UK
p.charalampopoulos@bbk.ac.uk
[2] University of Wrocław, Wrocław, Poland
gawry@cs.uni.wroc.pl
[3] Reichman University, Herzliya, Israel
smozes@idc.ac.il
[4] University of Haifa, Haifa Israel
oren@cs.haifa.ac.il

Abstract. We consider the edit distance problem on rooted ordered trees parameterized by the trees' depth. For two trees of size at most n and depth at most d, the state-of-the-art solutions of Zhang and Shasha [SICOMP 1989] and Demaine et al. [TALG 2009] have runtimes $\mathcal{O}(n^2d^2)$ and $\mathcal{O}(n^3)$, respectively, and are based on so-called *decomposition algorithms*. It has been recently shown by Bringmann et al. [TALG 2020] that, when $d = \Theta(n)$, one cannot compute the edit distance of two trees in $\mathcal{O}(n^{3-\epsilon})$ time (for any constant $\epsilon > 0$) under the APSP hypothesis. However, for small values of d, it is not known whether the $\mathcal{O}(n^2d^2)$ upper bound of Zhang and Shasha is optimal. We make the following twofold contribution. First, we show that under the APSP hypothesis there is no algorithm with runtime $\mathcal{O}(n^2d^{1-\epsilon})$ (for any constant $\epsilon > 0$) when $d = \text{poly(n)}$. Second, we show that there is no *decomposition algorithm* that runs in time $o(\min\{n^2d^2, n^3\})$.

1 Introduction

Let F and G be two rooted and ordered trees of size n where each node is assigned a label from an alphabet Σ. The edit distance between trees F and G is the minimum cost of transforming F into G by a sequence of elementary *edit operations*: changing the label of a node v, deleting a node v and setting the children of v as the children of v's parent (in the place of v in the left-to-right order), and inserting a node v (defined as the inverse of a deletion); see Figure 1. The cost of these elementary operations is given by two *cost functions*: $c_{del}(x)$ is the cost of deleting or inserting a node with label x, and $c_{match}(x, y)$ is the cost of changing the label of a node from x to y.

⋆ Supported by Israel Science Foundation grant 810/21.

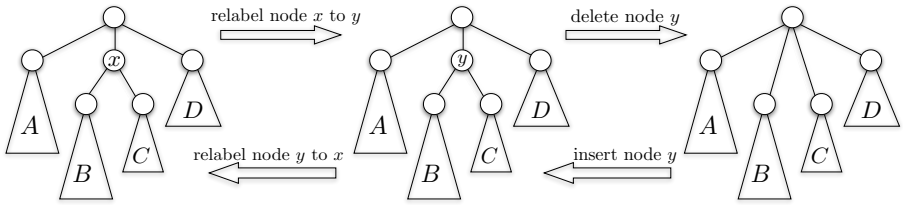

Fig. 1. The three edit operations on a node-labeled tree.

Tree edit distance is the most common similarity measure between labeled trees. It is instrumental in computational biology [7, 17, 25, 31], structured text processing [10, 11, 16], programming languages [18], computer vision [6, 19], character recognition [23], automatic grading [3], answer extraction [33], and many more (see the popular survey of Bille [7] and the books of Apostolico and Galil [4] and Valiente [29]).

The tree edit distance (TED) problem was introduced by Tai [27] as a generalization of the well known string edit distance problem [30]. Zhang and Shasha [35] showed that the classical dynamic-programming algorithm for string edit distance naturally extends to tree edit distance. Namely, to compute the edit distance of two forests F and G, consider the rightmost roots of F and G: they are either matched or (at least) one of them is deleted. Checking all these options generates a constant number of (smaller) recursive subproblems. Zhang and Shasha [35] showed that when the depths of F and G are bounded by d the total number of generated recursive subproblems (and hence the algorithm's running time) is $\mathcal{O}(n^2d^2)$. This is appealing for shallow trees, but can be as high as $\Omega(n^4)$ for trees of large depth.

Obviously, the choice of recursing on the rightmost root (and not the leftmost) is arbitrary. Klein [20] observed that if we carefully alternate between recursing on the rightmost and the leftmost roots the running time improves to $\mathcal{O}(n^3 \log n)$ (regardless of d). Dulucq and Touzet [14] called such algorithms (i.e., that are based on the same dynamic programming but only differ in their choices of rightmost and leftmost) *decomposition algorithms* and showed that there is no $o(n^2 \log^2 n)$-time decomposition algorithm. Demaine et al. [12] gave an $\mathcal{O}(n^3)$-time decomposition algorithm and showed that for trees of depth $d = \Omega(n)$ there is no $o(n^3)$-time decomposition algorithm. Of course there may be a faster TED algorithm that is not a decomposition algorithm. This however is probably not the case for trees of depth $d = \Omega(n)$. For such trees, it was shown in [9] that: (1) assuming the APSP hypothesis, there is no $\mathcal{O}(n^{3-\epsilon})$ algorithm for TED with alphabet-size $|\Sigma| = \Omega(n)$, and (2) assuming the stronger k-Clique hypothesis, there is no $\mathcal{O}(n^{3-\epsilon})$ algorithm for TED with alphabet-size $|\Sigma| = \mathcal{O}(1)$. An important exception is the special case of *unweighted* TED where $c_{del}(a) = c_{match}(a, b) = 1$ and $c_{match}(a, a) = 0$ (aka the Levenshtein distance) for which very recently

a non-decomposition strongly subcubic algorithm (for any d) was devised by Mao [21]; the exponent of n was further reduced to 2.9149 by Dürr [15].

To sum it up, the fastest known algorithm for (weighted) TED runs in $\mathcal{O}(\min\{n^3, n^2d^2\})$ time, it is a decomposition algorithm, and its cubic runtime when $d = \Omega(n)$ can probably not be improved by any polynomial factor. However, for smaller values of $d = \text{poly(n)}$ (i.e., $d = n^\delta$ for some constant $0 < \delta < 1$) we do not yet know the right complexity. This raises the following questions:

1. Is there an $\mathcal{O}(n^2d^{2-\epsilon})$-time decomposition algorithm?
   (The lower bound of [12] does not rule this out.)
2. Is there an $\mathcal{O}(n^2d^{2-\epsilon})$-time algorithm that is not a decomposition algorithm?
   (The lower bound of [9] does not rule this out.)

We show that the answer to the first question is no. As for the second question, we do not yet know if an $\mathcal{O}(n^2d^{2-\epsilon})$-time algorithm exists, but we show that an $\mathcal{O}(n^2d^{1-\epsilon})$-time algorithm does not (assuming the APSP hypothesis).

Related work. Pawlik and Augsten [22] developed a decomposition algorithm whose performance on any input is not worse (and possibly better) than that of any of the existing decomposition algorithms. Other attempts achieved better running times by restricting the edit operations or the scoring schemes [1, 11, 21, 24, 26, 28, 34], or by resorting to approximation [2, 5, 8]. However, in the worst case no algorithm currently beats $\mathcal{O}(\min\{n^3, n^2d^2\})$ (not even by a logarithmic factor). Finally, the edit distance between edge-labeled unrooted trees, first studied by Klein [20], can be computed in $\mathcal{O}(n^3)$ time as shown by Dudek and Gawrychowski [13]. In addition, Dudek and Gawrychowski [13] presented a simple $\mathcal{O}(|\mathsf{input}|)$-time reduction from TED on node-labeled rooted trees to TED on edge-labeled unrooted trees. This reduction replaces the two rooted trees by unrooted trees with the same size and diameter asymptotically, and hence our lower bounds also apply to the TED problem on edge-labeled unrooted trees parameterized by the trees' diameter.

2 Preliminaries

We denote the tree edit distance of two trees F and G by $\text{TED}(F, G)$. The alphabet is denoted by Σ.

Now, let us look more closely at the allowed edit operations. First, observe that the insertion operation is redundant: an insertion to one of the trees is equivalent to a deletion in the other. We can thus consider the problem of computing a minimum-cost sequence of deletions and relabelings to both F and G that yields identical trees. Further, as we argue next, without loss of generality, we can assume that $c_{del}(x) = 0$ for all $x \in \Sigma$. Starting from general cost functions, we can define new cost functions $c'_{match}(x, y) := c_{match}(x, y) - c_{del}(x) - c_{del}(y)$, for all $x, y \in \Sigma$, and $c'_{del}(x) := 0$, for all $x \in \Sigma$, that preserve the tree edit distance up to a linear-time computable additive constant equal to the cost of deleting

both trees with the original c_{del} function. Intuitively, we pay for the deletion of all nodes up front and get refunded for nodes that are not deleted.

A *left comb* (resp. *right comb*) of depth n is a tree with $2n-1$ nodes that consists of a path P of length n, with one endpoint of the path being the root of the tree and the other one being a leaf, and $n-1$ more leaf nodes, each being the right (resp. left) child of a distinct node of P. We call a pair of left and a right combs *opposing combs*. See Fig. 2 for an illustration.

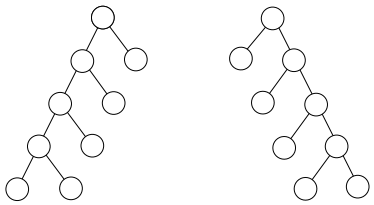

Fig. 2. Two opposing combs of depth 5; the left comb is shown in the left.

3 Lower Bound Conditioned on the APSP Hypothesis

In this section we present a hardness proof of TED on trees of depth $d = \text{poly(n)}$, conditioned on the All-Pairs Shortest Paths (APSP) hypothesis.

Conjecture 1 (APSP hypothesis). For any $\epsilon > 0$, there exists $c > 0$ such that APSP on n-vertex graphs with edge weights in $\{1, \ldots, n^c\}$ cannot be solved in time $\mathcal{O}(n^{3-\epsilon})$.

Instead of reducing APSP to TED, we will reduce from the equivalent (see [32]) NegativeTriangle problem:

NegativeTriangle
Input: A complete tripartite graph $H = (V, E)$ with three parts I, J, and K, each of size at most n, and a weight function $w : E \to \{-n^c, \ldots, n^c\}$.
Output: **Yes** if and only if there exist vertices $i \in I$, $j \in J$, and $k \in K$ such that $w(i,j) + w(j,k) + w(k,i) < 0$.

Lemma 1. *Consider an instance of* NegativeTriangle *comprised of a complete tripartite graph $H = (V, E)$ with parts of size at most n and a weight-function $w : E \to \{-n^c, \ldots, n^c\}$. For any integer $d \leq n$, this instance can be reduced to deciding whether any of $\mathcal{O}((n/d)^3)$ complete graphs on $3d$ vertices contains a negative triangle. The time required for this reduction is $\mathcal{O}(n^2 + n^3/d)$.*

Proof. Let us split each of the three parts I, J, and K into $\lceil n/d \rceil$ subsets, each of size at most d. Then, it suffices to solve separately, for each of the $\mathcal{O}((n/d)^3)$ triplets of subsets $A \subseteq I$, $B \subseteq J$, and $C \subseteq K$, an instance of the NegativeTriangle problem for the subgraph of G induced by $A \cup B \cup C$. We consider each such induced subgraph and pad it with dummy vertices and edges so that it is a complete graph on $3d$ vertices, ensuring that we do not introduce any negative triangles. The latter can be achieved by setting the weights of dummy edges to be twice as large as the largest absolute value of an edge-weight in H. This reduction requires time linear in the total size of the input and the output and hence the stated bound follows. □

We will use the following reduction from NegativeTriangle to TED that was presented in [9].

Lemma 2 ([9, Lemma 2 and Theorem 2]). *Given a complete undirected n-vertex graph $H = (V, E)$ and a weight function $w : E \to \{1, \ldots, n^c\}$, we can construct, in linear time in the output size, an instance of* TED *of size $\mathcal{O}(n)$ such that the minimum weight of a triangle in H can be extracted from the edit distance. In particular, the constructed instance of* TED *satisfies the following:*[5]

- *c_{del} is an all-zeroes function;*
- *the trees are two opposing combs of depth $2n + 1$;*
- *the edit distance of the two trees is equal to $-3M^2$ plus the minimum weight of a triangle in H, where M is a (sufficiently large) integer parameter that is used to define c_{match}.*

In particular, the fact that the (shapes of the) trees in the above lemma are fixed means that information about the NegativeTriangle instance (H, w) is only encoded in the assignment of letters to nodes and the cost function $c_{match}(\cdot, \cdot)$. Hence, given t instances of NegativeTriangle of the same size, one can construct $\sqrt{t}$ left combs and $\sqrt{t}$ right combs, such that each of the t pairs of left and right combs corresponds to one of the NegativeTriangle instances. We can then assign a distinct letter to each node in each of the combs and define the cost function so that its restriction to any particular pair of left and right combs coincides with the cost function that Lemma 2 would yield for the NegativeTriangle instance corresponding to this pair.

In the following lemma, we combine the above idea with Lemma 1 into a subcubic-time reduction from NegativeTriangle to TED.

Lemma 3. NegativeTriangle *reduces in $\mathcal{O}(n^2 + n^3/d)$ time to an instance of* TED *over $\mathcal{O}(n^{1.5}/\sqrt{d})$-size and $\mathcal{O}(d)$-depth trees.*

Proof. We first apply Lemma 1 to reduce our NegativeTriangle instance to the problem of deciding whether any of $t = \mathcal{O}((n/d)^3)$ complete graphs on $3d$

[5] Not all of these properties are explicitly stated in [9, Lemma 2 and Theorem 2], but they are evident from their proofs.

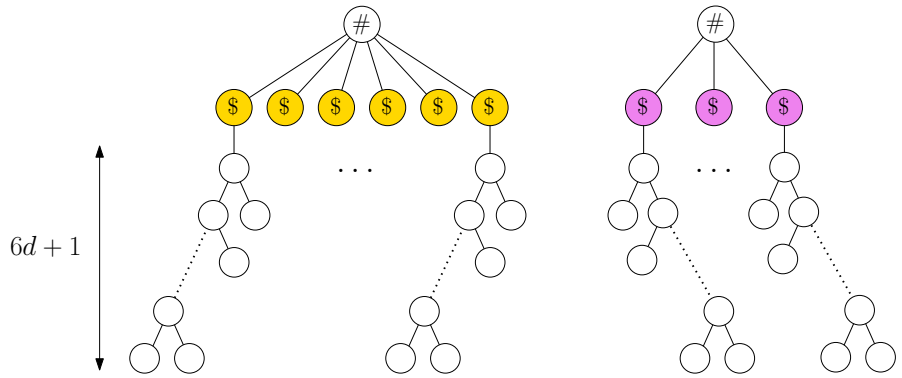

Fig. 3. A depiction of the instance of TED from the proof of Lemma 3 with $t = 4$ (and $s = 2$). Available nodes (in F) and selector nodes (in G) are colored yellow and purple, respectively. A comb of depth $6d+1$ is attached to *each* of the available/selector nodes—we only show a few of them for clarity.

vertices contains a negative triangle. Our goal is to efficiently reduce the latter problem to a TED instance with trees of depth $\mathcal{O}(d)$.

Let us denote the obtained graphs by $H_1, H_2, \ldots, H_t$. We construct a TED instance as follows. Let $s = \sqrt{t}$ and assume that it is an integer in order to avoid clutter. F consists of a root with $3s$ children, which we call *available nodes*. Each of these available nodes has a left comb of depth $2 \cdot 3d + 1 = 6d + 1$ attached to it. G consists of a root with s children, which we call *decider nodes*. Each of the decider nodes has a right comb of depth $6d + 1$ attached to it; see Figure 3. Observe that the sizes and the depths of these trees are as desired.

Our goal is to define a cost function $c_{match}(\cdot, \cdot)$ that ensures the following:

- the roots of the two trees are matched;
- each decider node is matched with an available node;
- the restriction of the cost function to the pair that consists of the $(s + i)$-th left comb in F and the j-th comb in G, for $i = 1, \ldots, s$ and $j = 1, \ldots, s$ is identical to the one yielded by Lemma 2 for H_m, where $m = s \cdot (i - 1) + j$;
- the restriction of the cost function to each other pair that consists of a left comb in F and a right comb in G is identical to the one yielded by Lemma 2 for some undirected graph on $3d$ vertices in which the minimum weight of a triangle is zero, where all such applications of Lemma 2 use the same (sufficiently large) integer parameter M.

To ensure the above properties, let us label the roots of both trees by # and all decider/available nodes by \$. In addition, let us label each other node with a unique letter from an alphabet Σ that is disjoint from $\{\#, \$\}$. We populate a table that corresponds to the c_{match} function as follows:

- For all $(x, y) \in \Sigma^2$ with x being a label of a node in F and y being a label of a node in G, i.e., pairs of labels of nodes from opposing combs, the cost

$c_{match}(x, y)$ is given by an application of Lemma 2. Let us denote the sum of the absolute values of all these costs by ψ.
- $c_{match}(\#, \#) = c_{match}(\$, \$) = -2\psi$;
- $c_{match}(\#, \$) = \infty$;
- $c_{match}(x, y) = \infty$ for all pairs $(x, y) \in \{\#, \$\} \times \Sigma$.

Claim. There is a negative triangle in at least one of $H_1, H_2, \ldots, H_t$ if and only if $\mathrm{TED}(F, G) < \eta := -2\psi \cdot (s + 1) - 3M^2 \cdot s$.

Proof. Let us denote the comb attached to the i-th available node by F_i and the one attached to the j-th decider node by G_j.

Now, the optimal solution must match the roots of the two trees and match every decider node with an available node. This is because these yield a cost of $-2\psi \cdot (s + 1)$, while the cost of any sequence of operations that do not match these nodes cannot be smaller than $-2\psi \cdot s - \psi$. Thus, the edit distance of F and G equals

$$\mathrm{TED}(F, G) = \min_p \left\{ -2\psi \cdot (s + 1) + \sum_{j=1}^{s} \mathrm{TED}(F_{p(j)}, G_j) \right\}$$

where the minimization is over all increasing functions $p : \{1, 2, \ldots, s\} \to \{1, 2, \ldots, 3s\}$. We therefore have two cases:

- If none of the graphs H_i contains a negative triangle, we might as well set every $p(j)$ to be j since the pairs (F_j, G_j) correspond to a graph in which the minimum triangle is of zero weight. So in this case we have

$$\mathrm{TED}(F, G) = -2\psi \cdot (s + 1) + \sum_{j=1}^{s} \mathrm{TED}(F_j, G_j) = \eta.$$

- Else, some graph H_i contains a negative triangle of weight $-w$. Let $q = \lceil i/s \rceil$ and r be an integer in $\{1, \ldots, s\}$ satisfying $r \equiv i \pmod{s}$. Notice that matching the pair (F_{s+q}, G_r) is cheaper than matching a pair corresponding to a minimum weight triangle of value zero. We therefore have

$$\mathrm{TED}(F, G) \leq -2\psi \cdot (s + 1) + \sum_{j=1}^{r-1} \mathrm{TED}(F_j, G_j) + \mathrm{TED}(F_{s+q}, G_r) + \sum_{j=r+1}^{s} \mathrm{TED}(F_{j+2s}, G_j) = \eta - w < \eta.$$

This completes the proof of the claim. □

The lemma follows. □

Theorem 1. *There exists no algorithm that solves* TED *for trees of size at most n and depth at most $d = \mathrm{poly(n)}$, with node labels from an alphabet of size $\Omega(n)$, in $\mathcal{O}(n^2 d^{1-\epsilon})$ time, for any constant $\epsilon > 0$, unless the APSP hypothesis fails.*

Proof. To the contrary, suppose that there is such an algorithm with $\epsilon < 1$. Let N denote the size of an APSP instance. Using Lemma 3 with $d = \text{poly(N)}$, we obtain an algorithm for APSP with runtime $\mathcal{O}(N^2 + N^3/d + (N^{1.5}/\sqrt{d})^2 \cdot d^{1-\epsilon}) = \mathcal{O}(N^3/d^\epsilon)$, contradicting the APSP hypothesis. □

4 Lower Bound for Decomposition Algorithms

Let us recall that the *decomposition algorithm* paradigm for the computation of tree edit distance is based on the following observation: given two forests F and G, the rightmost (or leftmost) roots of F and G are either matched or (at least) one of them is deleted. This observation leads to a dynamic programming approach: consider all three such options and recurse. The algorithm of Zhang and Shasha [35] proceeds by always considering the rightmost roots of the forests, while the algorithms of Klein [20] and Demaine et al. [12] use more intricate strategies (based on heavy-path decompositions) to decide whether to consider the rightmost or the leftmost roots of the forests in each step. In general, we call a mapping S from pairs of forests to the set $\{\mathsf{left}, \mathsf{right}\}$ a *strategy*. Previous lower bounds on decomposition algorithms were established by proving a lower bound on the number of different pairs of forests F' of F and G' of G that a decomposition algorithm will consider irrespective of the strategy S that it follows; we do not deviate from this approach.

Let us introduce some more terminology and notation. For a node v in a tree T, we denote by T_v the subtree of T rooted at v. Further, we call v's child u such that T_u is largest *heavy*, resolving ties arbitrarily; all other children of v are called *light*. If a node v in T has two children, this allows us to naturally refer to the two subtrees of the children of v as v's heavy and light subtrees. Further, for a tree T, we denote by T° the forest obtained by deleting the root of T.

We next specify our hard instance of TED. It consists of trees of size $\Theta(n)$ and depth $\Theta(d)$ for any parameters $d, n \in \mathbb{Z}_+$ with $n > 100d \geq 300$. For simplicity, we assume that d divides n. Each of the trees that we will consider in this lower bound consists of a path (also called *spine*) P of length $d+1$, with one endpoint of the path being the root of the tree and the other one being a leaf, and d trees of size $\lceil n/d \rceil$ and depth $\mathcal{O}(d)$, each attached to a distinct non-leaf node of P, in an alternating fashion with regards to being left/right subtrees. (The exact shape of these trees is not important.) See Figure 4 for an illustration. We call a non-leaf spine node u of F and a non-leaf spine node v of G *opposing* if and only if their children that lie on the corresponding spine are in different directions (i.e., left and right); two such opposing nodes are indicated in Figure 4.

Let us fix an arbitrary strategy S and denote by $\mathcal{U}(\mathsf{S})$ the set of subproblems that a decomposition algorithm with strategy S will encounter. Our goal is to give an $\Omega(\min\{n^2d^2, n^3\})$ lower bound on $|\mathcal{U}(\mathsf{S})|$.

Consider a subproblem $(F', G') \in \mathcal{U}(\mathsf{S})$. Suppose that the strategy S for this subproblem is right. In this case, the subproblems obtained by (i) deleting the root of the rightmost tree in F', (ii) deleting the root of the rightmost tree in G', and (iii) matching the roots of the rightmost trees in F' and G', all belong

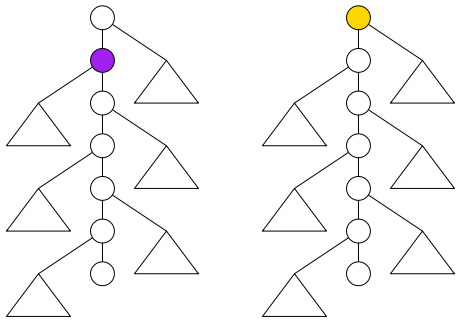

Fig. 4. An illustration of our hard instance for decomposition algorithms over shallow trees. Here, $d = 6$ and each of the subtrees attached to a spine node is of size $\mathcal{O}(\lceil n/6 \rceil) = \mathcal{O}(n)$ and depth $\mathcal{O}(1)$. A pair of opposing spine nodes is colored.

to $\mathcal{U}(\mathsf{S})$. In what follows, when we say that in such a scenario we *delete from* F (resp. G), we mean that we concentrate our attention on the subproblem created in option (i) (resp. (ii)) above. We stress that in reality both these subproblems (as well as those created in option (iii) above) belong to $\mathcal{U}(\mathsf{S})$, but that for our purposes it suffices to focus on a particular subproblem.

We rely on the following lemma from [12], stating that any strategy must consider matching every pair of nodes.

Lemma 4 ([12, Lemma 2.3]). *For any strategy S, for all $u \in F$, $v \in G$, $(F_u^\circ, G_v^\circ) \in \mathcal{U}(\mathsf{S})$.*

Our plan is to start from such subproblems $(F_u^\circ, G_v^\circ) \in \mathcal{U}(\mathsf{S})$, and, by choosing an appropriate sequence of deletions from F and from G, to obtain sufficiently many new subproblems. To make sure we do not double count subproblems obtained in this way, we will charge a subproblem consisting of a pair of forests F' and G' to the pair of spine nodes $p \in F$ and $q \in G$ that are the lowest common ancestors (LCAs) of the nodes of F' in F and of the nodes of G' in G, respectively. For a node v in a tree T, and for a positive integer x (resp. y) smaller than the size of the left (resp. right) subtree of T_v, we denote by $\mathsf{L}^x\mathsf{R}^y(T_v)$ the forest obtained from T_v° by x deletions of the leftmost root and y deletions of the rightmost root—we stress that the root of T_v has already been deleted prior to these $x + y$ deletions. Observe that v is the LCA of the nodes in $\mathsf{L}^x\mathsf{R}^y(T_v)$. In what follows, we refer to deletions of the leftmost and rightmost root as left deletions and right deletions, respectively.

Lemma 5. *Consider a spine node p in F of depth at most $d/2$ whose right child c is heavy. Let q be an opposing spine node in G of depth at most $d/2$ whose heavy child is w. Then, the total number of subproblems charged to (p, q), (c, q), and (p, w) is $\Omega(\min\{n^2, n^3/d^2\})$.*

Proof. Observe that the heavy subtree of each of p, c, q, and w has at least $n/3$ nodes. Let $\Delta := \{1, \ldots, \lceil n/4d \rceil\}$. We distinguish between two cases.

Case I: For every $(k, \ell) \in \Delta^2$, there is a subproblem in $\mathcal{U}(\mathsf{S})$ with exactly k deletions in the left subtree of F_p and exactly ℓ deletions in the right subtree of G_q, at least $\lfloor n/8 \rfloor$ nodes in the heavy (i.e., right) subtree of F_p, and at least $\lfloor n/8 \rfloor$ nodes in the heavy (i.e., left) subtree of G_q. In this case, starting from each of these $\lceil n/4d \rceil^2 = \Omega(n^2/d^2)$ pairs of forests, we consider the subproblems generated by always performing deletions only in the heavy subtrees of both F_p and G_q, i.e., deleting from F if and only if $\mathsf{S} = \mathsf{right}$. As each of these subtrees has at least $\lfloor n/8 \rfloor$ nodes, we obtain $\Omega(n)$ distinct subproblems of the form $(\mathsf{L}^k\mathsf{R}^x(F_p), \mathsf{L}^y\mathsf{R}^\ell(G_q))$ for each fixed k, ℓ. Since $k, \ell < \lceil n/4d \rceil$, $\mathsf{L}^k\mathsf{R}^x(F_p)$ (resp. $\mathsf{L}^y\mathsf{R}^\ell(G_q)$) contains nodes from both the left and right subtrees of F_p (resp. G_q). Hence the LCA of the nodes of $\mathsf{L}^k\mathsf{R}^x(F_p)$ (resp. $\mathsf{L}^y\mathsf{R}^\ell(G_q)$) is p (resp. q). We thus obtain a total of $\Omega(n^3/d^2)$ subproblems that are charged to the pair (p, q) of spine nodes. We are therefore done in this case.

For the remainder of the proof we can thus focus on the complementary case.

Case II: There is some pair $(k, \ell) \in \Delta^2$ for which we do not have a subproblem with k deletions in the left subtree of F_p and ℓ deletions in the right subtree of G_q, and at least $\lfloor n/8 \rfloor$ nodes in both heavy subtrees of F_p and G_q.

Claim. One of the following holds:

- for every integer $y \in [n/16, \lfloor n/8 \rfloor)$, there exists $\ell' < \ell$ such that
$$(\mathsf{L}^k(F_p), \mathsf{L}^y\mathsf{R}^{\ell'}(G_q)) \in \mathcal{U}(\mathsf{S}),$$
- for every integer $y \in [n/16, \lfloor n/8 \rfloor)$, there exists $k' < k$ such that
$$(\mathsf{L}^{k'}\mathsf{R}^y(F_p), \mathsf{R}^\ell(G_q)) \in \mathcal{U}(\mathsf{S}).$$

Proof. Consider applying the following sequence of operations starting from the pair (F_p°, G_q°), which is in $\mathcal{U}(\mathsf{S})$ by Lemma 4: delete from F whenever $\mathsf{S} = \mathsf{left}$ until we have reached $\mathsf{L}^k(F_p)$ and delete from G whenever $\mathsf{S} = \mathsf{right}$ until we have reached $\mathsf{R}^\ell(G_q)$. Let us only consider the case where the strategy says left k times before it says right ℓ times as the other case is symmetric.

Let A denote the subproblem obtained by performing k left deletions from F_p and some number $\ell'' < \ell$ right deletions from G_q. That is, $A = (\mathsf{L}^k(F_p), \mathsf{R}^{\ell''}(G_q))$. Since we are in Case II, for every integer $y \in [n/16, \lfloor n/8 \rfloor)$ we can, starting from A, only make deletions in G_q, making y left deletions from G_q and making less than $\ell - \ell''$ right deletions from G_q, thus obtaining the subproblem $(\mathsf{L}^k(F_p), \mathsf{L}^y\mathsf{R}^{\ell'}(G_q))$ for some ℓ' satisfying $\ell'' \leq \ell' < \ell$. □

Let us assume without loss of generality that we are in the first case of the above claim, as the other case is symmetric. Let us fix some value of $y \in [n/16, \lfloor n/8 \rfloor)$ and the corresponding subproblem $(\mathsf{L}^k(F_p), \mathsf{L}^y\mathsf{R}^{\ell'}(G_q))$, where $\ell' < \ell$. We prove the following claim.

Claim. There exist $\Omega(\min\{n, n^2/d^2\})$ quadruples (x, m, s, v) where x, m, and s are integers, and $v \in \{p, c\}$, such that
$$(\mathsf{L}^x\mathsf{R}^m(F_v), \mathsf{L}^y\mathsf{R}^s(G_q)) \in \mathcal{U}(\mathsf{S}).$$

Proof. Starting from our fixed subproblem $(\mathsf{L}^k(F_p), \mathsf{L}^y\mathsf{R}^{\ell'}(G_q))$, we consider making all left deletions in F. However, for each $t \in \Delta$, we consider making the first t right deletions in G and the remaining ones in F. We distinguish between two cases depending on whether, for each pair $(m,t) \in \Delta^2$, we obtain a subproblem of the form:

$$(\mathsf{L}^x\mathsf{R}^m(F_v), \mathsf{L}^y\mathsf{R}^{\ell'+t}(G_q)), \text{ for some integer } x \text{ and } v \in \{p,c\}.$$

1. If this is the case, we obtain $\Omega(n^2/d^2)$ of the desired quadruples, and we are thus done.
2. Else, let $(m',t') \in \Delta^2$ be a pair for which there is no integer x and node $v \in \{p,c\}$ such that $(\mathsf{L}^x\mathsf{R}^{m'}(F_v), \mathsf{L}^y\mathsf{R}^{\ell'+t'}(G_q)) \in \mathcal{U}(\mathsf{S})$. This can only be the case if we eliminate the entire heavy (i.e., left) subtree of T_c prior to making the intended $m'+t'$ right deletions. In other words, this can only happen if along this computational path of the recursion, S says left $\Omega(n)$ times before it says right $m'+t'$ times. In this case, for each $x \in [n/16, \lfloor n/8 \rfloor)$, there exist $m'', t'' \in \Delta^2$ such that $(\mathsf{L}^x\mathsf{R}^{m''}(F_c), \mathsf{L}^y\mathsf{R}^{\ell'+t''}(G_q)) \in \mathcal{U}(\mathsf{S})$. In this case, we thus obtain $\Omega(n)$ quadruples of the desired form.

This completes the proof of the claim. □

Thus, for each of $\Omega(n)$ values of y, we obtain $\Omega(\min\{n, n^2/d^2\})$ subproblems. Over all such y, we thus obtain $\Omega(\min\{n^2, n^3/d^2\})$ subproblems charged to (p,q) and (c,q), thus completing the analysis of Case II. □

As our instance of TED has $\Omega(d^2)$ pairs of spine nodes p and q that satisfy the conditions of Lemma 5, we obtain the main result of this section:

Theorem 2. *Any decomposition algorithm for tree edit distance on trees of size at most n and depth at most d requires $\Omega(\min\{n^3, n^2d^2\})$ time.*

References

1. Akmal, S., Jin, C.: Faster algorithms for bounded tree edit distance. In: 48th ICALP. pp. 12:1–12:15 (2021). https://doi.org/10.4230/LIPIcs.ICALP.2021.12
2. Akutsu, T., Fukagawa, D., Takasu, A.: Approximating tree edit distance through string edit distance. In: 17th ISAAC. pp. 90–99 (2006). https://doi.org/10.1007/11940128_11
3. Alur, R., D'Antoni, L., Gulwani, S., Kini, D., Viswanathan, M.: Automated grading of dfa constructions. In: 23rd IJCAI. pp. 1976–1982 (2013), http://dl.acm.org/citation.cfm?id=2540128.2540412
4. Apostolico, A., Galil, Z. (eds.): Pattern matching algorithms. Oxford University Press, Oxford, UK (1997)
5. Aratsu, T., Hirata, K., Kuboyama, T.: Approximating tree edit distance through string edit distance for binary tree codes. Fundam. Inform. **101**(3), 157–171 (2010). https://doi.org/10.3233/FI-2010-282

6. Bellando, J., Kothari, R.: Region-based modeling and tree edit distance as a basis for gesture recognition. In: 10th International Conference on Image Analysis and Processing, ISIAP 1999. pp. 698–703 (1999). https://doi.org/10.1109/ICIAP.1999.797676
7. Bille, P.: A survey on tree edit distance and related problems. Theoretical Computer Science **337**(1-3), 217–239 (2005). https://doi.org/10.1016/j.tcs.2004.12.030
8. Boroujeni, M., Ghodsi, M., Hajiaghayi, M., Seddighin, S.: $1+\epsilon$ approximation of tree edit distance in quadratic time. In: 51st STOC. pp. 709–720. ACM (2019). https://doi.org/10.1145/3313276.3316388
9. Bringmann, K., Gawrychowski, P., Mozes, S., Weimann, O.: Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). ACM Trans. Algorithms **16**(4), 48:1–48:22 (2020). https://doi.org/10.1145/3381878
10. Buneman, P., Grohe, M., Koch, C.: Path queries on compressed XML. In: VLDB. pp. 141–152 (2003). https://doi.org/10.1016/B978-012722442-8/50021-5
11. Chawathe, S.: Comparing hierarchical data in external memory. In: VLDB. pp. 90–101 (1999), http://www.vldb.org/conf/1999/P8.pdf
12. Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An optimal decomposition algorithm for tree edit distance. ACM Trans. Algorithms **6**(1), 2:1–2:19 (2009). https://doi.org/10.1145/1644015.1644017
13. Dudek, B., Gawrychowski, P.: Edit distance between unrooted trees in cubic time. In: 45th ICALP. pp. 45:1–45:14 (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.45
14. Dulucq, S., Touzet, H.: Decomposition algorithms for the tree edit distance problem. J. Discrete Algorithms **3**(2-4), 448–471 (2005). https://doi.org/10.1016/j.jda.2004.08.018
15. Dürr, A.: Improved bounds for rectangular monotone min-plus product and applications. Arxiv 2208.02862v1 (2022)
16. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and indexing labeled trees, with applications. J. ACM **57**, 1–33 (2009). https://doi.org/10.1145/1613676.1613680
17. Gusfield, D.: Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge University Press (1997)
18. Hoffmann, C.M., O'Donnell, M.J.: Pattern matching in trees. J. ACM **29**(1), 68–95 (1982). https://doi.org/10.1145/322290.322295
19. Klein, P.N., Tirthapura, S., Sharvit, D., Kimia, B.B.: A tree-edit-distance algorithm for comparing simple, closed shapes. In: 11th SODA. pp. 696–704 (2000), http://dl.acm.org/citation.cfm?id=338219.338628
20. Klein, P.N.: Computing the edit-distance between unrooted ordered trees. In: 6th ESA. pp. 91–102 (1998). https://doi.org/10.1007/3-540-68530-8_8
21. Mao, X.: Breaking the cubic barrier for (unweighted) tree edit distance. In: 62nd FOCS. pp. 792–803 (2021). https://doi.org/10.1109/FOCS52979.2021.00082
22. Pawlik, M., Augsten, N.: Efficient computation of the tree edit distance. ACM Trans. Database Syst. **40**(1), 3:1–3:40 (Mar 2015). https://doi.org/10.1145/2699485
23. Rico-Juan, J.R., Micó, L.: Comparison of AESA and LAESA search algorithms using string and tree-edit-distances. Pattern Recognition Letters **24**(9-10), 1417–1426 (2003). https://doi.org/10.1016/S0167-8655(02)00382-3
24. Selkow, S.: The tree-to-tree editing problem. Information Processing Letters **6**(6), 184–186 (1977). https://doi.org/10.1016/0020-0190(77)90064-3
25. Shapiro, B.A., Zhang, K.: Comparing multiple RNA secondary structures using tree comparisons. Computer Applications in the Biosciences **6**(4), 309–318 (1990). https://doi.org/10.1093/bioinformatics/6.4.309

26. Shasha, D., Zhang, K.: Fast algorithms for the unit cost editing distance between trees. Journal of Algorithms **11**(4), 581–621 (1990). https://doi.org/10.1016/0196-6774(90)90011-3
27. Tai, K.: The tree-to-tree correction problem. J. ACM **26**(3), 422–433 (1979). https://doi.org/10.1145/322139.322143
28. Touzet, H.: Comparing similar ordered trees in linear-time. J. Discrete Algorithms **5**(4), 696–705 (2007). https://doi.org/10.1016/j.jda.2006.07.002
29. Valiente, G.: Algorithms on Trees and Graphs. Springer-Verlag (2002)
30. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM **21**(1), 168–173 (1974). https://doi.org/10.1145/321796.321811
31. Waterman, M.: Introduction to computational biology: maps, sequences and genomes, *Chapters 13, 14*. Chapman and Hall (1995)
32. Williams, V.V., Williams, R.R.: Subcubic equivalences between path, matrix, and triangle problems. J. ACM **65**(5), 27:1–27:38 (2018). https://doi.org/10.1145/3186893
33. Yao, X., Durme, B.V., Callison-Burch, C., Clark, P.: Answer extraction as sequence tagging with tree edit distance. In: HLT-NAACL 2013. pp. 858–867 (2013), http://aclweb.org/anthology/N/N13/N13-1106.pdf
34. Zhang, K.: Algorithms for the constrained editing distance between ordered labeled trees and related problems. Pattern Recognition **28**(3), 463–474 (1995). https://doi.org/10.1016/0031-3203(94)00109-Y
35. Zhang, K., Shasha, D.E.: Simple fast algorithms for the editing distance between trees and related problems. SIAM J. Comput. **18**(6), 1245–1262 (1989). https://doi.org/10.1137/0218082