Faster Algorithm for Bounded Tree Edit Distance in the Low-Distance Regime

Tomasz Kociumaka
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Ali Shahali
Sharif University of Technology, Tehran, Iran

Abstract

The *tree edit distance* is a natural dissimilarity measure between rooted ordered trees whose nodes are labeled over an alphabet Σ. It is defined as the minimum number of node edits – insertions, deletions, and relabelings – required to transform one tree into the other. The weighted variant assigns costs ≥ 1 to edits (based on node labels), minimizing total cost rather than edit count.

The unweighted tree edit distance between two trees of total size n can be computed in $\mathcal{O}(n^{2.6857})$ time; in contrast, determining the weighted tree edit distance is fine-grained equivalent to the All-Pairs Shortest Paths (APSP) problem and requires $n^3/2^{\Omega(\sqrt{\log n})}$ time [Nogler, Polak, Saha, Vassilevska Williams, Xu, Ye; STOC'25]. These impractical super-quadratic times for large, similar trees motivate the bounded version, parameterizing runtime by the distance k to enable faster algorithms for $k \ll n$.

Prior algorithms for bounded unweighted edit distance achieve $\mathcal{O}(nk^2 \log n)$ [Akmal & Jin; ICALP'21] and $\mathcal{O}(n + k^7 \log k)$ [Das, Gilbert, Hajiaghayi, Kociumaka, Saha; STOC'23]. For weighted, only $\mathcal{O}(n + k^{15})$ is known [Das, Gilbert, Hajiaghayi, Kociumaka, Saha; STOC'23].

We present an $\mathcal{O}(n + k^6 \log k)$-time algorithm for bounded tree edit distance in both weighted/unweighted settings. First, we devise a simpler weighted $\mathcal{O}(nk^2 \log n)$-time algorithm. Next, we exploit periodic structures in input trees via an optimized universal kernel: modifying prior $\mathcal{O}(n)$-time $\mathcal{O}(k^5)$-size kernels to generate such structured instances, enabling efficient analysis.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases tree edit distance, edit distance, kernelization, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.94

Related Version *Full Version*: `https://arxiv.org/abs/2507.02701` [27]

Funding *Ali Shahali*: The work of was carried out mostly during a summer internship at the Max Planck Institute for Informatics.

1 Introduction

The edit distance between two strings – the minimum cost of insertions, deletions, and substitutions needed to transform one string into the other – is one of the most widely used string similarity measures with numerous algorithmic applications. It provides a robust model for comparing sequential data and underpins techniques in fields such as computational biology, natural language processing, and text correction. Nevertheless, many data types exhibit hierarchical rather than linear structure. For example, RNA secondary structures, syntactic parse trees of natural language sentences, and hierarchical representations of documents and code all store information in tree-like forms. One can linearize such data and still use string edit distance, but the resulting alignments disregard the original hierarchical structure, and thus more expressive similarity measures are needed in most scenarios.

First introduced by Selkow [38], tree edit distance generalizes the notion of string edit distance to rooted ordered trees and forests with nodes labeled over an alphabet Σ. It quantifies the dissimilarity between two forests as the minimal cost of a sequence of node

edits – insertions, deletions, and relabelings – required to transform one forest into the other. This distance naturally captures both structural and label-based differences and serves as a core primitive in various algorithmic and applied domains, including computational biology [22, 39, 23, 45], analysis of structured data (such as XML and JSON files) [9, 13, 12, 20, 44], image processing [5, 26, 25, 36], and information extraction [16, 48]; see [6, 2] for surveys.

Tai [41] proposed the first polynomial-time algorithm for computing the tree edit distance: a dynamic programming procedure running in $\mathcal{O}(n^6)$ time, where n is the total number of nodes in the input forests. In the following decades, a sequence of works progressively improved the runtime. Zhang and Shasha [49] introduced an $\mathcal{O}(n^4)$-time algorithm, and then Klein [24] brought the complexity down to $\mathcal{O}(n^3 \log n)$. Subsequently, Demaine, Mozes, Rossman, and Weimann [17] presented an $\mathcal{O}(n^3)$-time solution, whereas Bringmann, Gawrychowski, Mozes, and Weimann [8] proved that a hypothetical $n^{3-\Omega(1)}$-time algorithm would violate the All-Pairs Shortest Paths (APSP) hypothesis in fine-grained complexity. Very recently, Nogler, Polak, Saha, Vassilevska Williams, Xu, and Ye [33] showed that computing the tree edit distance is, in fact, equivalent to the APSP problem, and the $n^3/2^{\Omega(\sqrt{\log n})}$ running time can be inherited from the state of the art for the latter task [46].

The aforementioned conditional lower bounds apply to the weighted tree edit distance only, where the cost of each edit depends on the involved labels. In the unweighted version with unit costs, a series of recent results achieved truly subcubic running time using fast matrix multiplication. Mao [29] presented an $\mathcal{O}(n^{2.9546})$-time solution, which was subsequently improved by Dürr [18] to $\mathcal{O}(n^{2.9148})$ and by Nogler et al. [33] to $\widetilde{\mathcal{O}}(n^{(3+\omega)/2}) = \mathcal{O}(n^{2.6857})$, where ω is the fast matrix multiplication exponent and $\widetilde{\mathcal{O}}(\cdot)$ hides $\operatorname{poly}\log n$ factors. The state-of-the-art conditional lower bound, inherited from string edit distance [4] and assuming the Orthogonal Vectors Hypothesis, prohibits $n^{2-\Omega(1)}$-time algorithms already if $|\Sigma| = 1$.

Bounded Tree Edit Distance. Even though the decades of research resulted in significantly faster tree edit distance algorithms, they also revealed fundamental challenges that explain the difficulty of this problem. A natural way of circumventing these barriers, paved by classical work for string edit distance [43, 31, 28], is to parameterize the running time not only in terms of the length n of the input forests but also the value k of the computed distance. This version of tree edit distance has originally been studied in the unweighted setting only. Touzet [42] adapted the ideas of Zhang and Shasha [49] to derive an $\mathcal{O}(nk^3)$-time algorithm, whereas Akmal and Jin [1] improved the running time to $\mathcal{O}(nk^2 \log n)$ based on the approach of Klein [24]. The latter running time remains the state-of-the-art for medium distances. In the low-distance regime, Das, Gilbert, Hajiaghayi, Kociumaka, Saha, and Saleh [15] achieved an $\widetilde{\mathcal{O}}(n + k^{15})$-time solution and subsequently improved the running time to $\mathcal{O}(n + k^7 \log k)$ [14]. They also studied the weighted version of the problem and presented an $\mathcal{O}(n + k^{15})$-time algorithm assuming that the weight function is normalized (each edit costs at least one unit; this prohibits arbitrary scaling) and satisfies the triangle inequality.

Still, the $\widetilde{\mathcal{O}}(n + \operatorname{poly}(k))$ running times for tree edit distance are much larger than for string edit distance and the related Dyck edit distance problem, which asks for the minimum cost of edits needed to make a given string of parentheses balanced (so that it represents a node-labeled forest). In the unweighted setting, the state-of-the-art running times are $\mathcal{O}(n + k^2)$ for string edit distance [28] and $\mathcal{O}(n + k^{5.442})$ for Dyck edit distance [21]. In the presence of weights, these complexities increase to $\widetilde{\mathcal{O}}(n + \sqrt{nk^3}) \le \widetilde{\mathcal{O}}(n + k^3)$ [10] and $\mathcal{O}(n + k^{12})$ [14], respectively. Thus, tree edit distance is a natural candidate for improvements.

Our Results. As the main contribution, we bring the time complexity of tree edit distance to $\widetilde{\mathcal{O}}(n+k^6)$. Even for unit weights, this improves upon the state of the art if $n^{1/7} \ll k \ll n^{1/4}$.

▶ **Theorem 1.1.** *There exists a deterministic algorithm that, given two forests F and G with n nodes in total, each with a label from an alphabet Σ, and oracle access to a normalized weight function $w : (\Sigma \cup \{\varepsilon\})^2 \to \mathbb{R}_{\geq 0}$ satisfying the triangle inequality, determines the tree edit distance $k \coloneqq \mathsf{ted}^w(F,G)$ in $\mathcal{O}(n + k^6 \log k)$ time.*

We also prove that the $\mathcal{O}(nk^2 \log n)$ running time by Akmal and Jin [1] remains valid in the presence of weights. This approach gives the best complexity when $n^{1/4} \ll k \ll n$.

▶ **Theorem 1.2.** *There exists a deterministic algorithm that, given two forests F and G, with n nodes in total, each with a label from an alphabet Σ, and oracle access to a normalized weight function $w : (\Sigma \cup \{\varepsilon\})^2 \to \mathbb{R}_{\geq 0}$, determines $k \coloneqq \mathsf{ted}^w(F,G)$ in $\mathcal{O}(nk^2 \log n)$ time.*

We believe that the original approach of Akmal and Jin [1] supports the weighted setting with minimal adjustments; nevertheless, we prove Theorem 1.2 using a slightly different algorithm that is better-suited for further optimizations. Both solutions are variants of Klein's dynamic programming [24] with some states pruned. The main difference is that we view the input forests as balanced sequences of parentheses and cast the tree edit distance as the minimum cost of a string alignment satisfying a certain consistency property. In our interpretation (Section 3), it is almost trivial to see that the classic pruning rules, originally devised to compute string edit distance in $\mathcal{O}(nk)$ time [43, 31], can be reused for tree edit distance in $\mathcal{O}(nk^2 \log n)$ time. In contrast, Akmal and Jin [1, Lemma 9] spend over four pages (on top of the analysis of [24]) to bound the number of states surviving their pruning rules.

Theorem 1.2 combined with the results of [14] lets us derive an $\mathcal{O}(n + k^7 \log k)$-time algorithm for weighted tree edit distance, matching the previously best running time for unit weights. This is due to the *universal kernel* that transforms the input forests to equivalent forests of size $\mathcal{O}(k^5)$ preserving the following capped tree edit distance value:

$$\mathsf{ted}^w_{\leq k}(F,G) = \begin{cases} \mathsf{ted}^w(F,G) & \text{if } \mathsf{ted}^w(F,G) \leq k, \\ \infty & \text{otherwise.} \end{cases}$$

▶ **Theorem 1.3** ([14, Corollary 3.20])**.** *There exists a linear-time algorithm that, given forests F, G and an integer $k \in \mathbb{Z}_+$, constructs forests F', G' of size $\mathcal{O}(k^5)$ such that $\mathsf{ted}^w_{\leq k}(F,G) = \mathsf{ted}^w_{\leq k}(F',G')$ holds for every normalized weight function w satisfying the triangle inequality.*

To bring the time complexity from $\widetilde{\mathcal{O}}(n+k^7)$ to $\widetilde{\mathcal{O}}(n+k^6)$, we adapt both Theorems 1.2 and 1.3. The main novel insight is that the dynamic-programming procedure behind Theorem 1.2 behaves predictably while processing regions with certain repetitive (periodic) structure. Upon an appropriate relaxation of the invariant that the dynamic-programming values satisfy, processing each repetition of the period can be interpreted as a single min-plus matrix-vector multiplication. When the period repeats many times, the same matrix is used each time, and thus we can raise the matrix to an appropriate power (with fast exponentiation) and then compute a single matrix-vector product; see Section 4 for details. This can be seen as a natural counterpart of an optimization used in [10] for string edit distance.[1]

[1] The matrices arising in the string edit distance computation in [10] are $\mathcal{O}(k) \times \mathcal{O}(k)$ Monge matrices, which allows for computing each matrix-vector product in $\mathcal{O}(k)$ time and each matrix-matrix product in $\mathcal{O}(k^2)$ time. In the context of tree edit distance, the Monge property is no longer satisfied, so the complexities increase to $\mathcal{O}(k^2)$ and $\mathcal{O}(k^3)$ respectively; in fact, computing the (weighted) tree edit distance is, in general, as hard as computing the min-plus product of two $n \times n$ arbitrary matrices [33].

Intuitively, the worst-case instances for the kernelization algorithm of Theorem 1.3 are close to having the necessary structure for our optimization to improve the worst-case running time. Unfortunately, the implementation in [14] controls the forest sizes only, and it needs to be modified to keep track of the more subtle instance difficulty measure. The biggest challenge is that the original algorithm proceeds in logarithmically many steps, each shrinking the forests by a constant fraction. In every step, the forests are partitioned into $\mathcal{O}(k)$ pieces of size $\mathcal{O}(n/k)$, and a constant fraction of pieces is shrunk to size $\mathcal{O}(k^4)$ each. A partition into pieces of equal "difficulty" would be much more challenging, so we instead take a different approach that lets us implement the reduction in a single step; see Section 5 for details. Notably, the kernelization algorithms for weighted string and Dyck edit distance already have single-step implementations in [14], but they crucially rely on the fact that the unit-weight variants of these problems can be solved faster, unlike for tree edit distance.

Related Work

The classical definition of tree edit distance [38] involves labeled rooted trees (or forests) with node labels. Many other variants have also been studied allowing, among others, unordered [47, 40], unrooted [40], and edge-labeled [3] trees; see also the surveys [2, 6].

The tree edit distance problem becomes easier not only when the computed distance is small but also when the input forests are of small depth [49, 11] or when the approximate distance suffices [3, 7, 37]. There is also a body of work focusing on practical performance and empirical evaluation, including sequential [34, 35] and parallel [19] implementations.

2 Preliminaries

Consistently with [14], we identify forests with the underlying Euler tours, interpreted as balanced strings of parentheses. This representation is at the heart of space-efficient data structures on trees [30, 32], and it allows for a simple definition of tree edits and a seamless transfer of many tools from strings to forests. Notably, Klein's algorithm [24] already uses substrings of the Euler tours of the input forests to identify the dynamic-programming states.

For an alphabet Σ, let $\mathsf{P}_\Sigma := \bigcup_{a\in\Sigma}\{(_a,)_a\}$ denote the set parentheses with labels in Σ. As formalized next, a *forest* with node labels over Σ is a *balanced* string of parentheses in P_Σ.

▶ **Definition 2.1.** *The set of* forests *with labels over* Σ *(*balanced strings *over* P_Σ*) is the smallest subset* $\mathcal{F}_\Sigma \subseteq \mathsf{P}_\Sigma^*$ *satisfying the following conditions:*

- $\varepsilon \in \mathcal{F}_\Sigma$,
- $F \cdot G \in \mathcal{F}_\Sigma$ *for every* $F, G \in \mathcal{F}_\Sigma$,
- $(_a \cdot F \cdot)_a \in \mathcal{F}_\Sigma$ *for every* $F \in \mathcal{F}_\Sigma$ *and* $a \in \Sigma$.

For a forest F, we define the set of *nodes* V_F as the set of intervals $[i\,..\,j] \subseteq [0\,..\,|F|)$ such that $F[i]$ is an opening parenthesis, $F[j]$ is a closing parenthesis, and $F(i\,..\,j)$ is balanced. A forest F is a *tree* if $[0\,..\,|F|) \in V_F$. For a node $u = [i\,..\,j] \in V_F$, we denote the positions of the opening and the closing parenthesis by $o(u) := i$ and $c(u) := j$. The label of u, that is, the character $a \in \Sigma$ such that $F[o(u)] = (_a$ and $F[c(u)] =)_a$, is denoted by $\mathsf{lbl}(u)$. A simple inductive argument shows that V_F forms a laminar family and, for every $i \in [0\,..\,|F|)$, there is a unique node $u \in V_F$ such that $i \in \{o(u), c(u)\}$; we denote this node by $\mathsf{node}_F(i)$. We also write $\mathsf{mate}_F(i)$ for the paired parenthesis, that is, $\mathsf{mate}_F(i) = j$ if $\{i, j\} = \{o(u), c(u)\}$.

We say that a node $u \in V_F$ *is contained* in a fragment $F[i\,..\,j)$ of a forest F if $i \le o(u) < c(u) < j$; we denote by $V_{F[i\,..\,j)} \subseteq V_F$ the set of nodes contained in $F[i\,..\,j)$. Moreover, u *enters* $F[i\,..\,j)$ if $o(u) < i \le c(u) < j$ and u *exits* $F[i\,..\,j)$ if $i \le o(u) < j \le c(u)$. In either of these two cases, we also say that u *straddles* $F[i\,..\,j)$.

We denote by $F_{[i..j)}$ the *subforest of F induced by $F[i..j)$*, which we obtain from F by deleting (the parentheses corresponding to) all nodes except for those contained in $F[i..j)$. Alternatively, one can obtain $F_{[i..j)}$ from $F[i..j)$ by deleting the opening parenthesis of every node that exits $F[i..j)$ and the closing parenthesis of every node that enters $F[i..j)$.

2.1 Tree Edits, Forest Alignments, and Tree Edit Distance

For an alphabet Σ, we define $\bar{\Sigma} := \Sigma \cup \{\varepsilon\}$, where ε is the empty string. We say that a function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$ is *normalized* if $w(a,a) = 0$ and $w(a,b) \geq 1$ hold for distinct $a, b \in \bar{\Sigma}$.

Tree edit distance is classically defined using elementary edits transforming $F \in \mathcal{F}_\Sigma$:

Node insertion produces $F[0..i) \cdot (_a \cdot F[i..j) \cdot)_a \cdot F[j..|F|)$ for a balanced fragment $F[i..j)$ and a label $a \in \Sigma$, at cost $w(\varepsilon, a)$.

Node relabeling produces $F[0..o(u)) \cdot (_a \cdot F(o(u)..c(u)) \cdot)_a \cdot F(c(u)..|F|)$ for a node $u \in V_F$ and a label $a \in \Sigma$, at cost $w(\mathsf{lbl}(u), a)$.

Node deletion produces $F[0..o(u)) \cdot F(o(u)..c(u)) \cdot F(c(u)..|F|)$ for a node $u \in V_F$, at cost $w(\mathsf{lbl}(u), \varepsilon)$.

The tree edit distance $\mathsf{ted}^w(F,G)$ of two forests $F, G \in \mathcal{F}_\Sigma$ is then defined as the minimum cost of a sequence of edits transforming F to G. In this context, without loss of generality, we can replace w by its transitive closure. If, say $w(a,\varepsilon) > w(a,b) + w(b,\varepsilon)$, instead of directly deleting a node with label a, it is more beneficial to first change its label to b and only then perform the deletion. When w satisfies the triangle inequality, we are guaranteed that an inserted or relabeled node is never modified (deleted or relabeled) again. Consistently with modern literature [8, 14, 33], we use a more general *alignment-based* definition of $\mathsf{ted}^w(F,G)$ that enforces the latter condition even if w does not necessarily satisfy the triangle inequality.

▶ **Definition 2.2** (Alignment Graph [10]). *For strings $X, Y \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, we define the* alignment graph $\mathsf{AG}^w(X,Y)$ *as a grid graph with vertices $[0..|X|] \times [0..|Y|]$ and the following directed edges:*

- *horizontal edges $(x,y) \to (x+1,y)$ of cost $w(X[x],\varepsilon)$ for $(x,y) \in [0..|X|) \times [0..|Y|]$,*
- *vertical edges $(x,y) \to (x,y+1)$ of cost $w(\varepsilon, Y[y])$ for $(x,y) \in [0..|X|] \times [0..|Y|)$, and*
- *diagonal edges $(x,y) \to (x+1,y+1)$ of cost $w(X[x],Y[y])$ for $(x,y) \in [0..|X|) \times [0..|Y|)$.*

The alignment graph allows for a concise definition of a string *alignment*.

▶ **Definition 2.3** (Alignment). *For strings $X, Y \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, an* alignment *of $X[x..x')$ onto $Y[y..y')$, denoted by $\mathcal{A} : X[x..x') \rightsquigarrow Y[y..y')$, is a path from (x,y) to (x',y') in $\mathsf{AG}^w(X,Y)$, interpreted as a sequence of vertices. The* cost $\mathsf{ed}^w_{\mathcal{A}}(X[x..x'), Y[y..y'))$ *of the alignment $\mathcal{A}$ is the total costs of the edges that belong to $\mathcal{A}$.*

We write $\mathbf{A}(X[x..x'), Y[y..y'))$ for the set of all alignments of $X[x..x')$ onto $Y[y..y')$.

The edges of an alignment $\mathcal{A} = \mathbf{A}(X[x..x'), Y[y..y'))$ can be interpreted as follows:

- If $\mathcal{A}$ includes an edge $(\hat{x},\hat{y}) \to (\hat{x}+1,\hat{y})$ for some $\hat{x} \in [x..x')$ and $\hat{y} \in [y..y']$, then $\mathcal{A}$ *deletes* $X[\hat{x}]$, denoted by $X[\hat{x}] \rightsquigarrow_{\mathcal{A}} \varepsilon$.
- If $\mathcal{A}$ includes an edge $(\hat{x},\hat{y}) \to (\hat{x},\hat{y}+1)$ for some $\hat{x} \in [x..x']$ and $\hat{y} \in [y..y')$, then $\mathcal{A}$ *inserts* $Y[\hat{y}]$, denoted by $\varepsilon \rightsquigarrow_{\mathcal{A}} Y[\hat{y}]$.
- If $\mathcal{A}$ includes an edge $(\hat{x},\hat{y}) \to (\hat{x}+1,\hat{y}+1)$ for some $\hat{x} \in [x..x')$ and $\hat{y} \in [y..y')$, then $\mathcal{A}$ *aligns* $X[\hat{x}]$ to $Y[\hat{y}]$, denoted by $X[\hat{x}] \rightsquigarrow_{\mathcal{A}} Y[\hat{y}]$. If $X[\hat{x}] \neq Y[\hat{y}]$, then $\mathcal{A}$ *substitutes* $X[\hat{x}]$ for $Y[\hat{y}]$. If $X[\hat{x}] = Y[\hat{y}]$, then $\mathcal{A}$ *matches* $X[\hat{x}]$ with $Y[\hat{y}]$.

Insertions, deletions, and substitutions are jointly called character *edits*.

The weighted edit distance of fragments $X[x\,..\,x')$ and $Y[y\,..\,y')$ with respect to a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$ is defined as the minimum cost of an alignment:

$$\mathsf{ed}^w(X[x\,..\,x'), Y[y\,..\,y')) = \min_{\mathcal{A}\in\mathbf{A}(X[x\,..\,x'),Y[y\,..\,y'))} \mathsf{ed}^w_{\mathcal{A}}(X[x\,..\,x'), Y[y\,..\,y')).$$

We often consider alignments of the entire string X onto the entire string Y; we then used simplified notation including $\mathbf{A}(X,Y)$, $\mathsf{ed}^w_{\mathcal{A}}(X,Y)$, and $\mathsf{ed}^w(X,Y)$.

▶ **Definition 2.4** (Forest alignment)**.** *Consider forests $F, G \in \mathcal{F}_\Sigma$. An alignment $\mathcal{A} \in \mathbf{A}(F[f\,..\,f'), G[g\,..\,g'))$ is a* forest alignment *if it satisfies the following consistency condition:*

For every two aligned characters $F[\hat{f}] \rightsquigarrow_{\mathcal{A}} G[\hat{g}]$, also $F[\mathsf{mate}_F(\hat{f})] \rightsquigarrow_{\mathcal{A}} G[\mathsf{mate}_G(\hat{g})]$.

We write $\mathbf{FA}(F[f\,..\,f'), G[g\,..\,g')) \subseteq \mathbf{A}(F[f\,..\,f'), G[g\,..\,g'))$ for the set of forest alignments.

▶ Remark 2.5. Consider a forest alignment $\mathcal{A} \in \mathbf{FA}(F[f\,..\,f'), G[g\,..\,g'))$. Then,

- $\mathcal{A}$ deletes every character $F[\hat{f}]$ with $\hat{f} \in [f\,..\,f')$ and $\mathsf{mate}_F(\hat{f}) \notin [f\,..\,f')$,
- $\mathcal{A}$ inserts every character $G[\hat{g}]$ with $\hat{g} \in [g\,..\,g')$ and $\mathsf{mate}_G(\hat{g}) \notin [g\,..\,g')$.

Define $\overline{\mathsf{P}_\Sigma} = \mathsf{P}_\Sigma \cup \{\varepsilon\}$ and a mapping $\lambda : \overline{\mathsf{P}_\Sigma} \to \bar{\Sigma}$ such that $\lambda(\langle_a) = \lambda(\rangle_a) = a$ for each $a \in \Sigma$, and $\lambda(\varepsilon) = \varepsilon$. For a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, we define a corresponding weight function $\tilde{w} : \overline{\mathsf{P}_\Sigma}^2 \to \mathbb{R}_{\geq 0}$ so that $\tilde{w}(p,q) = \frac{1}{2}w(\lambda(p), \lambda(q))$ for all $p, q \in \overline{\mathsf{P}_\Sigma}$. The *cost* of a forest alignment $\mathcal{A} \in \mathbf{FA}(F[f\,..\,f'), G[g\,..\,g'))$ with respect to a weight function w is defined as $\mathsf{ted}^w_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g')) := \mathsf{ed}^{\tilde{w}}_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g'))$. Moreover, we define

$$\mathsf{ted}^w(F[f\,..\,f'), G[g\,..\,g')) = \min_{\mathcal{A}\in\mathbf{FA}(F[f\,..\,f'),G[g\,..\,g'))} \mathsf{ted}^w_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g')).$$

For a threshold $k \in \mathbb{R}_{\geq 0}$, we set

$$\mathsf{ted}^w_{\leq k}(F[f\,..\,f'), G[g\,..\,g')) = \begin{cases} \mathsf{ted}^w(F[f\,..\,f'), G[g\,..\,g')) & \text{if } \mathsf{ted}^w(F[f\,..\,f'), G[g\,..\,g')) \leq k, \\ \infty & \text{otherwise.} \end{cases}$$

By Remark 2.5, the following value is non-negative for every $\mathcal{A} \in \mathbf{FA}(F[f\,..\,f'), G[g\,..\,g'))$:

$$\widetilde{\mathsf{ted}}^w_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g')) \coloneqq \mathsf{ted}^w_{\mathcal{A}}(F[f\,..\,f'), G[g\,..\,g')) - \sum_{\substack{\hat{f}\in[f\,..\,f'):\\ \mathsf{mate}_F(\hat{f})\notin[f\,..\,f')}} \tilde{w}(F[\hat{f}], \varepsilon) - \sum_{\substack{\hat{g}\in[g\,..\,g'):\\ \mathsf{mate}_G(\hat{g})\notin[g\,..\,g')}} \tilde{w}(\varepsilon, G[\hat{g}]).$$

We naturally generalize this value to $\widetilde{\mathsf{ted}}^w(F[f\,..\,f'), G[g\,..\,g'))$.

▶ **Observation 2.6.** *For all forests $F, G \in \mathcal{F}_\Sigma$, fragments $F[f\,..\,f')$ and $G[g\,..\,g')$, and weight functions $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, we have $\widetilde{\mathsf{ted}}^w(F[f\,..\,f'), G[g\,..\,g')) = \mathsf{ted}^w(F_{[f\,..\,f')}, G_{[g\,..\,g')})$.*

3 $\mathcal{O}(nk^2 \log n)$-Time Algorithm

In this section, we reinterpret Klein's algorithm [24] and develop its $\mathcal{O}(nk^2 \log n)$-time variant.

3.1 Klein's Algorithm

Klein's algorithm [24] uses dynamic programming to compute $\mathsf{ted}(F_{[l_F..r_F)}, G_{[l_G..r_G)}) = \widetilde{\mathsf{ted}}(F[l_F..r_F), G[l_G..r_G))$ for $\mathcal{O}(n \log n)$ selected fragments $F[l_F..r_F)$ of F and all $\mathcal{O}(n^2)$ fragments $G[l_G..r_G)$ of G. We modify the algorithm slightly so that the computed value $\mathsf{ted}(F[l_F..r_F), G[l_G..r_G))$ includes the costs of deleting the single parentheses of nodes straddling $F[l_F..r_F)$ and inserting the single parentheses of nodes straddling $G[l_G..r_G)$.

Algorithm 1 `Klein`(l_F, r_F, l_G, r_G): Klein's algorithm for computing the tree edit distance.

Input: Two fragments $F[l_F..r_F)$ and $G[l_G..r_G)$ of the input forests
Output: Compute and store the value of $\mathsf{dp}[l_F, r_F, l_G, r_G]$

1 **if** $l_F = r_F$ **then** $\mathsf{result} \leftarrow \sum_{i \in [l_G..r_G)} \tilde{w}(\varepsilon, G[i])$;
2 **else if** $l_G = r_G$ **then** $\mathsf{result} \leftarrow \sum_{i \in [l_F..r_F)} \tilde{w}(F[i], \varepsilon)$;
3 **else**
4 $\quad u_F = \mathsf{node}_F(l_F)$, $v_F = \mathsf{node}_F(r_F - 1)$, $u_G = \mathsf{node}_G(l_G)$, $v_G = \mathsf{node}_G(r_G - 1)$;
5 $\quad$ **if** $u_F \notin V_{F[l_F..r_F)}$ **or** ($v_F \in V_{F[l_F..r_F)}$ **and** $\mathsf{size}(u_F) \le \mathsf{size}(v_F)$) **then**
6 $\quad\quad \mathsf{result} \leftarrow \mathsf{dp}[l_F + 1, r_F, l_G, r_G] + \tilde{w}(F[l_F], \varepsilon)$;
7 $\quad\quad \mathsf{result} \overset{\min}{\leftarrow} \mathsf{dp}[l_F, r_F, l_G + 1, r_G] + \tilde{w}(\varepsilon, G[l_G])$;
8 $\quad\quad$ **if** $u_F \in V_{F[l_F..r_F)}$ **and** $u_G \in V_{G[l_G..r_G)}$ **then**
9 $\quad\quad\quad \mathsf{result} \overset{\min}{\leftarrow} \tilde{w}(F[l_F], G[l_G]) + \mathsf{dp}[l_F + 1, c(u_F), l_G + 1, c(u_G)] + \tilde{w}(F[c(u_F)], G[c(u_G)]) + \mathsf{dp}[c(u_F) + 1, r_F, c(u_G) + 1, r_G]$;
10 $\quad$ **else**
11 $\quad\quad \mathsf{result} \leftarrow \mathsf{dp}[l_F, r_F - 1, l_G, r_G] + \tilde{w}(F[r_F - 1], \varepsilon)$;
12 $\quad\quad \mathsf{result} \overset{\min}{\leftarrow} \mathsf{dp}[l_F, r_F, l_G, r_G - 1] + \tilde{w}(\varepsilon, G[r_G - 1])$;
13 $\quad\quad$ **if** $v_F \in V_{F[l_F..r_F)}$ **and** $v_G \in V_{G[l_G..r_G)}$ **then**
14 $\quad\quad\quad \mathsf{result} \overset{\min}{\leftarrow} \mathsf{dp}[l_F, o(v_F), l_G, o(v_G)] + \tilde{w}(F[o(v_F)], G[o(v_G)]) + \mathsf{dp}[o(v_F) + 1, r_F - 1, o(v_G) + 1, r_G - 1] + \tilde{w}(F[r_F - 1], G[r_G - 1])$;
15 $\mathsf{dp}[l_F, r_F, l_G, r_G] \leftarrow \mathsf{result}$

The algorithm's implementation is provided in Algorithm 1. We use an operator $x \overset{\min}{\leftarrow} y$ that assigns $x \leftarrow y$ if $y < x$. We also remember which of these assignments were applied so that we can later trace back the optimal alignment based on this extra information stored.

In the corner case when $F[l_F..r_F)$ or $G[l_G..r_G)$ is empty, then the unique (and thus optimal) forest alignment pays for inserting or deleting all characters in the other fragment. If both $F[l_F..r_F)$ and $G[l_G..r_G)$ are non-empty, the algorithm considers nodes $u_F = \mathsf{node}_F(l_F)$, $u_G = \mathsf{node}_G(l_G)$, $v_F = \mathsf{node}_F(r_F - 1)$, and $v_G = \mathsf{node}_G(r_G - 1)$.

Let us first suppose that u_F and v_F are contained in $F[l_F..r_F)$. If the subtree of v_F is at least as large as the subtree of u_F, that is, $\mathsf{size}(v_F) \ge \mathsf{size}(u_F)$, where $\mathsf{size}(x) = c(x) - o(x) + 1$, then algorithm considers three possibilities: $F[l_F]$ is deleted, $G[l_G]$ is inserted, or $F[l_F]$ is aligned with $G[l_G]$. In the first two possibilities, we can pay for the deleted or inserted opening parenthesis and align the remaining fragments; see Lines 6–7. In the third possibility, the consistency condition in the definition of the forest alignments requires that $F[\mathsf{mate}_F(l_F)]$ is also aligned with $G[\mathsf{mate}_G(l_G)]$. In particular, the node u_G needs to be contained in $G[l_G..r_G)$ so that $\mathsf{mate}_G(l_G) = c(u_G)$ and $\mathsf{mate}_F(l_F) = c(u_F)$. Thus, we align $F(l_F..c(u_F))$ with $G(l_G..c(u_G))$, align $F(c(u_F)..r_F)$ with $G(c(u_G)..r_G)$, and pay for aligning u_F with u_G, that is $F[l_F]$ with $G[l_G]$ and $F[c(u_F)]$ with $G[c(u_G)]$; see Line 9. If $\mathsf{size}(v_F) < \mathsf{size}(u_F)$, the algorithm handles v_F and v_G in a symmetric way; see Lines 10–14.

If $u_F \notin V_{F[l_F..r_F)}$, we follow Lines 5–9 and effectively consider deleting $F[l_F]$ and inserting $G[l_G]$. Else, if $v_F \notin V_{F[l_F..r_F)}$, we follow the symmetric Lines 10–14.

Zhang and Shasha [49] use a very similar dynamic programming; in a baseline version, their algorithm always constructs the alignment from left to right instead of picking the side depending on u_F and v_F. Klein's optimization allows for an improved running time of $\mathcal{O}(n^3 \log n)$ instead of $\mathcal{O}(n^4)$. In the full version [27], we recall the proof of the following lemma.

▶ **Lemma 3.1.** *The recursive implementation of Algorithm 1 visits $\mathcal{O}(n \log n)$ fragments of F.*

Akmal and Jin's Algorithm. Akmal and Jin [1] reduce the time complexity of Klein's algorithm to $\mathcal{O}(nk^2 \log n)$ when computing $\mathsf{ted}_{\le k}(F, G)$. Their solutions prunes some dynamic programming states so that, in the surviving states, the differences between the sizes of the subforests $F_{[l_F..r_F)}$ and $G_{[l_G..r_G)}$ is at most k. This condition also holds for the difference between the sizes of $F_{[0..r_F)} \setminus F_{[l_F..r_F)}$ and $G_{[0..l_G)} \setminus G_{[l_G..r_G)}$, as well as between $F_{[l_F..|F|)} \setminus F_{[l_F..r_F)}$ and $G_{[l_G..|G|)} \setminus G_{[l_G..r_G)}$; see [1, Lemma 12]. As shown in [1, Lemma 13], for each subforest $F_{[l_F..r_F)}$, there are $\mathcal{O}(k^2)$ subforests $G_{[l_G..r_G)}$ satisfying these three conditions. With a careful implementation, the total number of states becomes $\mathcal{O}(nk^2 \log n)$.

3.2 Our Algorithm

Our variant of Klein's algorithm simply prunes all states with $|l_F - l_G| > 2k$ or $|r_F - r_G| > 2k$. In other words, we modify Algorithm 1 so that $\mathsf{dp}[l_F, r_F, l_G, r_G]$ is set to ∞ if either condition holds, and the existing instructions are executed otherwise. This does not increase the number $\mathcal{O}(n \log n)$ of visited fragments $F[l_F..r_F)$; see Lemma 3.1. For each of these fragments, trivially, at most $\mathcal{O}(k^2)$ fragments of G survive pruning, so the total number of states is $\mathcal{O}(nk^2 \log n)$. The correctness of the pruning rules follows from the fact that, for all forests F, G, the width of any alignment in $\mathbf{FA}(F, G)$ does not exceed twice its cost, where the width of an alignment $\mathcal{A} \in \mathbf{A}(X[x..x'), Y[y..y'))$ is defined as $\mathsf{width}(\mathcal{A}) = \max\{|\hat{x} - \hat{y}| : (\hat{x}, \hat{y}) \in \mathcal{A}\}$.

In the full version [27], we formalize this intuition using the notion of bounded forest alignments.

▶ **Definition 3.2.** *Let us fix a threshold $k \in \mathbb{Z}_+$ and consider fragments $F[f..f')$ and $G[g..g')$ of forests $F, G \in \mathcal{F}_\Sigma$. We call a forest alignment $\mathcal{A} \in \mathbf{FA}(F[f..f'), G[g..g'))$* bounded *if $\mathsf{width}(\mathcal{A}) \le 2k$. The family of* bounded forest alignments *is $\mathbf{BFA}_k(F[f..f'), G[g..g')) \subseteq \mathbf{FA}(F[f..f'), G[g..g'))$. For a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\ge 0}$, we denote*

$$\mathsf{bted}^w_k(F[f..f'), G[g..g')) = \min_{\mathcal{A} \in \mathbf{BFA}_k(F[f..f'), G[g..g'))} \mathsf{ted}^w_{\mathcal{A}}(F[f..f'), G[g..g')).$$

▶ **Lemma 3.3.** *The dp values computed using the pruned version of Algorithm 1 satisfy*

$$\mathsf{ted}^w(F[l_F..r_F), G[l_G..r_G)) \le \mathsf{dp}[l_F, r_F, l_G, r_G] \le \mathsf{bted}^w_k(F[l_F..r_F), G[l_G..r_G)).$$

In particular, the $\mathsf{dp}[0, |F|, 0, |G|]$ entry stores a value between $\mathsf{ted}^w(F, G)$ and $\mathsf{bted}^w_k(F, G)$. The following observation implies that this is enough to retrieve $\mathsf{ted}^w_{\le k}(F, G)$.

▶ **Observation 3.4.** *If $\mathsf{ted}^w(F, G) \le k$, then $\mathsf{ted}^w(F, G) = \mathsf{bted}^w_k(F, G)$.*

Proof. Consider the optimal forest alignment $\mathcal{A} \in \mathbf{FA}(F, G)$ with $\mathsf{ted}^w_{\mathcal{A}}(F, G) \le k$. For each edge $(f, g) \to (f', g')$, either $f - g = f' - g'$ (if the edge is diagonal) or $\big|(f-g)-(f'-g')\big| = 1$ and the edge cost is at least $\frac{1}{2}$ (if the edge is vertical or horizontal). Since $(0, 0) \in \mathcal{A}$ and the total cost of edges in $\mathcal{A}$ is at most k, every $(f, g) \in \mathcal{A}$ satisfies $|f - g| \le 2k$, i.e., $\mathsf{width}(\mathcal{A}) \le 2k$ and $\mathcal{A} \in \mathbf{BFA}_k(F, G)$. Hence, $\mathsf{bted}^w_k(F, G) \le \mathsf{ted}^w_{\mathcal{A}}(F, G) = \mathsf{ted}^w(F, G)$. The converse inequality $\mathsf{ted}^w(F, G) \le \mathsf{bted}^w_k(F, G)$ holds trivially due to $\mathbf{BFA}_k(F, G) \subseteq \mathbf{FA}(F, G)$. ◀

With minor implementation details needed to avoid logarithmic overheads for memoization using a sparse table dp (Akmal and Jin [1] ignore this issue), we achieve the following result.

▶ **Theorem 1.2.** *There exists a deterministic algorithm that, given two forests F and G, with n nodes in total, each with a label from an alphabet Σ, and oracle access to a normalized weight function $w : (\Sigma \cup \{\varepsilon\})^2 \to \mathbb{R}_{\geq 0}$, determines $k := \mathsf{ted}^w(F, G)$ in $\mathcal{O}(nk^2 \log n)$ time.*

4 Faster Algorithm for Repetitive Inputs

In this section, we present an optimized version of our $\mathcal{O}(nk^2 \log n)$-time algorithm capable of exploiting certain repetitive structures within the input forests F and G. The following notion of *free pairs* captures the structure that our algorithm is able to utilize.

▶ **Definition 4.1** (Free pair, free block)**.** *Consider forests $F, G \in \mathcal{F}_\Sigma$ and a fixed threshold $k \in \mathbb{Z}_+$. We call a pair of fragments $F[p_F \mathinner{..} q_F)$ and $G[p_G \mathinner{..} q_G)$ a* free pair *if*

- $|p_F - p_G| \leq 2k$, *and*
- *there exists a balanced string $R \in \mathcal{F}_\Sigma$ with $4k \leq |R| < 8k$ such that $F[p_F - |R| \mathinner{..} q_F + |R|) = R^{e+2} = G[p_G - |R| \mathinner{..} q_G + |R|)$ holds for some integer exponent $e \in \mathbb{Z}_+$.*

For that free pair, we call the fragment $F[p_F \mathinner{..} q_F)$ a free block *with period R and exponent e.*

Our improved algorithm assumes that the input forests F and G are augmented with a collection $\mathbf{F}$ of disjoint free blocks $F[p_F \mathinner{..} q_F)$, each associated with the underlying period R, exponent e, and the corresponding fragment $G[p_G \mathinner{..} q_G)$. The speed-up compared to the algorithm of Section 3 is noticeable if the free blocks in $\mathbf{F}$ jointly cover most of the characters of F, that is, the number of remaining *non-free* characters is asymptotically smaller than $|F|$.

▶ **Theorem 4.2.** *There exists a deterministic algorithm that, given forests $F, G \in \mathcal{F}_\Sigma$ of total length n, oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\geq 0}$, a threshold $k \in \mathbb{Z}_+$, and a collection $\mathbf{F}$ of t disjoint free blocks in F such that m characters of F are not contained in any free block, computes $\mathsf{ted}^w_{\leq k}(F, G)$ in $\mathcal{O}(n \log n + mk^2 \log n + tk^3 \log n)$ time.*

Intuitively, the algorithm skips all the free blocks, which allows reducing the $\mathcal{O}(nk^2 \log n)$ running time of Theorem 1.2 to $\mathcal{O}(mk^2 \log n)$, i.e., the algorithm needs to pay for non-free characters only. Nevertheless, processing each free block takes $\mathcal{O}(k^3 \log n)$ extra time, and $\mathcal{O}(n \log n)$-time preprocessing time is still needed to avoid overheads for memoization.

Processing Free Blocks. To understand our speed-up, let us consider a free pair with period R and exponent e, that is, $F[p_F \mathinner{..} q_F) = R^e = G[p_G \mathinner{..} q_G)$. We picked the parameters so that $|R| \geq 4k \geq \mathsf{width}(\mathcal{A}) + |p_F - p_G|$, and thus every alignment $\mathcal{A} \in \mathbf{BFA}_k(F, G)$ aligns $F[p_F \mathinner{..} q_F) = R^e$ with a fragment $G[p'_G \mathinner{..} q'_G)$ contained in $G[p_G - |R| \mathinner{..} q_G + |R|) = R^{e+2}$. By the same argument, the image of every copy of R within $F[p_F \mathinner{..} q_F) = R^e$ is contained within the corresponding copy of R^3 within $G[p_G - |R| \mathinner{..} q_G + |R|) = R^{e+2}$. Moreover, when we align $F[p_F \mathinner{..} q_F)$ to $G[p'_F \mathinner{..} q'_F) \subseteq G[p_G - |R| \mathinner{..} q_G + |R|)$, it suffices to partition $G[p'_F \mathinner{..} q'_F)$ into e fragments and *independently* optimally align each copy of R in $F[p_F \mathinner{..} q_F)$ with the corresponding fragment of $G[p'_F \mathinner{..} q'_F)$. This is because R is balanced, so every node of R^e is contained within a single copy of R, and thus the consistency condition in Definition 2.4 does not impose any constraints affecting multiple copies of R.

The optimal costs of aligning R with relevant fragments of R^3 can be encoded in the following matrix, constructible in $\mathcal{O}(|R|^3 \log |R|)$ time using Algorithm 1 (Klein's algorithm).

▶ **Definition 4.3.** *For a balanced string $R \in \mathcal{F}_\Sigma$, we define a matrix M_R of size $(2|R|+1) \times (2|R|+1)$ with indices i, j in the range $[-|R| \,..\, |R|]$ as follows:*

$$M_R[i,j] = \begin{cases} \mathsf{ted}^w(R, R^3[|R|+i \,..\, 2|R|+j)) & \text{if } |R|+i \le 2|R|+j, \\ \infty & \text{otherwise.} \end{cases}$$

In order to derive the optimal costs of aligning $R^e = F[p_F \,..\, q_F)$ with the relevant fragments of $R^{e+2} = G[p_G - |R| \,..\, q_G + |R|)$, we simply compute the e-th power of M_R with respect to the min-plus product. Formally, the min-plus product of matrices $A \in \mathbb{R}^{I \times J}$ and $B \in \mathbb{R}^{J \times K}$ is a matrix $C \in \mathbb{R}^{I \times K}$ such that $C[i,k] = \min_{j \in J} A[i,j] + B[j,k]$ for $(i,k) \in I \times K$.

For each free block $F[p_F \,..\, q_F) = R^e \in \mathbf{F}$, we construct the matrix M_R^e in $\mathcal{O}(k^3 \log n)$ time using Algorithm 1 followed by fast exponentiation, which reduces to computing $\mathcal{O}(\log e)$ min-plus products. We apply the matrix whenever we are tasked with filling a $\mathsf{dp}[l_F, r_F, l_G, r_G]$ entry such that $F[p_F \,..\, q_F)$ is a prefix or a suffix of $F[l_F \,..\, r_F)$. Formally, if $l_F \le p_F < q_F = r_F$, then we use the following formula instead of following Algorithm 1:

$$\mathsf{dp}[l_F, r_F, l_G, r_G] \leftarrow \min_{p'_G \in [p_F - |R| \,..\, p_F + |R|]} \mathsf{dp}[l_F, p_F, l_G, p'_G] + M_R^e[p'_G - p_G, r_G - q_G]. \tag{1}$$

In the symmetric case of $l_F = p_F < q_F \le r_F$, we apply

$$\mathsf{dp}[l_F, r_F, l_G, r_G] \leftarrow \min_{q'_G \in [q_F - |R| \,..\, q_F + |R|]} \mathsf{dp}[q_F, r_F, q'_G, r_G] + M_R^e[l_G - p_G, q'_G - q_G]. \tag{2}$$

In the full version [27], we formalize the intuition above to prove that Equations (1) and (2) preserve the invariant of Lemma 3.3, that is,

$$\mathsf{ted}^w(F[l_F \,..\, r_F), G[l_G \,..\, r_G)) \le \mathsf{dp}[l_F, r_F, l_G, r_G] \le \mathsf{bted}_k^w(F[l_F \,..\, r_F), G[l_G \,..\, r_G)).$$

For (1), this boils down to the following lemma; the case of (2) is symmetric.

▶ **Lemma 4.4.** *Consider fragments $F[l_F \,..\, r_F)$, $G[l_G \,..\, r_G)$ of forests $F, G \in \mathcal{F}_\Sigma$, a normalized weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\ge 0}$, and a threshold $k \in \mathbb{Z}_+$ such that $|r_F - r_G| \le 2k$. If there is a free pair $F[p_F \,..\, q_F) = G[p_G \,..\, q_G) = R^e$ such that $F[p_F \,..\, q_F)$ is a suffix of $F[l_F \,..\, r_F)$, then*

$$\begin{aligned} &\mathsf{bted}_k^w(F[l_F \,..\, r_F), G[l_G \,..\, r_G)) \\ &\quad \ge \min_{p'_G \in [p_F - |R| \,..\, p_F + |R|]} \mathsf{bted}_k^w(F[l_F \,..\, p_F), G[l_G \,..\, p'_G)) + M_R^e[p'_G - p_G, r_G - q_G] \\ &\quad \ge \min_{p'_G \in [p_F - |R| \,..\, p_F + |R|]} \mathsf{ted}^w(F[l_F \,..\, p_F), G[l_G \,..\, p'_G)) + M_R^e[p'_G - p_G, r_G - q_G] \\ &\quad \ge \mathsf{ted}^w(F[l_F \,..\, r_F), G[l_G \,..\, r_G)). \end{aligned}$$

We further argue in the full version [27] that the recursive implementation of Algorithm 1 augmented with the optimizations of (1) and (2) visits $\mathcal{O}(m \log n)$ fragments $F[l_F \,..\, r_F)$, where m is the number of non-free characters, including $\mathcal{O}(t \log n)$ fragments $F[l_F \,..\, r_F)$ for which (1) or (2) apply. Specifically, our optimized algorithm visits fragments $F[l_F \,..\, r_F)$ visited by the original Algorithm 1 and satisfying the following additional property: every free block $F[p_F \,..\, q_F) \in \mathbf{F}$ is either disjoint with $F[l_F \,..\, r_F)$ or contained in $F[l_F \,..\, r_F)$. Our implementation takes $\mathcal{O}(n \log n)$ extra preprocessing time to list fragments visited by Algorithm 1 and filter those satisfying the aforementioned property.

5 Universal Kernel with Improved Repetitiveness Guarantees

In this section, we outline our approach to strengthen Theorem 1.3 into the following result:

▶ **Theorem 5.1.** *There exists a linear-time algorithm that, given forests F, G and an integer $k \in \mathbb{Z}_+$, constructs forests F', G' of size $\mathcal{O}(k^5)$ such that $\mathsf{ted}^w_{\le k}(F', G') = \mathsf{ted}^w_{\le k}(F, G)$ holds for every normalized* quasimetric *w (weight function satisfying the triangle inequality), and a collection of $\mathcal{O}(k^3)$ disjoint free blocks in F' with $\mathcal{O}(k^4)$ non-free characters.*

Compared to Theorem 1.3, we require the presence of $\mathcal{O}(k^3)$ free blocks that jointly capture all but $\mathcal{O}(k^4)$ characters of the output forest F'; consult Definition 4.1. At the very high level, the proofs of both Theorems 1.3 and 5.1 consist in three steps: decomposing the input forests F and G into several pieces, identifying pieces that can be matched exactly, and replacing (pairs of) identical pieces with smaller equivalent counterparts.

Forest Decompositions and Piece Matchings. Following [14], we say that a *piece* of a forest F is a *subforest* – a balanced fragment $F[i\,..\,j)$ – or a *context* – a pair of fragments $\langle F[i\,..\,i'); F[j'\,..\,j)\rangle$ such that $F[i\,..\,j)$ is a tree and $F[i'\,..\,j')$ is balanced. We denote the set of pieces contained in a fragment $F[i\,..\,j)$ of F by $\mathcal{P}(F[i\,..\,j))$; we set $\mathcal{P}(F) = \mathcal{P}(F[0\,..\,|F|))$.

In isolation from F, a context can be interpreted as a pair of non-empty strings $C = \langle C_L; C_R\rangle \in \mathsf{P}^+_\Sigma \times \mathsf{P}^+_\Sigma$ such that $C_L \cdot C_R$ is a tree. The *composition* of contexts C, D results in a context $C \star D := \langle C_L \cdot D_L; D_R \cdot C_R\rangle$. Moreover, the composition of a context C and a forest H results in a tree $C \star H := C_L \cdot H \cdot C_R$. For any *decomposition* of a forest F into disjoint pieces, one can recover F using the concatenation and composition operations.

We define the *depth* of a context $C = \langle C_L; C_R\rangle$ to be the number nodes of $C_L \cdot C_R$ with the opening parenthesis in C_L and the closing parenthsis in C_R. Note that the depth of the context $C \star D$ is equal to the sum of the depths of C and D.

The following notion formalizes the concept of a matching between pieces of F and G.

▶ **Definition 5.2.** *For two forests F and G and a fixed threshold $k \in \mathbb{Z}_+$, a* piece matching *between F and G is a set of pairs $\mathcal{M} \subseteq \mathcal{P}(F) \times \mathcal{P}(G)$ such that:*

- *across all pairs $(f, g) \in \mathcal{M}$, the pieces $f \in \mathcal{P}(F)$ are pairwise disjoint, and*
- *there exists a forest alignment $\mathcal{A} \in \mathbf{FA}(F, G)$ of width at most $2k$ (i.e., $\mathcal{A} \in \mathbf{BFA}_k(F, G)$) that matches f to g perfectly for every $(f, g) \in \mathcal{M}$.*

The kernelization algorithm behind Theorem 1.3 repeatedly identifies a piece matching $\mathcal{M}$ of size $|\mathcal{M}| = \mathcal{O}(k)$ covering $\Omega(n)$ vertices of F and replaces each pair of matching pieces $(f, g) \in \mathcal{M}$ with a pair of "equivalent" pieces (f', g') of size $\mathcal{O}(k^4)$. After $\mathcal{O}(\log n)$ steps, this yields forests of size $\mathcal{O}(k^5)$. Our strategy relies on the following new result:

▶ **Theorem 5.3.** *There exists a linear-time algorithm that, given forests $F, G \in \mathcal{F}_\Sigma$ and a threshold $k \in \mathbb{Z}_{\ge 0}$, either certifies that $\mathsf{ted}(F, G) > k$ or constructs a size-$\mathcal{O}(k)$ piece matching $\mathcal{M}$ between F and G that leaves $\mathcal{O}(k^4)$ unmatched characters.*

The proof of Theorem 5.3, presented in the full version [27], reuses a subroutine of [14] to construct a decomposition $\mathcal{D} \subseteq \mathcal{P}(F)$ of F into $\mathcal{O}(n/k^3)$ pieces of size $\mathcal{O}(k^3)$ each. The next step is to build a piece matching $\mathcal{M} \subseteq \mathcal{D} \times \mathcal{P}(G)$ that leaves at most k pieces of $\mathcal{D}$ unmatched. We modify a dynamic-programming procedure from [14] so that, additionally, the unmatched characters of G form $\mathcal{O}(k)$ fragments. As a result, even though the obtained matching is of size $|\mathcal{M}| = \mathcal{O}(n/k^3)$, it is possible to reduce its size to $\mathcal{O}(k)$. For this, it suffices to repeatedly identify pairs of adjacent pieces $f, f' \in \mathcal{P}(F)$ matched to adjacent

pieces $g, g' \in \mathcal{P}(G)$, and then replace these pieces with their unions $f \cup f'$ and $g \cup g'$ (as formalized in the full version [27], two disjoint pieces are adjacent if their union, consisting of the characters contained in at least one of these pieces, can be interpreted as a piece).

Periodic Blocks and Red Characters. The main ingredient of our kernelization algorithm is a procedure that replaces a pair of matching pieces $(f, g) \in \mathcal{M}$ with a pair of "equivalent" smaller pieces (f', g'). Unlike [14], where the goal was to reduce the piece size to $\mathcal{O}(k^4)$, we aim to identify $\mathcal{O}(k^2)$ disjoint free blocks with $\mathcal{O}(k^3)$ non-free nodes within the replacement piece f'. Unfortunately, free blocks lack a canonical construction, and it would be tedious to maintain a specific selection while the forests change. Instead, we use the following notions:

▶ **Definition 5.4.** *For a fixed threshold $k \in \mathbb{Z}_+$, we say that a string $S \in \mathsf{P}^*_\Sigma$ forms a* periodic block *if it satisfies the following properties:*

- *$|S| \geq 42k$, that is, the fragment is of length at least $42k$, and*
- *S has a string period of length at most $4k$ with equally many opening and closing parentheses.*

For a string $T \in \mathsf{P}^*_\Sigma$, we denote by $\mathbf{B}_k(T)$ the set of fragments of T that are periodic blocks. For a context $C = \langle C_L; C_R \rangle$, we denote by $\mathbf{B}_k(C)$ the disjoint union of $\mathbf{B}_k(C_L)$ and $\mathbf{B}_k(C_R)$.

We partition the characters of $T \in \mathsf{P}^*_\Sigma$ into *black* and *red* based on the family $\mathbf{B}_k(T)$.

▶ **Definition 5.5.** *A character $T[i]$ of a string $T \in \mathsf{P}^*_\Sigma$ is* black *if there exists a periodic block $T[l\,..\,r) \in \mathbf{B}_k(T)$ such that $i \in [l + 5k\,..\,r - 5k)$. The remaining characters are* red*. We denote by $\mathsf{black}_k(T)$ and $\mathsf{red}_k(T)$ the sets of black and red characters of T.*

For a context $C = \langle C_L; C_R \rangle$, the sets $\mathsf{black}_k(C) = \mathsf{black}_k(C_L) \sqcup \mathsf{black}_k(C_R)$ and $\mathsf{red}_k(C) = \mathsf{red}_k(C_L) \sqcup \mathsf{red}_k(C_R)$ are defined as disjoint unions.

Piece Reduction. The following definitions in [14] formalize the concept of equivalent pieces.

▶ **Definition 5.6** ([14, Definition 3.4])**.** *For a threshold $k \in \mathbb{Z}_{\geq 0}$ and a weight function w, forests P, P' are called* $\mathsf{ted}^w_{\leq k}$-equivalent *if*

$$\mathsf{ted}^w_{\leq k}(F, G) = \mathsf{ted}^w_{\leq k}(F[0\,..\,l_F) \cdot P' \cdot F[r_F\,..\,|F|), G[0\,..\,l_G) \cdot P' \cdot G[r_G\,..\,|G|))$$

holds for all forests F and G with matching pieces $F[l_F\,..\,r_F) = P = G[l_G\,..\,r_G)$ satisfying $|l_F - r_G| \leq 2k$.

▶ **Definition 5.7** ([14, Definition 3.9])**.** *For a threshold $k \in \mathbb{Z}_{\geq 0}$ and a weight function w, contexts $P = \langle P_L; P_R \rangle$ and $P' = \langle P'_L; P'_R \rangle$ are called* $\mathsf{ted}^w_{\leq k}$-equivalent *if*

$$\mathsf{ted}^w_{\leq k}(F, G) = \mathsf{ted}^w_{\leq k}(F[0\,..\,l_F) \cdot P'_L \cdot F[l'_F\,..\,r'_F) \cdot P'_R \cdot F[r_F\,..\,|F|), \\ G[0\,..\,l_G) \cdot P'_L \cdot G[l'_G\,..\,r'_G) \cdot P'_R \cdot G[r_G\,..\,|G|))$$

holds for all forests F and G with matching pieces $\langle F[l_F\,..\,l'_F); F[r'_F\,..\,r_F)\rangle = P = \langle G[l_G\,..\,l'_G); G[r'_G\,..\,r_G)\rangle$ satisfying $|l_F - l_G| \leq 2k$ and $|r_F - r_G| \leq 2k$.

In the full version [27], we slightly modify the arguments of [14] to prove the following results:

▶ **Lemma 5.8** (see [14, Lemma 3.17])**.** *There is a linear-time algorithm that, given a forest P and a threshold $k \in \mathbb{Z}_+$, computes a forest P' with $|P'| \leq |P|$ and $|\mathsf{red}_k(P')| \leq 158k^2$ such that P and P' are $\mathsf{ted}^w_{\leq k}$-equivalent for every normalized quasimetric weight function w.*

▶ **Lemma 5.9** (see [14, Lemma 3.18])**.** *There is a linear-time algorithm that, given a context P and a threshold $k \in \mathbb{Z}_+$, computes a context P' with $|P'| \leq |P|$ and $|\mathsf{red}_k(P')| \leq 1152k^3$ such that P and P' are $\mathsf{ted}^w_{\leq k}$-equivalent for every normalized quasimetric weight function w.*

Complete Kernelization Algorithm. In the full version [27], we combine the above ingredients to formally prove Theorem 5.1. Our procedure first applies Theorem 5.3. Then, for every pair of matched pieces $(f, g) \in \mathcal{M}$, we use Lemma 5.8 or 5.9 (depending on piece type) to obtain an equivalent piece P' with $\mathcal{O}(k^3)$ red characters. The remaining (black) characters in P' can be traced back to $\mathcal{O}(k^2)$ periodic blocks and each periodic block within P' yields a free block in the output forest F' that covers the underlying black characters.

6 Summary

▶ **Theorem 1.1.** *There exists a deterministic algorithm that, given two forests F and G with n nodes in total, each with a label from an alphabet Σ, and oracle access to a normalized weight function $w : (\Sigma \cup \{\varepsilon\})^2 \to \mathbb{R}_{\geq 0}$ satisfying the triangle inequality, determines the tree edit distance $k := \mathsf{ted}^w(F, G)$ in $\mathcal{O}(n + k^6 \log k)$ time.*

Proof. First, suppose that the task is to compute $\mathsf{ted}^w_{\leq k}(F, G)$ for a given threshold k. In this case, we use the algorithm of Theorem 5.1, resulting in a pair of forests F', G' of size $\mathcal{O}(k^5)$ such that $\mathsf{ted}^w_{\leq k}(F', G') = \mathsf{ted}^w_{\leq k}(F, G)$, as well as a collection of $\mathcal{O}(k^3)$ free blocks in F' with $\mathcal{O}(k^4)$ non-free characters. Based on this, the algorithm of Theorem 4.2 computes $\mathsf{ted}^w_{\leq k}(F', G') = \mathsf{ted}^w_{\leq k}(F, G)$ in $\mathcal{O}(k^5 \cdot \log k^5 + k^4 \cdot k^2 \log k^5 + k^3 \cdot k^3 \log k^5) = \mathcal{O}(k^6 \log k)$ time. Including the $\mathcal{O}(n)$ running time of Theorem 5.1, we get $\mathcal{O}(n + k^6 \log k)$ total time.

In the absence of a given threshold, we consider a geometric sequence of thresholds $(d_i)_{i \in \mathbb{Z}_{\geq 0}}$, with $d_i = 2^i \cdot \lceil (n/\log n)^{1/6} \rceil$, and we compute $\mathsf{ted}^w_{\leq d_i}(F, G)$ for subsequent $i \in \mathbb{Z}_{\geq 0}$ until $\mathsf{ted}^w_{\leq d_j}(F, G) \leq d_j$ holds for some $j \in \mathbb{Z}_{\geq 0}$, which indicates $\mathsf{ted}^w(F, G) = \mathsf{ted}^w_{\leq d_j}(F, G)$.

Since $d_0 = \mathcal{O}((n/\log n)^{1/6})$, the initial iteration costs $\mathcal{O}(n + d_0^6 \log d_0) = \mathcal{O}(n)$ time. Consequently, if $k \leq d_0$, then the whole algorithm runs in $\mathcal{O}(n)$ time.

Due to $d_i \geq d_0 \geq (n/\log n)^{1/6}$, the running time of the ith iteration iteration is $\mathcal{O}(d_i^6 \log d_i)$. This sequence grows geometrically, so the total running time of the algorithm is dominated by the running time of the last iteration, which is $\mathcal{O}(d_j^6 \log d_j)$. If $k > d_0$, then $j > 0$ and, since the algorithm has not terminated one iteration earlier, $d_j = 2d_{j-1} < 2k$. Consequently, the overall running time is $\mathcal{O}(k^6 \log k)$ when $k > d_0$. ◀

References

1 Shyan Akmal and Ce Jin. Faster algorithms for bounded tree edit distance. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 12:1–12:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.12.

2 Tatsuya Akutsu. Tree edit distance problems: Algorithms and applications to bioinformatics. *IEICE Trans. Inf. Syst.*, 93-D(2):208–218, 2010. doi:10.1587/TRANSINF.E93.D.208.

3 Tatsuya Akutsu, Daiji Fukagawa, and Atsuhiro Takasu. Approximating tree edit distance through string edit distance. *Algorithmica*, 57(2):325–348, 2010. doi:10.1007/s00453-008-9213-z.

4 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

5 John Bellando and Ravi Kothari. Region-based modeling and tree edit distance as a basis for gesture recognition. In *1oth International Conference on Image Analysis and Processing (ICIAP 1999), 27-29 September 1999, Venice, Italy*, pages 698–703. IEEE Computer Society, 1999. doi:10.1109/ICIAP.1999.797676.

6 Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1–3):217–239, June 2005. doi:10.1016/j.tcs.2004.12.030.

7 Mahdi Boroujeni, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. $1+\epsilon$ approximation of tree edit distance in quadratic time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 709–720. ACM, 2019. doi:10.1145/3313276.3316388.

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

9 Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed xml. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 141–152. VLDB Endowment, 2003. doi:10.1016/b978-012722442-8/50021-5.

10 Alejandro Cassis, Tomasz Kociumaka, and Philip Wellnitz. Optimal algorithms for bounded weighted edit distance. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2177–2187. IEEE, 2023. doi:10.1109/FOCS57990.2023.00135.

11 Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. On the hardness of computing the edit distance of shallow trees. In Diego Arroyuelo and Barbara Poblete, editors, *String Processing and Information Retrieval - 29th International Symposium, SPIRE 2022, Concepción, Chile, November 8-10, 2022, Proceedings*, volume 13617 of *Lecture Notes in Computer Science*, pages 290–302. Springer, 2022. doi:10.1007/978-3-031-20643-6_21.

12 Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 90–101. Morgan Kaufmann, 1999. URL: http://www.vldb.org/conf/1999/P8.pdf.

13 Gregory Cobena, Serge Abiteboul, and Amélie Marian. Detecting changes in XML documents. In Rakesh Agrawal and Klaus R. Dittrich, editors, *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pages 41–52. IEEE Computer Society, 2002. doi:10.1109/ICDE.2002.994696.

14 Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, and Barna Saha. Weighted edit distance computation: Strings, trees, and dyck. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 377–390. ACM, 2023. doi:10.1145/3564246.3585178.

15 Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, Barna Saha, and Hamed Saleh. $\tilde{O}(n + poly(k))$-time algorithm for bounded tree edit distance. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*. IEEE, 2022. doi:10.1109/FOCS54457.2022.00071.

16 Davi de Castro Reis, Paulo Braz Golgher, Altigran Soares da Silva, and Alberto H. F. Laender. Automatic web news extraction using tree edit distance. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 502–511. ACM, 2004. doi:10.1145/988672.988740.

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

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

19 Dayi Fan, Rubao Lee, and Xiaodong Zhang. X-TED: massive parallelization of tree edit distance. *Proc. VLDB Endow.*, 17(7):1683–1696, 2024. doi:10.14778/3654621.3654634.

20 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1), November 2009. doi:10.1145/1613676.1613680.

21 Dvir Fried, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, Ely Porat, and Tatiana Starikovskaya. An improved algorithm for the k-dyck edit distance problem. *ACM Trans. Algorithms*, 20(3):26, 2024. doi:10.1145/3627539.

22 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, USA, 1997. doi:10.1017/cbo9780511574931.

23 Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in RNA secondary structures. In *2nd IEEE Computer Society Bioinformatics Conference, CSB 2003, Stanford, CA, USA, August 11-14, 2003*, pages 159–168. IEEE Computer Society, 2003. doi:10.1109/CSB.2003.1227315.

24 Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms*, ESA '98, pages 91–102, Berlin, Heidelberg, 1998. Springer-Verlag. doi:10.1007/3-540-68530-8_8.

25 Philip N. Klein, Thomas B. Sebastian, and Benjamin B. Kimia. Shape matching using edit-distance: an implementation. In S. Rao Kosaraju, editor, *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*, pages 781–790. ACM/SIAM, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365779.

26 Philip N. Klein, Srikanta Tirthapura, Daniel Sharvit, and Benjamin B. Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In David B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 696–704. ACM/SIAM, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338628.

27 Tomasz Kociumaka and Ali Shahali. Faster algorithm for bounded tree edit distance in the low-distance regime, 2025. doi:10.48550/arXiv.2507.02701.

28 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

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

30 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

31 Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446.

32 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014. doi:10.1145/2601073.

33 Jakob Nogler, Adam Polak, Barna Saha, Virginia Vassilevska Williams, Yinzhan Xu, and Christopher Ye. Faster weighted and unweighted tree edit distance and APSP equivalence. In *57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 2167–2178. ACM, 2025. doi:10.1145/3717823.3718116.

34 Mateusz Pawlik and Nikolaus Augsten. Efficient computation of the tree edit distance. *ACM Trans. Database Syst.*, 40(1):3:1–3:40, 2015. doi:10.1145/2699485.

35 Mateusz Pawlik and Nikolaus Augsten. Tree edit distance: Robust and memory-efficient. *Inf. Syst.*, 56:157–173, 2016. doi:10.1016/J.IS.2015.08.004.

36 Thomas B. Sebastian, Philip N. Klein, and Benjamin B. Kimia. Recognition of shapes by editing their shock graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):550–571, 2004. doi:10.1109/TPAMI.2004.1273924.

37 Masoud Seddighin and Saeed Seddighin. $3 + \epsilon$ approximation of tree edit distance in truly subquadratic time. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, volume 215, pages 115:1–115:22, 2022. doi:10.4230/LIPIcs.ITCS.2022.115.

38 Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977. doi:10.1016/0020-0190(77)90064-3.

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

40 Raghavendra Sridharamurthy, Talha Bin Masood, Adhitya Kamakshidasan, and Vijay Natarajan. Edit distance between merge trees. *IEEE Trans. Vis. Comput. Graph.*, 26(3):1518–1531, 2020. doi:10.1109/TVCG.2018.2873612.

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

42 Hélène Touzet. A linear tree edit distance algorithm for similar ordered trees. In *Proceedings of the 16th Annual Conference on Combinatorial Pattern Matching*, CPM'05, pages 334–345, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11496656_29.

43 Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1):100–118, 1985. International Conference on Foundations of Computation Theory. doi:10.1016/S0019-9958(85)80046-2.

44 Yuan Wang, David J. DeWitt, and Jin-yi Cai. X-diff: An effective change detection algorithm for XML documents. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 519–530. IEEE Computer Society, 2003. doi:10.1109/ICDE.2003.1260818.

45 Michael S. Waterman. *Introduction to computational biology - maps, sequences, and genomes: interdisciplinary statistics*. CRC Press, 1995.

46 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

47 Yoshiyuki Yamamoto, Kouichi Hirata, and Tetsuji Kuboyama. Tractable and intractable variations of unordered tree edit distance. *Int. J. Found. Comput. Sci.*, 25(3):307–330, 2014. doi:10.1142/S0129054114500154.

48 Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. Answer extraction as sequence tagging with tree edit distance. In Lucy Vanderwende, Hal Daumé III, and Katrin Kirchhoff, editors, *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 858–867. The Association for Computational Linguistics, 2013. URL: https://aclanthology.org/N13-1106/.

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