$\tilde{\mathcal{O}}(n + \text{poly}(k))$-time Algorithm for Bounded Tree Edit Distance

Debarati Das[1], Jacob Gilbert[2], MohammadTaghi Hajiaghayi[2], Tomasz Kociumaka[3], Barna Saha[4], and Hamed Saleh[2]

[1]Pennsylvania State University, United States
debaratix710@gmail.com
[2]University of Maryland, United States
jgilber8@umd.edu hajiaghayi@gmail.com hamed@cs.umd.edu
[3]Max Planck Institute for Informatics, Germany
tomasz.kociumaka@mpi-inf.mpg.de
[4]University of California, San Diego, United States
barnas@ucsd.edu

Abstract

Computing the *edit distance* of two strings is one of the most basic problems in computer science and combinatorial optimization. *Tree edit distance* is a natural generalization of edit distance in which the task is to compute a measure of dissimilarity between two (unweighted) rooted trees with node labels. Perhaps the most notable recent application of tree edit distance is in NoSQL big databases, such as MongoDB, where each row of the database is a JSON document represented as a labeled rooted tree and finding dissimilarity between two rows is a basic operation. Until recently, the fastest algorithm for tree edit distance ran in cubic time (Demaine, Mozes, Rossman, Weimann; TALG'10); however, Mao (FOCS'21) broke the cubic barrier for the tree edit distance problem using fast matrix multiplication.

Given a parameter k as an upper bound on the distance, an $\mathcal{O}(n + k^2)$-time algorithm for edit distance has been known since the 1980s due to works of Myers (Algorithmica'86) and Landau and Vishkin (JCSS'88). The existence of an $\tilde{\mathcal{O}}(n + \text{poly}(k))$-time algorithm for tree edit distance has been posed as open question, e.g., by Akmal and Jin (ICALP'21), who give a state-of-the-art $\tilde{\mathcal{O}}(nk^2)$-time algorithm. In this paper, we answer this question positively.

1 Introduction

Computing the *edit distance* of two strings is one of the most fundamental problems in theoretical computer science and combinatorial optimization studied since the 1960s. *Tree edit distance* is a natural generalization of edit distance in which the task is to compute a measure of dissimilarity between two (unweighted) rooted trees with node labels in which every insertion, deletion, or relabeling operation has unit cost. Tree edit distance, first introduced by Selkow [Sel77], extends the applications of edit distance to areas such as computational biology (e.g., analysis of RNA molecules, where the secondary structure of RNA is represented as a rooted tree) [Bil05, Gus97, HT84, SZ90, LMS98], structured data analysis (e.g., XML) [BGK03, Cha99, FLMM09], image analysis [BS98], and compiler optimization [DMRW10]. Perhaps the most notable recent application of tree edit distance is in NoSQL big databases, such as MongoDB [Mon], where each row of the database is a JSON document represented as a labeled rooted tree, and finding dissimilarity between two rows is a basic operation.

The computational aspect of tree edit distance is also widely studied. Tai [Tai79] gave the first solution for tree edit distance that runs in time $\mathcal{O}(n^6)$, where n is the total number of nodes in both trees. This running time was later improved in a series of works to $\mathcal{O}(n^4)$ [ZS89], $\mathcal{O}(n^3 \log n)$ [Kle98], and $\mathcal{O}(n^3)$ [DMRW10]. Very recently, Mao [Mao21] broke the cubic barrier by showing an $\mathcal{O}(n^{2.9546})$-time algorithm via a reduction to max-plus product of bounded-difference matrices. In a follow-up preprint, Dürr further improved the running time to $\mathcal{O}(n^{2.9149})$ [Dür22].

Boroujeni, Ghodsi, Hajiaghayi, and Seddighin [BGHS19] presented a $(1+\epsilon)$-approximation algorithm for tree edit distance that runs in $\tilde{\mathcal{O}}(n^2)$ time.[1] They also obtained an $\mathcal{O}(\sqrt{n})$-factor approximation algorithm that runs in $\tilde{\mathcal{O}}(n)$ time. Very recently, Seddighin and Seddighin [SS22] gave an $\mathcal{O}(n^{1.99})$-time $(3+\epsilon)$-approximation algorithm for tree edit distance.

The problem has also been considered from the lower-bound perspective. In particular, Bringmann, Gawrychowski, Mozes, and Weimann [BGMW20] proved that the cubic running time barrier for weighted tree edit distance cannot be broken unless APSP admits a truly subcubic time solution and weighted k-clique admits an $\mathcal{O}(n^{k-\epsilon})$-time solution. The existence of such a lower bound was previously conjectured by Abboud [Abb14] in a collection of open problems in fine-grained complexity.

In contrast to tree edit distance, approximation algorithms for string edit distance have been subject to many studies [AKO10, AO12, BYJKK04, BES06, HRS19, Ind01, LMS98, BEG+21, CDG+20, AN20]. After a series of recent developments [BEG+21, CDG+20, GRS20, KS20, BR20], the current best bound is an algorithm by Andoni and Nosatzki [AN20], for any constant $\epsilon > 0$, provides a constant-factor approximation of edit distance between strings of length n in time $\mathcal{O}(n^{1+\epsilon})$.

Given a parameter k as an upper bound on the distance, an $\mathcal{O}(n+k^2)$-time algorithm for edit distance is known since the 1980s, due to Myers [Mye86] and Landau and Vishkin [LV88], who combined suffix trees with an elegant greedy algorithm. The existence of such an $\tilde{\mathcal{O}}(n+\mathrm{poly}(k))$-time algorithm for tree edit distance (even for unlabeled trees) remained open despite persistent effort from researchers in the field. In particular, this question was posed by Mao [Mao21] and Akmal and Jin [AJ21]. The current fastest algorithm for the problem runs in $\tilde{\mathcal{O}}(nk^2)$ time [AJ21], improving on the previous results of $\mathcal{O}(nk^3)$ time by Touzet [Tou05]. In this paper, we answer this open question affirmatively by providing an $\tilde{\mathcal{O}}(n+k^{15})$-time algorithm for bounded tree edit distance.

The previous algorithms for computing tree edit distance [Tai79, ZS89, Kle98, DMRW10, Mao21] are dynamic-programming-based procedures, and the solutions for bounded tree edit distance [Tou05, AJ21] are obtained by appropriately pruning the set of states in earlier general-purpose algorithms. Our strategy is very different: the main effort is to greedily match all but $\mathcal{O}(\mathrm{poly}(k))$ nodes of the input trees so that any polynomial-time algorithm can be used to solve the residual instances of the problem. The greedy approach is sufficiently powerful only for trees avoiding certain synchronized periodicity, and therefore we start with a preprocessing step that eliminates appropriate periodic structures. Although such an approach is fairly simple to realize for strings, implementing it on trees requires several novel components of their own interest. This is because, so far, the underlying techniques have not been used on trees, and this setting brings many challenges absent in the context of strings. For example, periodicity in trees comes in two flavors, which we name *vertical* and *horizontal*, and we provide efficient procedures detecting both kinds. Another obstacle is that, whereas greedily matching two characters yields two independent instances of the string edit distance problem, greedily matching two nodes does not produce two independent instances of the tree edit distance problem, and thus we need a dedicated algorithm to optimally extend a partial alignment to a complete alignment of two trees. For more details, see

[1] The $\tilde{\mathcal{O}}(\cdot)$ notation suppresses polylogarithmic factors.

the technical overview in Section 2.

Finally, a problem related to (tree) edit distance is the *Dyck edit distance* problem, in which, given a sequence of n parentheses, the task is to find the minimum number of edits (character insertions, deletions, and substitutions) needed to make the sequence well-balanced. The Dyck edit distance has numerous applications [Har78, Koz97] and has been subject to many theoretical studies designing exact [BGSW19, CDX22, BO16, FGK+22a, Dür22] and approximation algorithms [Sah14, DKS22]. It is also known that this problem is at least as hard as Boolean matrix multiplication [ABW18]. Though Dyck edit distance problem has a different flavor than tree edit distance (since the goal is to find the minimum number of edits to completion, i.e., to a target of well-balanced parentheses), the existence of an $\tilde{\mathcal{O}}(n + \mathrm{poly}(k))$-time algorithm for the bounded version was still a very important open problem (motivated by fixing hierarchical data files, such as XML and JSON). Backurs and Onak [BO16] solved the open problem by providing an exact algorithm for Dyck edit distance that runs in $\mathcal{O}(n + k^{16})$ time, which has recently been improved by Fried, Golan, Kociumaka, Kopelowitz, Porat, and Starikovskaya [FGK+22a] to run in $\mathcal{O}(n+k^5)$ time, and further to $\tilde{\mathcal{O}}(n + k^{4.5442})$ using fast matrix multiplication [FGK+22b, Dür22]. Dyck edit distance falls under the umbrella of a general language edit distance problem [BGSW19], which, however, is at least as hard as Boolean matrix multiplication already for $k = 0$.

Our Result

While in edit distance, the goal is to transform a string S into another string S', in tree edit distance the goal is to transform a tree T into another tree T' using the least number of edit operations. In the most common version of the problem, it is assumed that both trees T and T' are rooted and that there is a left-to-right order between the children of any node. Moreover, each node has a label (independent of its the degree). The elementary operations are node deletion, node insertion, and node relabeling. In node deletion, we remove a node v and replace it with all of its children, preserving their order. The reverse of node deletion is node insertion, which allows us to select a consecutive set of siblings and bring them under a new node v which appears at the previous position of the relocated nodes. A node relabeling simply modifies the label of an existing node.

In fact, we solve a slightly more general problem of computing the edit distance between two labeled forests, which are defined as sequences of labeled, rooted, and ordered trees. For two labeled forests F and G, we denote their tree edit distance with $\mathsf{ted}(F, G)$. Moreover, for a threshold k, we denote with $\mathsf{ted}_{\le k}(F, G)$ a value equal to $\mathsf{ted}(F, G)$ (if it is at most k) or ∞ (otherwise). The main result of this paper is summarized in the following theorem:

Theorem 1.1. *There exists a randomized algorithm that, given forests F, G of total size n and an integer $k \in \mathbb{Z}_+$, computes $\mathsf{ted}_{\le k}(F, G)$ in $\mathcal{O}(n \log n + k^{15} \log k \log n)$ time correctly with high probability.*

2 Technical Overview

Given two strings $X, Y \in \Sigma^{\le n}$ and a threshold $k \in \mathbb{Z}_{\ge 0}$, the classic Landau–Vishkin algorithm [LV88] computes $\mathsf{ed}_{\le k}(X, Y)$ in time $\mathcal{O}(n + k^2)$. The algorithm uses dynamic programming: for each $i \in [0\,..\,k]$ and $j \in [-k\,..\,k]$, it computes an index $d_{i,j} = \max\{x : \mathsf{ed}(X[0\,..\,x), Y[0\,..\,x + j)) \le i\}$. In terms of the standard quadratic-size DP table, $d_{i,j}$ can be interpreted as the row of the farthest cell on the jth diagonal that can be reached with cost at most i. After a linear-time reprocessing of X, Y, each value $d_{i,j}$ can be computed in $\mathcal{O}(1)$ time; thus, the algorithm takes $\mathcal{O}(n+k^2)$ time in total. From the definition of $d_{i,j}$, we can observe that the alignments produced by the

Landau–Vishkin algorithm satisfy the following greedy property: if a prefix $X[0\,..\,x)$ is aligned to a prefix $Y[0\,..\,y)$ and the characters $X[x] = Y[y]$ match, then these two characters are also aligned (instead of being deleted). This greedy matching strategy is crucial in achieving $\mathcal{O}(n + \mathrm{poly}(k))$ time complexity as it allows the algorithm to focus on $\mathcal{O}(\mathrm{poly}(k))$ mismatches and quickly slide through the bulk of the input strings.

A natural question is whether a similar greedy strategy can be applied in the context of the edit distance of two labeled forests F and G. While there could be several ways to formalize such a greedy property, all definitions should capture the following scenario: if the leftmost roots of F and G have the same label, we should be able to greedily match these roots. Unfortunately, this is not the case, as illustrated in Fig. 1.

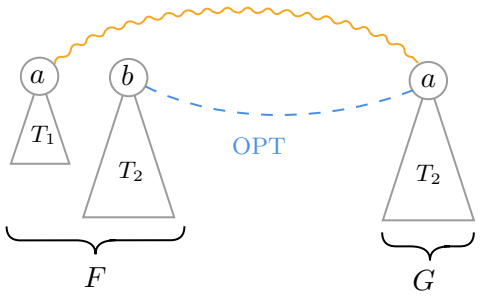


Figure 1: An example of simple greedy strategy where the alignment matches the leftmost roots, both having labels a. However, the unique optimal alignment deletes the entire tree T_1 and substitutes the label at the node of T_2 from b to a.

However, it is fairly obvious that if the entire leftmost trees of F and G are the same, then these trees can be matched greedily. In general, as proved in Proposition 4.4, if, while constructing an alignment, we are given a chance to match a node u in F with a node v in G (or to delete one or both of these nodes), then we can greedily match the two nodes provided that the entire subtrees rooted at u and v, respectively, are identical. We call alignments following this principle *greedy alignments*. Unfortunately, these subtrees can be large. Thus, first we need to design a process that, for any node $u \in V_F$ (or $v \in V_G$), can efficiently encode the information (e.g., structure, labeling) of the subtree rooted at u. We do it using *Look-Ahead Labeling* discussed next.

Look-Ahead Labelling: Given unlabeled forests F, G and a labeling $\lambda : V_{\mathsf{F}}, V_{\mathsf{G}} \to \Sigma$ (here, V_{F} stands for the node set of F) and a parameter d, the depth-d look-ahead labeling $\lambda' = L(\lambda, d)$ satisfies the following property for any pair of nodes $u, v \in V_F \cup V_G$: $\lambda'(u) = \lambda'(v)$ if and only if $\mathsf{ted}(\mathsf{sub}_{\le d}(u), \mathsf{sub}_{\le d}(v)) = 0$. Here, $\mathsf{sub}_{\le d}(u)$ denotes the λ-labeled subtree rooted at u trimmed to depth d.

To construct λ', we first define the parentheses representation of F and G with respect to labeling λ. For forest F with labeling λ, this is denoted as $\mathsf{P}_\lambda(\mathsf{F})$ and can be defined using the following recursion: If F consists of trees $\mathsf{T}_1, \ldots, \mathsf{T}_m$, then $\mathsf{P}_\lambda(\mathsf{F}) = \bigodot_{i=1}^{m} \mathsf{P}_\lambda(\mathsf{T}_i)$, where $\mathsf{P}_\lambda(\mathsf{T}_i) = (_{\lambda(v_i)} \cdot \mathsf{P}_\lambda(\mathsf{F}_i) \cdot)_{\lambda(v_i)}$ and $\cdot$ as well as $\bigodot$ denote concatenation. Here, v_i is the root of T_i and F_i is the forest obtained from T_i by removal of v_i. To create λ' in linear time, we replace the label of each node $u \in V_{\mathsf{F}} \cup V_{\mathsf{G}}$ with a Karp–Rabin fingerprint of $P_\lambda(\mathsf{sub}_{\le d}(v))$. Note by setting d larger than the heights of F and G, we can create a labeling that for each node encodes the entire subtree rooted at that node. The details can be found in Section 4.1.

As an additional challenge, it turns out that this greedy approach is beneficial only if the input trees F, G avoid certain periodic structures, which may be present in the input forests. Thus, we first

design an $\tilde{\mathcal{O}}(n)$-time algorithm that, given arbitrary labelled forests F, G, produces a pair of forests F', G' such that $\mathsf{ted}_{\leq k}(F, G) = \mathsf{ted}_{\leq k}(F', G')$ and both F', G' avoid synchronized *horizontal* and *vertical* k-periodicity, the two types of periodicity which we describe in the following paragraphs. Moreover, with an $\tilde{\mathcal{O}}(n+k^3)$-time post-processing, we also compute an alignment $\tilde{\mathcal{A}}$ mapping $\mathsf{P}(F')$ to $\mathsf{P}(G')$ such that if we consider any optimal alignment $\mathcal{B}$ between F' and G' (and identify it with the corresponding alignment of $\mathsf{P}(F')$ and $\mathsf{P}(G')$), then $\tilde{\mathcal{A}}$ and $\mathcal{B}$ differ at $\mathcal{O}(k^4)$ positions. Next, we discuss the key ideas of the periodicity reduction process. We remark that this is the first algorithm that provides an efficient way of reducing periodicity in trees or forests, and it can be of independent interest.

Periodicity Reduction: One of the most important novel contributions of our algorithm is the definition and identification of periodicity in forests. An integer p is *period* of a string S if $S[i] = S[i+p]$ holds for all $i \in [0\,..\,n-p)$; we then call $\frac{|S|}{p}$ the *exponent* of this period of S. If $p \leq k$, then S is called k-periodic. Ideally, we would like to identify and reduce the exponent of periodic regions of forests F and G somehow. Given two strings S and T, if there exist periodic fragments $Q = S[\alpha_s\,..\,\beta_s) = T[\alpha_t\,..\,\beta_t)$ such that $|\alpha_s - \alpha_t| \leq 2k$, we say that there are k-synchronized occurrences of Q in S and T. Synchronized periodicity in strings has previously been studied for string edit distance algorithms that take advantage of such periodicity when constructing optimal alignments (e.g., in [KPS21]), but never in forests. Unfortunately, reducing periodic fragments in the parentheses representations of forests does not preserve tree edit distance. In particular, if a periodic substring is unbalanced, then any edit to an opening parenthesis in the periodic region also needs to be applied to the corresponding closing parenthesis, which may lie outside the periodic region. We propose and identify two new types of "balanced" periodicity in forests, *horizontal* and *vertical* periodicity. By constructing forests F' and G' which avoid k-synchronized horizontal and vertical k-periodicity in F and G without affecting the total edit distance between the forests, we prove that we also avoid any periodicity in the underlying parentheses representations of F and G (with an appropriate labeling).

In Section 6, the first type of periodicity we identify and reduce is *horizontal periodicity*. If a sequence of subtrees of a given node repeats periodically, then we consider the repeated forest as a horizontal period. Therefore, horizontal periodicity appears in the parentheses representation of F and G as a balanced periodic fragment. Since horizontal periods are already balanced, any edits can be applied locally within the horizontally periodic fragment. By computing all maximal periodic fragments in the parentheses representations of F and G, we can find and reduce all horizontally periodic regions at nearby locations in both forests. In order to avoid interactions between overlapping horizontally periodic regions, we are careful to only reduce horizontal periodicity with large exponent and small period. Algorithm 3 gives detailed pseudocode for the $\tilde{\mathcal{O}}(n)$-time construction of forests F' and G' which avoid k-synchronized horizontal k-periodicity from input forests F and G.

In Section 7, the second type of periodicity we identify and reduce is *vertical periodicity*. Vertical periodicity occurs in a forest when there is some path in which the subtrees to the left and right of the path repeat periodically at subsequent levels of the path (see Fig. 4). To find all vertical periods in a forest F, we identify nodes whose corresponding opening and closing parenthesis are each contained in periodic substrings of the parentheses representation $\mathsf{P}_\lambda(\mathsf{F})$. Once we find all vertical periods of F and G, we use orthogonal range queries (in two dimensions) to find nodes $u \in V_F$ and $v \in V_G$ which are both contained in a vertically periodic region and whose opening parentheses and closing parentheses are in similar locations of $\mathsf{P}_\lambda(\mathsf{F})$ and $\mathsf{P}_\lambda(\mathsf{G})$. Once we find all such pairs of nodes, we can reduce the exponent of the periodic path as in the case of horizontal

periods. Algorithm 7 performs these vertical periodic reductions in $\tilde{\mathcal{O}}(n)$-time without introducing any new horizontal k-periodicity, and it outputs forests F' and G' which avoid both k-synchronized horizontal and vertical k-periodicity.

In Section 8, we prove that since F' and G' avoid k-synchronized horizontal and vertical k-periodicity of large exponent, then $\mathsf{P}_\lambda(\mathsf{F}')$ and $\mathsf{P}_\lambda(\mathsf{G}')$ avoids all k-synchronized $2k$-periodicity of large exponent, where λ is an appropriate refinement of the depth-$8k$ look-ahead labeling. We show that any periodic substring of a forest's parentheses representation must be either horizontal or vertical period depending on whether the period string is balanced or unbalanced. At the end of the section, using an $\mathcal{O}(n+k^3)$-time DP algorithm described in Section 3, we compute an alignment $\tilde{\mathcal{A}}$ of $\mathsf{P}_\lambda(F')$ and $\mathsf{P}_\lambda(G')$ that differs from any optimal tree alignment of F' and G' on at most k^4 characters.

Given the two trees F' and G' that avoid synchronized horizontal k-periodicity and synchronized vertical k-periodicity, in Section 9.5 we design an algorithm using the greedy strategy that computes $\mathsf{ted}_{\le k}(F', G')$ in time $\tilde{\mathcal{O}}(n + h^2k^7)$, where h is an upper bound on the height of F' and G'. We provide a brief sketch of the algorithm in the following.

Tree Edit Distance of Shallow Forests: Here, we first show that if we consider any optimal alignment $\mathcal{A}$ between F' and G', then, for all but $2hk$ pairs of nodes (u, v) where $u \in V_{F'}$ and $v \in V_{G'}$, $\mathcal{A}$ matches u with v and the subtree rooted at u is identical as the one rooted at v. If we assume that F' and G' avoid synchronized horizontal k-periodicity, this allows constructing a large set of matching pairs $M \subseteq \{(u, v) : u \in V_F, v \in V_G\}$ (of size $|V_F| - \mathcal{O}(h^2k^4)$) that is common to all optimal greedy alignments; i.e., each pair $(u, v) \in M$ is matched by every greedy alignment and the subtrees rooted at u and v are at distance 0. The time required to construct M is $\mathcal{O}(n + hk^2)$.

Given this partial matching M common to every optimal greedy alignment, our next objective is to extend it in order to construct an optimal alignment between F' and G'. An analogous task is relatively straightforward for strings: given a partial matching, we can first partition the strings along this matching and then independently compute the optimal distance between subsequent pieces. This strategy fails for trees, but we still provide a linear-time algorithm that, given a pair of forests F' and G' and a non-crossing partial matching M between them, constructs forests F'' and G'' such that $\mathsf{ted}^M_{\le k}(F', G') = \mathsf{ted}_{\le k}(F'', G'')$ and $|F''| + |G''| = \mathcal{O}(k(|F'| + |G'| - 2|M|))$. Here, $\mathsf{ted}^M_{\le k}(F', G')$ is the minimum of the cost of all alignments $\mathcal{A}$ between F', G' such that each pair of nodes in M is also matched by $\mathcal{A}$ (or ∞ if that cost exceeds k).

Thus, following this construction and using the fact that the partial matching M is common to every optimal greedy alignment between F', G' (thus $\mathsf{ted}^M_{\le k}(F', G') = \mathsf{ted}_{\le k}(F', G')$) and $|M| = |V_F| - \mathcal{O}(h^2k^4)$, in linear time we can construct trees F'', G'' such that $\mathsf{ted}_{\le k}(F', G') = \mathsf{ted}_{\le k}(F'', G'')$ and $|F''| + |G''| = \mathcal{O}(h^2k^5)$. Next using the $\tilde{\mathcal{O}}(nk^2)$ time algorithm of [AJ21], we compute $\mathsf{ted}_{\le k}(F'', G'')$ in time $\tilde{\mathcal{O}}(h^2k^7)$. Below, we discuss how to extend a partial matching to a complete alignment.

Partial Forest Matching: As mentioned earlier, here, given a pair of forests F' and G' and a non-crossing partial matching M, our objective is to construct forests F'' and G'' such that $\mathsf{ted}^M_{\le k}(F', G') = \mathsf{ted}_{\le k}(F'', G'')$ and $|F''| + |G''| = \mathcal{O}(k(|F'| + |G'| - 2|M|))$. Also, if the height of F'' (or G'') is $h > 2$, then there is a length $h - 2$ top down-path in F' avoiding nodes from M. Thus, if M is large and hits all long paths, then this significantly reduces both the size and the depth of the input forests while preserving their distance.

We start by partitioning F' and G' along the matching pairs in M, creating forests $\hat{F}$ and $\hat{G}$, and a set of partial matching pairs $\hat{M}$ between $\hat{F}, \hat{G}$. Our aim is to ensure that each node appearing

in a matching pair of $\hat{M}$ is a leaf node of some tree in $\hat{F}$ (or $\hat{G}$) and $\mathsf{ted}^{M}(F', G') = \mathsf{ted}^{\hat{M}}(\hat{F}, \hat{G})$. For this, we mark all the nodes that are present in M and assign each node to the nearest marked proper ancestor. This creates a partition of V_F and of V_G. Next for each subset S of nodes in the partition, we create a tree by deleting all the nodes in $V_F \setminus S$ from F, and we add the tree to $\hat{F}$ (while preserving the order of the nodes as present in F). Similarly create $\hat{G}$ from G'. Note that, by construction, each node present in M appears as a leaf node of some tree in $\hat{F}$ (or in $\hat{G}$). We also copy all the matching pairs from M to $\hat{M}$. Lastly to ensure $\mathsf{ted}^{M}(F', G') = \mathsf{ted}^{\hat{M}}(\hat{F}, \hat{G})$, between each pair of consecutive trees in $\hat{F}$ and their corresponding trees in $\hat{G}$, we insert a pair of single-node trees whose roots are also added to $\hat{M}$. Now, each node present in $\hat{M}$ appears as a leaf node of some tree in $\hat{F}$ (or in $\hat{G}$) and, if height of $\hat{F}$ (or $\hat{G}$) is $h > 1$, then there is a length $h-1$ top down path in F' (or G') avoiding nodes from M.

In the second step we further reduce $\hat{M}$ by removing redundant matching pairs. The idea here is that, for each matching pair $(\hat{u}, \hat{v}) \in \hat{M}$, if their immediate left siblings u and v (respectively) are also matched by $\hat{M}$, i.e., $(u, v) \in \hat{M}$, then we can discard $(\hat{u}, \hat{v})$ to create a new set $\bar{M}$ and create forests $\bar{F}$ and $\bar{G}$ by deleting all the nodes present in some discarded matching pair. This is because (u, v) serves as a representative of $(\hat{u}, \hat{v})$ and thus reintroducing them would not violate the non-crossing property of the matching set, and we can ensure $\mathsf{ted}^{\bar{M}}(\bar{F}, \bar{G}) = \mathsf{ted}^{\hat{M}}(\hat{F}, \hat{G}) = \mathsf{ted}^{M}(F', G')$. We also show $|\hat{M}| \leq 2/5(|\hat{F}| + |\hat{G}| + 1)$.

Lastly, using $\bar{F}$, $\bar{G}$, and $\bar{M}$, we construct forests F'', G'' such that $\mathsf{ted}^{\bar{M}}_{\leq k}(\bar{F}, \bar{G}) = \mathsf{ted}_{\leq k}(F'', G'')$. To construct F'' and G'', we attach a gadget containing $k+1$ uniquely-labeled children to each node present in $\bar{M}$. As the cost of an optimal alignment of F'' and G'' is bounded by k, it should match at least one node from each gadget; following this, we can show the alignment will indeed match all the nodes from the gadget and from set $\bar{M}$ thus proving $\mathsf{ted}^{\bar{M}}_{\leq k}(\bar{F}, \bar{G}) = \mathsf{ted}_{\leq k}(F'', G'')$. Moreover, from the above construction and using the bound of $|\hat{M}|$, we can show $|F''| + |G''| = \mathcal{O}(\min(|F'| + |G'| + k|M|, k(|F'| + |G'| - 2|M|)))$. Also, by the gadget construction, we ensure the height of F'' (or G'') is at most the height of $\bar{F}$ (or $\bar{G}$) plus one. Thus, if the height of F'' (or G'') is $h > 2$, then there is a length $h-2$ top down path in F' (or G', respectively) avoiding M. We provide the details in Section 9.1.

To summarize, given arbitrary labelled forests F, G and a threshold k, we first design an algorithm that in time $\tilde{\mathcal{O}}(n)$ produces a pair of forests F', G' such that $\mathsf{ted}_{\leq k}(F, G) = \mathsf{ted}_{\leq k}(F', G')$ and F', G' avoid synchronized horizontal and vertical k-periodicity. At $\tilde{\mathcal{O}}(n + k^3)$ time, this algorithm also computes an alignment $\tilde{\mathcal{A}}$ between $\mathsf{P}(F')$ and $\mathsf{P}(G')$ such that, any optimal alignment $\mathcal{B}$ between F' and G', differs from $\tilde{\mathcal{A}}$ at $\mathcal{O}(k^4)$ positions. Next, using the greedy strategy and the partial matching technique, we design an algorithm that can compute $\mathsf{ted}_{\leq k}(F', G')$ in time $\tilde{\mathcal{O}}(n + h^2k^7)$, given that the heights of F' and G' is at most h. However, the challenge is that the height h can be much larger than k. To solve this issue, instead of directly computing the distance between F', G', we design Section 9.3, which first creates a partial matching $\tilde{M}$ using random sampling within alignment $\tilde{\mathcal{A}}$, and then extends this partial matching to construct an optimal alignment between F' and G'. The random sampling constructs $\tilde{M}$ in such a way so that, given F', G', and $\tilde{M}$, the partial matching reduction generates forests F'', G'' where $\mathsf{ted}^{\tilde{M}}_{\leq k}(F', G') = \mathsf{ted}_{\leq k}(F'', G'')$ and the heights of F'' and G'' are bounded by $O(k^4)$. Thus, one can compute the distance in time $\tilde{\mathcal{O}}(n + k^{15})$. We now give a sketch of the sampling technique.

Level Sampling Set the height threshold $h = O(k^4)$. Select the sampling parameter r from $[0\,..\,h)$ uniformly at random. Mark all the nodes in F' and G' at depth congruent to r modulo h. Next, we create a partial matching $\tilde{M}$ as follows: for each marked node $u \in V_{F'}$ (or in $V_{G'}$), add

(u, v) to $\tilde{M}$, where $v \in V_{G'}$ (or $v \in V_{F'}$) and $\tilde{\mathcal{A}}$ matches the parentheses representing u with the parentheses representing v. Apply partial matching reduction on F', G' and $\tilde{M}$ to create forests F'', G'' given (i) every marked node is present in $\tilde{M}$ (ii) $|\tilde{M}| = O(n/k^4)$. Using the bound on $|\tilde{M}|$, we claim $|F''|+|G''| = O(n)$. Moreover, neither of F'', G'' has a top-down length-h path avoiding $\tilde{M}$. Hence, the height of F'', G'' is at most $h + 1 = O(k^4)$. Thus, we can compute $\mathsf{ted}_{\le k}(F'', G'')$ using the shallow forest algorithm in time $\tilde{\mathcal{O}}(n + \mathrm{poly}(k))$. Note that the partial matching reduction ensures $\mathsf{ted}^{\tilde{M}}_{\le k}(F', G') = \mathsf{ted}_{\le k}(F'', G'')$. As $\mathsf{ted}(F', G') \le k$ and any optimal alignment between F' and G' and $\tilde{\mathcal{A}}$ differs in at most $O(k^4)$ matching pairs, following our sampling strategy we further can argue that $\mathsf{ted}_{\le k}(F', G') = \mathsf{ted}^{\tilde{M}}_{\le k}(F', G')$ holds with constant probability. To ensure concentration, we repeat this $\Theta(\log n)$ times and report the minimum distance computed.

3 Preliminaries

3.1 Strings

A *string* $Y \in \Sigma^n$ is a sequence of $|Y| := n$ characters from an *alphabet* Σ. For $i \in [0\,..\,n)$, we denote the ith character of Y with $Y[i]$. The *reverse* of a string Y is $\overline{Y} := Y[n-1]Y[n-2]\cdots Y[0]$. We say that a string X *occurs* as a *substring* of a string Y if $X = Y[i]\cdots Y[j-1]$ holds for some indices $0 \le i \le j \le |Y|$. We denote the underlying *occurrence* of X as $Y[i\,..\,j)$. Formally, $Y[i\,..\,j)$ is a *fragment* of Y that can be represented using a reference to Y as well as its endpoints i, j. The fragment $Y[i\,..\,j)$ can be alternatively denoted as $Y[i\,..\,j-1]$, $Y(i-1\,..\,j-1]$, or $Y(i-1\,..\,j)$. A fragment of the form $Y[0\,..\,j)$ is a *prefix* of Y, whereas a fragment of the form $Y[i\,..\,n)$ is a *suffix* of Y.

An integer $p \in [1\,..\,n]$ is a *period* of a string $Y \in \Sigma^n$ if $Y[i] = Y[i+p]$ holds for all $i \in [0\,..\,n-p)$. In this case, the prefix $Y[0\,..\,p)$ is called a *string period* of Y. By $\mathsf{per}(Y)$ we denote the smallest period of Y. The exponent of a string Y is defined as $\exp(Y) := \frac{|Y|}{\mathsf{per}(Y)}$, and we say that a string Y is *periodic* if $\exp(Y) \ge 2$.

For a string Y and an integer $m \ge 0$, we define the mth power of Y, denoted Y^m, as the concatenation of m copies of Y. For a string $Y \in \Sigma^n$, we define a *forward rotation* $\mathsf{rot}(Y) = Y[1]\cdots Y[n-1]Y[0]$. In general, a *cyclic rotation* $\mathsf{rot}^s(Y)$ with *shift* $s \in \mathbb{Z}$ is obtained by iterating rot or the inverse operation rot^{-1}. A non-empty string $Y \in \Sigma^n$ is *primitive* if it is distinct from its non-trivial rotations, i.e., if $Y = \mathsf{rot}^s(Y)$ holds only when s is a multiple of n. A string Y is primitive if and only if it cannot be expressed as $Y = X^m$ for some string X and integer $m > 1$.

3.2 Edit-distance Alignments

The *edit distance* (also known as the *Levenshtein distance*) $\mathsf{ed}(X, Y)$ between two strings X and Y is defined as the smallest number of character insertions, deletions, and substitutions required to transform X to Y.

Equivalently, the edit distance $\mathsf{ed}(X, Y)$ can be defined as the cost of the cheapest *alignment* $\mathcal{A} : X \to Y$. There are many ways to formally define an alignment; all of them, however, need to identify which characters of X are *deleted* and which characters of Y are *inserted*. The remaining characters are *aligned* (either *substituted* or *matched*) in the left-to-right order. Since parts of this work rely on the techniques of [KPS21], we chose to stick to their formalization.

Definition 3.1. A sequence $\mathcal{A} = (x_t, y_t)_{t=0}^{m}$ is an *alignment* of a string $X \in \Sigma^*$ onto a string $Y \in \Sigma^*$, denoted $\mathcal{A} : X \to Y$, if $(x_0, y_0) = (0, 0)$, $(x_m, y_m) = (|X|, |Y|)$, and $(x_{t+1}, y_{t+1}) \in \{(x_t + 1, y_t + 1), (x_t + 1, y_t), (x_t, y_t + 1)\}$ for $t \in [0\,..\,m)$.

Given an alignment $\mathcal{A} = (x_t, y_t)_{t=0}^{m} : X \to Y$, for every $t \in [0\,..\,m)$:

- If $(x_{t+1}, y_{t+1}) = (x_t + 1, y_t)$, we say that $\mathcal{A}$ *deletes* $X[x_t]$.
- If $(x_{t+1}, y_{t+1}) = (x_t, y_t + 1)$, we say that $\mathcal{A}$ *inserts* $Y[y_t]$.
- If $(x_{t+1}, y_{t+1}) = (x_t+1, y_t+1)$, we say that $\mathcal{A}$ *aligns* $X[x_t]$ and $Y[y_t]$, denoted $X[x_t] \sim_{\mathcal{A}} Y[y_t]$. If additionally $X[x_t] = Y[y_t]$, we say that $\mathcal{A}$ *matches* $X[x_t]$ and $Y[y_t]$, denoted $X[x_t] \simeq_{\mathcal{A}} Y[y_t]$. Otherwise, we say that $\mathcal{A}$ *substitutes* $X[x_t]$ for $Y[y_t]$.

The *cost* of an edit distance alignment $\mathcal{A}$ is the total number characters that $\mathcal{A}$ deletes, inserts, or substitutes. We denote the cost by $\mathsf{ed}_{\mathcal{A}}(X, Y)$. The cost of an alignment $\mathcal{A} = (x_t, y_t)_{t=0}^{m}$ is at least its *width*, defined as $\max_{t=0}^{m} |x_t - y_t|$. Observe that $\mathsf{ed}(X, Y)$ can be defined as the minimum cost of an alignment of X and Y, that is, $\mathsf{ed}(X, Y) = \min_{\mathcal{A}:X\to Y} \mathsf{ed}_{\mathcal{A}}(X, Y)$. An alignment of X and Y is *optimal* if its cost is equal to $\mathsf{ed}(X, Y)$.

Given an alignment $\mathcal{A} = (x_t, y_t)_{t=0}^{m}$ of $X, Y \in \Sigma^+$, we partition the elements (x_t, y_t) of $\mathcal{A}$ into *matches* (for which $X[x_t] \simeq_{\mathcal{A}} Y[y_t]$) and *breakpoints* (the remaining elements). We denote the set of matches and breakpoints by $\mathsf{M}_{\mathcal{A}}(X, Y)$ and $\mathsf{B}_{\mathcal{A}}(X, Y)$, respectively. Observe that $|\mathsf{B}_{\mathcal{A}}(X, Y)| = 1 + \mathsf{ed}_{\mathcal{A}}(X, Y)$.

We say that a set $M \subseteq \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$ is *non-crossing* if there are no distinct pairs $(x, y), (x', y') \in M$ with $x \leq x'$ and $y \geq y'$. Observe that, for every alignment $\mathcal{A}$ of X, Y, the set $\{(x, y) \in [0\,..\,|X|) \times [0\,..\,|Y|) : X[x] \sim_{\mathcal{A}} Y[y]\}$ is non-crossing. We further say that a non-crossing set $M \subseteq [0\,..\,|X|) \times [0\,..\,|Y|)$ is a *non-crossing matching* of strings X, Y if $X[x] = Y[y]$ holds for every $(x, y) \in M$. Note that, for every alignment $\mathcal{A}$ of X, Y, the set $\mathsf{M}_{\mathcal{A}}(X, Y)$ is a non-crossing matching.

Given an alignment $\mathcal{A} = (x_t, y_t)_{t=0}^{m}$ of X and Y, for every $\ell, r \in [0\,..\,m]$ with $\ell \leq r$, we say that $\mathcal{A}$ *aligns* $X[x_\ell\,..\,x_r)$ to $Y[y_\ell\,..\,y_r)$, denoted $X[x_\ell\,..\,x_r) \sim_{\mathcal{A}} Y[y_\ell\,..\,y_r)$. If there is no breakpoint (x_t, y_t) with $t \in [\ell\,..\,r)$, we further say that $\mathcal{A}$ *matches* $X[x_\ell\,..\,x_r)$ to $Y[y_\ell\,..\,y_r)$, denoted $X[x_\ell\,..\,x_r) \simeq_{\mathcal{A}} Y[y_\ell\,..\,y_r)$.

Definition 3.2 (Greedy alignment [KPS21])**.** We say that an alignment $\mathcal{A}$ of two strings $X, Y \in \Sigma^*$ is *greedy* if $X[x] \neq Y[y]$ holds for every $(x, y) \in \mathsf{B}_{\mathcal{A}}(X, Y)$ such that $x \neq |X|$ and $y \neq |Y|$.

As proven in [KPS21], any two strings $X, Y \in \Sigma^*$ have an optimal alignment that is also greedy.

For strings $X, Y \in \Sigma^*$ and integers $k \geq w \geq 0$, we denote by $\mathsf{A}_{k,w}(X, Y)$ the family of alignments of $\mathcal{A} : X \to Y$ whose cost is at most k and whose width is at most w. Furthermore, we define $\mathsf{GA}_{k,w}(X, Y)$ as the set consisting of all greedy alignments in $\mathsf{A}_{k,w}(X, Y)$. The following result can be obtained using a straightforward adaptation of the Landau–Vishkin algorithm [LV88]. The only difference compared to the baseline implementation is that the DP table is artificially restricted to diagonals $[-w\,..\,w]$.

Lemma 3.3. *Given strings $X, Y \in \Sigma^{\leq n}$ and integers $k, w \in \mathbb{Z}_{\geq 0}$, one can in $\mathcal{O}(n + kw)$ time decide whether $\mathsf{A}_{k,w}(X, Y) \neq \emptyset$. If the answer is positive, the algorithm reports a witness alignment $\mathcal{A} \in \mathsf{GA}_{k,w}(X, Y)$.*

3.3 Trees and Forests

Definition 3.4 (Forest)**.** We say that a forest F is a (possibly empty) sequence of non-empty rooted ordered trees $\mathsf{T}_1, \ldots, \mathsf{T}_m$. Formally, each such tree T_i consists of a root node v_i and a forest F_i representing the descendants of v_i. We write V_{F} to denote the node set of F.

For a node $v \in V_{\mathsf{F}}$ in a forest F, we denote the parent of v by $\mathsf{parent}_{\mathsf{F}}(v)$. If v is a root, then we set $\mathsf{parent}_{\mathsf{F}}(v) = \bot_{\mathsf{F}}$; consistently with [Mao21, AJ21], we refer to $\bot_{\mathsf{F}}$ as the *virtual root* of F. We denote $\bar{V}_{\mathsf{F}} = V_{\mathsf{F}} \cup \{\bot_{\mathsf{F}}\}$ as the set of nodes in F including the virtual root. For each node $v \in \bar{V}_{\mathsf{F}}$, we

denote the sequence of children of v by $\mathsf{children}_{\mathsf{F}}(v)$. Observe that the forest F is uniquely specified by the lists $\mathsf{children}_{\mathsf{F}}(v)$ for $v \in \bar{V}_{\mathsf{F}}$ and these lists, in total, contain every node $u \in V_{\mathsf{F}}$ exactly once.

We define the *height* of a forest F, denoted $\mathsf{height}(\mathsf{F})$, as the maximum number of nodes on the root-to-leaf path in F. Thus, an empty forest has height 0, whereas a forest consisting of roots only has height 1.

Definition 3.5 (Labelled forest)**.** A *labeled forest* F consists of a forest F and a labeling function $\lambda : V_{\mathsf{F}} \to \Sigma$ assigning labels (from an integer *alphabet* $\Sigma = [0\,..\,\sigma)$) to nodes of F.

In this work, we typically consider two labeled forests F, G. We then assume that the underlying unlabeled forests F, G have disjoint node sets and a joint labeling $\lambda : V_{\mathsf{F}} \cup V_{\mathsf{G}} \to \Sigma$.

Definition 3.6 (Tree edit distance)**.** The following operations are jointly called *edits* of a labeled forest F:

Substitution (relabeling) Change the label of a node in F with another symbol from Σ.

Deletion Delete a node v from F. As a result, the children of v become the children of $\mathsf{parent}_F(v)$ while preserving their order. Formally, we modify $\mathsf{children}_F(\mathsf{parent}_F(v))$ by replacing v with $\mathsf{children}_F(v)$.

Insertion Insert a labeled node v into F so that deletion of v produces F again. Formally, an insertion is specified by a $u \in \bar{V}_{\mathsf{F}}$ (which will become the parent of v), a (possibly empty) fragment of $\mathsf{children}_F(u)$ (identifying nodes which will become the children of v), and the label of the new node.

The tree edit distance $\mathsf{ted}(F, G)$ is defined as the minimum number of edits (listed above) required to transform one forest to the other.

Just like the string edit distance, the tree edit distance admits an equivalent definition in terms of *alignments* $\mathcal{A} : F \to G$. Such an alignment must specify which nodes in F are *deleted* and which nodes in G are *inserted*. The remaining nodes are *aligned* (either *relabeled* or *matched*). Unlike for string edit distance, where the alignment must only be consistent with the left-to-right ordering, here, the alignment must the consistent with both the pre-order and the post-order forest traversal.

3.4 Parenthesis Representation of Forests

Aiming to adapt string techniques into the forest setting, we map each forest to a string through the *parentheses representation* (closely related to the *bi-order traversal* considered in [Mao21]).

Definition 3.7 (Parenthesis representation)**.** For every unlabeled forest F, we define the *parentheses representation* of F, denoted $\mathsf{P}(\mathsf{F}) \in \{(,)\}^*$, using the following recursion: If F consists of trees $\mathsf{T}_1, \ldots, \mathsf{T}_m$, where each T_i consists of a root node v_i and a forest F_i representing the descendants of v_i, then $\mathsf{P}(\mathsf{F}) = \bigodot_{i=1}^{m} \mathsf{P}(\mathsf{T}_i)$, where $\mathsf{P}(\mathsf{T}_i) = (\cdot\,\mathsf{P}(\mathsf{F}_i)\cdot)$ and $\cdot$ as well as $\bigodot$ denote concatenation.

For a labelling $\lambda : V_{\mathsf{F}} \to \Sigma$, we further set $\mathsf{P}_\lambda(\mathsf{F}) \in (\{(,)\} \times \Sigma)^*$ so that $\mathsf{P}_\lambda(\mathsf{F}) = \bigodot_{i=1}^{m} \mathsf{P}_\lambda(\mathsf{T}_i)$ and $\mathsf{P}_\lambda(\mathsf{T}_i) = (_{\lambda(v_i)} \cdot \mathsf{P}_\lambda(\mathsf{F}_i) \cdot)_{\lambda(v_i)}$. For a labeled forest $F = (\mathsf{F}, \lambda)$, we also write $\mathsf{P}(F) := \mathsf{P}_\lambda(\mathsf{F})$.

Observe that $\mathsf{P}(\mathsf{F})$ forms a well-parenthesized sequence and there is a bijection between nodes of F and the pairs of matching parentheses in $\mathsf{P}(\mathsf{F})$. We define functions $o_{\mathsf{F}}, c_{\mathsf{F}} : V_{\mathsf{F}} \to [0\,..\,2|\mathsf{F}|)$ so that the opening and the closing parenthesis representing u are located at positions $o_{\mathsf{F}}(u)$ and $c_{\mathsf{F}}(u)$, respectively.

Definition 3.8. We say that $\mathcal{A} : \mathsf{P}(\mathsf{F}) \to \mathsf{P}(\mathsf{G})$ is a *tree alignment* of forests F and G, denoted $\mathcal{A} : \mathsf{F} \to \mathsf{G}$ if the following *consistency* conditions are satisfied for each $u \in V_{\mathsf{F}}$:

- either $\mathcal{A}$ deletes both $\mathsf{P}(\mathsf{F})[o_{\mathsf{F}}(u)]$ and $\mathsf{P}(\mathsf{F})[c_{\mathsf{F}}(u)]$, or
- there exists $v \in V_{\mathsf{G}}$ such that $\mathsf{P}(\mathsf{F})[o_{\mathsf{F}}(u)] \sim_{\mathcal{A}} \mathsf{P}(\mathsf{G})[o_{\mathsf{G}}(v)]$ and $\mathsf{P}(\mathsf{F})[c_{\mathsf{F}}(u)] \sim_{\mathcal{A}} \mathsf{P}(\mathsf{G})[c_{\mathsf{G}}(v)]$.

We denote the set of all tree alignments $\mathcal{A} : \mathsf{F} \to \mathsf{G}$ with $\mathsf{TA}(\mathsf{F}, \mathsf{G}) \subseteq \mathsf{A}(\mathsf{P}(\mathsf{F}), \mathsf{P}(\mathsf{G}))$.

Definition 3.9. Consider a joint labeling λ of forests F, G and an alignment $A \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$. We denote $\mathsf{ted}_{\lambda,\mathcal{A}}(\mathsf{F}, \mathsf{G}) := \frac{1}{2}\mathsf{ed}_{\mathcal{A}}(\mathsf{P}_\lambda(\mathsf{F}), \mathsf{P}_\lambda(\mathsf{G}))$. In terms of the underlying labeled forests F, G, we express this as $\mathsf{ted}_{\mathcal{A}}(F, G) := \frac{1}{2}\mathsf{ed}_{\mathcal{A}}(\mathsf{P}(F), \mathsf{P}(G))$.

We observe that Definitions 3.6 and 3.9 are equivalent as each tree edit operation can be represented using two string edit operations. For example, a deletion of a node $v \in V_F$ corresponds to deleting the characters at positions $o_F(u)$ and $c_F(u)$ in $P(F)$.

Observation 3.10. *For any two forests F, G, we have* $\mathsf{ted}(F, G) = \min_{\mathcal{A} \in \mathsf{TA}(F,G)} \mathsf{ted}_{\mathcal{A}}(F, G)$.

Furthermore, for an integer $k \in \mathbb{Z}_{\geq 0}$, we denote $\mathsf{TA}_k(F, G) = \{\mathcal{A} \in \mathsf{TA}(F, G) : \mathsf{ted}_{\mathcal{A}}(F, G) \leq k\}$. We also denote

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

4 Labeling Refinements

Consider two joint labelings $\lambda, \lambda' : V_{\mathsf{F}} \cup V_{\mathsf{G}} \to \Sigma$ of forests F, G. We say that λ is a *refinement* of λ' if $\lambda'(u) = \lambda'(v)$ holds for any two nodes $u, v \in V_{\mathsf{F}} \cup V_{\mathsf{G}}$ such that $\lambda(u) = \lambda(v)$. If simultaneously λ is a refinement of λ' and λ' is a refinement of λ, we say that λ is equivalent to λ'.

4.1 Look-Ahead Refinement

For a node v of an unlabeled forest F, let $\mathsf{sub}(v)$ denote the subtree of F rooted at v. Further, for an integer $d \in \mathbb{Z}_+$, let $\mathsf{sub}_{<d}(v)$ denote the subtree of $\mathsf{sub}(v)$ consisting of nodes at distance less than d from the root v.

Definition 4.1 (Look-ahead refinement)**.** Let λ be a joint labeling of forests F, G. We say that λ' is a *depth-d look-ahead refinement* of λ if, for any $u, v \in V_{\mathsf{F}} \cup V_{\mathsf{G}}$, we have $\lambda'(u) = \lambda'(v)$ if and only if $\mathsf{P}_\lambda(\mathsf{sub}_{<d}(u)) = \mathsf{P}_\lambda(\mathsf{sub}_{<d}(v))$. We use $\mathsf{L}(\lambda, d)$ to denote a depth-d look-ahead refinement of λ (chosen arbitrarily up to equivalence of labelings).

Lemma 4.2. *Consider forests F, G, their joint labeling λ, an alignment $\mathcal{A} \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$, and an integer $d \in \mathbb{Z}_+$. Then,*

$$\mathsf{ted}_{\mathsf{L}(\lambda,d),\mathcal{A}}(\mathsf{F}, \mathsf{G}) \leq d \cdot \mathsf{ted}_{\lambda,\mathcal{A}}(\mathsf{F}, \mathsf{G}).$$

Proof. Let the *level* of a node in $V_{\mathsf{F}} \cup V_{\mathsf{G}}$ denote its distance to the root. We *mark* some nodes of F, G and prove that $\mathsf{ted}_{\mathsf{L}(\lambda,d),\mathcal{A}}(\mathsf{F}, \mathsf{G})$ is bounded by the number of marks.

1. If $\mathcal{A}$ deletes a node $u \in V_{\mathsf{F}}$ at level ℓ or aligns such a node to a node $v \in V_{\mathsf{G}}$ with $\lambda(u) \neq \lambda(v)$, then we mark all ancestors of u at levels in $(\ell - d \,..\, \ell]$.
2. If $\mathcal{A}$ inserts a node $v \in V_{\mathsf{G}}$ at level ℓ, then we mark all ancestors of v at levels in $(\ell - d \,..\, \ell]$.

Clearly, the total number of marks does not exceed $d \cdot \mathsf{ted}_{\lambda,\mathcal{A}}(\mathsf{F}, \mathsf{G})$.

It remains to prove that each unit of $\mathsf{ted}_{\mathsf{L}(\lambda,d),\mathcal{A}}(\mathsf{F}, \mathsf{G})$ can be charged to a marked node. If $\mathcal{A}$ deletes a node in $V_{\mathsf{F}} \cup V_{\mathsf{G}}$, then this node is already marked. We shall prove that if $\mathcal{A}$ aligns a node $u \in V_{\mathsf{F}}$ with a node $v \in V_{\mathsf{G}}$ such that $\mathsf{L}(\lambda, d)(u) \neq \mathsf{L}(\lambda, d)(v)$, then either u or v is marked.

For a proof by contradiction, suppose that this is not the case. In particular, this means that $\mathcal{A}$ does not delete nor insert any node in $\mathsf{sub}_{<d}(u)$ and $\mathsf{sub}_{<d}(v)$. Consequently, these trees are isomorphic, i.e., $\mathsf{P}(\mathsf{sub}_{<d}(u)) = \mathsf{P}(\mathsf{sub}_{<d}(v))$, and $\mathcal{A}$ aligns nodes according to this isomorphism. Moreover, λ assigns the same label to any two nodes matched by this isomporhism (otherwise, u would have been marked), and thus $\mathsf{P}_\lambda(\mathsf{sub}_{<d}(u)) = \mathsf{P}_\lambda(\mathsf{sub}_{<d}(v))$, contradicting the assumption that $\mathsf{L}(\lambda, d)(u) \neq \mathsf{L}(\lambda, d)(v)$. □

Lemma 4.3. *Given forests F, G of total size n, their joint labeling λ, and an integer $d \in \mathbb{Z}_{\geq 0}$, a labeling equivalent to $\mathsf{L}(\lambda, d)$ can be constructed in $\mathcal{O}(n)$ using a randomized algorithm correct with high probability.*

Proof. We replace the label of each node $v \in V_\mathsf{F} \cup V_\mathsf{G}$ with the Karp–Rabin fingerprint [KR87] of $\mathsf{P}_\lambda(\mathsf{sub}_{<d}(v))$. Choosing a sufficiently large prime number $p = n^{\Theta(1)}$ and a uniformly random seed $r \in [0\,..\,p)$, we can guarantee that, with high probability, the fingerprints will have no collisions among the n strings $\mathsf{P}_\lambda(\mathsf{sub}_{<d}(v))$.

As far as the implementation is concerned, for each node v at level ℓ, we first identify all its descendants $v_1, \ldots, v_m$ at level $\ell + d$. Such lists can be produced in linear time by a traversal of F and G that maintains the ancestors of the currently visited node in an array indexed by the level of the ancestor. When visiting a node at level $\ell \geq d$, we add it to the list of descendants of the ancestor at level $\ell - d$. Next, we note that $\mathsf{P}_\lambda(\mathsf{sub}_{<d}(v))$ can be obtained from $\mathsf{P}_\lambda(\mathsf{sub}(v))$ by cutting out the fragments representing $\mathsf{P}_\lambda(\mathsf{sub}(v_i))$. Thus, given the list $v_1, \ldots, v_m$, we can represent $\mathsf{P}_\lambda(\mathsf{sub}_{<d}(v))$ as the concatenation of $m+1$ fragments of $\mathsf{P}_\lambda(\mathsf{sub}(v))$. The fingerprint of each fragment of $\mathsf{P}_\lambda(F)$ and $\mathsf{P}_\lambda(G)$ can be constructed in $\mathcal{O}(1)$ time and the Karp–Rabin fingerprints support concatenations in $\mathcal{O}(1)$ time. Consequently, the desired fingerprints can be constructed in $\mathcal{O}(n)$ time in total. □

Next, we show that there exists an optimum alignment that becomes greedy if we refine the labeling using look-ahead $d \geq \max\{\mathsf{height}(F), \mathsf{height}(G)\}$.

Proposition 4.4. *Let λ be a joint labeling of unlabeled forests F, G of height at most h and let $\hat{\lambda} = \mathsf{L}(\lambda, h)$. Then, there exists a tree alignment $\mathcal{A} \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$ of optimum cost $\mathsf{ted}_{\lambda,\mathcal{A}}(\mathsf{F}, \mathsf{G}) = \mathsf{ted}_\lambda(\mathsf{F}, \mathsf{G})$ such that $\mathcal{A} \in \mathsf{GA}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}))$.*

Proof. Consider a tree alignment $\mathcal{A} \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$ of optimum cost $\mathsf{ted}_{\lambda,\mathcal{A}}(\mathsf{F}, \mathsf{G}) = \mathsf{ted}_\lambda(\mathsf{F}, \mathsf{G})$. We perform the following steps repeatedly for $2|\mathsf{F}|$ rounds: at round i, we construct a tree alignment $\mathcal{A}_i \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$ such that $\mathsf{ted}_{\lambda,\mathcal{A}_i}(F, G) = \mathsf{ted}_{\lambda,\mathcal{A}}(F, G)$ and $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x] \neq \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y]$ for every $(x, y) \in \mathsf{B}_{\mathcal{A}_i}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G})) \cap ([0\,..\,i) \times [0\,..\,2|\mathsf{G}|))$. Trivially, $\mathcal{A}_0 := \mathcal{A}$ satisfies this condition for $i = 0$.

Round i. At round i, we construct $\mathcal{A}_i$ given $\mathcal{A}_{i-1} = (x_t, y_t)_{t \in [0\,..\,m]}$. If $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x] \neq \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y]$ already holds for every $(x, y) \in \mathsf{B}_{\mathcal{A}_{i-1}}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G})) \cap ([0\,..\,i) \times [0\,..\,2|\mathsf{G}|))$, then we simply set $\mathcal{A}_i := \mathcal{A}_{i-1}$. Otherwise, let (x, y) be the leftmost breakpoint with $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x] = \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y]$. By the assumptions on $\mathcal{A}_{i-1}$, we must have $x = i - 1$.

Case 1. First, we consider the case where $(x, y) = (o_\mathsf{F}(u), o_\mathsf{G}(v))$ for some $(u, v) \in V_\mathsf{F} \times V_\mathsf{G}$. Let $(\bar{x}, \bar{y}) = (c_\mathsf{F}(u), c_\mathsf{G}(v))$ and note that, due to $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x] = \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y]$ and $\hat{\lambda} = \mathsf{L}(\lambda, h)$, we have $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x\,..\,\bar{x}] = \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y\,..\,\bar{y}]$. We define $q := \min\{t : x_t > \bar{x} \text{ and } y_t > \bar{y}\}$ and create $\mathcal{A}_i$ so that it:

- aligns $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[0\,..\,x)$ against $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})[0\,..\,y)$ in the same way as $\mathcal{A}_{i-1}$ did,
- matches $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x\,..\,\bar{x}]$ against $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y\,..\,\bar{y}]$,
- deletes $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})(\bar{x}\,..\,x_q)$ and inserts $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})(\bar{y}\,..\,y_q)$ (at least one of these fragments is empty),
- aligns $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x_q\,..\,2|\mathsf{F}|)$ against $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y_q\,..\,2|\mathsf{G}|)$ in the same way as $\mathcal{A}_{i-1}$ did.

Formally, $\mathcal{A}_i$ is defined as follows, where $\odot$ denotes concatenation of sequences and $p \in [0\,..\,m]$ is such that $(x, y) = (x_p, y_p)$:

$$\mathcal{A}_i = (x_t, y_t)_{t\in[0\,..\,p)} \odot (x+\delta, y+\delta)_{\delta\in[0\,..\,\bar{x}-x]} \odot (\hat{x}, y_q)_{\hat{x}\in(\bar{x}\,..\,x_q)} \odot (x_q, \hat{y})_{\hat{y}\in(\bar{y}\,..\,y_q)} \odot (x_t, y_t)_{t\in[q\,..\,m]}.$$

First, we show $\mathcal{A}_i \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$. For this, take a node $u' \in V_\mathsf{F}$ and consider several cases:

- $\boldsymbol{o_\mathsf{F}(u') \in [x\,..\,\bar{x}]}$ or $\boldsymbol{c_\mathsf{F}(u') \in [x\,..\,\bar{x}]}$. In this case, $u' \in \mathsf{sub}(u)$ and thus both $o_\mathsf{F}(u') \in [x\,..\,\bar{x}]$ and $c_\mathsf{F}(u') \in [x\,..\,\bar{x}]$. Moreover, due to $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x\,..\,\bar{x}] \simeq_{\mathcal{A}_i} \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y\,..\,\bar{y}]$, in this case, $\mathcal{A}_i$ matches u' with the corresponding node of the subtree $\mathsf{sub}(v)$ identical to $\mathsf{sub}(u)$.
- $\boldsymbol{o_\mathsf{F}(u'), c_\mathsf{F}(u') \in (\bar{x}\,..\,x_q)}$. In this case, $\mathcal{A}_i$ deletes both $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[o_\mathsf{F}(u')]$ and $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[c_\mathsf{F}(u')]$.
- $\boldsymbol{o_\mathsf{F}(u') \in (\bar{x}\,..\,x_q)}$ and $\boldsymbol{c_\mathsf{F}(u') \in [x_q\,..\,2|\mathsf{F}|)}$. In this case, $\mathcal{A}_i$ deletes $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[o_\mathsf{F}(u')]$ and handles $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[c_\mathsf{F}(u')]$ in the same way as $\mathcal{A}_{i-1}$ did. Thus, we must prove that $\mathcal{A}_{i-1}$ deleted u'. For a proof by contradiction, suppose that $\mathcal{A}_{i-1}$ aligned u' with some node $v' \in V_\mathsf{G}$. By the non-crossing property of $\mathcal{A}_{i-1}$, due to $(x, y), (x_q, y_q) \in \mathcal{A}_{i-1}$, we must have $o_\mathsf{G}(v') \in [y\,..\,y_q)$ and $c_\mathsf{G}(v') \in [y_q\,..\,2|\mathsf{G}|)$. Moreover, since $(\bar{x}\,..\,x_q) \neq \emptyset$, we must have $(\bar{y}\,..\,y_q) = \emptyset$, which means that $o_\mathsf{G}(v') \in [y\,..\,\bar{y}] = [y\,..\,y_q)$. Consequently, $v' \in \mathsf{sub}(v)$ and hence $c_\mathsf{G}(v') \in [y\,..\,\bar{y}] = [y\,..\,y_q)$; this contradicts $c_\mathsf{G}(v') \in [y_q\,..\,2|\mathsf{G}|)$.
- $\boldsymbol{o_\mathsf{F}(u') \in [0\,..\,x)}$ and $\boldsymbol{c_\mathsf{F}(u') \in (\bar{x}\,..\,x_q)}$. In this case, $\mathcal{A}_i$ deletes $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[c_\mathsf{F}(u')]$ and handles $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[o_\mathsf{F}(u')]$ in the same way as $\mathcal{A}_{i-1}$ did. Thus, we must prove that $\mathcal{A}_{i-1}$ deleted u'. For a proof by contradiction, suppose that $\mathcal{A}_{i-1}$ aligned u' with some node $v' \in V_\mathsf{G}$. By the non-crossing property of $\mathcal{A}_{i-1}$, due to $(x, y), (x_q, y_q) \in \mathcal{A}_{i-1}$, we must have $o_\mathsf{G}(v') \in [0\,..\,y)$ and $c_\mathsf{G}(v') \in [y\,..\,y_q)$. Moreover, since $(\bar{x}\,..\,x_q) \neq \emptyset$, we must have $(\bar{y}\,..\,y_q) = \emptyset$, which means that $c_\mathsf{G}(v') \in [y\,..\,\bar{y}] = [y\,..\,y_q)$. Consequently, $v' \in \mathsf{sub}(v)$ and hence $o_\mathsf{G}(v') \in [y\,..\,\bar{y}] = [y\,..\,y_q)$; this contradicts $o_\mathsf{G}(v') \in [0\,..\,y)$.
- $\boldsymbol{o_\mathsf{F}(u'), c_\mathsf{F}(u') \in [0\,..\,x) \cup [x_q\,..\,2|\mathsf{F}|)}$. In this case, $\mathcal{A}_i$ handles u' in the same way as $\mathcal{A}_{i-1}$ did.

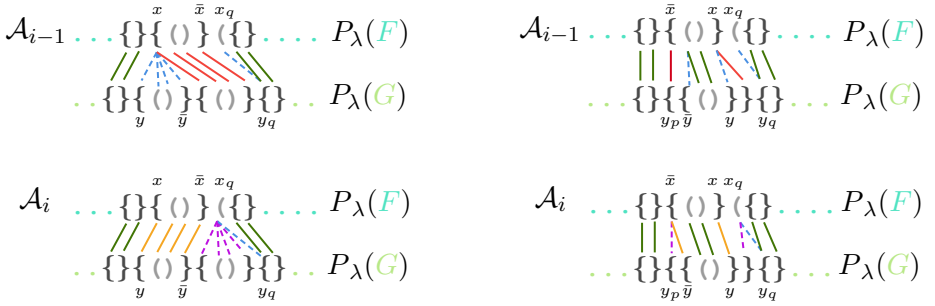
This case analysis above is exhaustive and, in all cases, the two parentheses corresponding to u' are handled consistently. Consequently, we indeed have $\mathcal{A}_i \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$.

Next we argue that $\mathsf{ted}_{\lambda,\mathcal{A}_i}(\mathsf{F}, \mathsf{G}) \leq \mathsf{ted}_{\lambda,\mathcal{A}_{i-1}}(\mathsf{F}, \mathsf{G})$ or, equivalently, $\mathsf{ed}_{\mathcal{A}_i}(\mathsf{P}_\lambda(\mathsf{F}), \mathsf{P}_\lambda(G)) \leq \mathsf{ed}_{\mathcal{A}_{i-1}}(\mathsf{P}_\lambda(\mathsf{F}), \mathsf{P}_\lambda(G))$. The two alignments differ only on $\mathsf{P}_\lambda(\mathsf{F})[x\,..\,x_q)$ and $\mathsf{P}_\lambda(\mathsf{G})[y\,..\,y_q)$. The contribution of these fragments to the cost of $\mathcal{A}_i : \mathsf{P}_\lambda(\mathsf{F}) \rightsquigarrow \mathsf{P}_\lambda(\mathsf{G})$ equals $|(\bar{x}\,..\,x_q)| + |(\bar{y}\,..\,y_q)| = \big||(\bar{x}\,..\,x_q)| - |(\bar{y}\,..\,y_q)|\big| = \big||[x\,..\,x_q) - |[y\,..\,y_q)|\big|$. Due to $(x, y), (x_q, y_q) \in \mathcal{A}_{i-1}$, the contribution of these fragments to the cost of $\mathcal{A}_{i-1} : \mathsf{P}_\lambda(\mathsf{F}) \rightsquigarrow \mathsf{P}_\lambda(\mathsf{G})$ must be at least that large.

Finally, we note that the breakpoints of $\mathcal{A}_i$ in $[0\,..\,i) \times [0\,..\,2|\mathsf{G}|)$ satisfy the greedy property. For breakpoints to the left of (x, y), this follows by definition of (x, y). Moreover, due to $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x] \simeq \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y]$, the pair (x, y) is not a breakpoint in $\mathcal{A}_i$; in particular, the subsequent pair $(x{+}1, y{+}1) \in \mathcal{A}_i$ satisfies $x + 1 = i$.

Case 2. Next, we consider the case where $(x, y) = (c_\mathsf{F}(u), c_\mathsf{G}(v))$ for some $(u, v) \in V_\mathsf{F} \times V_\mathsf{G}$. Define $(\bar{x}, \bar{y}) = (o_\mathsf{F}(u), o_\mathsf{G}(v))$ and note that, due to $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x] = \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y]$ and $\hat{\lambda} = \mathsf{L}(\lambda, h)$, we have $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[\bar{x}\,..\,x] = \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[\bar{y}\,..\,y]$. Define $q = \min\{t : x_t > x \text{ and } y_t > y\}$ and $p = \max\{t : x_t \leq \bar{x} \text{ and } y_t \leq \bar{y}\}$. We create $\mathcal{A}_i$ so that it:

- aligns $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[0\,..\,x_p)$ against $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})[0\,..\,y_p)$ in the same way as $\mathcal{A}_{i-1}$ did,
- deletes $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x_p\,..\,\bar{x})$ and inserts $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y_p\,..\,\bar{y})$ (at least one of these fragments is empty),
- matches $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[\bar{x}\,..\,x]$ against $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})[\bar{y}\,..\,y]$,
- deletes $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})(x\,..\,x_q)$ and inserts $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})(y\,..\,y_q)$ (at least one of these fragments is empty),
- aligns $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x_q\,..\,2|\mathsf{F}|)$ against $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y_q\,..\,2|\mathsf{G}|)$ in the same way as $\mathcal{A}_{i-1}$ did.



(a) Case 1 (b) Case 2

Figure 2: Example of the construction of alignment $\mathcal{A}_i$ from alignment $\mathcal{A}_{i-1}$

Formally, $\mathcal{A}_i$ is defined as follows:

$$\mathcal{A}_i = (x_t, y_t)_{t\in[0..p)} \odot (\hat{x}, \bar{y})_{\hat{x}\in[x_p..\bar{x})} \odot (\bar{x}, \hat{y})_{\hat{y}\in[y_p..\bar{y})} \odot (\bar{x}+\delta, \bar{y}+\delta)_{\delta\in[0..x-\bar{x}]} \odot (\hat{x}, y_q)_{\hat{x}\in(x..x_q)} \odot (x_q, \hat{y})_{\hat{y}\in(y..y_q)} \odot (x_t, y_t)_{t\in[q..m]}.$$

First, we show $\mathcal{A}_i \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$. For this, take a node $u' \in V_\mathsf{F}$ and consider several cases:

- $\boldsymbol{o_\mathsf{F}(u') \in [\bar{x}\,..\,x]}$ or $\boldsymbol{c_\mathsf{F}(u') \in [\bar{x}\,..\,x]}$. In this case, $u' \in \mathsf{sub}(u)$ and thus both $o_\mathsf{F}(u') \in [\bar{x}\,..\,x]$ and $c_\mathsf{F}(u') \in [\bar{x}\,..\,x]$. Moreover, due to $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[\bar{x}\,..\,x] \simeq_{\mathcal{A}_i} \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[\bar{y}\,..\,y]$, in this case, $\mathcal{A}_i$ matches u' with the corresponding node of the subtree $\mathsf{sub}(v)$ identical to $\mathsf{sub}(u)$.
- $\boldsymbol{o_\mathsf{F}(u'), c_\mathsf{F}(u') \in [x_p\,..\,\bar{x}) \cup (x\,..\,x_q)}$. In this case, $\mathcal{A}_i$ deletes both $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[o_\mathsf{F}(u')]$ and $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[c_\mathsf{F}(u')]$.
- $\boldsymbol{o_\mathsf{F}(u') \in [x_p\,..\,\bar{x})}$ and $\boldsymbol{c_\mathsf{F}(u') \in [x_q\,..\,2|\mathsf{F}|)}$. In this case, $\mathcal{A}_i$ deletes $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[o_\mathsf{F}(u')]$ and handles $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[c_\mathsf{F}(u')]$ in the same way as $\mathcal{A}_{i-1}$ did. Thus, we must prove that $\mathcal{A}_{i-1}$ deleted u'. For a proof by contradiction, suppose that $\mathcal{A}_{i-1}$ aligned u' with some node $v' \in V_\mathsf{G}$. By the non-crossing property of $\mathcal{A}_{i-1}$, due to $(x_p, y_p), (x, y), (x_q, y_q) \in \mathcal{A}_{i-1}$, we must have $o_\mathsf{G}(v') \in [y_p\,..\,y)$ and $c_\mathsf{G}(v') \in [y_q\,..\,2|\mathsf{G}|)$. Moreover, since $[x_p\,..\,\bar{x}) \neq \emptyset$, we must have $[y_p\,..\,\bar{y}) = \emptyset$, which means that $o_\mathsf{G}(v') \in [\bar{y}\,..\,y) = [y_p\,..\,y)$. Consequently, $v' \in \mathsf{sub}(v)$ and hence $c_\mathsf{G}(v') \in [s\,..\,y] = [y_p\,..\,y]$; this contradicts $c_\mathsf{G}(v') \in [y_q\,..\,2|\mathsf{G}|)$ due to $y_q > y$.
- $\boldsymbol{o_\mathsf{F}(u') \in (x\,..\,x_q)}$ and $\boldsymbol{c_\mathsf{F}(u') \in [x_q\,..\,2|\mathsf{F}|)}$. In this case, $\mathcal{A}_i$ deletes $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[o_\mathsf{F}(u')]$ and handles $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[c_\mathsf{F}(u')]$ in the same way as $\mathcal{A}_{i-1}$ did. Thus, we must prove that $\mathcal{A}_{i-1}$ deleted u'. For a proof by contradiction, suppose that $\mathcal{A}_{i-1}$ aligned u' with some node $v' \in V_\mathsf{G}$. By the non-crossing property of $\mathcal{A}_{i-1}$, due to $(x_p, y_p), (x, y), (x_q, y_q) \in \mathcal{A}_{i-1}$, we must have $o_\mathsf{G}(v') \in [y\,..\,y_q)$ and $c_\mathsf{G}(v') \in [y_q\,..\,2|\mathsf{G}|)$. Moreover, since $(x\,..\,x_q) \neq \emptyset$, we must have $(y\,..\,y_q) = \emptyset$, which means that $o_\mathsf{G}(v') = y$; this is a contradiction because $y = c_\mathsf{G}(v)$.
- $\boldsymbol{o_\mathsf{F}(u') \in [0\,..\,x_p)}$ and $\boldsymbol{c_\mathsf{F}(u') \in [x_p\,..\,\bar{x})}$. In this case, $\mathcal{A}_i$ deletes $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[c_\mathsf{F}(u')]$ and handles $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[o_\mathsf{F}(u')]$ in the same way as $\mathcal{A}_{i-1}$ did. Thus, we must prove that $\mathcal{A}_{i-1}$ deleted u'. For a proof by contradiction, suppose that $\mathcal{A}_{i-1}$ aligned u' with some node $v' \in V_\mathsf{G}$. By the non-crossing property of $\mathcal{A}_{i-1}$, due to $(x_p, y_p), (x, y), (x_q, y_q) \in \mathcal{A}_{i-1}$, we must have $o_\mathsf{G}(v') \in [0\,..\,y_p)$ and $c_\mathsf{G}(v') \in [y_p\,..\,y)$. Moreover, since $[x_p\,..\,\bar{x}) \neq \emptyset$, we must have $[y_q\,..\,\bar{y}) = \emptyset$, which means that $o_\mathsf{G}(v') \in [\bar{y}\,..\,y) = [y_p\,..\,y)$. Consequently, $v' \in \mathsf{sub}(v)$ and hence $o_\mathsf{G}(v') \in [\bar{y}\,..\,y] = [y_p\,..\,y]$; this contradicts $o_\mathsf{G}(v') \in [0\,..\,y_p)$.
- $\boldsymbol{o_\mathsf{F}(u') \in [0\,..\,x_p)}$ and $\boldsymbol{c_\mathsf{F}(u') \in (x\,..\,x_q)}$. In this case, $\mathcal{A}_i$ deletes $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[c_\mathsf{F}(u')]$ and handles $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[o_\mathsf{F}(u')]$ in the same way as $\mathcal{A}_{i-1}$ did. Thus, we must prove that $\mathcal{A}_{i-1}$ deleted u'. For a proof by contradiction, suppose that $\mathcal{A}_{i-1}$ aligned u' with some node $v' \in V_\mathsf{G}$. By the non-crossing property of $\mathcal{A}_{i-1}$, due to $(x_p, y_p), (x, y), (x_q, y_q) \in \mathcal{A}_{i-1}$, we must have $o_\mathsf{G}(v') \in$

$[0\,..\,y_p)$ and $c_{\mathsf{G}}(v') \in [y\,..\,y_q)$. Moreover, since $(x\,..\,x_q) \neq \emptyset$, we must have $(y\,..\,y_q) = \emptyset$, which means that $c_{\mathsf{G}}(v') = y$. Consequently, $v = v'$ and hence $o_{\mathsf{G}}(v') = \bar{y}$; this contradicts $o_{\mathsf{G}}(v') \in [0\,..\,y_p)$ due to $y_p \leq \bar{y}$.

- $\boldsymbol{o_{\mathsf{F}}(u'), c_{\mathsf{F}}(u') \in [0\,..\,x_p) \cup [x_q\,..\,2|\mathsf{F}|)}$. In this case, $\mathcal{A}_i$ handles u' in the same way as $\mathcal{A}_{i-1}$ did.

This case analysis above is exhaustive and, in all cases, the two parentheses corresponding to u' are handled consistently. Consequently, we indeed have $\mathcal{A}_i \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$.

Next we argue that $\mathsf{ted}_{\lambda,\mathcal{A}_i}(\mathsf{F}, \mathsf{G}) \leq \mathsf{ted}_{\lambda,\mathcal{A}_{i-1}}(\mathsf{F}, \mathsf{G})$ or, equivalently, $\mathsf{ed}_{\mathcal{A}_i}(\mathsf{P}_\lambda(\mathsf{F}), \mathsf{P}_\lambda(G)) \leq \mathsf{ed}_{\mathcal{A}_{i-1}}(\mathsf{P}_\lambda(\mathsf{F}), \mathsf{P}_\lambda(G))$. The two alignments differ only in how on $\mathsf{P}_\lambda(\mathsf{F})[x_p\,..\,x_q)$ and $\mathsf{P}_\lambda(\mathsf{G})[y_p\,..\,y_q)$. The contribution of these fragments to the cost of $\mathcal{A}_i : \mathsf{P}_\lambda(\mathsf{F}) \rightsquigarrow \mathsf{P}_\lambda(\mathsf{G})$ equals $|[x_p\,..\,\bar{x})| + |[y_p\,..\,\bar{y})| + |(x\,..\,x_q)| + |(y\,..\,y_q)| = \big||[x_p\,..\,\bar{x})| - |[y_p\,..\,\bar{y})|\big| + \big||(x\,..\,x_q)| - |(y\,..\,y_q)|\big| = \big||[x_p\,..\,x) - |[y_p\,..\,y)|\big| + \big||[x\,..\,x_q) - |[y\,..\,y_q)|\big|$. Due to $(x_p, y_p)(x, y), (x_q, y_q) \in \mathcal{A}_{i-1}$, the contribution of these fragments to the cost of $\mathcal{A}_{i-1} : \mathsf{P}_\lambda(\mathsf{F}) \rightsquigarrow \mathsf{P}_\lambda(\mathsf{G})$ must be at least that large.

Finally, we shall prove that $\mathcal{A}_i$ is greedy up to index $i-1$. For breakpoints to the left of (x_p, y_p), this follows by definition of (x, y). Moreover, none of the pairs $(\bar{x}+\delta, \bar{y}+\delta)_{\delta \in [0\,..\,x-\bar{x}]}$ is a breakpoint and, in particular, the subsequent pair $(x+1, y+1) \in \mathcal{A}_i$ satisfies $x + 1 = i$. Hence, it suffices to consider breakpoints of the form $(\hat{x}, \bar{y})$ for some $\hat{x} \in [x_p\,..\,\bar{x})$ or $(\bar{x}, \hat{y})$ for some $\hat{y} \in [y_p\,..\,\bar{y})$. These two cases are symmetric, so let us consider $(\bar{x}, \hat{y})$ for some $\hat{y} \in [y_p\,..\,\bar{y})$. For a proof by contradiction, suppose that $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[\bar{x}] = \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[\hat{y}]$; consequently, $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[\bar{x}\,..\,x] = \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[\hat{y}\,..\,\hat{y}+x-\bar{x}]$ because $\hat{\lambda} = \mathsf{L}(\lambda, h)$. Moreover, due to $[y_p\,..\,\bar{y}) \neq \emptyset$, so we must have $[x_p\,..\,\bar{x}) = \emptyset$, i.e., $x_p = \bar{x}$. Define $r = \min\{t \in [p\,..\,q) : y_t - x_t \geq \hat{y} - \bar{x}\}$. Note that $y_p - x_p \leq \hat{y} - \bar{x}$ and $y - x = \bar{y} - \bar{x} > \hat{y} - \bar{x}$; hence, $x_r \in [x_p\,..\,x] = [\bar{x}\,..\,x]$ and $y_r - x_r = \hat{y} - \bar{x}$. Since all breakpoints to the left of (x, y) satisfy the greedy property and since $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[\bar{x}\,..\,x] = \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[\hat{y}\,..\,\hat{y}+x-\bar{x}]$, we conclude that $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x_r\,..\,x] \simeq_{\mathcal{A}_{i-1}} \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y_r\,..\,\hat{y}+x-\bar{x}]$. In particular, $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x] \simeq_{\mathcal{A}_{i-1}} \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[\hat{y}+x-\bar{x}]$, which contradicts $(x, y) \in \mathcal{A}_{i-1}$ by the non-crossing property of $\mathcal{A}_{i-1}$ because $\hat{y} + x - \bar{x} < \bar{y} + x - \bar{x} = y$.

After $2|\mathsf{F}|$ rounds, we construct a tree alignment $\mathcal{A}_{2|\mathsf{F}|} \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$ such that $\mathsf{ted}_{\lambda,\mathcal{A}_{2|\mathsf{F}|}}(F, G) = \mathsf{ted}_{\lambda,\mathcal{A}}(F, G)$ and $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x] \neq \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y]$ holds for every $(x, y) \in \mathsf{B}_{\mathcal{A}_{2|\mathsf{F}|}}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G})) \cap ([0\,..\,2|\mathsf{F}|-1] \times [0\,..\,2|\mathsf{G}|))$. Thus, $\mathcal{A}_{2|\mathsf{F}|} \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$ is an optimum tree alignment and $\mathcal{A}_{2|\mathsf{F}|} \in \mathsf{GA}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}))$. □

4.2 Compatibility Refinement

Definition 4.5 (Compatibility)**.** Let λ be a joint labeling of forests F, G, and let $w \in \mathbb{Z}_{\geq 0}$. We say that nodes $u \in V_{\mathsf{F}}$ and $v \in V_{\mathsf{G}}$ are *w-compatible* if $\lambda(u) = \lambda(v)$, $|o_{\mathsf{F}}(u) - o_{\mathsf{G}}(v)| \leq w$, and $|c_{\mathsf{F}}(u) - c_{\mathsf{G}}(v)| \leq w$.

Definition 4.6 (Compatibilty refinement)**.** Let λ be a joint labeling of forests F, G, and let $w \in \mathbb{Z}_{\geq 0}$. We say that λ' is a *w-compatibility refinement* of λ if, for any $u, v \in V_{\mathsf{F}} \cup V_{\mathsf{G}}$, we have $\lambda'(u) = \lambda'(v)$ if and only if (u, v) belongs to the transitive closure of the w-compatibility relation. We use $\mathsf{C}(\lambda, w)$ to denote a *w-compatibility refinement* of λ (chosen arbitrarily up to equivalence of labelings).

Lemma 4.7. *Given forests F, G of total size n, their joint labeling λ, and an integer w, a w-compatibility refinement labeling $\lambda' = \mathsf{C}(\lambda, w)$ can be constructed in $\mathcal{O}(n \log n)$ time.*

Proof. We process nodes according to the original label. For each label ℓ, we use a BFS-based algorithm to compute connected components with respect to the w-compatibility relation. While doing so, we maintain two instances of a data structure for dynamic orthogonal range reporting [Mor06], one with points $(o_{\mathsf{F}}(u), c_{\mathsf{F}}(u))$ for all unvisited nodes $u \in V_{\mathsf{F}}$ with label ℓ, and one with points $(o_{\mathsf{G}}(v), c_{\mathsf{G}}(v))$ for all unvisited nodes $v \in V_{\mathsf{G}}$ with label ℓ. While we do not have direct access

to edges incident to any vertex, an orthogonal range reporting query allows listing all unvisited neighbors of any vertex. Each listed node is immediately visited (and thus the corresponding point is removed from the range reporting data structure), so, in total, we have $\mathcal{O}(n)$ queries producing output of total size $\mathcal{O}(n)$. Thus, the algorithm works in $\mathcal{O}(n \log n)$ time. □

Observation 4.8. *Consider a joint labeling λ of forests F, G, an integer $w \in \mathbb{Z}_{\geq 0}$, and an alignment $\mathcal{A} \in \mathsf{TA}(\mathsf{F}, \mathsf{G})$ of width at most w. Then, $\mathsf{ted}_{\mathsf{C}(\lambda,w),\mathcal{A}}(\mathsf{F}, \mathsf{G}) = \mathsf{ted}_{\lambda,\mathcal{A}}(\mathsf{F}, \mathsf{G})$.*

5 Periodicity Reduction in Strings

In this section, we analyze the structure of alignments $\mathcal{A} \in \mathsf{GA}_{k,w}(X, Y)$ under the assumption that X, Y avoid certain periodic structure. This is loosely inspired by the proof of [KPS21, Lemma III.10].

Definition 5.1. We say that a pattern P has *s-synchronized occurrences* in strings X, Y if $P = X[x \,..\, x + |P|) = Y[y \,..\, y + |P|)$ holds for some positions x, y satisfying $|x - y| \leq s$.

Definition 5.2. Consider integers $s, e, \ell \in \mathbb{Z}_+$. We say that strings $X, Y \in \Sigma^*$ *avoid s-synchronized e-powers with root at most ℓ* if there is no non-empty string $Q \in \Sigma^{\leq \ell}$ such that Q^e has s-synchronized occurrences in X, Y.

Lemma 5.3. *Consider strings X, Y, integers $e, k, w \in \mathbb{Z}_+$, and two alignments $\mathcal{A}, \mathcal{A}' \in \mathsf{A}_{k,w}(X, Y)$. If X, Y avoid w-synchronized e-powers with root at most $2w$, then $|\mathcal{A} \setminus \mathcal{A}'| \leq 7wke$.*

Proof. Let us partition X into individual characters representing deletions or substitutions of $\mathcal{A}$ or $\mathcal{A}'$ and maximal fragments $X[x \,..\, x+\ell)$ that $\mathcal{A}$ and $\mathcal{A}'$ match perfectly: $X[x \,..\, x+\ell) \simeq_{\mathcal{A}} Y[y \,..\, y+\ell)$ and $X[x \,..\, x+\ell) \simeq_{\mathcal{A}'} Y[y' \,..\, y'+\ell)$ for some $y, y' \in [0 \,..\, |Y| - \ell] \cap [x - w \,..\, x + w]$. Observe that the number of such maximal fragments is at most $2k + 1$ and their total length is at least $|X| - 2k$.

If $y = y'$, then $(x + \delta, y + \delta) \in \mathcal{A} \cap \mathcal{A}'$ holds for all $\delta \in [0 \,..\, \ell]$. Otherwise, $P := X[x \,..\, x + \ell)$ has w-synchronized occurrences in X, Y (starting at positions x, y) and $\mathsf{per}(P) \leq |y - y'| \leq 2w$. Denote $Q := X[x \,..\, x + \mathsf{per}(P))$ and observe that, if $\ell \geq 2we$, then Q^e is a prefix of P and thus also has w-synchronized occurrences in X, Y, contradicting our assumption. Hence, $y \neq y'$ is only possible if $\ell < 2we$. In either case, $X[x \,..\, x + \ell)$ contributes at least $\ell + 1 - 2we$ elements to $\mathcal{A} \cap \mathcal{A}'$.

Overall, the fragments contribute at least $|X| - 2k + (1 - 2we)(2k + 1) \geq |X| - 4wke - 2we + 1$ elements to $\mathcal{A} \cap \mathcal{A}'$. Consequently, $|\mathcal{A} \setminus \mathcal{A}'| \leq |\mathcal{A}| - |\mathcal{A} \cap \mathcal{A}'| \leq |X| + 1 + k - (|X| - 4wke - 2we + 1) = 4wke + 2we + k \leq 7wke$. □

Lemma 5.4 (Compare [KPS21, Lemma III.10])**.** *Consider strings $X, Y \in \Sigma^{\leq n}$ and integers $e, k, w \in \mathbb{Z}_+$. If X, Y avoid w-synchronized e-powers with root at most $2w$, then there exists a set M of size $|M| \geq |X| - 15wk^2e$ such that $M \subseteq \mathsf{M}_{\mathcal{A}}(X, Y)$ holds for every alignment $\mathcal{A} \in \mathsf{GA}_{k,w}(X, Y)$. Moreover, such a set M can be constructed in $\mathcal{O}(n + kw)$ time.*

Proof. Let us fix an alignment $\mathcal{A} \in \mathsf{GA}_{k,w}(X, Y)$ and partition X into individual characters representing deletions or substitutions of $\mathcal{A}$ and maximal fragments $X[x \,..\, x + \ell)$ that $\mathcal{A}$ matches perfectly: $X[x \,..\, x+\ell) \simeq_{\mathcal{A}} Y[y \,..\, y+\ell)$ for some $y \in [0 \,..\, |Y| - \ell]$. Observe that the number of such maximal fragments is at most $k + 1$ and their total length it at least $|X| - k$.

Let us fix a maximal fragment $X[x \,..\, x + \ell) \simeq_{\mathcal{A}} Y[y \,..\, y + \ell)$. We claim that $(x + \delta, y + \delta)$ can be added to M for every $\delta \in [7wke \,..\, \ell)$. For this, consider another alignment $\mathcal{A}' \in \mathsf{GA}_{k,w}(X, Y)$. By Lemma 5.3, we have $|\mathcal{A} \setminus \mathcal{A}'| \leq 7wke$. By the pigeonhole principle, this means that $(x + \delta, y + \delta) \in \mathcal{A}'$ holds for some $\delta \in [0 \,..\, 7wke)$. Then, the greedy nature of $\mathcal{A}'$ guarantees that $\mathcal{A}'$ must

greedily match $X[x+\delta\,..\,x+\ell)$ and $Y[y+\delta\,..\,y+\ell)$ and, in particular, $X[x+7wke\,..\,x+\ell)$ and $Y[y+7wke\,..\,y+\ell)$. Hence, $X[x\,..\,x+\ell)$ contributes at least $\ell-7wke$ pairs to M. The total contribution of all maximal fragments $X[x\,..\,x+\ell)$ is $|X|-k-(k+1)\cdot 7wke \geq |X|-15wk^2e$.

As for the efficient algorithm, we use Lemma 3.3 to build $\mathcal{A}$ (if one exists) in $\mathcal{O}(n+kw)$ time. The remaining steps of the construction above can be easily implemented in $\mathcal{O}(n)$ time. □

6 Horizontal Periodicity Reduction

In this section, we discuss periodic sections of the input forests F and G. Specifically, we look at "horizontal periodicity," which refers to the case when the children of a given node are a periodic sequence of subtrees. If a horizontal period is repeated a large number of times in both F and G, we know that any minimum cost alignment of F and G must align the periodic subtrees in a predictable way. We then argue that we can consider two forests F' and G' that are the exact same as F and G with a bound on the number of repetitions of horizontal periodic subtrees.

For ease of analysis, we do these periodic reductions on the parentheses representations of $\mathsf{P}(F)$ and $\mathsf{P}(G)$ with tree alignments. Furthermore, we borrow some useful definitions and results from [BII+17] on maximal periodic substrings in strings, called runs.

Definition 6.1 (Runs, [BII+17])**.** A *run* in a string S is a triple (i,j,p) such that $S[i\,..\,j)$ is periodic with $\mathsf{per}(S[i\,..\,j))=p$ and maximal in length, i.e., $S[i-1]\neq S[i-1+p]$ and $S[j]\neq S[j-p]$. We call $\frac{j-i}{p}$ the power of the run. Let $\mathsf{Runs}(S)$ denote the entire list of runs in S.

Theorem 6.2 (Theorem 9 of [BII+17])**.** *The number of runs in a string S of length n is less than n.*

Theorem 6.3 (Algorithm 1 of [BII+17])**.** *Given a string S with $n=|S|$, it is possible to compute all runs of S in $\mathcal{O}(n)$ time.*

Using the results of the above theorems, we can find all horizontal periodic substrings of $\mathsf{P}(F)$ and $\mathsf{P}(G)$ in $\mathcal{O}(n)$ time. Ideally, we could just iterate through a list of computed runs and reduce the exponent of each run in constant time. However, some runs may be overlapping, and so by reducing one run, we may significantly change the length of a previously reduced run, or even worse introduce a new run to the string. So, by requiring that the period of runs that we reduce is small, we also can guarantee that the overlap of runs will be manageable.

Lemma 6.4 (Fact 2.2.4, [Koc18])**.** *Any two distinct runs (i_1,j_1,p_1) and (i_2,j_2,p_2) in a string S satisfy $|[i_1\,..\,j_1)\cap[i_2\,..\,j_2)|<p_1+p_2-\gcd(p_1,p_2)$.*

Corollary 6.5. *Consider two distinct runs (i_1,j_1,p_1) and (i_2,j_2,p_2) in a string S with periods $p_1,p_2\leq 4k$ and exponents $\frac{j_1-i_1}{p_1},\frac{j_2-i_2}{p_2}\geq 8k$. If $i_1\leq i_2$, then $i_2-8k<j_1<j_2$.*

Proof. By Lemma 6.4, we have $|[i_1\,..\,j_1)\cap[i_2\,..\,j_2)|<p_1+p_2-\gcd(p_1,p_2)<8k$. Due to $|[i_2\,..\,j_2)|\geq 8kp_2\geq 8k$, this means that $[i_2\,..\,j_2)$ is not contained in $[i_1\,..\,j_1)$, i.e., $j_1<j_2$. Moreover, $j_1-i_2=\min(j_1,j_2)-\max(i_1,i_2)\leq|[i_1\,..\,j_1)\cap[i_2\,..\,j_2)|\leq 8k$. □

Since horizontal periodic reductions take place on subtrees, we require that any periodic substring we reduce has a balanced string period in the parentheses representation. For a string X, we define a balance function $\sigma(X)$ which is mapped to the minimum number of rotations of X needed to make it balanced. If X cannot be balanced, $\sigma(X):=-1$. For example, let $X=$ `)][()(`, then $\sigma(X)=2$ since $\mathsf{rot}^2(X)=$ `[()()]`; let $Y=$ `(((((`, then $\sigma(Y)=-1$ since there is no rotation that balances Y. We give the following lemma for computing the σ function.

Lemma 6.6. *$\sigma(X)$ can be computed in time $\mathcal{O}(|X|)$.*

Proof. To compute $\sigma(X)$, we run a folklore stack matching algorithm for checking balance on X. The algorithm iterates across the characters of X and every opening parenthesis the algorithm comes across gets pushed onto the stack. When the algorithm reaches a closing parenthesis, it pops an opening parenthesis from the stack and matches the two. We modify this basic algorithm so that if the stack is empty when a closing parenthesis is reached, it ignores the failed match and continues to the next character. Additionally, the algorithm will keep track of the largest index $m < |X|$ such that $X[m]$ is a closing parenthesis which had no match due to the stack being empty when this parenthesis was reached. Clearly we must rotate X at least $m+1$ times in order for closing parenthesis $X[m]$ to potentially have an opening parenthesis before it in X to match to. Moreover, since m is the index of the last unmatched closing parenthesis in X then for any index $i > m$ there can only be unmatched opening parenthesis. Therefore, rotating more than $m+1$ times will only place unmatched opening parenthesis at the end of X with no potential matches afterwards. Thus, we rotate exactly $m+1$ times, and run the stack matching algorithm on $\mathsf{rot}^{m+1}(X)$. If the rotated string is balanced and all matching parentheses share the same parenthesis type, then $\sigma(X) = m+1$. Otherwise, $\sigma(X) := -1$. □

Note that the above lemma shows that checking balance and finding the minimum number of rotations to balance a string only takes linear time. Since we only want to find the balance of periodic substrings of length at most $4k$, then computing σ will only take time $\mathcal{O}(k)$.

From the above definition and results, we can now give our algorithm to reduce horizontal periodicity seen below in Algorithms 1, 2, and 3. The goal of Algorithm 3 is to take two input forests F and G and output forests F' and G' which are the same as the input forests except any long horizontal periodic synchronized occurrences are reduced to an exponent of at most $18k$. First, Algorithm 2 uses Algorithm 1 as a subroutine to compute all runs of the parentheses representations of input forests F and G, sorted by starting index. Before the algorithm can actually reduce any pair of runs, it must make sure the runs adhere to strict requirements. First, to make sure the overlap between runs that we want to reduce is not too large compared to the run size, we only reduce runs whose period is at most $4k$ in length and whose exponent is larger than $16k$. Algorithm 1 does exactly this and returns a filtered list of runs that meet these criteria. Afterwards, Algorithm 2 iterates over pairs of runs, one from $\mathsf{P}(F)$ and one from $\mathsf{P}(G)$ in sorted order by their starting index. Note that we only want to reduce periodic substrings who have $2k$-synchronized occurrences; otherwise we cannot align the matching periodic subtrees with less than k edits. If we find a very large run in one input forest with no synchronized occurrence in the other input forest, the algorithm should not reduce these occurrences.

After checking for these properties in each run, Algorithm 2 does one final check that the string period of the run is a balanced parentheses string, i.e., it actually corresponds to a sequence of subtrees in the original forests F and G. If a run's string period is not balanced, we do not actually have horizontal periodicity and cannot reduce these runs as easily (these runs are instead handled in Sections 7 and 8). If a run is not balanced, did not have a small enough period, or did not have a large enough exponent, the run is simply copied over to the corresponding output forest F' or G' without any changes. In the case that Algorithm 2 does find a pair of runs representing $2k$-synchronized occurrences of large horizontal periodicity, a triple representing the pair of runs and their common period and exponent is added to a set S. Then, Algorithm 3 iterates across set S and reduces all found $2k$-synchronized occurrences to only $14k$ repetitions of the string period. Algorithm 3 copies all characters in $\mathsf{P}(F)$ to a new string $\mathsf{P}(F')$ except any substrings contained in $2k$-synchronized occurrences of horizontal periodicity, in which instead the algorithm skips all but

the last $14k$ repetitions of the string period. Once we have the output forests F' and G', it is still necessary to show that reduced runs have not changed the tree edit distance at all nor introduced any new horizontal periodicity. The rest of the section is devoted to the analysis of these three algorithms and their outputs.

Algorithm 1: FilterRuns(S).

```
1  R ← SortByStartingIndex(Runs(S));
2  R' ← empty list;
3  for (i, j, p) ∈ R do
4  |  if p ≤ 4k and (j−i)/p ≥ 16k then
5  |  |  R'.append((i, j, p));
6  return R';
```

Algorithm 2: SyncOccurrences(F, G).

```
1  R_F ← FilterRuns(P(F)), R_G ← FilterRuns(P(G));
2  ℓ_F, ℓ_G ← 0;
3  S ← empty list;
4  while ℓ_F < |R_F| and ℓ_G < |R_G| do
5  |  (i_F, j_F, p_F) ← R_F[ℓ_F], (i_G, j_G, p_G) ← R_G[ℓ_G];
6  |  X ← P(F)[i_F .. i_F + p_F), Y ← P(G)[i_G .. i_G + p_G);
7  |  e ← ⌊ |[i_F .. j_F) ∩ [i_G .. j_G)| / p_F ⌋;
8  |  if e ≥ 16k and ∃α ≤ p_F, rot^α(X) = Y and σ(X) ≥ 0 then
   |  |  // Add synchronized occurrences to set S
9  |  |  S.append((max(i_F, i_G), p_F, e − 2k));
   |  // Go to a new run
10 |  if j_F < j_G then
11 |  |  ℓ_F ← ℓ_F + 1;
12 |  else
13 |  |  ℓ_G ← ℓ_G + 1;
14 return S;
```

We begin by proving that the runtime of Algorithm 2 is linear. Since we know that computing all runs in a string only takes $\mathcal{O}(n)$ time from Theorem 6.3, the call to Algorithm 1 do not cause any issues. As for the main loop of Algorithm 2, either the algorithm finds a reason that a pair of runs do not contain $2k$-synchronized occurrences and moves on to the next run, or the algorithm creates a synchronized occurrence triple and adds it to the output list S. Therefore, we must be careful that all the checks done in each case do not take superlinear total time.

Lemma 6.7. *Given forests F and G of total size n, Algorithm 2 runs in time $\mathcal{O}(n)$.*

Proof. The runtime of Algorithm 2 is mostly affected by the call to Algorithm 1 for sorting and computation of runs in step 1 as well as the number of iterations and work done in each iteration of the **while** loop spanning steps 4–14. First in step 1, by Theorem 6.3, computing all runs of both forests can be done in time $\mathcal{O}(n)$, and there are less than $2n$ runs total by Theorem 6.2. Therefore, sorting all $\mathcal{O}(n)$ runs by their starting index can be done using a radix sort in time

$\mathcal{O}(n)$. The rest of Algorithm 1 filters runs in linear time to make sure the period is small enough and the exponent is large enough. Due to these filtering steps and Corollary 6.5, we can observe that $|R_F|, |R_G| = \mathcal{O}(n/k)$.

We look individually at the work done in the subsequent iterations of the **while** loop. An important observation is that one of the run counters ℓ_F or ℓ_G always increments by 1 (steps 11 and 13) per iteration of the loop. Line 8 involves three checks. First, the overlap of the runs is computed, which takes constant time. Then, we check if a rotation of string period X is equal to a rotation of string period Y. Since $p_F \leq 4k$, checking all rotations takes time $\mathcal{O}(k)$ using a rolling hash function [KR87]. The last check just requires computing $\sigma(X)$, which by Lemma 6.6 takes time $\mathcal{O}(|X|) = \mathcal{O}(k)$. The remaining instructions within each iteration of the **while** loop take $\mathcal{O}(1)$ time, so each iteration costs $\mathcal{O}(k)$ time in total. As mentioned earlier, $|R_F|, |R_G| = \mathcal{O}(n/k)$, and hence the entire algorithm is done in linear time. □

Now, we prove that Algorithm 2 indeed finds $2k$-synchronized occurrences of horizontal periodicity with large exponent in F and G. Any triple (i, p, e) added to set S by Algorithm 2 satisfies these requirements as per the checks in step 8. Therefore, the following proof is fairly straightforward and mostly just formalizes this intuition.

Lemma 6.8. *Given forests F, G, let $S = \mathsf{SyncOccurrences}(F, G)$. Then for any $(i, p, e) \in S$, there must be runs (i_F, j_F, p) in $\mathsf{P}(F)$, (i_G, j_G, p) in $\mathsf{P}(G)$ each containing $2k$-synchronized occurrences of string Q^e in $\mathsf{P}(F)$ and $\mathsf{P}(G)$ where $|Q| = p \leq 4k$, $e \geq 14k$, $i = \max(i_F, i_G)$, and Q is balanced.*

Proof. First, we observe that triples (i, p, e) are only inserted to S in step 9 of Algorithm 2. Let (i_F, j_F, p_F) and (i_G, j_G, p_G) be the runs considered by the algorithm during such an insertion, and note that $p = p_F = p_G \leq 4k$ and $i = \max(i_F, i_G)$. Moreover, from the checks done in step 8, the string periods $X = \mathsf{P}(F)[i_F \,..\, i_F + p)$ and $Y = \mathsf{P}(G)[i_G \,..\, i_G + p)$ are the same up to rotation, i.e., $\mathsf{rot}^\alpha(X) = Y$ for some $\alpha \in [0 \,..\, p)$. Hence, $\mathsf{P}(F)[i_F + \alpha \,..\, i_F + \alpha + p) = \mathsf{P}(G)[i_G \,..\, i_G + p)$. From step 8, we also know that $e + 2k \geq 16k$, which implies that the length of the set of overlapping indices between the two runs has a lower bound of

$$|[i_F \,..\, j_F) \cap [i_G \,..\, j_G)| \geq p_F \left\lfloor \frac{|[i_F \,..\, j_F) \cap [i_G \,..\, j_G)|}{p_F} \right\rfloor = p(e + 2k) \geq 16kp.$$

Without loss of generality, we assume that $i_F \leq i_G$. Then, there exists some $m \in \mathbb{N}$ such that $i_G \leq i'_F = i_F + \alpha + pm \leq i_G + p$ and $\mathsf{P}(F)[i'_F \,..\, i'_F + (e+k)p) = \mathsf{P}(G)[i_G \,..\, i_G + (e+k)p)$. Clearly, we have $2k$-synchronized occurrences of Y^{e+k}, with $e + k \geq 15k$. The check in step 8 guarantees that $\sigma(X) \neq -1$, and by extension, $\sigma(Y) \neq -1$. Recall that $\sigma(Y)$ is the minimum number of rotations needed for Y to be a balanced parentheses string, i.e. $Q := \mathsf{rot}^{\sigma(Y)}(Y) = \mathsf{P}(G)[i_G + \sigma(Y) \,..\, i_G + \sigma(Y) + p)$ is balanced. Since $\sigma(Y) \leq |Y| = p$, we have that $\mathsf{P}(F)[i'_F + \sigma(Y) \,..\, i'_F + \sigma(Y) + ep) = \mathsf{P}(G)[i_G + \sigma(Y) \,..\, i_G + \sigma(Y) + ep)$ are $2k$-synchronized occurrences of Q^e in $\mathsf{P}(F)$ and $\mathsf{P}(G)$ where $|Q| = p \leq 4k$ and $e \geq 14k$. □

Next, we prove the main statement that we need in order to show that the outputted forests F' and G' from Algorithm 3 have the same edit distance as the original forests F and G. Conceptually, the proof just shows that for long synchronized occurrences of a run in both forests, any two minimal cost alignments of F and G must match long segments of these periodic sections together. In fact, we show that given any minimal cost tree alignment $\mathcal{A}$ of F and G, we can build a minimal cost tree alignment of F' and G' that follows $\mathcal{A}$ almost exactly.

Lemma 6.9. *Consider forests F, G such that $\mathsf{P}(F)[\alpha_F \,..\, \beta_F) = \mathsf{P}(G)[\alpha_G \,..\, \beta_G) = Q^e$ for a balanced string Q of length $0 < |Q| \leq 4k$, an integer exponent $e \geq 6k$, and indices $\alpha_F, \alpha_G, \beta_F, \beta_G$ satisfying*

Algorithm 3: SyncReductions(F, G)

```
1  S ← SyncOccurrences(F, G);
2  s_F, s_G ← ε;
3  i ← 0;
4  for (i', p', e') ∈ S do
       // Copy from start of previous synchronized occurrences to start of next
          synchronized occurrences
5      s_F ← s_F · P(F)[i .. i');
6      s_G ← s_G · P(G)[i .. i');
       // Reduce synchronized occurrences to exponent of 14k
7      i ← i' + p'(e' − 14k);
8  s_F ← s_F · P(F)[i .. |P(F)|);
9  s_G ← s_G · P(G)[i .. |P(G)|);
10 return s_F, s_G;
```

$|\alpha_F - \alpha_G| \leq 2k$. *Let* F', G' *be forests such that* $\mathsf{P}(F') = \mathsf{P}(F)[0\,..\,\alpha_F) \cdot Q^{e'} \cdot \mathsf{P}(F)[\beta_F\,..\,|\mathsf{P}(F)|)$ *and* $\mathsf{P}(G') = \mathsf{P}(G)[0\,..\,\alpha_G) \cdot Q^{e'} \cdot \mathsf{P}(G)[\beta_G\,..\,|\mathsf{P}(G)|)$ *for some integer exponent* $e' \geq 6k$. *Then,* $\mathsf{ted}_{\leq k}(F, G) = \mathsf{ted}_{\leq k}(F', G')$.

Proof. We assume without loss of generality that Q is primitive (otherwise, we replace Q by its primitive root) and denote $p := |Q| \leq 4k$. Let $\mathcal{A}$ be an optimal tree alignment such that $\mathsf{ted}(F, G) = \mathsf{ted}_{\mathcal{A}}(F, G) \leq k$.

Claim 6.10. *There exist* $i_F, i_G \in [0\,..\,3k]$ *such that* $(\alpha_F + i_F \cdot p, \alpha_G + i_G \cdot p) \in \mathcal{A}$ *and* $j_F, j_G \in [0\,..\,3k]$ *such that* $(\beta_F - j_F \cdot p, \beta_G - j_G \cdot p) \in \mathcal{A}$.

Proof. Let $(x_F, x_G) \in \mathcal{A}$ be the leftmost element of $\mathcal{A}$ such that $x_F \geq \alpha_F$ and $x_G \geq \alpha_G$. By symmetry between F and G, we assume without loss of generality that $x_F = \alpha_F$. Consider the $k+1$ occurrences of Q starting at positions $\alpha_F + i \cdot p$ for $i \in [0\,..\,k]$. Since Q is balanced, the alignment $\mathcal{A}$ (of cost at most k) matches at least one of them exactly; we can thus define $i_F \in [0\,..\,k]$ so that $\mathcal{A}$ matches $\mathsf{P}(F)[\alpha_F + i_F \cdot p\,..\,\alpha_F + (i_F + 1) \cdot p)$ exactly to some fragment $\mathsf{P}(G)[y_G\,..\,y_G + p)$. Due to $(x_F, x_G) \in \mathcal{A}$, the non-crossing property of $\mathcal{A}$ implies that $y_G \geq \alpha_G$. Moreover, since $\mathsf{ted}_{\mathcal{A}}(F, G) \leq k$ and $\mathsf{P}(F)[\alpha_F + i_F \cdot p] \sim_{\mathcal{A}} \mathsf{P}(G)[y_G]$, we have $y_G \leq (\alpha_F + i_F \cdot p) + 2k \leq \alpha_F + kp + 2k \leq \alpha_G + kp + 4k \leq \alpha_G + 3kp$, where the last inequality follows from $p \geq 2$ (recall that Q is balanced, so its length is even). Furthermore, since Q is primitive (i.e., distinct from all its non-trivial cyclic rotations), we conclude that $y_G = \alpha_G + i_G \cdot p$ for some $i_G \in [0\,..\,3k]$. The second claim is symmetric (with respect to reversal). □

Observe that $\mathsf{P}(F)[\alpha_F + i_F p\,..\,\beta_F - j_F p) = Q^{d_F}$ for $d_F := e - j_F - i_F$ and, symmetrically, $\mathsf{P}(G)[\alpha_G + i_G p\,..\,\beta_G - j_G p) = Q^{d_G}$ for $d_G := e - j_G - i_G$. We denote $d = \min(d_F, d_G)$, observe that $d \geq e - 6k \geq 0$, and construct a tree alignment $\mathcal{A}'$ so that it

- aligns $\mathsf{P}(F)[0\,..\,\alpha_F + i_F p)$ with $\mathsf{P}(G)[0\,..\,\alpha_G + i_G p)$ in the same way as $\mathcal{A}$ does;
- matches $\mathsf{P}(F)[\alpha_F + i_F p\,..\,\alpha_F + (i_F + d)p) = Q^d$ with $\mathsf{P}(G)[\alpha_G + i_G p\,..\,\alpha_G + (i_G + d)p) = Q^d$;
- deletes $\mathsf{P}(F)[\alpha_F + (i_F + d)p\,..\,\beta_F - j_F p) = Q^{d_F - d}$ and $\mathsf{P}(G)[\alpha_G + (i_G + d)p\,..\,\beta_G - j_G p) = Q^{d_G - d}$;
- aligns $\mathsf{P}(F)[\beta_F - j_F p\,..\,|\mathsf{P}(F)|)$ with $\mathsf{P}(G)[\beta_G - j_G p\,..\,|\mathsf{P}(G)|)$ in the same way as $\mathcal{A}$ does.

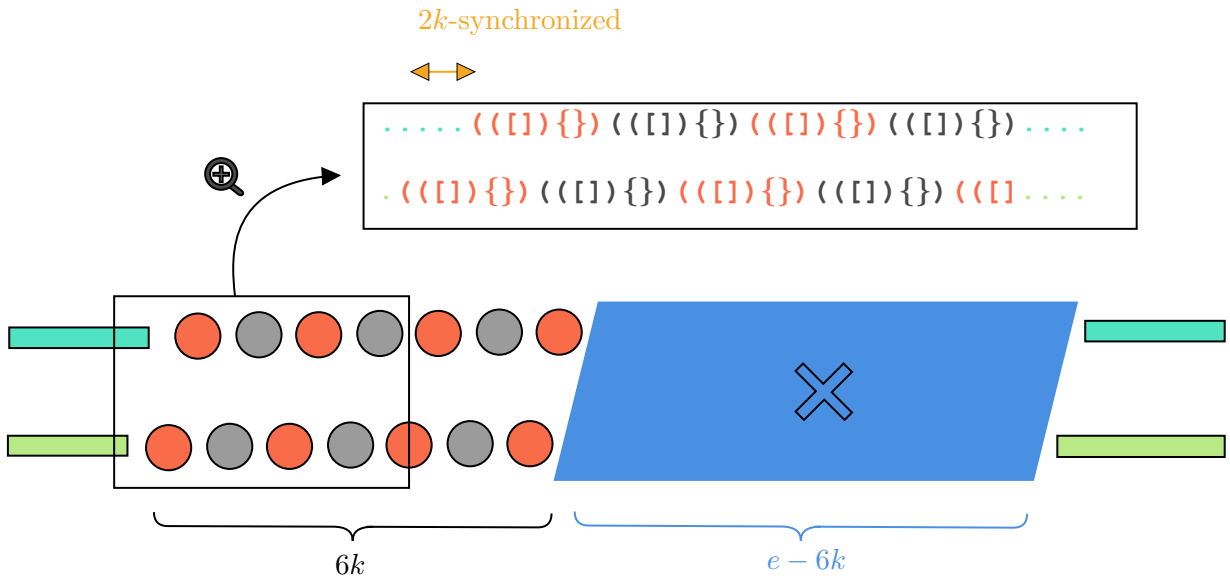


Figure 3: A $2k$-synchronized horizontal periodicity with $|Q| = 8$ is demonstrated. Each gray and red circle correspond to a balanced period block in $\mathsf{P}(F)$ and $\mathsf{P}(G)$ similar to the zoomed portion. According to Lemma 6.9 we can reduce the period exponent to $6k$ by removing the blue part of $\mathsf{P}(F)$ and $\mathsf{P}(G)$ so that the tree edit distance $\mathsf{ted}_{\le k}(F, G)$ remains unchanged.

Note that $\mathcal{A}'$ is a tree alignment: for any node of F, the corresponding parentheses are either both outside $\mathsf{P}(F)[\alpha_F \,..\, \beta_F)$ (and then they are handled as in $\mathcal{A}$) or both contained in a single copy of Q (which is either deleted or matched perfectly to a copy of Q in $\mathsf{P}(G)$). Moreover, the cost of $\mathcal{A}'$ does not exceed the cost of $\mathcal{A}$: the two alignments only differ in how they align Q^{d_F} with Q^{d_G}, and $\mathcal{A}'$ provides an optimum alignment of these fragments.

Now, if the exponent e of $Q^e = \mathsf{P}(F)[\alpha_F \,..\, \beta_F) = \mathsf{P}(G)[\alpha_G \,..\, \beta_G)$ is modified to $e' \ge 6k$, we can interpret this as modifying exponent d of the fragments Q^d matched perfectly by $\mathcal{A}'$ to $d' = d + e' - e \ge 0$. Thus, $\mathcal{A}'$ can be trivially adapted without modifying its cost and hence $\mathsf{ted}(F', G') \le \mathsf{ted}_{\mathcal{A}'}(F, G) = \mathsf{ted}(F, G)$. The converse inequality follows by symmetry between (F, G) and (F', G'). □

From the previous lemma, it is clear that reducing a long run does not affect edit distance. Utilizing this idea, we finally prove that the forests outputted by Algorithm 3 have the same edit distance as the input forests and avoid $2k$-synchronized runs with an exponent more than $14k$ without changing the edit distance.

Definition 6.11 (Synchronized horizontal periodicity)**.** We say that forests F, G *avoid synchronized horizontal k-periodicity* if there is no non-empty balanced string Q of length $|Q| \le 4k$ such that Q^{18k} has $2k$-synchronized occurrences in $\mathsf{P}(F), \mathsf{P}(G)$.

Proposition 6.12 (Avoiding Horizontal k-Periodicity)**.** *There exists an $\mathcal{O}(n)$-time algorithm that, given labeled forests F, G of total size n and an integer $k \in \mathbb{Z}_+$, produces labeled forests F', G' that satisfy $\mathsf{ted}_{\le k}(F, G) = \mathsf{ted}_{\le k}(F', G')$ and, moreover, avoid synchronized horizontal k-periodicity.*

Proof. Let $s_F, s_G = \mathsf{SyncReductions}(F, G)$. By Lemma 6.8, for any triple $(i, p', e') \in S$ of Algorithm 3, there is $2k$-synchronized occurrences in F, G of some $Q^{e'}$ with $|Q| = p$ satisfying the constraints of Lemma 6.9 within runs $r_1 = (i_1, j_1, p)$ and $r_2 = (i_2, j_2, p)$ of F and G, respectively.

In steps 7 and 8, we move the start of the synchronized occurrences forward by $(e' - 14k)p$ and do not copy the skipped over indices to s_F and s_G. This is equivalent to reducing the exponent of r_1 and r_2 by $e' - 14k$ since $\frac{j_1-(i_1+(e'-14k)p)}{p} = \frac{j_1-i_1}{p} - (e' - 14k)$. Furthermore, since the $2k$-synchronized occurrences with exponent e' lies completely within r_1 and r_2, steps 9 and 10 are actually equivalent to reducing the synchronized occurrences of $Q^{e'}$ to Q^{14k} in each forest. Note that since Q is balanced, there exist forests F', G' such that $\mathsf{P}(F') = s_F$ and $\mathsf{P}(G') = s_G$. Therefore, by Lemma 6.9, we have that $\mathsf{ted}_{\leq k}(F, G) = \mathsf{ted}_{\leq k}(F', G')$.

Now, we must show that F', G' avoid synchronized horizontal k-periodicity. Assume for contradiction that there exists $2k$-synchronized occurrences of some Q^{18k} with balanced string period Q such that $|Q| \leq 4k$ in F' and G'. If Q^{18k} was initially found in $2k$-synchronized occurrences in F and G, then since it is periodic, it must be the case that it was contained in runs $(i_F, j_F, p_F), (i_G, j_G, p_G)$, in F and G respectively, with $p_F, p_G \leq |Q| \leq 4k$ and $\frac{j_F-i_F}{p_F}, \frac{j_G-i_G}{p_G} \geq 18k$. Without loss of generality, let $j_F < j_G$. Since Q^{18k} is a $2k$-synchronized occurrence, it must be that the overlap of these runs is at least $18k - 2k = 16k$ and so, $j_F > i_G + 8k$. Let (i'_G, j'_G, p') be the run in R_G preceding (i_G, j_G, p_G). Note that by Corollary 6.5, $j'_G < i_G + 8k < j_F$. Furthermore, we only increment the run counter of the forest whose current run ends before the other forest's current run. Since $j'_G < j_F$ and $j_F < j_G$, we will always reach an iteration of Algorithm 2 that considers the pair of runs $(i_F, j_F, p_F), (i_G, j_G, p_G)$. Since the overlap of the two runs is at least $16k$, Algorithm 2 will therefore add triple $(\max(i_F, i_G), p_F, e - 2k)$ to S for some $e \geq 16k$. Then Algorithm 3 reduces the $2k$-synchronized occurrences containing Q^{16k} to an exponent of at most $14k$ in steps 9 and 10. Therefore, if any $2k$-synchronized occurrences with exponent at least $18k$ and period at most $4k$ is present in F and G, it will be reduced to an exponent of at most $14k$ in F', G'.

Now, if F', G' do not avoid synchronized horizontal k-periodicity it must be the case that by reducing the exponent of overlapping runs, we created new $2k$-synchronized occurrences Q^{18k} where $|Q| \leq 4k$. Clearly, Q^{18k} must lie in some new run $r'_Q = (i'_Q, j'_Q, p'_Q)$ in F' since it is periodic. We will show that this is not possible for such a run to form due to the small overlap between periodic substrings with period at most $4k$. By Corollary 6.5, the overlap between r'_Q and any other reduced runs is at most $8k$. Since r'_Q has length at least $18k$, r'_Q may overlap at most two reduced runs. We refer to the two reduced runs as $r'_1 = (i'_1, j'_1, p'_1)$ and $r'_2 = (i'_2, j'_2, p'_2)$ and without loss of generality assume they are in $\mathsf{P}(F')$. Let $\mathsf{P}(F')[j'_1 - \beta_1 \mathinner{..} j'_1]$ be the overlap between r'_1 and r'_Q, and similarly let $\mathsf{P}(F')[i'_2 \mathinner{..} i'_2 + \beta_2]$ be the overlap between r'_2 and r'_Q. Since r'_1 and r'_2 are reduced, there are two corresponding runs $r_1 = (i_1, j_1, p_1), r_2 = (i_1, j_2, p_1)$ in $\mathsf{P}(F)$, such that $\mathsf{P}(F)[j_1 - \beta_1 \mathinner{..} j_1) = \mathsf{P}(F')[j'_1 - \beta_1 \mathinner{..} j'_1)$ and $\mathsf{P}(F)[i_2 \mathinner{..} i_2 + \beta_2) = \mathsf{P}(F')[i'_2 \mathinner{..} i'_2 + \beta_2)$. Note that since we do not change any characters between any runs, we also know that the middle substrings $\mathsf{P}(F')[j'_1 \mathinner{..} i'_2] = \mathsf{P}(F)[j_1 \mathinner{..} i_2]$ are equal as well. Combining these three substrings we have that $\mathsf{P}(F')[i'_Q \mathinner{..} j'_Q) = \mathsf{P}(F)[j_1 - \beta_1 \mathinner{..} i_2 + \beta_2)$, which implies that Q^{18k} is a periodic substring of $\mathsf{P}(F)$. In other words, Q^{18k} is contained in a run in $\mathsf{P}(F)$ before any reductions occur, which is a contradiction.

Finally, we discuss the runtime of Algorithm 3. First Algorithm 3 calls Algorithm 2 in step 1, and by Lemma 6.7, this takes time $\mathcal{O}(n)$. Now, we consider the loop in steps 4–7. We copy substrings from $\mathsf{P}(F)$ and $\mathsf{P}(G)$ of the start of one pair of synchronized occurrences to the start of the next pair of synchronized occurrences. Note that synchronized occurrences we copy have periods at most $4k$ and exponents at least $14k$, and so by Corollary 6.5, we know that no two synchronized occurrences will start at the same index. Therefore, we only copy each character of $\mathsf{P}(F)$ and $\mathsf{P}(G)$ at most once across the entire algorithm, and so Algorithm 3 takes time $\mathcal{O}(n + |\mathsf{P}(F)| + |\mathsf{P}(G)|) = \mathcal{O}(n)$. □

7 Vertical Periodicity Reduction

In order to remove periodicity from regions of $\mathsf{P}(F), \mathsf{P}(G)$ which may be unbalanced, we consider a second type of periodicity, *vertical* periodicity, in addition to horizontal periodicity. Avoiding horizontal periodicity allows us to reduce large powers of repeated balanced substrings; in this section, we essentially aim to reduce large powers of pairs of periodic substrings which are balanced together but may be unbalanced separately. We do so by finding paths of nodes in forests F and G such that the children to the left and right of the path of each node in the path are the same, which we call vertical periodicity. At the bottom of the path, we may no longer have vertical periodicity, and so in a parentheses representation of the forest we will have two separate periodic substrings, one to the left of the path and one to the right of the path. We define some useful notation for vertical periodicity as follows:

Definition 7.1 (Context)**.** We define a *context* as a pair $C = (C_L, C_R)$ such that $C_L \cdot C_R = \mathsf{P}(T)$ for some labeled forest T.

Definition 7.2 (Vertical composition)**.** For a context $C = (C_L, C_R)$ and a labeled forest F, we denote by $C\langle F\rangle$ the labeled forest H such that $\mathsf{P}(H) = C_L \cdot \mathsf{P}(F) \cdot C_R$. Similarly, for two contexts $C = (C_L, C_R)$ and $D = (D_L, D_R)$, we denote $C\langle D\rangle = (C_L D_L, D_R C_R)$.

Observe that the vertical composition of contexts is associative. For a context C and an integer $e \in \mathbb{Z}_+$, we define C^e as the context obtained by vertical composition of e copies of C. We say a forest context C *occurs* at node u of a labeled forest F if the subtree of F rooted at node u is of the form $C\langle H\rangle$ for some labeled forest H.

We say that context C has s-synchronized occurrences in labeled forests F, G if C occurs at a node u of F and at a node v of G such that $|o_F(u) - o_G(v)| \leq s$ and $|c_F(u) - c_G(v)| \leq s$.

Definition 7.3 (Synchronized vertical periodicity)**.** We say that forests F, G *avoid synchronized vertical k-periodicity* if there is no context $C = (C_L, C_R)$ with $|C_L|, |C_R| \leq 4k$ such that C^{16k} has $2k$-synchronized occurrences in F, G.

To avoid synchronized vertical k-periodicity, we first compute periodic contexts which occur in forests F and G individually without concern for synchronicity. Note that if a forest has such a periodic context $C^{16k} = (C_L, C_R)^{16k}$, then that forest has two separate periodic substrings we want to identify, namely C_L^{16k} and C_R^{16k}. Additionally, from the definition of context it is clear that while C_L and C_R do not need to be balanced parentheses strings, $C_L \cdot C_R$ does have to be balanced. Therefore, C_L cannot begin with a closing parenthesis and C_R cannot end with an opening parenthesis since such parentheses would have no match in $C_L \cdot C_R$. For this reason, we will want to find periodic substrings starting at opening parentheses in $\mathsf{P}(F), \mathsf{P}(G)$ as well as periodic substrings ending at closing parentheses

For a node u in a forest F, consider $q_L \in [1\,..\,4k]$ and $e \in \mathbb{Q}$ such that q_L is a period of $\mathsf{P}(F)[o(u)\,..\,o(u) + q_L \cdot e)$ and e is maximized. In other words, we want to find the longest periodic substring starting at $o(u)$ with period at most $4k$ and exponent at least $16k$. If $e \geq 16k$, we may have vertical periodicity that we want to avoid, and so we define an array Q_F to store these values. Let $Q_F[o(u)] := (q_L, o(u) + q_L e)$. If $e < 16k$, we do not need to worry about reducing any substring starting at $o(u)$ and so, we set a default value $Q_F[o(u)] := (1, o(u))$. Since we want to find periodic substrings ending in closing parentheses as well, we define $Q_F[c(u)] := (q_R, i)$ where $\mathsf{P}(F)(c(u) - i\,..\,c(u)]$ is the longest periodic substring ending at $c(u)$ with a period $q_R \leq 4k$ and exponent at least $16k$. Again if the exponent is less than $16k$, we define $Q_F[c(u)] := (1, c(u))$.

Lemma 7.4. *Given a forest F, Q_F can be computed in time $\mathcal{O}(n)$.*

Proof. Initially, we set $Q_F[\ell] := (1, \ell)$ as the default value for all $\ell \in [0\,..\,2|F|)$ and use Algorithm 1 to compute the set R_F of runs (i, j, p) of $\mathsf{P}(F)$ with period $p \leq 4k$ and exponent $\frac{j-i}{p} \geq 16k$; as discussed in Section 6, this costs $\mathcal{O}(n)$ time. For every run $(i, j, p) \in R_F$, take any index $\ell \in [i\,..\,j)$ such that $\mathsf{P}(F)[\ell]$ is an opening parenthesis. If the exponent of the substring $\mathsf{P}(F)[\ell\,..\,j)$, that is, $\frac{j-\ell}{p}$, is at least $16k$, we update $Q_F[\ell]$ to (p, j). By iterating over all indices in each run, we will find the longest periodic substring starting at every opening parenthesis in Q_F with exponent at least $16k$. Note that any two runs with period at most $4k$ has less than $8k$ overlapping characters by Corollary 6.5, and therefore, we only consider each index ℓ twice. Furthermore, we update each value $Q_F[\ell]$ at most once because the indices in the overlap of the runs cannot have an exponent of at least $16k$ in each run (otherwise the overlap would exceed $8k$).

To finish the computation of Q_F, we do the analogous steps for closing parentheses. For every run $(i, j, p) \in R_F$, we take any index $\ell \in [i\,..\,j)$. If the exponent of the substring $\mathsf{P}(F)[i\,..\,\ell] = \mathsf{P}(F)(i-1\,..\,\ell]$, that is, $\frac{\ell-(i-1)}{p}$, is at least $16k$, we update $Q_F[\ell]$ to $(p, i-1)$.Now, note that by Corollary 6.5, runs with period at most $4k$ can only overlap in at most $8k$ indices. Furthermore, since we only update Q_F if the exponent of a run is at least $16k = 2(8k)$, any index $\ell \leq |\mathsf{P}(F)|$ can be contained in at most two runs of period at most $4k$ and exponent at least $16k$. Therefore, iterating through all such runs of $\mathsf{P}(F)$ and computing Q_F for each index takes time $\mathcal{O}(n)$. □

Algorithm 4: ComputeQ(F).

```
1 R_F ← FilterRuns(P(F));
2 Q_F[ℓ] ← (1, ℓ)   ∀ℓ ∈ [|P(F)|];
3 for (i, j, p) ∈ R_F do
4 |   for ℓ ∈ [i..j) do
5 |   |   if P(F)[ℓ] is an opening parenthesis and (j−ℓ)/p ≥ 16k then
6 |   |   |   Q_F[ℓ] ← (p, j);
7 |   |   if P(F)[ℓ] is a closing parenthesis and (ℓ−(i−1))/p ≥ 16k then
8 |   |   |   Q_F[ℓ] ← (p, i−1);
```

We give Algorithm 4 for pseudocode on computing Q_F from forest F. Now that we have computed Q_F, we can iterate over all nodes of forest F to compute the maximal periodic context that occur at each node $u \in F$. Let $Q_F[o(u)] = (q_L, j_L)$ and $Q_F[c(u)] = (q_R, j_R)$, and denote the string periods by $P_L = \mathsf{P}(F)[o(u)\,..\,o(u)+q_L)$ and $P_R = \mathsf{P}(F)(c(u)-q_R\,..\,c(u)]$. Assume that there is a periodic context C^e that occurs at u such that $C = (C_L, C_R)$, e is maximized, and the subtree rooted at u is of form $C^e\langle H\rangle$. This means that for two positive integers $r_L, r_R \in \mathbb{Z}_+$, $|C_L| = r_L q_L$ and $|C_R| = r_R q_R$ since C_L and C_R could be periodic themselves. To find the correct coefficients r_L and r_R, we need to compute the number of unmatched opening parentheses in P_L and the number of unmatched closing parentheses in P_R, denoted by d_L and d_R respectively. Values d_L and d_R correspond to the depth that P_L and P_R go down in the tree. The highest node in which the runs starting at $o(u)$ and ending at $c(u)$ synchronize is located at depth $d := \mathrm{lcm}(d_L, d_R)$. By synchronizing we mean that the unmatched opening and closing parentheses are matched so that $C_L \cdot C_R$ is balanced. Hence, $C_L = P_L^{d/d_L}$ and $C_R = P_R^{d/d_R}$. After this step we need to check whether the conditions $|C_L| \leq 4k$, $|C_R| \leq 4k$, and $e \geq 16k$ still hold.

The next step is to find the maximum exponent e such that C^e occurs at node u. There is no guarantee that $\mathsf{P}(F)[j_L\,..\,j_R]$ forms a balanced substring. For example, the left run could finish earlier than the right run, or the two runs could *diverge* as illustrated in Figure 4. To fix this issue, we first find the vertices v_L and v_R corresponding to indices j_L and j_R in $\mathsf{P}(F)$ respectively. If v^*

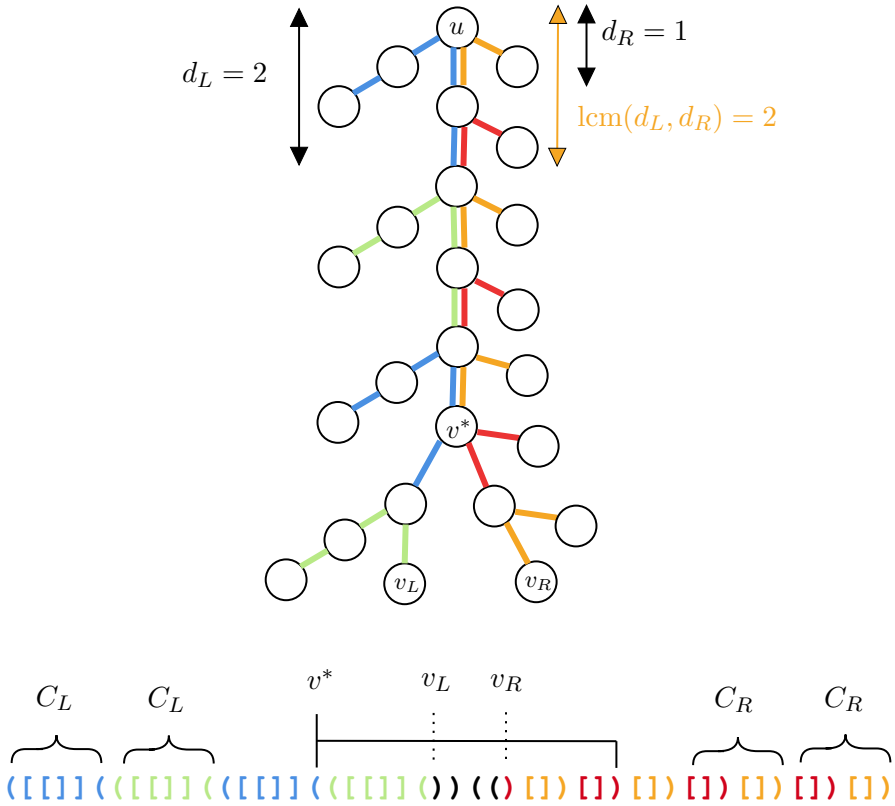


Figure 4: An example tree in which the left run starting at $o(u)$ and the right run ending at $c(u)$ diverge. The depth of the runs are $d_L = 2$ and $d_R = 1$. Therefore, the depth of the context is $\mathrm{lcm}(d_L, d_R) = 2$, and the left and right runs end at v_L and v_R respectively. In this case, the LCA v^* of v_L and v_R is at distance 5 from u, which indicates the exponent $e = \lfloor \frac{5+1}{2} \rfloor = 3$ of context C^e.

is the *Lowest Common Ancestor* (LCA) of v_L and v_R, then

$$e = \left\lfloor \min \left\{ \frac{j_L - o(u)}{|C_L|}, \frac{c(u) - j_R}{|C_R|}, \frac{c(u) - o(u) + 1}{|C_L| + |C_R|}, \frac{D[o(v^*)] - D[o(u)] + 1}{d} \right\} \right\rfloor$$

where the value $D[o(v)] = D[c(v)]$ is the depth of node v (i.e., the distance to the root of the corresponding tree). The arguments of the min function indicate that the left run is long enough, the right run is long enough, the subtree of u is large enough (which is needed if the left run overlaps the right run), and that the runs do not diverge too early. Again, we need to check whether $e \geq 16k$.

Definition 7.5. Given a forest F, let $\mathcal{C}(F)$ be a set of quadruples (u, q_L, q_R, e) such that

1. u is a node in F
2. $q_L, q_R \leq 4k$, $e \geq 16k$.
3. $C_L = \mathsf{P}(F)[o(u) \,..\, o(u) + q_L), C_R = \mathsf{P}(F)(c(u) - q_R \,..\, c(u)]$ form context $C = (C_L, C_R)$ in the subtree rooted at u.
4. C^e is a maximal context, i.e., C^{e+1} does not occur at u.

Lemma 7.6. *$\mathcal{C}(F)$ can be computed in $\mathcal{O}(n \log n)$ time.*

Proof. Given forest F, we showed that we can find Q_F in $\mathcal{O}(n)$ time in Lemma 7.4. In the rest of the algorithm, we iterate over the nodes of F. For each node $u \in V_F$, we check $Q_F[o(u)]$ and

Algorithm 5: ComputeContexts(F, D).

```
1  Q_F ← ComputeQ(P(F));
2  C ← empty set;
3  for u ∈ V_F do
4  |  (q_L, j_L) ← Q_F[o(u)], (q_R, j_R) ← Q_F[c(u)];
   |  // Check for dummy Q_F values and make sure o(u) and c(u) are not in the
   |     same period
5  |  if j_L ≠ o(u) and j_R ≠ c(u) and (c(u) − o(u)) ≥ max{q_L, q_R} then
6  |  |  d_L ← D[o(u) + q_L] − D[o(u)];
7  |  |  d_R ← D[c(u) − q_R] − D[c(u)];
8  |  |  d ← lcm(d_L, d_R);
9  |  |  C_L ← P(F)[o(u) .. o(u) + q_L)^{d/d_L};
10 |  |  C_R ← P(F)(c(u) − q_R .. c(u)]^{d/d_R};
11 |  |  if |C_L| > 4k or |C_R| > 4k then
12 |  |  |  continue;
   |  |  // Let v_L, v_R be the nodes corresponding to j_L, j_R
13 |  |  v* ← LCA(v_L, v_R);
14 |  |  e ← ⌊min{(j_L − o(u))/|C_L|, (c(u) − j_R)/|C_R|, (c(u) − o(u) + 1)/(|C_L| + |C_R|), (D[o(v*)] − D[o(u)] + 1)/d}⌋;
15 |  |  if e ≥ 16k then
16 |  |  |  C.insert((u, |C_L|, |C_R|, e));
17 return C;
```

$Q_F[c(u)]$ to find a potential periodic context rooted at vertex u. If the desired context exists with period at most $4k$ and exponent at least $16k$, it satisfies the required conditions to enter $\mathcal{C}(F)$.

In order to find the period of a maximal context, we first define array $D[0\,..\,2|F|)$ so that, for each node $u \in V_F$, the value $D[o(u)] = D[c(u)]$ is the depth of node u (i.e., the distance to the root of the corresponding tree). Recall that d_L is the number of unmatched opening parentheses in P_L, the shortest period starting from $o(u)$ which is stored in array Q_F, and similarly d_R is the number of the number of unmatched closing parenthesis in P_R. The shortest period of a maximal context is equal to $\mathrm{lcm}(d_L, d_R)$, which can be found in $\mathcal{O}(\log k)$ time.

$$d_L = D[o(u) + q_L] - D[o(u)], \tag{1}$$
$$d_R = D[c(u) - q_R] - D[c(u)]. \tag{2}$$

To find the correct exponent, we query the LCA of two nodes in the tree which indicate the end of $Q_F[o(u)]$ and $Q_F[c(u)]$; this operation takes $\mathcal{O}(1)$ time after $\mathcal{O}(n)$-time preprocessing of the forest F [HT84, BFP+05]. Once we have the LCA v^*, we compute the exponent in $\mathcal{O}(1)$ time using the D array. □

Once we compute the set of contexts of each node in forests F and G, we next find which contexts have $2k$-synchronized occurrences that we should reduce. We can use 2-D range queries to find such synchronized occurrences since we need to check both if the opening parentheses indices and closing parenthesis indices are within $2k$ of each other.

Definition 7.7. Given a set of points S and a query rectangle Q in the plane, *orthogonal range successor* (ORS) is the problem of finding the point with smallest y-coordinate in $S \cap Q$. Given a quadruple $(u, q_L, q_R, e) \in \mathcal{C}(F)$ with context $C = (\mathsf{P}(F)[o(u)\,..\,o(u) + q_L), \mathsf{P}(F)(c(u) - q_R\,..\,c(u)])$

we let $\mathsf{ORS}_{\mathcal{C}}(u)$ denote an ORS query which returns a node $v \in G$ such that $(v, q_L, q_R, e') \in \mathcal{C}(G)$, $(\mathsf{P}(G)[o(v)\,..\,o(v)+q_L), \mathsf{P}(G)(c(v)-q_R\,..\,c(v)]) = C$ and $|o(u)-o(v)| \leq 2k, |c(u)-c(v)| \leq 2k$. If no such query exists, we return $(0,0,0,0)$.

Theorem 7.8 (Linear time ORS queries [GHN20])**.** *Given a set of n points in the plane, a data structure can be computed in time $\mathcal{O}(n\sqrt{\log n})$ to answer* ORS *queries in $\mathcal{O}(\lg\lg n)$ time.*

Definition 7.9. Given a string s, a *fingerprint hash* of s is a function $h_s : s \to [|s|^3]$ such that $h_s(s[i_1\,..\,j_1]) = h_s(s[i_2\,..\,j_2])$ if and only if $s[i_1\,..\,j_1] = s[i_2\,..\,j_2]$.

Theorem 7.10 (Linear time fingerprint hash [Gaw11])**.** *Given a string s with $n = |s|$, after $\mathcal{O}(n)$ preprocessing time, fingerprint hash values can be computed in constant time for substrings of s.*

Lemma 7.11. *Given forests F, G and quadruple $(u, q_L, q_R, e) \in \mathcal{C}(F)$ with context $C = (\mathsf{P}(F)[o(u)\,..\,o(u)+qL), \mathsf{P}(F)(c(u)-q_R\,..\,c(u)])$, $\mathsf{ORS}_{\mathcal{C}}(u) \in G$ can be computed in $\mathcal{O}(\lg\lg n)$ time with $\mathcal{O}(n\sqrt{\lg n})$ preprocessing.*

Proof. We will build a separate ORS data structure $\mathcal{D}_C$ for each context $C = (C_L, C_R)$ that corresponds to a quadruple of $\mathcal{C}(F)$. Any node $v \in G$ will be contained in data structure $\mathcal{D}_C$ if $(v, q_L, q_R, e) \in \mathcal{C}(G)$ corresponds has context C. To determine which point belongs to which data structure, we first preprocess $\mathsf{P}(F), \mathsf{P}(G)$ and compute a fingerprint hash $h_{F,G}$ in $\mathcal{O}(n)$ time by Theorem 7.10. Then, given a quadruple $(v, q_L, q_R, e) \in \mathcal{C}(G)$, determining which data structure v belongs to can be done in constant time using $h_{F,G}$. Since there is only one quadruple per node of F and G in $\mathcal{C}(F), \mathcal{C}(G)$ and n nodes total per forest, we can construct all data structures of contexts in $\mathcal{C}(F)$ in time $\mathcal{O}(n\sqrt{\lg n})$ according to Theorem 7.8. Input queries to these data structures will be the $2k \times 2k$ rectangle surrounding a point $(o(u), c(u))$ such that $(u, q_L, q_R, e) \in \mathcal{C}(F)$ and outputs will be quadruples $(v, q_L, q_R, e') \in \mathcal{C}(G)$ where $(o(v), c(v))$ is the orthogonal range successor of $(o(u), c(u))$ in $\mathcal{D}_C$. Again, by Theorem 7.8, these queries can be answered in time $\mathcal{O}(\lg\lg n)$. □

Algorithm 6: VertPeriods(F, G)

```
1  i ← −1;
2  S ← ∅;
3  C(F) ← SortByStartingIndex(C(F));
4  for (u_F, q_L^F, q_R^F, e_F) ∈ C(F) do
5      if o(u_F) > i then
6          C ← (P(F)[o(u_F) .. o(u_F) + q_L^F), P(F)(c(u_F) − q_R^F .. c(u_F)]);
7          (v, q_L^F, q_R^G, e_G) ← ORS_C(u);
8          if e_G ≠ 0 then
9              e' ← min(e_F, e_G);
10             S ← S ∪ (u_F, u_G, q_L^F, q_R^F, e');
11             i ← o(u) + (e' − 8k) q_L^F;
12 return S;
```

Algorithm 6 and Algorithm 7 identify and output forests F', G' with reduced vertical periodicity. Algorithm 6 simply iterates through all nodes u of forest F that are the beginning of a vertical period with power at least $16k$. Then using ORS queries, we find any nodes v in G with a matching context to that of u and add the $2k$-synchronized occurrence to set S to be reduced in Algorithm 7. Algorithm 7 is fairly straightforward and simply outputs $\mathsf{P}(F')$ and $\mathsf{P}(G')$ by copying all characters

of $\mathsf{P}(F), \mathsf{P}(G)$ and skipping all but $14k$ repetitions of any $2k$-synchronized vertical periods of set S. We now show that these algorithms do correctly identify vertical periods and that reducing vertical period powers does not change tree edit distance. Most of the following proofs mimic the same structures as Section 6 with more details since each context $C^e = (C_L, C_R)^e$ has a left C_L^e and right C_R^e part we must consider rather than a single substring Q^e.

Lemma 7.12. *Given forests F, G and $S = \mathsf{VertPeriods}(F, G)$, then for any $(u, v, q_L, q_R, e) \in S$ there is $2k$-synchronized occurrences of C^e at nodes u in F, v in G where $C = (\mathsf{P}(F)[o(u)\,..\,o(u) + q_L), \mathsf{P}(F)(c(u) - q_R\,..\,c(u)])$ and $e \geq 16k$.*

Proof. The proof of this lemma is fairly straightforward from the steps of Algorithm 6. First, note that for any quintuple (u, v, q_L, q_R, e') added to S in step 10, there must be some quadruple $(u, q_L, q_R, e_F) \in \mathcal{C}(F)$ where by definition of $\mathcal{C}(F)$, $e_F \geq 16k$ and C^{e_F} occurs at node u where context $C = (\mathsf{P}(F)[o(u)\,..\,o(u) + q_L), \mathsf{P}(F)(c(u) - q_R\,..\,c(u)])$. Furthermore, by definition of $\mathsf{ORS}_C(u)$ any quadruple (v, q_L^G, q_R^G, e_G) returned in step 7 must also have an occurrence of C^{e_G} at node v where $|o(u) - o(v)| \leq 2k, |c(u) - c(v)| \leq 2k$. Moreover, (v, q_L, q_R, e_G) must be in $\mathcal{C}(G)$, and so, $e_G \geq 16k$. So, since in step 9 we set $e' = \min(e_F, e_G)$, we have that $e' \geq 16k$ and there is a $2k$-synchronized occurrence of C^e at nodes u in F and v in G. □

Algorithm 7: $\mathsf{VertSyncReductions}(F, G)$

```
1  S ← VertPeriods(F, G);
2  S' ← ∅;
3  for (u_F, u_G, q_L, q_R, e) ∈ S do
4  |  S' ← S' ∪ {(o(u_F), o(u_G), q_L, e), (c(u_F) − q_R · e + 1, c(u_G) − q_R · e + 1, q_R, e)} ;
5  S' ← SortByStartingIndex(S');
6  s_F, s_G ← ε;
7  i_F, i_G ← 0;
8  for (ℓ_F, ℓ_G, q, e) ∈ S' do
9  |  s_F ← s_F · P(F)[i_F .. ℓ_F);
10 |  s_G ← s_G · P(G)[i_G .. ℓ_G);
11 |  i_F ← ℓ_F + q(e − 14k);
12 |  i_G ← ℓ_G + q(e − 14k);
13 s_F ← s_F · P(F)[i_F .. |P(F)|) ;
14 s_G ← s_G · P(G)[i_G .. |P(G)|) ;
15 return s_F, s_G;
```

Lemma 7.13. *Consider forests F, G and a context $C = (C_L, C_R)$ such that $0 < |C_L|, |C_R| \leq 4k$ and, for some integer exponent $e \geq 10k$, the context C^e has $2k$-synchronized occurrences in F, G at nodes u and v, respectively. This means that*

$$\mathsf{P}(F) = \mathsf{P}(F)[0\,..\,o_F(u)) \cdot C_L^e \cdot \mathsf{P}(F)[o_F(u) + e|C_L|\,..\,c_F(u) - e|C_R|] \cdot C_R^e \cdot \mathsf{P}(F)(c_F(u)\,..\,|\mathsf{P}(F)|),$$
$$\mathsf{P}(G) = \mathsf{P}(G)[0\,..\,o_G(v)) \cdot C_L^e \cdot \mathsf{P}(G)[o_G(v) + e|C_L|\,..\,c_G(v) - e|C_R|] \cdot C_R^e \cdot \mathsf{P}(G)(c_G(v)\,..\,|\mathsf{P}(G)|).$$

We define F', G' so that the following holds for some exponent $e' \geq 10k$:

$$\mathsf{P}(F') = \mathsf{P}(F)[0\,..\,o_F(u)) \cdot C_L^{e'} \cdot \mathsf{P}(F)[o_F(u) + e|C_L|\,..\,c_F(u) - e|C_R|] \cdot C_R^{e'} \cdot \mathsf{P}(F)(c_F(u)\,..\,|\mathsf{P}(F)|),$$
$$\mathsf{P}(G') = \mathsf{P}(G)[0\,..\,o_G(v)) \cdot C_L^{e'} \cdot \mathsf{P}(G)[o_G(v) + e|C_L|\,..\,c_G(v) - e|C_R|] \cdot C_R^{e'} \cdot \mathsf{P}(G)(c_G(v)\,..\,|\mathsf{P}(G)|).$$

Then, $\mathsf{ted}_{\leq k}(F, G) = \mathsf{ted}_{\leq k}(F', G')$.

Proof. We assume without loss of generality that C is primitive (if C can be expressed as an integer power of a smaller context, we should consider that context instead) and denote $q_L := |C_L| \leq 4k$ and $q_R := |C_R| \leq 4k$. For $i \in [0\,..\,e)$, let u_i be the node of F with $o_F(u_i) = o_F(u) + iq_L$ (and $c_F(u_i) = c_F(u) - iq_R$) and let v_i be the node of G with $o_G(v_i) = o_G(v) + iq_L$ (and $c_G(v_i) = c_G(v) - iq_R$). Moreover, let $\mathcal{A}$ be an optimal tree alignment such that $\mathsf{ted}(F,G) = \mathsf{ted}_{\mathcal{A}}(F,G) \leq k$.

Claim 7.14. *There exist $i_F, i_G \in [0\,..\,5k]$ such that*

$$(o_F(u) + i_F \cdot q_L, o_G(v) + i_G \cdot q_L), (1 + c_F(u) - i_F \cdot q_R, 1 + c_G(v) - i_G \cdot q_R) \in \mathcal{A}.$$

Moreover, there exist $j_F, j_G \in [0\,..\,5k]$ such that

$$(o_F(u) + (e - j_F)q_L, o_G(v) + (e - j_G)q_L), (1 + c_F(u) - (e - j_F)q_R, 1 + c_G(v) - (e - j_G)q_R) \in \mathcal{A}.$$

Proof. Let $(x_F, x_G) \in \mathcal{A}$ be the leftmost element of $\mathcal{A}$ such that $x_F \geq o_F(u)$ and $x_G \geq o_G(v)$. By symmetry between F and G, we may assume without loss of generality that $x_F = o_F(u)$. The context C occurs at each of the nodes $u_0, \ldots, u_k$ and, since the occurrences are disjoint, the alignment $\mathcal{A}$ (of cost at most k) must match one of these occurrences perfectly. We pick the index $i_F \in [0\,..\,k]$ of one such perfectly matched occurrence and denote the node matched to u_{i_F} by w. In particular, $\mathsf{P}(F)[o_F(u_{i_F})\,..\,o_F(u_{i_F}) + q_L) \simeq_{\mathcal{A}} \mathsf{P}(G)[o_G(w)\,..\,o_G(w) + q_L)$ and $\mathsf{P}(F)(c_F(u_{i_F}) - q_R\,..\,c_F(u_{i_F})] \simeq_{\mathcal{A}} \mathsf{P}(G)(c_G(w) - q_R\,..\,c_G(w)]$. Since $(x_F, x_G) \in \mathcal{A}$, we must have $o_G(w) \geq o_G(v)$ by the non-crossing property of $\mathcal{A}$. At the same time, $o_G(w) \leq o_F(u_{i_F}) + 2k \leq o_F(u) + kq_L + 2k \leq o_G(v) + kq_L + 4k \leq o_G(v) + 5kq_L$. Similarly, $c_G(w) \geq c_G(v) - 5kq_R$, which also implies $c_G(w) \leq c_G(v)$.

Our next goal is to show that $w = v_{i_G}$ for some $i_G \in [0\,..\,5k]$. For a proof by contradiction, suppose that $o_G(v_i) < o_G(w) < o_G(v_{i+1})$ for some $i \in [0\,..\,5k)$. Due to $c_G(w) > c_G(v) - 5kq_R$, this also implies that $c_G(v_i) > c_G(w) > c_G(v_{i+1})$, i.e., that w is a node on the path between v_i and v_{i+1}. Suppose that the length of this path is ℓ and the node w is at distance ℓ' from v_i. Hence, $\mathsf{P}(G)[o_G(v_i)\,..\,o_G(w))$ has ℓ' unmatched opening parentheses out of the ℓ unmatched opening parentheses in C_L. Moreover, $\mathsf{P}(G)[o_G(v_i)\,..\,o_G(w)) \cdot \mathsf{P}(G)[o_G(w)\,..\,o_G(v_{i+1})) = C_L = \mathsf{P}(G)[o_G(w)\,..\,o_G(v_{i+1})) \cdot \mathsf{P}(G)[o_G(v_i)\,..\,o_G(w))$, and thus there is a primitive string Q_L such that $\mathsf{P}(G)[o_G(w)\,..\,o_G(v_{i+1}))$ and $\mathsf{P}(G)[o_G(v_i)\,..\,o_G(w))$ are both powers of Q_L. The number of unmatched opening parentheses is Q_L must be a common divisor of ℓ and ℓ', i.e., C_L can be expressed as a string power with exponent $\ell/\gcd(\ell,\ell')$. A symmetric argument shows that C_R can be expressed as a string power with exponent $\ell/\gcd(\ell,\ell')$. Overall, we conclude that C can be expressed as a context power with exponent $\ell/\gcd(\ell,\ell')$, contradicting the primitivity of C. Hence, $w = v_{i_G}$ for some $i_G \in [0\,..\,5k]$ holds as claimed and, in particular, $o_G(w) = o_G(v) + i_G q_L$ and $c_G(w) = c_G(v) - i_G q_R$.

The proof of the second part of the claim is analogous. □

Observe that $\mathsf{P}(F)[o_F(u)+i_F q_L\,..\,o_F(u)+(e-j_F)q_L) = C_L^{d_F}$ and $\mathsf{P}(F)(c_F(u)-(e-j_F)q_R\,..\,c_F(u)-i_F q_L] = C_R^{d_F}$ for $d_F := e - j_F - i_F$. Symmetrically, $\mathsf{P}(G)[o_G(v) + i_G q_L\,..\,o_G(v) + (e-j_G)q_L) = C_L^{d_G}$ and $\mathsf{P}(G)(c_G(v)-(e-j_G)q_R\,..\,c_G(v)-i_G q_L] = C_R^{d_G}$ for $d_G := e-j_G-i_G$. We denote $d = \min(d_F, d_G)$, observe that $d \geq e - 10k \geq 0$, and construct a tree alignment $\mathcal{A}'$ so that it:

- aligns $\mathsf{P}(F)[0\,..\,o_F(u) + i_F q_L)$ with $\mathsf{P}(G)[0\,..\,o_G(v) + i_G q_L)$ in the same way as $\mathcal{A}$ does;
- matches $\mathsf{P}(F)[o_F(u) + i_F q_L\,..\,o_F(u) + (i_F + d)q_L) = C_L^d$ with $\mathsf{P}(G)[o_G(v) + i_G q_L\,..\,o_G(v) + (i_G + d)q_L) = C_L^d$;
- deletes $\mathsf{P}(F)[o_F(u) + (i_F + d)q_L\,..\,o_F(u) + (e - j_F)q_L) = C_L^{d_F - d}$ and $\mathsf{P}(G)[o_G(v) + (i_G + d)q_L\,..\,o_G(v) + (e - j_G)q_L) = C_L^{d_G - d}$;

- aligns $\mathsf{P}(F)[o_F(u)+(e-j_F)q_L\,..\,c_F(u)-(e-j_F)q_R]$ with $\mathsf{P}(G)[o_G(v)+(e-j_G)q_L\,..\,c_G(v)-(e-j_G)q_R]$ in the same way as $\mathcal{A}$ does;
- deletes $\mathsf{P}(F)(c_F(u)-(e-j_F)q_R\,..\,c_F(u)-(i_F+d)q_R] = C_R^{d_F-d}$ and $\mathsf{P}(G)(c_G(v)-(e-j_G)q_R\,..\,c_G(v)-(i_G+d)q_R] = C_R^{d_G-d}$;
- matches $\mathsf{P}(F)(c_F(u)-(i_F+d)q_R\,..\,c_F(u)-i_Fq_R] = C_R^d$ with $\mathsf{P}(G)(c_G(v)-(i_G+d)q_R\,..\,c_G(v)-i_Gq_R] = C_R^d$;
- aligns $\mathsf{P}(F)(c_F(u)-i_Fq_R\,..\,|\mathsf{P}(F)|)$ with $\mathsf{P}(G)(c_G(v)-i_Gq_R\,..\,|\mathsf{P}(G)|)$ in the same way as $\mathcal{A}$ does.

This definition makes it clear that $\mathcal{A}'$ is an edit-distance (string) alignment. In terms of the forests F and G, the alignment $\mathcal{A}'$ can be interpreted so that it:

- perfectly matches the occurrences of C at nodes $u_{i_F},\dots,u_{i_F+d-1}$ to the occurrences of C at nodes $v_{i_G},\dots v_{i_G+d-1}$, respectively;
- deletes the occurrences of C at nodes $u_{i_F+d},\dots u_{i_F+d_F-1}$ and $v_{i_G+d},\dots v_{i_G+d_G-1}$;
- handles the remaining parts of F and G in the same way as $\mathcal{A}$ does.

This interpretation makes it clear that $\mathcal{A}'$ is a tree alignment. Moreover, $\mathcal{A}$ and $\mathcal{A}'$ only differ in how they align C^{d_F} to C^{d_G}, and $\mathcal{A}'$ provides an optimum alignments of these contexts.

Now, if the exponent e of the context C^e is modified to $e' \geq 10k$, we can interpret this as modifying the exponent d of the contexts C^d matched perfectly by $\mathcal{A}'$ to $d' = d+e'-e \geq 0$. Thus, $\mathcal{A}'$ can be trivially adapted without modifying its cost, and hence $\mathsf{ted}(F',G') \leq \mathsf{ted}_{\mathcal{A}'}(F,G) = \mathsf{ted}(F,G)$. The converse inequality follows by symmetry between (F,G) and (F',G'). □

Proposition 7.15 (Avoiding Vertical k-Periodicity)**.** *There exists an $\mathcal{O}(n\log n)$-time algorithm that, given labeled forests F,G of total size n and an integer $k\in\mathbb{Z}_+$, produces forests labeled forests F',G' that satisfy $\mathsf{ted}_{\leq k}(F,G) = \mathsf{ted}_{\leq k}(F',G')$ and, moreover, avoid both synchronized horizontal k-periodicity and synchronized vertical k-periodicity.*

Proof. We consider Algorithm 7 to prove this proposition. First we find $S = \mathsf{VertPeriods}(\mathsf{F},\mathsf{G})$ in step 1; by Lemma 7.12, any $(u,v,q_L,q_R,e)\in S$ represents a $2k$-synchronized occurrence of C^e where $C_L = \mathsf{P}(F)[o(u)\,..\,o(u)+q_L), C_R = \mathsf{P}(G)(c(u)-q_R\,..\,c(u)], C = (C_L,C_R)$ at nodes u and v. In step 4, we identify the start of the specific periodic substrings we want to reduce the exponent of, and construct a set S' with quadruples containing these starting indices, the length of the periods, and the exponent to be reduced. In step 5 we sort these quadruples by their indices and iterate through them in order from left to right. For each quadruple (ℓ_F,ℓ_G,q,e), in step 11 and step 12, we move the start of these synchronized occurrences ahead by $q(e-14k)$. Clearly, this is equivalent to shortening the power e of the synchronized occurrences of C to a power of $14k$ exactly since

$$\frac{(\ell_F+qe)-(\ell_F+q(e-14k))}{q} = 14k.$$

We also note that by construction in Algorithm 6, no two quintuples of S will represent overlapping synchronized occurrences with more than $8k \leq 14k$ overlapping characters in either the left or right part of the context (see step 11). Therefore, since $C_L\cdot C_R$ is a balanced string, we can construct forests F',G' such that $\mathsf{P}(F') = s_F, \mathsf{P}(G') = s_G$ and by Lemma 7.13, $\mathsf{ted}_{\leq k}(F,G) = \mathsf{ted}_{\leq k}(F',G')$.

Now, we want to show that F',G' avoid vertical k-periodicity. Assume for contradiction that there is some $2k$-synchronized occurrence C^{16k} in F',G'. If the occurrence already was present in F',G', there must have been some maximal $2k$-synchronized occurrence C^e for $e\geq 16k$ between nodes $u,w\in F$ and $v,x\in G$. Algorithm 6 iterates over all nodes of F with a vertical period and finds any synchronized occurrences of the period in G. Note that when a synchronized occurrence

with power e' and context length q_L^F is found, we skip the $(e'-8k)q_L^F$ nodes, i.e. we still check the last $8k$ nodes of every vertical period. By Corollary 6.5, there is at most an $8k$ overlap between any two maximal runs of period length at most $4k$, we know that we will check node u since it is the start of a maximal vertical period. Therefore, as shown previously, we will find the occurrence of C^e and reduce its power to $14k$ and so no such C^{16k} will occur in F', G'.

We have shown that if a $2k$-synchronized C^{16k} occurs in F, G, we will reduce it. However, it is possible that such an occurrence comes as a result of other reductions. In this case, by Corollary 6.5, such a synchronized occurrence in F', G' can overlap at most two reduced vertical periods C_1^{14k}, C_2^{14k} since both contexts had powers of at least $16k$ in F, G. Since the overlap with each context is at most $8k$ and we leave $14k$ repetitions of C_1 and C_2 in F', G', any context C^{16k} in F', G' must have also occurred in F, G. As shown previously, C^{16k} must have been reduced to C^{14k}, which is a contradiction. Therefore, all $2k$-synchronized occurrences of vertical periods have been reduced in F', G' to a power of $14k$. Note that by the same argument, no new horizontal periodicity can occur in F', G' as well.

Algorithm 6 takes $\mathcal{O}(n \log n)$ time to compute S by Lemmas 7.6 and 7.11. We add one quintuple to S per node in F, so $|S| \leq n$. Therefore, Algorithm 7 takes time $\mathcal{O}(n \log n)$ to compute forests F', G'. □

8 Full Periodicity Reduction

Lemma 8.1. *Let λ be a joint labeling of forests F, G resulting in labeled forests F, G, and let $\hat{\lambda} = \mathsf{C}(\mathsf{L}(\lambda, 8k), 2k)$ for some $k \in \mathbb{Z}_+$. If F, G avoid both synchronized horizontal k-periodicity and synchronized vertical k-periodicity, then $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})$ and $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})$ avoid $2k$-synchronized $(20k+2)$-powers with root at most $4k$.*

Proof. For a proof by contradiction, suppose that there is a string $\hat{Q}$ of length $\hat{q} \in (0\,..\,4k]$ such that $\hat{Q}^{20k+2} = \mathsf{P}_{\hat{\lambda}}(\mathsf{F})[x\,..\,x+(20k+2)\hat{q}) = \mathsf{P}_{\hat{\lambda}}(\mathsf{G})[y\,..\,y+(20k+2)\hat{q})$ for some positions x, y with $|x-y| \leq 2k$. Note that $\hat{\lambda}$ is a refinement of $\mathsf{L}(\lambda, 8k)$, which, in turn, is a refinement of $\lambda' := \mathsf{L}(\lambda, 4k)$. In particular, $\mathsf{P}_{\lambda'}(\mathsf{F})[x\,..\,x+(20k+2)\hat{q}) = \mathsf{P}_{\lambda'}(\mathsf{G})[y\,..\,y+(20k+2)\hat{q})$ is a string with period $\hat{q}$. Let q be the shortest period of this string and let Q be the underlying string period.

If Q contains the same number of opening and closing parentheses, then, since Q^2 occurs in a balanced string $\mathsf{P}_{\lambda'}(\mathsf{F})$, we conclude that some cyclic rotation of Q is balanced, and therefore F, G do not avoid synchronized horizontal k-periodicity (because λ' is a refinement of λ). Thus, by symmetry, we may assume without loss of generality that Q contains more opening than closing parentheses. Suppose that $Q[\delta]$ is the leftmost unmatched opening parenthesis within Q and define nodes $u_0, \ldots, u_{20k+1} \in V_{\mathsf{F}}$ such that $o_{\mathsf{F}}(u_i) = x + \delta + iq$ and nodes $v_0, \ldots, v_{20k+1} \in V_{\mathsf{G}}$ such that $o_{\mathsf{G}}(v_i) = y + \delta + iq$. Observe that $c_{\mathsf{F}}(u_0) > \cdots > c_{\mathsf{F}}(u_{20k+1}) > o_{\mathsf{F}}(u_{20k+1})$ and $c_{\mathsf{G}}(v_0) > \cdots > c_{\mathsf{G}}(v_{20k+1}) > o_{\mathsf{G}}(v_{20k+1})$.

Next, we prove two claims regarding $2k$-compatibility with respect to $\mathsf{L}(\lambda, 8k)$, which we simply refer to as compatibility in the remainder of the proof.

Claim 8.2. *If there exists $i \in [0\,..\,20k]$ and a node $u' \in V_{\mathsf{F}}$ with $o_{\mathsf{F}}(u') \in [o_{\mathsf{F}}(u_0)\,..\,o_{\mathsf{F}}(u_{20k})]$ such that $\hat{\lambda}(u_i) = \hat{\lambda}(u')$, then $u' = u_{i'}$ for some $i' \in [0\,..\,20k]$.*

Proof. Note that $\hat{\lambda}(u_i) = \hat{\lambda}(u')$ implies $\mathsf{P}_{\lambda}(\mathsf{sub}_{<8k}(u')) = \mathsf{P}_{\lambda}(\mathsf{sub}_{<8k}(u_i))$ and $\mathsf{P}_{\lambda'}(\mathsf{sub}_{<4k}(u')) = \mathsf{P}_{\lambda'}(\mathsf{sub}_{<4k}(u_i))$. We thus have $\mathsf{P}_{\lambda'}(\mathsf{F})[o_{\mathsf{F}}(u')\,..\,o_{\mathsf{F}}(u')+q) = \mathsf{P}_{\lambda'}(\mathsf{F})[o_{\mathsf{F}}(u_i)\,..\,o_{\mathsf{F}}(u_i)+q)$ because the common size of $\mathsf{sub}_{<4k}(u')$ and $\mathsf{sub}_{<4k}(u_i)$ is at least $4k \geq q$. Since q is the shortest period of $\mathsf{P}_{\lambda'}(\mathsf{F})[o_{\mathsf{F}}(u_0)\,..\,o_{\mathsf{F}}(u_{20k})+q)$, we conclude that $o_{\mathsf{F}}(u') - o_{\mathsf{F}}(u_0)$ is a multiple of q, and hence $u' = u_{i'}$ for some $i' \in [0\,..\,20k]$. □

Claim 8.3. *For all $i \in [0\,..\,20k)$, we have $c_{\mathsf{F}}(u_i) \le c_{\mathsf{F}}(u_{i+1}) + 4k$ and $c_{\mathsf{G}}(v_i) \le c_{\mathsf{G}}(v_{i+1}) + 4k$*

Proof. We focus on the claim regarding F (the claim regarding G is symmetric). For a proof by contradiction, suppose that $c_{\mathsf{F}}(u_i) > c_{\mathsf{F}}(u_{i+1}) + 4k$ holds for some $i \in [0\,..\,20k)$. By construction, the node u_i shares the $\hat{\lambda}$-label with its descendant whose opening parenthesis is located at position $o_{\mathsf{F}}(u_i) + \hat{q}$. Consider the underlying path in the compatibility graph, let $u' \in V_{\mathsf{F}}$ be the last node on this path that is an ancestor of u_i (possibly $u' = u_i$), and let $u'' \in V_{\mathsf{F}}$ be the subsequent node of F on this path. By definition of compatibility, we have $|o_{\mathsf{F}}(u') - o_{\mathsf{F}}(u'')| \le 4k$ and $|c_{\mathsf{F}}(u') - c_{\mathsf{F}}(u'')| \le 4k$. Moreover, since u' is an ancestor of u_i, we have $o_{\mathsf{F}}(u') \le o_{\mathsf{F}}(u_i)$ and $c_{\mathsf{F}}(u') \ge c_{\mathsf{F}}(u_i)$. We conclude the proof by deriving a contradiction for every possible location of $o_{\mathsf{F}}(u'')$.

- If $o_{\mathsf{F}}(u'') \le o_{\mathsf{F}}(u_i)$, then either $c_{\mathsf{F}}(u'') \le o_{\mathsf{F}}(u_i)$, which implies $c_{\mathsf{F}}(u'') \le o_{\mathsf{F}}(u_i) < o_{\mathsf{F}}(u_{i+1}) < c_{\mathsf{F}}(u_{i+1}) < c_{\mathsf{F}}(u_i) - 4k \le c_{\mathsf{F}}(u') - 4k \le c_{\mathsf{F}}(u'')$ (a contradiction) or $c_{\mathsf{F}}(u'') \ge c_{\mathsf{F}}(u_i)$, which means that u'' is an ancestor of u_i and contradicts the choice of u'.
- If $o_{\mathsf{F}}(u'') \in (o_{\mathsf{F}}(u_i)\,..\,o_{\mathsf{F}}(u_{i+1}))$, then a contradiction follows from Claim 8.2
- If $o_{\mathsf{F}}(u'') \in [o_{\mathsf{F}}(u_{i+1})\,..\,c_{\mathsf{F}}(u_{i+1})]$, then u'' is an ancestor of u_{i+1}, which means that $c_{\mathsf{F}}(u'') \le c_{\mathsf{F}}(u_{i+1}) < c_{\mathsf{F}}(u_i) - 4k \le c_{\mathsf{F}}(u') - 4k \le c_{\mathsf{F}}(u'')$ (a contradiction).
- If $o_{\mathsf{F}}(u'') > c_{\mathsf{F}}(u_{i+1})$, then $o_{\mathsf{F}}(u'') > c_{\mathsf{F}}(u_{i+1}) > o_{\mathsf{F}}(u_{i+1}) + 4k > o_{\mathsf{F}}(u_i) + 4k \ge o_{\mathsf{F}}(u') + 4k \ge o_{\mathsf{F}}(u'')$, which is also a contradiction. □

The nodes $(u_i)_{i\in[0\,..\,20k]}$ and $(v_i)_{i\in[0\,..\,20k]}$ share the same λ'-label, so the subtrees $\mathsf{P}_\lambda(\mathsf{sub}_{<4k}(u_i))$ and $\mathsf{P}_\lambda(\mathsf{sub}_{<4k}(v_i))$ are all isomorphic. Consequently, by Claim 8.3, the context $C := (C_L, C_R) := (\mathsf{P}_\lambda(\mathsf{F})[o_{\mathsf{F}}(u_0)\,..\,o_{\mathsf{F}}(u_1)), \mathsf{P}_\lambda(\mathsf{F})(c_{\mathsf{F}}(u_1)\,..\,c_{\mathsf{F}}(u_0)])$ satisfies $0 < |C_L|, |C_R| < 4k$ and occurs at all nodes $(u_i)_{i\in[0\,..\,20k)}$ and $(v_i)_{i\in[0\,..\,20k)}$. Its power C^{20k} occurs in F at node u_0 and in G at node v_0.

Observe that, for every $i \in [0\,..\,20k]$, the node v_i must be compatible with some $u' \in V_{\mathsf{F}}$. If $i \in [4k\,..\,16k]$, then $o_{\mathsf{F}}(u') \ge o_{\mathsf{G}}(v_i) - 2k \ge o_{\mathsf{F}}(u_i) - 4k \ge o_{\mathsf{F}}(u_{i-4k})$ and $o_{\mathsf{F}}(u') \le o_{\mathsf{G}}(v_i) + 2k \le o_{\mathsf{F}}(u_i) + 4k \le o_{\mathsf{F}}(u_{i+4k})$, so $o_{\mathsf{F}}(u') \in [o_{\mathsf{F}}(u_{i-4k})\,..\,o_{\mathsf{F}}(u_{i+4k})]$. By Claim 8.2, this means that $u' = u_j$ for some $j \in [i - 4k\,..\,i + 4k]$. In particular, $|o_{\mathsf{F}}(u_i) - o_{\mathsf{G}}(v_j)| \le 2k$ and $|c_{\mathsf{F}}(u_i) - c_{\mathsf{G}}(v_j)| \le 2k$. Since $(o_{\mathsf{F}}(u_t))_{t\in[0\,..\,20k)}$ and $(o_{\mathsf{G}}(v_t))_{t\in[0\,..\,20k)}$ form arithmetic progressions with difference $|C_L|$ and $(c_{\mathsf{F}}(u_t))_{i\in[0\,..\,20k)}$ and $(c_{\mathsf{G}}(v_i))_{t\in[0\,..\,20k)}$ form arithmetic progressions with difference $-|C_R|$, we conclude that there exists $\delta := j - i \in [-4k\,..\,4k]$ such that $|o_{\mathsf{F}}(u_t) - o_{\mathsf{G}}(v_{t+\delta})| \le 2k$ and $|c_{\mathsf{F}}(u_t) - c_{\mathsf{G}}(v_{t+\delta})| \le 2k$ hold whenever $t, t + \delta \in [0\,..\,20k]$. This means that C^{16k} has $2k$-synchronized occurrences in F and G (at nodes u_0, v_δ if $\delta \ge 0$, and at nodes $u_{-\delta}, v_0$ otherwise), contradicting the assumption that F, G avoid synchronized vertical k-periodicity. □

Proposition 8.4. *There exists a randomized algorithm that, given forests F, G and a threshold $k \in \mathbb{Z}_+$, produces forests F', G' and an alignment $\mathcal{A} : \mathsf{P}(F') \rightsquigarrow \mathsf{P}(G')$ such that:*

- $\mathsf{ted}_{\le k}(F, G) = \mathsf{ted}_{\le k}(F', G')$, *and*
- $|\mathcal{A} \triangle \mathcal{B}| \le 4928k^4$ *for every alignment $\mathcal{B} \in \mathsf{TA}_k(F', G')$.*

The running time is $\mathcal{O}(n \log n + k^3)$ and the algorithm is correct w.h.p.

Proof. The forests F', G' are produced by horizontal (Proposition 6.12) and vertical (Proposition 7.15) periodicity reduction; this guarantees $\mathsf{ted}_{\le k}(F, G) = \mathsf{ted}_{\le k}(F', G')$. Let F', G' be the underlying unlabeled forests and let λ' be their joint labeling. We use Lemmas 4.3 and 4.7 to construct $\hat{\lambda} = \mathsf{C}(\mathsf{L}(\lambda', 8k), 2k)$ and strings $\mathsf{P}_{\hat{\lambda}}(\mathsf{F}')$, $\mathsf{P}_{\hat{\lambda}}(\mathsf{G}')$. Note that the width of any $\mathcal{B} \in \mathsf{TA}_k(F', G')$ does not exceed $2k$ and, by Lemma 4.2 and Observation 4.8, we have $\mathsf{ed}_{\mathcal{B}}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}'), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}')) \le 2k \cdot 8k \le 16k^2$. Thus, $\mathcal{B} \in \mathsf{A}_{16k^2,2k}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}'), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}'))$. We construct $\mathcal{A}$ in $\mathcal{O}(n + k^3)$ time using Lemma 3.3 as an arbitrary alignment in $\mathsf{GA}_{16k^2,2k}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}'), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}'))$; if there is no such alignment,

then $\mathsf{A}_{16k^2,2k}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}'), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}')) = \emptyset$ and hence $\mathsf{TA}_k(F', G') = \emptyset$. By Lemmas 8.1 and 5.3, we have $|\mathcal{A} \triangle \mathcal{B}| \leq 7 \cdot 2k \cdot 16k^2 \cdot (20k + 2) \leq 4928k^4$. □

9 Main Algorithm

9.1 Partial Forest Matching

Let F and G be labeled forests. We say that a set $M \subseteq V_F \times V_G$ is *non-crossing* if the set $\bigcup_{(u,v)\in M}\{(o_F(u), o_F(v)), (c_F(u), c_F(v))\} \subseteq \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$ is non-crossing. Furthermore, we say that $M \subseteq V_F \times V_G$ is a *non-crossing matching* of forests F, G if $\bigcup_{(u,v)\in M}\{(o_F(u), o_F(v)), (c_F(u), c_F(v))\}$ is a non-crossing matching of $\mathsf{P}(F), \mathsf{P}(G)$.

We say that a tree alignment $\mathcal{A} \in \mathsf{TA}(F, G)$ *aligns* $u \in V_F$ with $v \in V_G$, denoted $u \sim_{\mathcal{A}} v$, if $\mathsf{P}(F)[o_F(u)] \sim_{\mathcal{A}} \mathsf{P}(G)[o_G(v)]$ and $\mathsf{P}(F)[c_F(u)] \sim_{\mathcal{A}} \mathsf{P}(G)[c_G(v)]$. If, additionally, u and v have the same labels, we say that $\mathcal{A}$ *matches* u and v, denoted $u \simeq_{\mathcal{A}} v$. For a set $M \subseteq V_F \times V_G$, we define $\mathsf{TA}^M(F, G) = \{\mathcal{A} \in \mathsf{TA}(F, G) : u \simeq_{\mathcal{A}} v \text{ for each } (u, v) \in M\}$; note that $\mathsf{TA}^M(F, G) = \emptyset$ unless M is a non-crossing matching. Moreover, we denote $\mathsf{ted}^M(F, G) = \min_{\mathcal{A}\in\mathsf{TA}^M(F,G)} \mathsf{ted}_{\mathcal{A}}(F, G)$.

In this section, we consider the problem of computing $\mathsf{ted}^M(F, G)$ given labeled forests F, G and a non-crossing matching $M \subseteq V_F \times V_G$. The following lemma provides a reduction that restricts the heights of F, G and lets us assume that $M \subseteq L_F \times L_G$, where $L_F \subseteq V_F$ is the set of leaves of F and $L_G \subseteq V_G$ is the set of leaves of G.

Lemma 9.1. *There exists a linear-time algorithm that, given labeled forests F, G and a non-crossing matching $M \subseteq V_F \times V_G$, produces labeled forests F', G' and a non-crossing matching $M' \subseteq L_{F'} \times L_{G'}$ such that:*

- $\mathsf{ted}^{M'}(F', G') = \mathsf{ted}^M(F, G)$;
- $|F'| = |M| + |F|$, $|G'| = |M| + |G|$, *and* $|M'| = 2|M|$;
- *if the height of F' (G') is $h > 1$, then there is an $(h-1)$-node top-down path in F (respectively, G) avoiding nodes participating in M.*

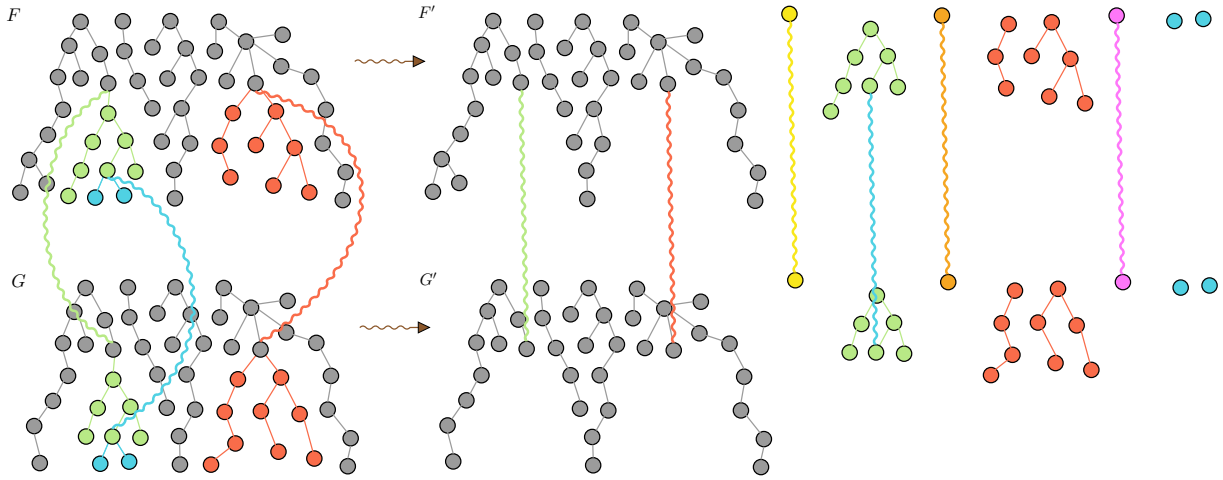


Figure 5: An illustration of the transformation implemented in Lemma 9.1. The non-crossing matching is represented using colorful wavy lines. In the input forests F and G, each node inherits its color from the nearest marked ancestor.

Proof. Let us denote $M = \{(u_i, v_i) : i \in [1\,..\,m]\}$ and *mark* all nodes participating in M. We decompose $V_F = \bigcup_{i=0}^{m} V_{F_i}$ so that $u \in V_{F_0}$ if u does not have any proper marked ancestor and, for every $i \in [1\,..\,m]$, we have $u \in V_{F_i}$ if u_i is the nearest proper marked ancestor of u. We further define forests $F_0, \ldots, F_m$ so that F_i is obtained from F by deleting all vertices in $V_F \setminus V_{F_i}$.

As far as the implementation is concerned, a top-down traversal of F allows classifying each node $u \in V_F$ into one of the classes V_{F_i}. This is because every root node belongs to V_{F_0}, every node with a marked parent u_i belongs to the class V_{F_i}, and every other node belongs to the same class as its unmarked parent. Consequently, the forest F can be decomposed into the forests $F_0, \ldots, F_m$ in linear time. Symmetrically, we decompose the forest G into analogously defined forests $G_0, \ldots, G_m$.

Due to the fact that M is non-crossing, this yields a decomposition $M = \bigcup_{i=0}^{m} M_i$, where $M_i = M \cap (V_{F_i} \times V_{G_i})$. Furthermore, if $u \sim_{\mathcal{A}} v$ holds for an alignment $\mathcal{A} \in \mathsf{TA}^M(F, G)$, then the set $M \cup \{(u, v)\} \subseteq V_F \times V_G$ is non-crossing, and thus there exists $i \in [0\,..\,m]$ such that $u \in V_{F_i}$ and $v \in V_{G_i}$. Consequently, we have $\mathsf{ted}^M(F, G) = \sum_{i=0}^{m} \mathsf{ted}^{M_i}(F_i, G_i)$.

In the second step of the algorithm, we create nodes $\hat{u}_1, \ldots, \hat{u}_m$ and $\hat{v}_1, \ldots, \hat{v}_m$, all sharing the same label. The forest F' is obtained as a (horizontal) concatenation $F' := F_0 \cdot \hat{u}_1 \cdot F_1 \cdots \hat{u}_m \cdot F_m$, where each node $\hat{u}_i$ is interpreted as a single-node forest. Symmetrically, $G' := G_0 \cdot \hat{v}_1 \cdot G_1 \cdots \hat{v}_m \cdot G_m$. Finally, we set $M' := M \cup \{(\hat{u}_i, \hat{v}_i) : i \in [1\,..\,m]\}$. The forests F', G' and the set M' can be easily constructed in linear time.

The construction trivially yields $|F'| = |F| + |M|$, $|G'| = |G| + |M|$, and $|M'| = 2|M|$. Since the separator nodes are included in M', we further have $\mathsf{ted}^{M'}(F', G') = \sum_{i=0}^{m} \mathsf{ted}^{M_i}(F_i, G_i) = \mathsf{ted}^{M}(F, G)$. The separator nodes are leaves and $M_i \subseteq L_{F_i} \times L_{G_i}$ holds for each $i \in [0\,..\,m]$ (by definition of the decompositions of F and G), so we have $M' \subseteq L_{F'} \times L_{G'}$. Finally, observe that if the height of F' is $h > 1$, then one of the forests F_i has an h-node root-to-leaf path. Trimming the leaf, we obtain an $(h-1)$-node top-down path avoiding marked nodes. By construction of F_i, this path is also present in F. A symmetric reasoning lets us characterize the height of G'. □

The following reduction prunes unnecessary leaves. This is useful if $M \subseteq L_F \times L_G$ is very large.

Lemma 9.2. *There exists a linear-time algorithm that, given labeled forests F, G and a non-crossing matching $M \subseteq L_F \times L_G$ produces labeled forests F', G' and a non-crossing matching $M' \subseteq L_{F'} \times L_{G'}$ such that:*

- *F' and G' are obtained by deleting some leaves in F and G, respectively;*
- *$M' = M \cap (V_{F'} \times V_{G'}) = M \cap (V_F \times V_{G'}) = M \cap (V_{F'} \times V_G)$ satisfies $|M'| \leq \frac{2}{5}(|F'| + |G'| + 1)$;*
- *$\mathsf{ted}^{M'}(F', G') = \mathsf{ted}^{M}(F, G)$.*

Proof. The algorithm identifies a subset $\hat{M}$ of *redundant leaf pairs* and constructs (F', G', M') so that $M' = M \setminus \hat{M}$ whereas the forests F' and G' are obtained be deleting all nodes participating in $\hat{M}$. Specifically, $\hat{M}$ contains all pairs $(\hat{u}, \hat{v}) \in M$ such that $\hat{u}$ and $\hat{v}$ have immediate left siblings u and v, respectively, such that $(u, v) \in M$. The construction trivially yields $\mathsf{ted}^{M'}(F', G') \leq \mathsf{ted}^{M}(F, G)$. As for the inverse inequality, we shall prove that every alignment $\mathcal{A}' \in \mathsf{ted}^{M'}(F', G')$, can be converted to an alignment $\mathcal{A} \in \mathsf{ted}^{M}(F, G)$ of the same cost. For this, we simply extend $\mathcal{A}'$ so that $\hat{u} \simeq_{\mathcal{A}} \hat{v}$ holds for all $(\hat{u}, \hat{v}) \in \hat{M}$. A simple inductive argument (reintroducing $(\hat{u}, \hat{v}) \in \hat{M}$ in the left-to-right order) shows that this does not introduce any crossings among the aligned pairs of vertices. This is because any existing pair crossing $(\hat{u}, \hat{v})$ would also cross the pair (u, v) of immediate left siblings.

Next, we note that if $(u', v') \in M'$, then one of the following cases holds:

- u' or v' is the leftmost root of F' (resp. G');
- u' or v' is the leftmost child of its parent (and the parent is not participating in M');

- u' or v' has an immediate left sibling that is not participating in M'.

The number of nodes not participating in M' is $|F'|+|G'|-2|M'|$, and each of them can be charged twice (by its right sibling and by its leftmost child). Hence, $|M'| \leq 2(1+|F'|+|G'|-2|M'|)$, and this simplifies to $|M'| \leq \frac{2}{5}(|F'|+|G'|+1)$.

Finally, we observe that F', G', and M' can be easily constructed in linear time. □

The last result of this section reduces computing $\mathsf{ted}^M_{\leq k}(F,G)$ to computing $\mathsf{ted}_{\leq k}(F',G')$ for some F' and G'.

Lemma 9.3. *There exists an algorithm that, given labeled forests F, G, a non-crossing matching $M \subseteq L_F \times L_G$, and an integer $k \in \mathbb{Z}_{\geq 0}$, produces labeled forests F', G' satisfying the following conditions:*

- $\mathsf{ted}_{\leq k}(F',G') = \mathsf{ted}^M_{\leq k}(F,G)$;
- $|F'| = |F| + (k+1)|M|$ *and* $|G'| = |G| + (k+1)|M|$;
- *The height of F' (G') does not exceed the height of F (resp., G) plus one.*

The running time of the algorithm is $\mathcal{O}(|F'|+|G'|)$.

Proof. Let $M = \{(u_i, v_i) : i \in [1\,..\,m]\}$. For each $i \in [1\,..\,m]$, we attach $k+1$ new nodes, $u_{i,0}, \ldots, u_{i,k}$, as children of u_i and $k+1$ new nodes, $v_{i,0}, \ldots, v_{i,k}$, as children of v_i. The nodes $u_{i,j}$ and $v_{i,j}$ share a unique label $\$_{i,j}$ that is not present anywhere else in the constructed forests F', G'. This completes the description of the constructed forests F', G'. It is easy to see that F', G' can be constructed in time proportional to their sizes, which are $|F|+(k+1)|M|$ and $|G|+(k+1)|M|$, respectively. Moreover, the height of F' (resp. G') may increase compared to the height of F (G) by at most one (since we attach new leaves only to existing nodes).

Thus, it remains to prove that $\mathsf{ted}^M_{\leq k}(F,G) = \mathsf{ted}_{\leq k}(F',G')$. For this, let us first observe that $\mathsf{ted}(F',G') \leq \mathsf{ted}^M(F,G)$. This is because, any $\mathcal{A} \in \mathsf{TA}^M(F,G)$ can be extended to an alignment $\mathcal{A}' \in \mathsf{TA}(F',G')$ such that $u_{i,j} \simeq_{\mathcal{A}'} v_{i,j}$ holds for all $i \in [1\,..\,m]$ and $j \in [0\,..\,k]$. If any of the newly matched pairs $(u_{i,j}, v_{i,j})$ crossed some other pair of vertices aligned by $\mathcal{A}'$, already (u_i, v_i), with $u_i \simeq_{\mathcal{A}} v_i$, would cross that pair. As for the converse inequality, consider an alignment $\mathcal{A}' \in \mathsf{TA}(F',G')$ of cost at most k. By Proposition 4.4, we can choose $\mathcal{A}'$ so that it can be interpreted as a greedy alignment with respect to (unbounded) look-ahead labels. We shall prove that, for each $i \in [1\,..\,m]$, we have $u_i \simeq_{\mathcal{A}'} v_i$ and $u_{i,j} \simeq_{\mathcal{A}'} v_{i,j}$ for $j \in [0\,..\,k]$. As a result, a matching $\mathcal{A} \in \mathsf{TA}^M(F,G)$ of the same cost can be obtained by restricting $\mathcal{A}'$ to the nodes of F and G.

Let us fix $i \in [1\,..\,m]$. As the cost of $\mathcal{A}'$ is at most k, this alignment must match the node $u_{i,\hat{\jmath}}$ for some $\hat{\jmath} \in [0\,..\,k]$. The only vertex in G' sharing the label with $u_{i,\hat{\jmath}}$ is $v_{i,\hat{\jmath}}$, so we must have $u_{i,\hat{\jmath}} \simeq_{\mathcal{A}'} v_{i,\hat{\jmath}}$ and, in particular, $\mathsf{P}(F')[c_{F'}(u_{i,\hat{\jmath}})] \simeq_{\mathcal{A}'} \mathsf{P}(G')[c_{G'}(v_{i,\hat{\jmath}})]$. Furthermore, observe that the nodes (u_i, v_i) and the nodes $(u_{i,j}, v_{i,j})$ for $j \in [0\,..\,k]$ not only share their regular labels, but also their (unbounded) look-ahead labels. Consequently, the greedy nature of $\mathcal{A}'$ yields $\mathsf{P}(F')[c_{F'}(u_{i,\hat{\jmath}})\,..\,c_{F'}(u_i)] \simeq_{\mathcal{A}'} \mathsf{P}(G')[c_{G'}(v_{i,\hat{\jmath}})\,..\,c_{G'}(v_i)]$. Since $\mathcal{A}'$ is a tree alignment, $\mathsf{P}(F')[c_{F'}(u_i)] \simeq_{\mathcal{A}'} \mathsf{P}(G')[c_{G'}(v_i)]$ implies $\mathsf{P}(F')[o_{F'}(u_i)] \simeq_{\mathcal{A}'} \mathsf{P}(G')[o_{G'}(v_i)]$. Using the greedy nature of $\mathcal{A}'$ once again, we finally conclude that $\mathsf{P}(F')[o_{F'}(u_i)\,..\,c_{F'}(u_i)] \simeq_{\mathcal{A}'} \mathsf{P}(G')[o_{G'}(v_i)\,..\,c_{G'}(v_i)]$. Thus, $u_i \simeq_{\mathcal{A}'} v_i$ and $u_{i,j} \simeq_{\mathcal{A}'} v_{i,j}$ for $j \in [0\,..\,k]$ hold as claimed. □

We conclude with a corollary that applies Lemmas 9.1 to 9.3 one after another.

Corollary 9.4. *There exists an algorithm that, given labeled forests F, G, a non-crossing matching $M \subseteq V_F \times V_G$, and an integer $k \in \mathbb{Z}_{\geq 0}$, produces labeled forests F', G' such that:*

- $\mathsf{ted}_{\leq k}(F',G') = \mathsf{ted}^M_{\leq k}(F,G)$;

- $|F'| + |G'| = \mathcal{O}(\min(|F| + |G| + k|M|, 1 + (k+1)(|F| + |G| - 2|M|)))$;
- *if the height of F' (or G') is $h > 2$, then there is an $(h-2)$-node top-down path in F (respectively, G) avoiding nodes participating in M.*

The running time of the algorithm is $\mathcal{O}(|F| + |G| + |F'| + |G'|)$.

Proof. As hinted above, the algorithm behind Corollary 9.4 simply chains the procedures underlying Lemmas 9.1 to 9.3. Let us denote the intermediate results by $(\hat{F}, \hat{G}, \hat{M})$ and $(\bar{F}, \bar{G}, \bar{M})$, respectively. Due to $\mathsf{ted}_{\leq k}(F', G') = \mathsf{ted}^{\bar{M}}_{\leq k}(\bar{F}, \bar{G})$ and $\mathsf{ted}^{\bar{M}}(\bar{F}, \bar{G}) = \mathsf{ted}^{\hat{M}}(\hat{F}, \hat{G}) = \mathsf{ted}^{M}(F, G)$, we conclude that $\mathsf{ted}_{\leq k}(F', G') = \mathsf{ted}^{M}_{\leq k}(F, G)$ holds as claimed. As for the heights, observe that if the height of F' is $h > 2$, then the heights of $\bar{F}$ are $\hat{F}$ are at least $h - 1 > 1$. By Lemma 9.1, this means that F contains a top-down path of at least $h - 2$ nodes none of which participate in M. A symmetric argument can be used to bound the height of G'.

It remains to bound $|F'|+|G'|$. Applying Lemma 9.3, Lemma 9.2, and Lemma 9.1 in subsequent steps, we obtain

$$|F'|+|G'| \leq |\bar{F}|+|\bar{G}|+2(k+1)|\bar{M}| \leq |\hat{F}|+|\hat{G}|+2(k+1)|\hat{M}| \leq |F|+|M|+|G|+|M|+4(k+1)|M| \\ = \mathcal{O}(|F| + |G| + k|M|).$$

However, we further have

$$2|\bar{M}| \leq |\bar{F}| + |\bar{G}| \leq |\bar{F}| + |\bar{G}| - 10|\bar{M}| + 10|\bar{M}| \leq |\bar{F}| + |\bar{G}| - 10|\bar{M}| + 4(|\bar{F}| + |\bar{G}| + 1) \\ = 5(|\bar{F}| + |\bar{G}| - 2|\bar{M}|) + 4 = 5(|\hat{F}| + |\hat{G}| - 2|\hat{M}|) + 4 = 5(|F| + |G| - 2|M|) + 4,$$

and thus

$$|F'| + |G'| \leq |\bar{F}| + |\bar{G}| + 2(k+1)|\bar{M}| \leq (k+2)(|\bar{F}| + |\bar{G}|) = \mathcal{O}(1 + (k+1)(|F| + |G| - 2|M|)).$$

Consequently, we indeed have $|F'|+|G'| = \mathcal{O}(\min(|F|+|G|+k|M|, 1+(k+1)(|F|+|G|-2|M|)))$. □

9.2 Tree Edit Distance of Shallow Forests

Theorem 9.5. *There exists a randomized algorithm that, given forests F, G of height at most $h \in \mathbb{Z}_+$ and a threshold $k \in \mathbb{Z}_+$, computes $\mathsf{ted}_{\leq k}(F, G)$ in $\mathcal{O}(n \log n + h^2 k^7 \log(hk))$ time correctly with high probability.*

Proof. Let F and G be the underlying unlabeled forests and let λ be their joint labeling. Using Proposition 6.12, at the cost of $\mathcal{O}(n)$ time, we can assume without loss of generality that F, G avoid synchronized horizontal k-periodicity, i.e., there is no balanced string Q of length $|Q| \leq 4k$ such Q^{18k} has $2k$-synchronized occurrences in $\mathsf{P}_\lambda(\mathsf{F})$ and $\mathsf{P}_\lambda(\mathsf{G})$. In the next step, we construct a labeling $\hat{\lambda}$ equivalent to $\mathsf{L}(\lambda, h)$ (using Lemma 4.3) as well as the strings $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})$ and $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})$. Since $\hat{\lambda}$ is a refinement of λ, we conclude that there is no balanced string Q of length $|Q| \leq 4k$ such Q^{18k} has $2k$-synchronized occurrences in $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})$ and $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})$. Moreover, $\hat{\lambda}$ assigns distinct labels to nodes in any root-to-leaf path; thus, for any unbalanced string Q, even Q^2 cannot occur in $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})$ or $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})$. Consequently, $\mathsf{P}_{\hat{\lambda}}(\mathsf{F})$ or $\mathsf{P}_{\hat{\lambda}}(\mathsf{G})$ avoid $2k$-synchronized $18k$-powers of length at most $4k$. This lets us apply Lemma 5.4 to build a set M_{P} of size $|M_{\mathsf{P}}| \geq 2|F| - 15 \cdot (18k) \cdot (2hk)^2 \cdot 2k \geq 2|F| - \mathcal{O}(h^2 k^4)$ such that $M_{\mathsf{P}} \subseteq \mathsf{M}_{\mathcal{A}}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}))$ holds for every $\mathcal{A} \in \mathsf{GA}_{2hk,2k}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}))$; the construction of M_{P} costs $\mathcal{O}(n + hk^2)$ time.

Claim 9.6. *Let* $M = \{(u,v) \in V_F \times V_G : (o_F(u), o_G(v)) \in M_\mathsf{P} \text{ and } (c_F(u), c_G(v)) \in M_\mathsf{P}\}$. *If* $\mathsf{ted}(F,G) \le k$, *then* M *is a non-crossing matching of size* $|M| \ge |F| - \mathcal{O}(h^2k^4)$ *and, moreover,* $\mathsf{ted}^M(F,G) = \mathsf{ted}(F,G)$.

Proof. By Proposition 4.4, there is an optimum alignment $\mathcal{A} \in \mathsf{TA}(F,G)$ of cost $\mathsf{ted}_\mathcal{A}(F,G) = \mathsf{ted}(F,G)$ such that $\mathcal{A} \in \mathsf{GA}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}))$. If $\mathsf{ted}(F,G) \le k$, then the width of $\mathcal{A}$ is at most $2k$ and, by Lemma 4.2, its cost $\mathsf{ed}_\mathcal{A}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}))$ does not exceed $2hk$. Hence, $\mathcal{A} \in \mathsf{GA}_{2hk,2k}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}))$, which means that $M_\mathsf{P} \subseteq \mathsf{M}_\mathcal{A}(\mathsf{P}_{\hat{\lambda}}(\mathsf{F}), \mathsf{P}_{\hat{\lambda}}(\mathsf{G}))$ and, in particular, $\mathcal{A} \in \mathsf{TA}^M(F,G)$. Consequently, $\mathsf{ted}^M(F,G) = \mathsf{ted}(F,G)$ and the number of vertices $u \in V_\mathsf{F}$ not participating in M does not exceed $2|F| - |M_\mathsf{P}| = \mathcal{O}(h^2k^4)$. □

In the light of Claim 9.6, we construct M and check whether it is a non-crossing matching of size $|M| \ge |F| - \mathcal{O}(h^2k^4)$; if it is not, we report that $\mathsf{ted}(F,G) > k$. The remaining task is to compute $\mathsf{ted}^M_{\le k}(F,G)$. For this, we use Corollary 9.4, which reduces this problem to computing $\mathsf{ted}_{\le k}(F',G')$, where $|F'| + |G'| = \mathcal{O}(k \cdot h^2k^4)$ because $|F| + |G| - 2|M| = \mathcal{O}(h^2k^4)$; this reduction costs $\mathcal{O}(n + h^2k^5)$ time. As for the final step of determining $\mathsf{ted}_{\le k}(F',G')$, we employ the algorithm of [AJ21], whose running time is $\mathcal{O}(h^2k^5 \cdot k^2 \cdot \log(h^2k^5)) = \mathcal{O}(h^2k^7 \log(hk))$. Overall, our procedure takes $\mathcal{O}(n + h^2k^7 \log(hk))$ time. □

9.3 Level Sampling

Theorem 1.1. *There exists a randomized algorithm that, given forests* F, G *of total size* n *and an integer* $k \in \mathbb{Z}_+$, *computes* $\mathsf{ted}_{\le k}(F,G)$ *in* $\mathcal{O}(n \log n + k^{15} \log k \log n)$ *time correctly with high probability.*

Algorithm 8: `TreeEditDistance`(F,G,k)

Input: labeled forests F, G, integer threshold $k \in \mathbb{Z}_+$
Output: $\mathsf{ted}_{\le k}(F,G)$

```
(F', G', A) := FullPeriodicityReduction(F, G, k);          // Proposition 8.4
d := ∞;
h := 19716k^4;
for i := 0 to Θ(log(|F| + |G|)) do
    Pick r_i ∈ [0 .. h) uniformly at random;
    Mark nodes in F', G' at depths ≡ r_i (mod h);
    M_i := {(u,v) ∈ V_F' × V_G' : P(F')[o_F'(u)] ≃_A P(G')[o_G'(v)], and
                P(F')[c_F'(u)] ≃_A P(G')[c_G'(v)], and u or v is marked};
    if |M_i| ≤ (4/h)(|F'| + |G'|) and all marked nodes participate in M_i then
        (F_i, G_i) := PartialMatchingReduction(F', G', M_i, k);   // Corollary 9.4
        d_i := ShallowTreeEditDistance(F_i, G_i, k);              // Theorem 9.5
        d := min(d, d_i);
return d;
```

Proof. As a first step, we apply Proposition 8.4, which produces forests F', G' and an alignment $\mathcal{A} : \mathsf{P}(F') \rightsquigarrow \mathsf{P}(G')$. Next, we pick $h := 19716k^4$ and draw $s = \Theta(\log n)$ uniformly random values $r_1, \ldots, r_s \in [0..h)$. As described below, each of these values r_i is either discarded or results in an upper bound on $\mathsf{ted}_{\le k}(F',G')$. We mark all nodes in F' and G' whose depths are congruent to r_i modulo h. We then attempt using $\mathcal{A}$ to construct a non-crossing matching M_i in which

all marked nodes participate. For this, we add to M_i every pair of nodes $(u,v) \in V_{F'} \times V_{G'}$ such that $\mathsf{P}(F')[o_{F'}(u)] \simeq_{\mathcal{A}} \mathsf{P}(G')[o_{G'}(v)]$, $\mathsf{P}(F')[c_{F'}(u)] \simeq_{\mathcal{A}} \mathsf{P}(G')[c_{G'}(v)]$, and u or v is marked. By construction, this guarantees that M_i is a non-crossing matching. However, we discard M_i if $|M_i| > \frac{4}{h}(|F'| + |G'|)$ or there exists a marked node that does not participate in M_i. If M_i is not discarded, we use Corollary 9.4 to build forests F_i, G_i such that $\mathsf{ted}_{\leq k}(F_i, G_i) = \mathsf{ted}^{M_i}_{\leq k}(F', G')$, and then we compute $d_i := \mathsf{ted}_{\leq k}(F_i, G_i)$ using Theorem 9.5. The final answer d is defined as the minimum among the values d_i computed for non-discarded matchings M_i.

Let us analyze the correctness of this approach. Proposition 8.4 provides the following guarantees with high probability:

- $\mathsf{ted}_{\leq k}(F, G) = \mathsf{ted}_{\leq k}(F', G')$,
- for every alignment $\mathcal{B} \in \mathsf{TA}_k(F', G')$, we have $|\mathcal{A} \triangle \mathcal{B}| \leq 4928k^4$.

Due to $\mathsf{ted}^{M_i}(F', G') \geq \mathsf{ted}(F', G')$, the reported value d must clearly satisfy $d \geq \mathsf{ted}_{\leq k}(F', G') = \mathsf{ted}_{\leq k}(F, G)$. The main challenge is to prove the converse inequality. This task is trivial if $\mathsf{ted}(F, G) > k$. Otherwise, it boils down to showing that, with high probability, there exists $i \in [1\,..\,s]$ such that $d_i \leq \mathsf{ted}_{\leq k}(F, G)$. Let us fix an optimum alignment $\mathcal{B} \in \mathsf{TA}(F', G')$. We shall first prove that $d_i \leq \mathsf{ted}_{\leq k}(F, G)$ holds conditioned on the following events:

- $|M_i| \leq \frac{4}{h}(|F'| + |G'|)$;
- the alignment $\mathcal{B}$ that does not make any edits on the marked nodes and the parentheses corresponding to the marked nodes do not contribute to $\mathcal{A} \triangle \mathcal{B}$.

Since $\mathcal{B}$ does not make any edits on the marked nodes, all the marked nodes participate in the following non-crossing matching: $\{(u,v) \in V_{F'} \times V_{G'} : u \simeq_{\mathcal{B}} v \text{ and } u \text{ or } v \text{ is marked}\}$. Furthermore, this matching is equal to M_i because the parentheses corresponding to marked nodes do not contribute to $\mathcal{A} \triangle \mathcal{B}$. Consequently, $\mathsf{ted}^{M_i}(F', G') \leq \mathsf{ted}_{\mathcal{B}}(F', G') \leq k$, and thus $d_i = \mathsf{ted}_{\leq k}(F_i, G_i) = \mathsf{ted}^{M_i}_{\leq k}(F', G') = \mathsf{ted}_{\leq k}(F', G') = \mathsf{ted}_{\leq k}(F, G)$. It remains to prove that the favorable events hold with high probability for at least one $i \in [1\,..\,s]$. For this, we analyze the complementary bad events. For a random remainder modulo h, each individual node in $V_{F'} \cup V_{G'}$ is marked with probability $\frac{1}{h}$. Hence, in expectation, the number of marked nodes is $\frac{1}{h}(|F'| + |G'|)$. Given that each pair in M_i contains a marked node, this means that $\mathbb{E}[|M_i|] \leq \frac{1}{h}(|F'| + |G'|)$. By Markov's inequality, we conclude that $|M_i| > \frac{4}{h}(|F'| + |G'|)$ holds with probability at most $\frac{1}{4}$. Moreover, there are at most $2k$ nodes affected by the edits of $\mathcal{B}$ and at most $9856k^4$ further nodes may contribute to $\mathcal{B} \triangle \mathcal{A}$. By the union bound, at least one of these nodes is marked with probability at most $\frac{2k+9856k^4}{h} \leq \frac{1}{2}$. Overall, for each $i \in [1\,..\,s]$, the probability of the bad events does not exceed $\frac{1}{4} + \frac{1}{2} = \frac{3}{4}$. Due to $s = \Theta(\log n)$, the probability that the bad events hold for all $i \in [1\,..\,s]$ does not exceed $(\frac{3}{4})^s = n^{-\Theta(1)}$. Thus, the algorithm is correct with high probability.

Let us complete the proof with the running time analysis. Applying Proposition 8.4 costs $\mathcal{O}(n \log n + k^3)$ time. For each $i \in [1\,..\,s]$, the set M_i can be constructed and verified in $\mathcal{O}(n)$ time. If r_i is not discarded, then $|M_i| = \mathcal{O}(\frac{1}{h}(|F'| + |G'|)) = \mathcal{O}(\frac{n}{k^4})$ and neither F' nor G' contains an h-node top-down path avoiding nodes participating in M_i. Consequently, the application of Corollary 9.4 takes $\mathcal{O}(n)$ time and produces forests of size $\mathcal{O}(n)$ and height at most $h + 1 = \mathcal{O}(k^4)$. The final application of Theorem 9.5 thus costs $\mathcal{O}(n + k^{15} \log k)$ time. Since all steps, except for the preprocessing of Proposition 8.4, are repeated $\mathcal{O}(\log n)$ times, the overall time complexity of our algorithm is $\mathcal{O}(n \log n + k^{15} \log k \log n)$. □

10 Conclusion

This paper gives an $\tilde{\mathcal{O}}(n + \text{poly}(k))$-time algorithm for bounded tree edit distance, solving an open problem posed in [AJ21, Mao21]. Multiple natural improvements and extensions of this result might be feasible. An immediate direction for future work is to significantly reduce the polynomial dependency on k, which currently stays at k^{15}, akin to the recent progress for Dyck edit distance [FGK+22a]. Another open question is whether the weighted version of tree edit distance admits an $\tilde{\mathcal{O}}(n + \text{poly}(k))$-time algorithm (assuming that the cost of each edit is at least one). To the best of our knowledge, no such algorithm is known even for weighted string edit distance.

Acknowledgment

Barna Saha and Tomasz Kociumaka were partly supported by NSF 1652303, 1909046, and HDR TRIPODS Phase II grant 2217058. MohammadTaghi Hajiaghayi and Jacob Gilbert were partly supported by NSF CCF grants 2114269 and 2218678.

References

[Abb14] Amir Abboud. Hardness for easy problems, 2014. Presented at Satellite Workshop of ICALP (YR-ICALP). URL: https://www.dropbox.com/s/jt9uzljjmormkb7/EasyHardness.pdf.

[ABW18] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. *SIAM J. Comput.*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.

[AJ21] Shyan Akmal and Ce Jin. Faster algorithms for bounded tree edit distance. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 12:1–12:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.12.

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *51st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, page 377–386. IEEE, 2010. doi:10.1109/FOCS.2010.43.

[AN20] Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *61st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2020*, pages 990–1001. IEEE, 2020. doi:10.1109/FOCS46700.2020.00096.

[AO12] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012. doi:10.1137/090767182.

[BEG+21] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. *J. ACM*, 68(3):19:1–19:41, 2021. doi:10.1145/3456807.

[BES06] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, page 792–801. SIAM, 2006. doi:10.1145/1109557.1109644.

[BFP+05] Michael A. Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005. doi:10.1016/j.jalgor.2005.08.001.

[BGHS19] Mahdi Boroujeni, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. $(1+\epsilon)$-approximation of tree edit distance in quadratic time. In *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 709–720. ACM, 2019. doi:10.1145/3313276.3316388.

[BGK03] Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed XML. In *29th International Conference on Very Large Data Bases, VLDB 2003*, page 141–152. Morgan Kaufmann, 2003. doi:10.1016/b978-012722442-8/50021-5.

[BGMW20] 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), 2020. doi:10.1145/3381878.

[BGSW19] Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM J. Comput.*, 48(2):481–512, 2019. doi:10.1137/17M112720X.

[BII+17] Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. doi:10.1137/15m1011032.

[Bil05] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1–3):217–239, 2005. doi:10.1016/j.tcs.2004.12.030.

[BO16] Arturs Backurs and Krzysztof Onak. Fast algorithms for parsing sequences of parentheses with few errors. In Tova Milo and Wang-Chiew Tan, editors, *35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016*, pages 477–488. ACM, 2016. doi:10.1145/2902251.2902304.

[BR20] Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 685–698. ACM, 2020. doi:10.1145/3357713.3384282.

[BS98] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recogn. Lett.*, 19(3–4):255–259, 1998. doi:10.1016/S0167-8655(97)00179-7.

[BYJKK04] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *45th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2004*, page 550–559. IEEE, 2004. doi:10.1109/FOCS.2004.14.

[CDG+20] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *J. ACM*, 67(6), 2020. doi:10.1145/3422823.

[CDX22] Shucheng Chi, Ran Duan, and Tianle Xie. Faster algorithms for bounded-difference min-plus product. In *33rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 1435–1447. SIAM, 2022. doi:10.1137/1.9781611977073.60.

[Cha99] Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *25th International Conference on Very Large Data Bases, VLDB 1999*, pages 90–101. Morgan Kaufmann, 1999. URL: http://www.vldb.org/conf/1999/P8.pdf.

[DKS22] Debarati Das, Tomasz Kociumaka, and Barna Saha. Improved approximation algorithms for Dyck edit distance and RNA folding. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 49:1–49:20. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.49.

[DMRW10] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, 6(1), 2010. doi:10.1145/1644015.1644017.

[Dür22] Anita Dürr. Improved bounds for rectangular monotone min-plus product, 2022. doi:10.48550/arXiv.2208.02862.

[FGK+22a] Dvir Fried, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, Ely Porat, and Tatiana Starikovskaya. An improved algorithm for the k-Dyck edit distance problem. In *33rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 3650–3669. SIAM, SIAM, 2022. doi:10.1137/1.9781611977073.144.

[FGK+22b] Dvir Fried, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, Ely Porat, and Tatiana Starikovskaya. An improved algorithm for the k-Dyck edit distance problem, 2022. arXiv:2111.02336v2.

[FLMM09] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1), 2009. doi:10.1145/1613676.1613680.

[Gaw11] Paweł Gawrychowski. Pattern matching in Lempel-Ziv compressed strings: Fast, simple, and deterministic. In *19th Annual European Symposium on Algorithms, ESA 2011*, volume 6942 of *LNCS*, pages 421–432. Springer, 2011. doi:10.1007/978-3-642-23719-5_36.

[GHN20] Younan Gao, Meng He, and Yakov Nekrich. Fast preprocessing for optimal orthogonal range reporting and range successor with applications to text indexing. In *28th Annual European Symposium on Algorithms, ESA 2020*, volume 173 of *LIPIcs*, pages 54:1–54:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.54.

[GRS20] Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. Does preprocessing help in fast sequence comparisons? In *52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 657–670, 2020. doi:10.1145/3357713.3384300.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, USA, 1997. doi:10.1017/cbo9780511574931.

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978.

[HRS19] Bernhard Haeupler, Aviad Rubinstein, and Amirbehshad Shahrasbi. Near-linear time insertion-deletion codes and $(1+\epsilon)$-approximating edit distance via indexing. In *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, page 697–708. ACM, 2019. doi:10.1145/3313276.3316371.

[HT84] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.

[Ind01] Piotr Indyk. Algorithmic applications of low-distortion geometric embeddings. In *42nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2001*, page 10. IEEE, 2001. doi:10.1109/sfcs.2001.959878.

[Kle98] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *6th Annual European Symposium on Algorithms, ESA 1998*, ESA '98, page 91–102. Springer, 1998. doi:10.1007/3-540-68530-8_8.

[Koc18] Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, 2018. URL: https://depotuw.ceon.pl/handle/item/3614.

[Koz97] Dexter C. Kozen. *Automata and Computability*. Springer New York, 1997. doi:10.1007/978-1-4612-1844-9.

[KPS21] Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. Small space and streaming pattern matching with k edits. In *62nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2021*, pages 885–896. IEEE, 2021. arXiv:2106.06037, doi:10.1109/FOCS52979.2021.00090.

[KR87] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

[KS20] Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 699–712. ACM, 2020. doi:10.1145/3357713.3384307.

[LMS98] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, 1998. doi:10.1137/s0097539794264810.

[LV88] Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

[Mao21] Xiao Mao. Breaking the cubic barrier for (unweighted) tree edit distance. In *62nd Annual IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 792–803. IEEE, 2021. doi:10.1109/FOCS52979.2021.00082.

[Mon] MongoDB JSON schema examples tutorial. https://www.mongodb.com/basics/json-schema-examp Accessed: 2022-03-03.

[Mor06] Christian Worm Mortensen. Fully dynamic orthogonal range reporting on RAM. *SIAM J. Comput.*, 35(6):1494–1525, 2006. doi:10.1137/S0097539703436722.

[Mye86] Eugene W. Myers. An $o(nd)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446.

[Sah14] Barna Saha. The Dyck language edit distance problem in near-linear time. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014*, pages 611–620. IEEE Computer Society, 2014. doi:10.1109/focs.2014.71.

[Sel77] Stanley M. Selkow. The tree-to-tree editing problem. *Inform. Process. Lett.*, 6(6):184–186, 1977. doi:10.1016/0020-0190(77)90064-3.

[SS22] 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 of *LIPIcs*, pages 115:1–115:22. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.115.

[SZ90] 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.

[Tai79] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979. doi:10.1145/322139.322143.

[Tou05] Hélène Touzet. A linear tree edit distance algorithm for similar ordered trees. In *16th Annual Conference on Combinatorial Pattern Matching, CPM 2005*, page 334–345. Springer, 2005. doi:10.1007/11496656_29.

[ZS89] 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.