

Weighted Edit Distance Computation: Strings, Trees and Dyck

Debarati Das¹, Jacob Gilbert², MohammadTaghi Hajiaghayi², Tomasz Kociumaka³, and Barna Saha⁴

¹Pennsylvania State University, United States
debaratix710@gmail.com

²University of Maryland, United States
jgilber8@umd.edu hajiaghayi@gmail.com

³Max Planck Institute for Informatics, Germany
tomasz.kociumaka@mpi-inf.mpg.de

⁴University of California, San Diego, United States
barnas@ucsd.edu

Abstract

Given two strings of length n over alphabet Σ , and an upper bound k on their edit distance, the algorithm of Myers (Algorithmica’86) and Landau and Vishkin (JCSS’88) from almost forty years back computes the unweighted string edit distance in $\mathcal{O}(n+k^2)$ time. Till date, it remains the fastest algorithm for exact edit distance computation, and it is optimal under the Strong Exponential Hypothesis (STOC’15). Over the years, this result has inspired many developments, including fast approximation algorithms for string edit distance as well as similar $\tilde{\mathcal{O}}(n+\text{poly}(k))$ -time algorithms for generalizations to tree and Dyck edit distances. Surprisingly, all these results hold only for unweighted instances.

While unweighted edit distance is theoretically fundamental, almost all real-world applications require weighted edit distance, where different weights are assigned to different edit operations (insertions, deletions, and substitutions), and the weights may vary with the characters being edited. Given a weight function $w : \Sigma \cup \{\varepsilon\} \times \Sigma \cup \{\varepsilon\} \rightarrow \mathbb{R}_{\geq 0}$ (such that $w(a, a) = 0$ and $w(a, b) \geq 1$ for all $a, b \in \Sigma \cup \{\varepsilon\}$ with $a \neq b$), the goal is to find an alignment that minimizes the total weight of edits. Except for the vanilla $\mathcal{O}(n^2)$ -time dynamic-programming algorithm and its almost trivial $\mathcal{O}(nk)$ -time implementation (k being an upper bound on the sought total weight), none of the aforementioned developments on the unweighted edit distance applies to the weighted variant. In this paper, we propose the first $\mathcal{O}(n + \text{poly}(k))$ -time algorithm that computes weighted string edit distance exactly, thus bridging a fundamental decades-old gap between our understanding of unweighted and weighted edit distance. We then generalize this result to weighted tree and Dyck edit distances, bringing in several new techniques, which lead to a deterministic algorithm that improves upon the previous work even for unweighted tree edit distance. Given how fundamental weighted edit distance is, we believe our $\mathcal{O}(n + \text{poly}(k))$ algorithm for weighted edit distance will be instrumental for further significant developments in the area.

1 Introduction

String edit distance and its several variants have been studied for decades since the 1960s [Lev65, NW70, WF74]. Historically, most work on these problems assumed that the edit operations have unit weights in order to simplify the problem and streamline theoretical results. Till date, the

fastest exact algorithm for unweighted edit distance is due to Myers [Mye86] and Landau and Vishkin [LV88], who obtained an $\mathcal{O}(n + k^2)$ -time solution for two strings of length n with an upper bound k on their edit distance. This bound is now known to be optimal (up to subpolynomial factors) under the Strong Exponential Hypothesis [BI18]. Over the years, the Holy-Grail result of [Mye86, LV88] has inspired many developments on fast approximation algorithms for (unweighted) string edit distance [CDG⁺20, GKS19, GRS20, KS20a] and similar $\tilde{\mathcal{O}}(n + \text{poly}(k))$ -time¹ algorithms for generalizations such as the (unweighted) Dyck and tree edit distances [BO16, FGK⁺22a, DGH⁺22]. However, almost all real-world applications require weighted edit distance, where different weights are assigned to different edit operations (insertions, deletions, and substitutions), and the weights may vary with the characters being edited [WF74, ZS89, Kur96, Gus97, PM02, JM09, FFF⁺16, Ski20, KXI20, GWK21, CIG22]. As a result, there is a major gap between the theoretical results of prior research and real-world utility of these results. In this paper, we bridge this fundamental gap between the understanding of unweighted and weighted edit distance: We provide the first non-trivial algorithm computing the weighted edit distance and its generalizations to weighted tree and Dyck tree edit distance.

More specifically, in this paper we propose the first $\mathcal{O}(n + \text{poly}(k))$ -time algorithm for exact weighted edit distance computation in which, given a weight function $w : \Sigma \cup \{\varepsilon\} \times \Sigma \cup \{\varepsilon\} \rightarrow \mathbb{R}_{\geq 0}$ (normalized so that $w(a, b) \geq 1$ for $a \neq b$), the goal is to find an alignment that minimizes the total weight of edit operations (insertions, deletions, and substitutions) assuming that it does not exceed a provided threshold k . Strikingly, except for the vanilla $\mathcal{O}(n^2)$ -time dynamic-programming algorithm and its almost trivial $\mathcal{O}(nk)$ -time implementation, none of the aforementioned developments on unweighted edit distance apply to this weighted variant. We then generalize our result to weighted tree and Dyck edit distances, bringing in several new techniques that lead to improvements even for the unweighted tree edit distance problem: As a byproduct of our results, we present a deterministic $\mathcal{O}(n + k^7 \log k)$ -time solution, which is much faster than the randomized $\mathcal{O}(n \log n + k^{15} \log^2 k)$ -time algorithm of Das, Gilbert, Hajiaghayi, Kociumaka, Saha, and Saleh [DGH⁺22].

Can this apparent lack of progress in weighted edit distance computation be explained? As we observe later, even basic properties like monotonicity, which was fundamental for efficient computation of unweighted edit distance [Mye86, LV88], break down when considering weighted operations. This precludes any local matching approach, which seemed necessary for a linear-time algorithm for bounded (unweighted) edit distance [Mye86, LV88, BO16, FGK⁺22a, DGH⁺22]; instead, a global view of the sequences is needed to find matching substrings and yet maintain the linear runtime. Faced with such barriers, our biggest contribution is a kernelization method for weighted edit distance, not just for strings, but also for tree and Dyck edit distance instances. Interestingly, our kernels are weight-agnostic, that is, the kernelization algorithms do not need to know the weight function w . Given how fundamental weighted edit distance is, we believe our $\mathcal{O}(n + \text{poly}(k))$ algorithm for weighted edit distance will be instrumental for further significant developments in the area.

1.1 Related Work

String Edit Distance: *Edit distance* is one of the most fundamental problems in computer science studied since the 1960s [Lev65, NW70, WF74]. In the unweighted edit distance problem, given two strings of length at most n , the goal is to find the minimum number of edit operations (insertions, deletions, and substitutions) required to transform one string into the other. Given a parameter k as an upper bound on the edit distance, an algorithm proposed in the 1980s by

¹The $\tilde{\mathcal{O}}(\cdot)$ notation suppresses factors polylogarithmic in the input size n .

Myers [Mye86] and Landau and Vishkin [LV88] achieves this task in $\mathcal{O}(n + k^2)$ time by combining suffix trees with an elegant greedy approach. As long as $k = \mathcal{O}(\sqrt{n})$, the running time of the above algorithm is linear in n . For larger values of k , approximation algorithms for edit distance have been studied extensively [LMS98, Ind01, BYJKK04, BES06, AO09, AKO10], especially recently [HRS19, CDG⁺20, GRS20, KS20b, BR20, BEG⁺21]. This culminated with the currently best bound by Andoni and Nosatzki [AN20], who obtained a constant-factor approximation algorithm with running time $\mathcal{O}(n^{1+\epsilon})$ for any constant $\epsilon > 0$. All of these works require monotonicity and assume that an optimal solution can be extended easily if matching suffixes are added to both strings, none of which may hold in weighted edit distance instances. As a result, the state-of-the-art approximation algorithm for weighted edit distance, by Kuszmaul [Kus19], offers much worse trade-off, with an $\mathcal{O}(n^\tau)$ -factor approximation in $\tilde{\mathcal{O}}(n^{2-\tau})$ time for any $0 \leq \tau \leq 1$.

Tree Edit Distance: The *tree edit distance* problem, first introduced by Selkow [Sel77], is a generalization of edit distance in which the task is to compute a measure of dissimilarity between two rooted ordered trees with node labels. In the unweighted version of tree edit distance, every node insertion, deletion, or relabeling operation has unit cost. The problem has numerous applications in compiler optimization [DMRW10], structured data analysis [Cha99, BGK03, FLMM09], image analysis [BS98], and computational biology [HT84, SZ90, Gus97, LMS98, Bil05]. The current best bound on running time of an algorithm for finding exact tree edit distance is due to Dürr [Dür22] who obtained an $\mathcal{O}(n^{2.9149})$ -time algorithm for the problem, after a long series of improvements from $\mathcal{O}(n^6)$ [Tai79] to $\mathcal{O}(n^4)$ [ZS89], to $\mathcal{O}(n^3 \log n)$ [Kle98], to $\mathcal{O}(n^3)$ [DMRW10], and to $\mathcal{O}(n^{2.9546})$ [Mao21]. Moreover, there is a $(1 + \epsilon)$ -approximation algorithm for tree edit distance with running time $\tilde{\mathcal{O}}(n^2)$ time due to Boroujeni, Ghodsi, Hajiaghayi, and Seddighin [BGHS19]. Recently, Seddighin and Seddighin [SS22] gave an $\mathcal{O}(n^{1.99})$ -time $(3 + \epsilon)$ -approximation algorithm for tree edit distance (building on a previous $\tilde{\mathcal{O}}(n)$ -time $\mathcal{O}(\sqrt{n})$ -factor approximation algorithm of [BGHS19]). Furthermore, Das, Gilbert, Hajiaghayi, Kociumaka, Saha, and Saleh [DGH⁺22] obtained an $\tilde{\mathcal{O}}(n + k^{15})$ -time algorithm for exact tree edit distance with an upper bound k on the distance (see also an $\tilde{\mathcal{O}}(nk^2)$ -time algorithm of Akmal and Jin [AJ21], which improves upon a previous algorithm with running time $\mathcal{O}(nk^3)$ for the bounded tree edit distance problem [Tou05]).

As far as the weighted tree edit distance is concerned, the fastest algorithm, by Demaine, Mozes, Rossman, and Weimann [DMRW10], takes $\mathcal{O}(n^3)$ time, which matches the conditional lower-bound of Bringmann, Gawrychowski, Mozes, and Weimann [BGMW20] (earlier conjectured by Abboud [Abb14]). Specifically, there is no truly subcubic-time algorithm for weighted tree edit distance unless APSP has a truly subcubic-time solution. The lower bound still holds for trees over a constant-size alphabet unless the weighted k -clique problem admits an $\mathcal{O}(n^{k-\epsilon})$ -time algorithm.

Dyck Edit Distance: The *Dyck edit distance* problem is another variation of edit distance which falls under the umbrella of general language edit distance [AP72, Mye95, Sah17, BGSW19] and has numerous practical applications, e.g., for fixing hierarchical data files, in particular XML and JSON files [Har78, Koz97]. In the unweighted version of this problem, given a string of n parentheses, the goal is to find the minimum number of edits (character insertions, deletions, and substitutions) to make the string well-balanced. Several algorithms for both exact [BGSW19, CDX22, Dür22] and approximation [Sah14, DKS22] versions of the problem have been obtained. Finding exact Dyck edit distance is at least as hard as Boolean matrix multiplication [ABW18]. The bounded Dyck edit problem was subject to several recent studies as well: Backurs and Onak [BO16] obtained the first algorithm with running time $\mathcal{O}(n + k^{16})$, which was further improved to $\mathcal{O}(n + k^5)$ [FGK⁺22a], and finally to $\mathcal{O}(n + k^{4.5442})$ using fast matrix multiplication [FGK⁺22b, Dür22]. Except for the

$\mathcal{O}(n^3)$ -time exact algorithm for language edit distance [Mye95], these results are not applicable to the weighted setting.

1.2 Our Contribution

The main contributions of our paper are new algorithms for weighted string, tree, and Dyck edit distance. We define a *weight function* as a function $w : \Sigma \cup \{\varepsilon\} \times \Sigma \cup \{\varepsilon\} \rightarrow \mathbb{R}_{\geq 0}$ such that $w(a, a) = 0$ and $w(a, b) \geq 1$ for $a \neq b$. If $a, b \in \Sigma$, then $w(a, \varepsilon)$ is the cost of deleting a , $w(\varepsilon, b)$ is the cost of inserting b , whereas $w(a, b)$ is the cost of substituting a for b . The assumption $w(a, a) = 0$ indicates that matching symbols can be aligned at no cost, whereas the assumption $w(a, b) \geq 1$ for $a \neq b$ indicates that the weights are normalized so that every edit costs at least one. A weight function is a *quasimetric* if it also satisfies the triangle inequality (which we assume for tree and Dyck edit distance). When it comes to computations on weights, we consider any uniform model in which real numbers are subject to only comparison and addition [PR05], e.g., the RAM model.

We define $\text{ed}^w(X, Y)$ to be the minimum cost of an alignment of strings X and Y for weight function w . Furthermore, we define $\text{ed}_{\leq k}^w(X, Y)$ as $\text{ed}^w(X, Y)$ (if it is at most k) or ∞ (otherwise). We give the first weighted bounded edit distance algorithm with runtime $\mathcal{O}(n + \text{poly}(k))$.

Theorem 1.1. *Given strings X, Y of length at most n , an integer $k \in \mathbb{Z}_+$, and a weight function w , the value $\text{ed}_{\leq k}^w(X, Y)$ can be computed in $\mathcal{O}(n + k^5)$ time.*

Similarly to string edit distance, we define $\text{ted}^w(F, G)$ as the minimum cost of a tree alignment of forests F and G for weight function w . We define $\text{ted}_{\leq k}^w(F, G)$ analogously and give the first weighted tree edit distance algorithm with runtime $\mathcal{O}(n + \text{poly}(k))$. In the unweighted case, our deterministic algorithm is significantly faster than the state-of-the-art randomized algorithm from [DGH⁺22].

Theorem 1.2. *Given forests F, G of length at most n , an integer $k \in \mathbb{Z}_+$, and a quasimetric w , the value $\text{ted}_{\leq k}^w(F, G)$ can be computed in $\mathcal{O}(n + k^{15})$ time. Moreover, $\text{ted}_{\leq k}(F, G)$ can be computed in $\mathcal{O}(n + k^7 \log k)$ time.*

Finally, we define $\text{dyck}_{\leq k}^w(X)$ to be the minimum distance $\text{ed}_{\leq k}^w(X, Y)$ between X and a string Y in the Dyck language. We give the first algorithm for weighted Dyck edit distance with runtime $\mathcal{O}(n + \text{poly}(k))$. In this setting, the alphabet consists of opening and closing parentheses, and we need to assume that the weight function, apart from satisfying the triangle inequality, treats opening and closing parentheses of the same type similarly. This is captured in the notion of a *skewmetric* formally defined in Section 4.2.

Theorem 1.3. *Given a string X of length n , an integer $k \in \mathbb{Z}_+$, and a skewmetric w , the value $\text{dyck}_{\leq k}^w(X)$ can be computed in $\mathcal{O}(n + k^{12})$ time.*

We note that, although our algorithms assume k is given, one can also obtain running times analogous to those of Theorems 1.1 to 1.3 but with the sought distance instead of the threshold k . For this, it suffices to start from the largest value k that results in the running time of $\mathcal{O}(n)$, e.g., $k = \Theta(n^{1/5})$ for strings, and keep doubling the threshold k as long as the algorithm outputs ∞ . The first finite outcome is guaranteed to be the sought distance and, since the running times of the subsequent iterations form a geometric progression, the overall runtime is dominated by the last iteration, where k is at most twice the sought distance.

1.3 Overview

The folklore algorithms to compute edit distance for unweighted and weighted instances use dynamic programming and runs in $\mathcal{O}(n^2)$ time. Given two strings X and Y , the entry $D[i, j]$ of the dynamic programming table D holds the weighted (unweighted) edit distance of prefixes of X and Y up to indices i and j respectively. That is $D[i, j] := \text{ed}(X[0..i], Y[0..j])$. Then

$$D[i+1, j+1] = \min\{D[i, j+1] + 1, D[i+1, j] + 1, D[i, j] + \delta(X[i], Y[j])\} \quad \text{:unweighted edit distance}$$

$$D[i+1, j+1] = \min\{D[i, j+1] + w(X[i], \varepsilon), D[i+1, j] + w(\varepsilon, Y[j]), D[i, j] + w(X[i], Y[j])\} \quad \text{:weighted edit distance}$$

The first entry in the recursive definition corresponds to deleting $X[i]$, the second entry corresponds to inserting $Y[j]$, and the third entry corresponds to either matching or substitution ($\delta(X[i], Y[j]) = 0$ if $X[i] = Y[j]$, otherwise $\delta(X[i], Y[j]) = 1$). Clearly, $D[|X|, |Y|]$ equals the total weighted (unweighted) edit distance between X and Y , and can be computed in $\mathcal{O}(n^2)$ time.

It is possible to improve the running time to $\mathcal{O}(nk)$ if the weighted (unweighted) edit distance is bounded by $k < n$. In this case the entries corresponding to only $2k + 1$ diagonals surrounding the main diagonal of D need to be computed. However, the similarities between the developments on unweighted and weighted edit distance computations end here.

The first major breakthrough in the unweighted edit distance computation came in the late eighties [Mye86, LV88]. An $\mathcal{O}(n + k^2)$ -time algorithm for unweighted edit distance was developed whenever edit distance is bounded by k , thereby giving a linear time algorithm for $k \leq \sqrt{n}$. The algorithm utilizes two simple but powerful properties of unweighted edit distance, namely (i) *monotonicity*: $D[i+1, j+1] \geq D[i, j]$, and (ii) *greedy extension*: if $X[i] = Y[j]$ then $D[i+1, j+1] = D[i, j]$. These two properties together imply that if we can find maximal equal substrings in X and Y through a preprocessing step, only $\mathcal{O}(k^2)$ entries of D need to be computed. More precisely, for each of the $2k + 1$ diagonals, these are the at most $k + 1$ entries with $k + 1 \geq D[i+1, j+1] > D[i, j]$. The preprocessing step utilizes a linear-time construction of a suffix tree to answer any maximal equal substring queries in constant time, leading to an overall running time of $\mathcal{O}(n + k^2)$. All subsequent developments on fast approximation algorithms for unweighted string edit distance rely on the above two properties without exception.

Unfortunately, none of the above two properties hold for weighted edit distance computation. The following simple examples will make this observation clear.

1. *No monotonicity*: Let $X = \mathbf{ab}$, $Y = \mathbf{c}$, and $w(\mathbf{a}, \mathbf{c}) + w(\mathbf{b}, \varepsilon) < w(\mathbf{a}, \varepsilon)$. Then $D[1, 0] = w(\mathbf{a}, \varepsilon)$ and $D[2, 1] = w(\mathbf{a}, \mathbf{c}) + w(\mathbf{b}, \varepsilon) < w(\mathbf{a}, \varepsilon)$.
2. *No greedy extension*: Let $X = \mathbf{ab}$, $Y = \mathbf{b}$, and $w(\mathbf{a}, \mathbf{b}) + w(\mathbf{b}, \varepsilon) < w(\mathbf{a}, \varepsilon)$. Then substituting \mathbf{a} to \mathbf{b} and deleting \mathbf{b} from X is cheaper than deleting \mathbf{a} and matching the subsequent \mathbf{b} .

In some sense, this explains the lack of progress on weighted instances in this field. We need a very different approach and new ideas.

When k , the minimum weighted edit distance, is small for two input strings, clearly most characters of the input strings are perfectly matched and contribute no cost to the edit distance computation. The main idea of our algorithm is to find small representative instances for the input strings and then run the $\mathcal{O}(nk)$ -time weighted edit distance solution on these representatives to find the original weighted edit distance. In fact, we prove that any instance of the bounded-weighted edit distance can be solved using strings of size $\mathcal{O}(k^4)$. Our algorithm constructs such an $\mathcal{O}(k^4)$ -size kernel from strings of size $\mathcal{O}(n)$ in time $\mathcal{O}(n)$, and then the resulting small instances can be solved in time $\mathcal{O}(k^5)$ using the $\mathcal{O}(nk)$ -time weighted extension of the dynamic programming.

We are also able to extend the idea of kernelization to weighted instances of tree and Dyck edit distances by giving the first $\mathcal{O}(n + \text{poly}(k))$ algorithms for them. Notably, our algorithms are deterministic and give significant improvements over the recent randomized algorithms on unweighted tree edit distance [DGH⁺22]. We show it is possible to compute small $\mathcal{O}(\text{poly}(k))$ -size kernels from the original instances of each problem in linear time, and then run dynamic programming based algorithms to compute the final edit distance values.

To find such kernels, we utilize substrings that have *synchronized occurrences* in both input strings X and Y , that is, they occur in X and Y at positions x and y , respectively, satisfying $|x - y| = \mathcal{O}(k)$. Our kernelization algorithm first tries to cover the input strings (almost entirely) with $\mathcal{O}(k)$ pairs of synchronized occurrences. If this is impossible, then we conclude that the edit distance must be large, that is, $\text{ed}_{\leq k}^w(X, Y) = \infty$. Otherwise, we apply a novel notion of *edit-distance equivalence* so that synchronized occurrences of a substring P can be substituted with synchronized occurrences of an equivalent substring P' without affecting the edit distance $\text{ed}_{\leq k}^w(X, Y)$. To this end, we provide a linear-time algorithm that, given any string P , computes an edit-distance equivalent string P' of size $\text{poly}(k)$.

A similar notion of equivalent pieces is also central to our algorithms for weighted tree and Dyck edit distance. Our three algorithms all utilize the following high-level steps:

1. Partition the input objects into $\mathcal{O}(k)$ pieces most of which can be paired up to form synchronized occurrences.
2. If the algorithm failed to find sufficiently long synchronized occurrences, report that the edit distance exceeds k .
3. Otherwise, for every pair of synchronized occurrences, substitute the original piece with a small equivalent replacement.
4. Solve the resulting small instance with a known dynamic-programming algorithm.

1.3.1 Weighted String Edit Distance

We now describe how to obtain Theorem 1.1 by implementing the aforementioned high-level scheme.

Edit-Distance Equivalent Strings The biggest technical contribution behind our weighted edit distance algorithm is a linear-time procedure (of Corollary 2.14) that, given a string P , computes an equivalent string of length $\mathcal{O}(k^3)$. In the first phase, it eliminates *k-periodicity*: as long as the processed string contains a fragment of the form Q^{4k+1} with $|Q| \in [1 \dots 2k]$, this fragment is replaced by Q^{4k} . As shown in Lemma 2.9, the strings Q^{4k+1} and Q^{4k} are equivalent, so this step preserves equivalence with the input string P . Eventually, the first phase results in a string that *avoids k-periodicity* and is equivalent with P (see Fig. 1 for example). It is implemented in Lemma 2.13, where the underlying algorithm processes the input string P from left to right and removes the first copy of Q for every encountered fragment of the form Q^{4k+1} with $|Q| \in [1 \dots 2k]$.

In Lemma 2.11, we prove that if P avoids k -periodicity and satisfies $|P| \geq 42k^3$, then it is equivalent to $P[0 \dots 21k^3) \cdot P[|P| - 21k^3 \dots |P|)$, that is, the concatenation of its prefix of length $21k^3$ and its suffix of length $21k^3$ (with the characters in the middle removed). For this, we consider synchronized occurrences of P in strings X and Y and an optimal alignment \mathcal{A} of cost $\text{ed}^w(X, Y) \leq k$ that maps X onto Y . We observe that \mathcal{A} must perfectly match a length- $10k^2$ fragment within the length- $21k^3$ prefix of the occurrence of P in X . Moreover, since P avoids k -periodicity, \mathcal{A} can only match this fragment to the corresponding fragment of the occurrence of P in Y . Symmetrically, \mathcal{A} must match the two copies of a length- $10k^2$ fragment within the length- $21k^3$ suffix of P . We conclude that \mathcal{A} aligns the two copies of $P[d \dots |P| - e)$ for some

$d, e \in [0 \dots 21k^3]$. Since \mathcal{A} is optimal, it must perfectly match the two copies of $P[d \dots |P| - e]$. Thus, $P[21k^3 \dots |P| - 21k^3]$ can be removed from the synchronized occurrences of P in X and Y without affecting the cost $\text{ed}_{\leq k}^w(X, Y)$. Consequently, if the first phase returns a string of length at least $42k^3$, then the algorithm of Corollary 2.14 removes all but the leading $21k^3$ and the trailing $21k^3$ characters of that string (see Fig. 2 for example).

Linear-Time Kernel In order to apply the notion of edit-distance equivalence, we need to identify synchronized occurrences within X and Y . To this end, we check whether $\text{ed}(X, Y) \leq k$. If this is not the case, then $\text{ed}^w(X, Y) \geq \text{ed}(X, Y) > k$ holds for every normalized weight function w , and thus we already know that $\text{ed}_{\leq k}^w(X, Y) = \infty$. If $\text{ed}(X, Y) \leq k$, on the other hand, then we construct an optimal unweighted \mathcal{A} alignment mapping X onto Y . As formally proved in Fact 2.7, the unedited characters of X form at most $k + 1$ fragments that \mathcal{A} matches perfectly. Each of these fragments of X forms a synchronized occurrence together with its image under \mathcal{A} in Y . Thus, we can replace the synchronized occurrences with occurrences of an equivalent string of length $\mathcal{O}(k^3)$. Since we have partitioned X and Y into $\mathcal{O}(k)$ edited characters plus $\mathcal{O}(k)$ synchronized occurrences, this yields strings X' and Y' of length $\mathcal{O}(k^4)$ satisfying $\text{ed}_{\leq k}^w(X', Y') = \text{ed}_{\leq k}^w(X, Y)$. In order to construct \mathcal{A} efficiently, we use the $\mathcal{O}(n + k^2)$ unweighted edit distance algorithm of [Mye86, LV88]. However, if $n \leq k^4$, then we do not need to worry about reducing the size of X and Y in the first place and therefore do not construct an optimal unweighted alignment; otherwise, $\mathcal{O}(n + k^2) = \mathcal{O}(n)$ and constructing the $\mathcal{O}(k^4)$ -size kernel takes linear time; see Theorem 2.15 for details on our kernel for weighted string edit distance.

As mentioned earlier, once we have a kernel (X', Y') of size $\mathcal{O}(k^4)$, we can run the $\mathcal{O}(nk)$ -time weighted edit-distance algorithm to compute $\text{ed}_{\leq k}^w(X', Y') = \text{ed}_{\leq k}^w(X, Y)$ in $\mathcal{O}(k^5)$ time.

1.3.2 Weighted Tree Edit Distance

Our algorithm for weighted tree edit distance follows the same high-level approach. However, compared to the string edit distance, two major challenges arise. First, the structure of periodicity is much richer and requires two notions: *horizontal periodicity* of *forests* and *vertical periodicity* of *contexts*. As a result, we need separate definitions of tree-edit-distance equivalence for forests and contexts. Nevertheless, assuming that the weight function w satisfies the triangle inequality, we can still construct equivalent forests and contexts of size $\mathcal{O}(k^3)$ and $\mathcal{O}(k^4)$, respectively. The second challenge is that the state-of-the-art algorithm for computing the unweighted tree edit distance is randomized and takes $\mathcal{O}(n \log n + \text{poly}(k))$ time rather than $\mathcal{O}(n + \text{poly}(k))$ time. Thus, in order to achieve a deterministic linear-time kernel, we need another method for identifying large synchronizing occurrences. Our workaround is to shrink the input in multiple iterations (essentially halving the size each time) rather than in a single shot. This way, we can still obtain a kernel of size $\mathcal{O}(k^5)$, which is asymptotically as small as we would get from an optimum unweighted alignment.

Periodicity in Trees Intuitively, the two types of periodicity in trees correspond to the two ways to interpret strings as trees. For a string X , the *horizontal embedding* constructs a tree with $|X|$ leaves attached to the root and labeled by subsequent characters of X , whereas the *vertical embedding* constructs a path with $|X|$ nodes labeled by subsequent characters of X . Similarly, forest algebras (see [BW08] for a survey) in formal language theory involve two natural monoids: a *horizontal monoid* of forests (with concatenation, denoted \cdot) and a *vertical monoid* of contexts (with composition, denoted \star). A context can be defined as a tree with a single *hole* in some leaf, and contexts can be composed by placing one of them in the hole of the other. Moreover, placing a forest in the hole of a context yields a forest. In order to formalize these notions and easily port

combinatorial and algorithmic tools designed for strings, we interpret forests as balanced strings of parentheses; see Section 3.1.

Following [DGH⁺22], a horizontal power is the concatenation of multiple copies of the same forest, whereas a vertical power is the composition of multiple copies of the same context; see Fig. 3 for an example. More specifically, we say that a forest contains horizontal k -periodicity if it has a subforest of the form Q^{4k+1} for some forest Q of size $|Q| \leq 4k$, whereas a context contains vertical k -periodicity if it can be expressed as a composition of several contexts, including Q^{6k+1} for some context Q of size $|Q| \leq 8k$.

Tree-Edit-Distance Equivalent Forests The first ingredient of our algorithm for weighted tree edit distance is a linear-time procedure that, given a forest P , constructs an equivalent forest of size $\mathcal{O}(k^3)$. The first phase of this subroutine eliminates horizontal k -periodicity: as long as the processed forest contains a subforest of the form Q^{4k+1} with $|Q| \in [1..4k]$, this subforest is replaced by Q^{4k} . As shown in Lemma 3.5, the forests Q^{4k+1} and Q^{4k} are equivalent, so this step preserves equivalence with the input forest P . An efficient implementation of this phase relies on the fact that, if P is interpreted as a string, then horizontal k -periodicity can be interpreted as a substring of the form Q^{4k+1} for a sufficiently short *balanced* string Q . Thus, we can reuse Lemma 2.13 to obtain a forest equivalent with P that avoids horizontal k -periodicity.

In Lemma 3.7, we show the equivalence of any two forests of size at least $74k^3$ that avoid horizontal k -periodicity.² Based on this result, if horizontal periodicity reduction yields a forest of size at least $74k^3$, we return a canonical forest of size exactly $74k^3$; see Lemma 3.17 for details.

Tree-Edit-Distance Equivalent Contexts Our next ingredient is a linear-time algorithm that, given a context P , constructs an equivalent context of size $\mathcal{O}(k^4)$. First, we use the previous procedure for every maximal forest in P (that does not contain the hole). Then, we eliminate vertical k -periodicity: as long as P contains a context of the form Q^{6k+1} with $|Q| \in [1..8k]$, this context is replaced by Q^{6k} . As shown in Lemma 3.10, the contexts Q^{6k+1} and Q^{6k} are equivalent, so this step preserves equivalence with the input context P . For an efficient implementation, the *spine*, i.e., the path from the root of P to the hole, is interpreted as a string, with each character encoding the label of the underlying node and the subtrees attached there to the left and to the right of the spine. This way, vertical- k periodicity can be interpreted as periodicity in the constructed string, and hence Lemma 2.13 can be used again.

In Lemma 3.12, we show the equivalence of any two contexts of size at least $578k^4$ that avoid vertical k -periodicity and subforests of size more than $74k^3$. Thus, if vertical periodicity reduction yields a context of size at least $578k^4$, we replace it with a canonical context of size exactly $578k^4$; see Lemma 3.18 for details.

Linear-Time Kernel As for strings, in order to apply the notion of tree-edit-distance equivalence, we need to identify synchronized occurrences of forests and contexts within the input forests F and G . As mention above, in order to obtain a deterministic linear-time kernel, we cannot use the algorithm of [DGH⁺22] to obtain a tree alignment mapping F to G with at most k edits. Instead, we develop an iterative workaround. At each step, we decompose F into $\mathcal{O}(k)$ contexts and forests (jointly called pieces) of size at most $\frac{n}{2k}$ each; see Lemma 3.15 for details. Next, we maximize the number of pieces (from the decomposition) that admit disjoint synchronized occurrences in G ; Lemma 3.16 implements this step in $\mathcal{O}(n + k^4)$ time using dynamic programming. If $\text{ted}(F, G) \leq k$,

²This statement is stronger than its counterpart for strings, Lemma 2.11, because we now assume that the weight function w satisfies the triangle inequality.

then no more than k of the pieces are left unmatched (an optimal alignment may edit at most k pieces). We replace the matched pieces with equivalent pieces of size $\mathcal{O}(k^5)$, obtaining forests of size at most $\frac{n}{2} + \mathcal{O}(k^5)$, where the first term corresponds to the unmatched pieces; see Theorem 3.19. As long as $n = \omega(k^5)$, this procedure essentially halves the input size. Hence, as shown in Corollary 3.20, this still yields a linear-time algorithm producing forests F' and G' of size $\mathcal{O}(k^5)$ such that $\text{ted}_{\leq k}^w(F', G') = \text{ted}_{\leq k}^w(F, G)$.

Once we have such a kernel (F', G') of size $\mathcal{O}(k^5)$, we can run the cubic-time weighted edit-distance algorithm [DMRW10] to compute $\text{ted}_{\leq k}^w(X', Y') = \text{ted}_{\leq k}^w(X, Y)$ in $\mathcal{O}(k^{15})$ time, for a total runtime of $\mathcal{O}(n + k^{15})$. Additionally, we significantly improve the state-of-the-art of the unweighted tree edit distance problem by using the $\mathcal{O}(nk^2 \log n)$ -time algorithm from [AJ21], which gives us a total runtime of $\mathcal{O}(n + k^7 \log k)$ for unweighted tree edit distance.

1.3.3 Weighted Dyck Edit Distance

In the final section of our paper, the weighted Dyck edit distance algorithm follows a similar approach to that of the string and tree edit distance algorithm. However, many of the proofs and details are specific to Dyck edit distance problem and come with their own set of intricacies and difficulties that we outline in the following.

Given a string X over an alphabet $\Sigma = T \cup \bar{T}$ (where T and \bar{T} are the sets of opening and closing parentheses, respectively), an integer $k \in \mathbb{Z}_+$, and a skewmetric weight function w representing the cost of each edit operation (parenthesis insertion, deletion, and substitution), our objective is to compute the minimum weight of a sequence of edits that convert X to a well-parenthesized expression over Σ provided the total weight of all edits is bounded by k . In this work we design a deterministic algorithm that achieves this goal in $\mathcal{O}(n + k^{12})$ time. For the unweighted counterpart of this problem, the recent solution of [FGK⁺22b, Dür22] computes the Dyck edit distance in time $\mathcal{O}(n + k^{4.5442})$. That algorithm, consistently with its predecessors [BO16, FGK⁺22a], starts with a greedy preprocessing step that exhaustively removes any two adjacent characters $X[i]X[i+1]$ such that $X[i]$ is an opening parenthesis and $X[i+1]$ is a closing parenthesis of the same type. Following a simple argument, it can be shown that the Dyck edit distance of the preprocessed string stays exactly the same as the input string X .

Preprocessing We tried to follow a similar approach for the weighted version, but it turns out that such a simple analysis is not enough to construct a reduced string. For example, let the input string be $(\{(\{)$. For a general weight function w , it is not evident that the optimal matching should always match the last two parentheses. In fact, if we consider a weight function where the cost of substituting $\{$ with $)$ is 10 whereas the cost of substituting $($ with $\}$ is 5, then any optimal matching should match the first and the last parentheses instead of the last two. Thus, in this work, we consider our weight function w to be a skewmetric. Formally, we assume that w satisfies the triangle inequality and skew-symmetry, that is, $w(p_1, p_2) = w(\bar{p}_2, \bar{p}_1)$ holds for all $p_1, p_2 \in \Sigma \cup \{\varepsilon\}$, where \bar{p} is the parenthesis complementary to p (and $\bar{\varepsilon} = \varepsilon$). Following this property of w , we show that one can apply a similar greedy preprocessing (as described for the unweighted version) to reduce X to a string X' while preserving the weighted Dyck edit distance. Our argument is substantially more elaborate, though, and follows a case-by-case analysis depending on the structure of the other alternate alignments (Claim 4.7). Nevertheless, it is trivial to observe the greedy preprocessing can be done in linear time.

Dyck-Edit-Distance Equivalent Strings Next, following a similar strategy as described for string edit distance, we further reduce X' to generate a string X'' of length $\mathcal{O}(k^4)$ while preserving

the weighted Dyck edit distance. For this first we introduce the concept of k -synchronicity. A substring P containing only opening parentheses and a substring \overline{P} containing only closing parentheses are k -synchronized if \overline{P} appears after P , they are of same length and their height difference is at most $2k$. Following this and the non-crossing property of Dyck matching, first we argue that if the lengths of P, \overline{P} are large and the distance is bounded by k , then there exist a substring $\ell \in P$ that is matched with a substring $\ell' \in \overline{P}$ in the optimal alignment (we fix one for the analysis purpose). Now if we replace P with $P \setminus \ell$ and P' with $P' \setminus \ell'$ then in the resulting string the distance stays the same (Fact 4.12). Following this, for any two k -synchronized substrings P, \overline{P} , we can reduce their periodicity as follows: if $P = Q^e$ and $\overline{P} = \overline{Q}^e$, (where Q is a primitive string with large exponent e) then at least one occurrence of Q is matched with its reverse complement counterpart \overline{Q} in \overline{P} . Thus, we can remove the matched part while not changing the distance, and it reduces the exponent by one. Repeat this until e become small (Lemma 4.13).

Next assuming that P, \overline{P} avoid periodicity, it can be shown that there exists a pair of indices $i, j \in [0..78k^3]$ such that $P[i]$ is matched with $\overline{P}[|P|-1-i]$ and $P[|P|-1-j]$ is matched with $\overline{P}[j]$ in the optimal alignment. Thus, following the fact that $|P| = |\overline{P}|$ and the non-crossing property of the Dyck optimal alignment, all the indices between i and $|P| - 1 - j$ are also matched, and thus removing these matched characters from both P, \overline{P} does not affect the Dyck edit distance. Consequently, we replace each k -synchronized pairs with substrings of length just $156k^3$ (replace P, \overline{P} with their first and last $78k^3$ characters) to generate a string X'' such that the weighted Dyck edit distance of X and X'' is the same (Lemma 4.15, Corollary 4.17).

Linear-Time Kernel Lastly, we show if the distance is bounded by k , then X can be partitioned in time $\mathcal{O}(n + k^5)$ into $\mathcal{O}(k)$ disjoint k -synchronized pairs of substrings (plus $\mathcal{O}(k)$ individual characters) and thus the total length of X'' is bounded by $\mathcal{O}(k^4)$. Start by preprocessing input string X to generate X' . Next, we check if $\text{ded}(X') \leq k$ and, if so, we compute an unweighted optimal Dyck alignment \mathcal{M} of X' in time $\mathcal{O}(n + k^5)$ [FGK⁺22a]. Then, we argue any pair of substrings of X' that are matched by \mathcal{M} are k -synchronized. Thus, using \mathcal{M} , we identify the set of maximal substrings from T^* and \overline{T}^* that are matched by \mathcal{M} . A substring is maximal in a sense that either the substring itself or its matched counterpart can not be extended to the right or left without paying an edit. As unweighted Dyck edit distance is no more than the weighted version and hence, assuming cost of \mathcal{M} is bounded by k , we can show string X' can be partitioned into $\mathcal{O}(k)$ different k -synchronized pairs. Also, these maximal fragments can be found in linear time with a left-to-right scan of X' . Subsequently, we create a string X'' from X' as follows: (i) for each k -synchronized pairs we reduce them following the algorithm as discussed above and add two corresponding strings each of length $\mathcal{O}(k^3)$ (ii) add all the characters that are edited by \mathcal{M} just the same to X'' (Theorem 4.19).

Finally, we compute the weighted Dyck edit distance of X'' using the dynamic program algorithm of [Mye95] in time $\mathcal{O}(k^{12})$.

2 String Edit Distance

2.1 Preliminaries

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 i th character of Y with $Y[i]$. 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 integers $0 \leq i \leq j \leq |Y|$. 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 .

Theorem 2.1 (LCE queries [LV88, FFM00]). *Strings X, Y can be preprocessed in linear time so that the following longest common extension (LCE) queries can be answered in $\mathcal{O}(1)$ time: given positions $x \in [0..|X|]$ and $y \in [0..|Y|]$, compute the largest ℓ such that $X[x..x+\ell) = Y[y..y+\ell)$.*

As mentioned in Section 1.3, high-power periodicity plays a key role in our algorithms, which we may now formally define for strings here. 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 $\text{per}(Y)$ we denote the smallest period of Y . The exponent of a string Y is defined as $\text{exp}(Y) := \frac{|Y|}{\text{per}(Y)}$, and we say that a string Y is *periodic* if $\text{exp}(Y) \geq 2$.

Theorem 2.2 (2-Period queries [KRRW15, BII+17]). *A string X can be preprocessed in linear time so that one can decide in constant time whether any given fragment $X[i..j)$ is periodic and, if so, compute its shortest period $\text{per}(X[i..j))$.*

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

2.2 Edit-Distance Alignments and Weighted Edit Distance

In this subsection, we discuss alignments and their weighted cost, which provide a formal way to describe a sequence of edits needed to transform a string X into Y .

Definition 2.3. A sequence $\mathcal{A} = (x_t, y_t)_{t=0}^m$ is an *alignment* of a fragment $X[x..x')$ onto a fragment $Y[y..y')$ if $(x_0, y_0) = (x, y)$, $(x_m, y_m) = (x', y')$, 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 $t \in [0..m)$. The set of all alignments of $X[x..x')$ onto $Y[y..y')$ is denoted with $\mathbf{A}(X[x..x'), Y[y..y'])$.

Given an alignment $\mathcal{A} = (x_t, y_t)_{t=0}^m \in \mathbf{A}(X[x..x'), Y[y..y'])$, for every $t \in [0..m)$, we say that

- \mathcal{A} *deletes* $X[x_t]$ if $(x_{t+1}, y_{t+1}) = (x_t + 1, y_t)$.
- \mathcal{A} *inserts* $Y[y_t]$ if $(x_{t+1}, y_{t+1}) = (x_t, y_t + 1)$.
- \mathcal{A} *aligns* $X[x_t]$ to $Y[y_t]$, denoted by $X[x_t] \sim_{\mathcal{A}} Y[y_t]$, if $(x_{t+1}, y_{t+1}) = (x_t + 1, y_t + 1)$.
- \mathcal{A} *matches* $X[x_t]$ with $Y[y_t]$, denoted by $X[x_t] \simeq_{\mathcal{A}} Y[y_t]$, if $X[x_t] \sim_{\mathcal{A}} Y[y_t]$ and $X[x_t] = Y[y_t]$.
- \mathcal{A} *substitutes* $X[x_t]$ for $Y[y_t]$ if $X[x_t] \sim_{\mathcal{A}} Y[y_t]$ but $X[x_t] \neq Y[y_t]$.

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

Example 2.4. For an example of an alignment, consider strings $X = \mathbf{abc}$ and $Y = \mathbf{bd}$. One optimal alignment \mathcal{A} might be $\{(0, 0), (1, 0), (2, 1), (3, 2)\}$. The pairs $(0, 0), (1, 0)$ represent a deletion of $X[0] = \mathbf{a}$ by \mathcal{A} . The pairs $(1, 0), (2, 1), (3, 2)$ signify that \mathcal{A} aligns $X[1..2] \sim_{\mathcal{A}} Y[0..1]$, i.e. $\mathbf{bc} \sim_{\mathcal{A}} \mathbf{bd}$. Moreover, $X[1]$ is matched to $Y[0]$ since $X[1] = Y[0] = \mathbf{b}$ while $X[2]$ is substituted for $Y[1]$ since $X[2] = \mathbf{c} \neq \mathbf{d} = Y[1]$.

For an alphabet Σ , we define $\bar{\Sigma} := \Sigma \cup \{\varepsilon\}$, where ε is the empty string over Σ . We say that a function $w : \bar{\Sigma} \times \bar{\Sigma} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ is a *weight function* if $w(a, a) = 0$ holds for all $a \in \bar{\Sigma}$. The *cost* of an alignment $\mathcal{A} \in \mathbf{A}(X[x..x'), Y[y..y'])$ with respect to a weight function w , denoted $\text{ed}_{\mathcal{A}}^w(X[x..x'), Y[y..y'])$, is defined as the total cost of edits that \mathcal{A} performs, where:

- the cost of deleting $X[x]$ is $w(X[x], \varepsilon)$,
- the cost of inserting $Y[y]$ is $w(\varepsilon, Y[y])$,
- the cost of substituting $X[x]$ for $Y[y]$ is $w(X[x], Y[y])$.

The *width* of an alignment $(x_t, y_t)_{t=0}^m \in \mathbf{A}(X[x..x'], Y[y..y'])$ is defined as $\max_{t=0}^m |x_t - y_t|$.

We usually consider alignments of the entire string $X[0..|X|]$ onto the entire string $Y[0..|Y|]$, and we denote the set of all such alignment with $\mathbf{A}(X, Y) = \mathbf{A}(X[0..|X|], Y[0..|Y|])$. The weighted edit distance of strings $X, Y \in \Sigma^*$ with respect to a weight function w is defined as $\text{ed}^w(X, Y) = \min_{\mathcal{A} \in \mathbf{A}(X, Y)} \text{ed}_{\mathcal{A}}^w(X, Y)$. For $k \in \mathbb{R}_{\geq 0}$, we also denote

$$\text{ed}_{\leq k}^w(X, Y) = \begin{cases} \text{ed}^w(X, Y) & \text{if } \text{ed}^w(X, Y) \leq k, \\ \infty & \text{otherwise.} \end{cases}$$

In the literature, the (weighted) edit distance of X and Y is sometimes defined as the minimum cost of a sequence of edits that transform X into Y . As shown in the following fact (whose technical proof is deferred to Appendix A), this sequence-based view is equivalent to our alignment-based view provided that w is a *quasimetric*, that is, it satisfies the triangle inequality $w(a, b) + w(b, c) \geq w(a, c)$ for every $a, b, c \in \bar{\Sigma}$. The assumption of w being quasimetric can be made without loss of generality in the sequence-based view (a single character can be edited multiple times, so one can replace w by its distance closure without affecting the edit distances). Our alignment-based view, on the other hand, is more general and captures weighted edit distances violating the triangle inequality.

Fact 2.5. *If w is a quasimetric on $\bar{\Sigma}$, then ed^w is a quasimetric on Σ^* . In this case, $\text{ed}^w(X, Y)$ can be equivalently defined as the minimum cost of a sequence of edits transforming X into Y .*

Although our algorithm for strings works for any weight function, its tree and Dyck counterparts assume that w is a quasimetric. Specifically, they rely on the following fact proved in Appendix A.

Fact 2.6. *Consider a string X and its fragment $X[i..j]$. Then, for every quasimetric w , we have $\text{ed}^w(X, X[i..j]) = \text{ed}^w(X[0..i] \cdot X[j..|X|], \varepsilon)$.*

While our main results are on the weighted version of edit distance, our algorithm relies on unweighted edit distance procedures as well. If w is the discrete metric on $\bar{\Sigma}$ (that is, for every $a, b \in \bar{\Sigma}$, we have $w(a, b) = 0$ if $a = b$ and $w(a, b) = 1$ otherwise), then we drop the superscript w in ed^w and $\text{ed}_{\mathcal{A}}^w$. This yields the unit-cost edit distance (also known as the unweighted edit distance or the Levenshtein distance). We consider weight function w to be *normalized* that is $w(a, b) \geq 1$ holds for all $a, b \in \bar{\Sigma}$ with $a \neq b$. In this case, $\text{ed}_{\mathcal{A}}^w(X, Y) \geq \text{ed}_{\mathcal{A}}(X, Y)$ holds for all strings $X, Y \in \Sigma^*$ and alignments $\mathcal{A} \in \mathbf{A}(X, Y)$.

Given an alignment $\mathcal{A} = (x_t, y_t)_{t=0}^m \in \mathbf{A}(X, Y)$, 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 $X[x_{\ell}..x_r] \sim_{\mathcal{A}} Y[y_{\ell}..y_r]$. In this case, for any weight function w , we write $\text{ed}_{\mathcal{A}}^w(X[x_{\ell}..x_r], Y[y_{\ell}..y_r])$ to denote the cost of the induced alignment of $X[x_{\ell}..x_r]$ onto $Y[y_{\ell}..y_r]$. If $\text{ed}_{\mathcal{A}}^w(X[x_{\ell}..x_r], Y[y_{\ell}..y_r]) = 0$, we say that \mathcal{A} *matches* $X[x_{\ell}..x_r]$ with $Y[y_{\ell}..y_r]$, denoted $X[x_{\ell}..x_r] \simeq_{\mathcal{A}} Y[y_{\ell}..y_r]$.

Fact 2.7. *Consider $k \in \mathbb{Z}_{\geq 0}$, strings X, Y , and an alignment $\mathcal{A} \in \mathbf{A}(X, Y)$ of cost $\text{ed}_{\mathcal{A}}(X, Y) \leq k$. Then, the string X can be partitioned into at most k individual characters (that \mathcal{A} deletes or substitutes) and at most $k + 1$ fragments that \mathcal{A} matches perfectly to fragments of Y .*

Proof. Let $\mathcal{A} = (x_t, y_t)_{t=0}^m$ and let $t_1 < \dots < t_e$ be the indices in $[0..m]$ corresponding to edits in \mathcal{A} . Then, the maximal fragments that \mathcal{A} matches perfectly are $X[0..x_{t_1}]$, $X[x_{t_i+1}..x_{t_{i+1}}]$ for $i \in [1..e)$, and $X[x_{t_e+1}..|X|]$. Moreover, \mathcal{A} deletes or substitutes $X[x_{t_i}]$ for every $i \in [1..e]$ such that $x_{t_{i+1}} > x_{t_i}$. Each edit contributes one unit to the cost of \mathcal{A} , so the decomposition contains at most $e \leq k$ edited characters and $e + 1 \leq k + 1$ fragments matched perfectly. \square

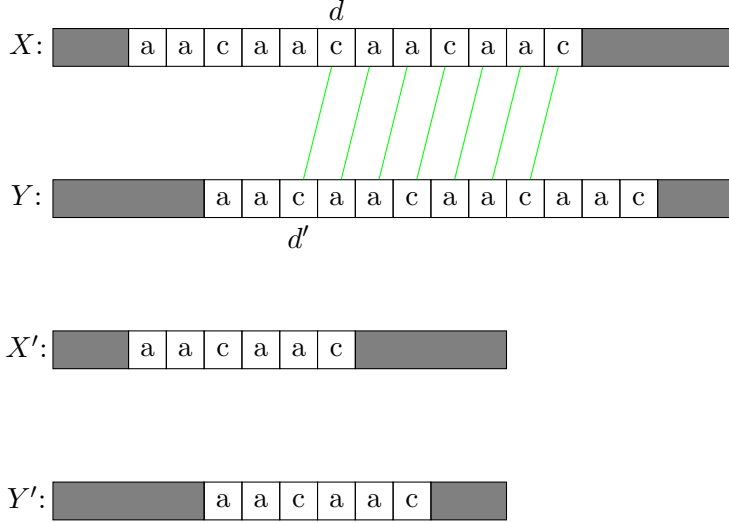


Figure 1: Periodicity reduction in X and Y with an optimal alignment depicted by lines connecting characters of the two strings. At indices d, d' in X and Y respectively, the periodic substring is fully aligned (depicted by green lines), and so, we may reduce the power of these periodic substrings to construct X' and Y' with $\text{ed}(X, Y) = \text{ed}(X', Y')$.

2.3 Combinatorial Foundations

Before giving our algorithms for weighted string edit distance, we discuss edit distance equivalent substrings, one of our main technical contributions.

Definition 2.8. For $k \in \mathbb{Z}_{\geq 0}$ and a weight function w , strings P, P' are called $\text{ed}_{\leq k}^w$ -equivalent if

$$\text{ed}_{\leq k}^w(X, Y) = \text{ed}_{\leq k}^w(X[0..p_X] \cdot P' \cdot X[p_X + |P|..|X|], Y[0..p_Y] \cdot P' \cdot Y[p_Y + |P|..|Y|])$$

holds for all strings X and Y in which P occurs at positions p_X and p_Y , respectively, satisfying $|p_X - p_Y| \leq k$. We say that such occurrences of P in X and Y are k -synchronized occurrences.

First, we prove that changing the power of a periodic substring does not change the edit distance cost of any synchronized occurrences of that substring. Second, we prove that we only need to consider the small prefixes and suffixes of substrings when calculating the edit distance. In both cases, we are able to show that any optimal alignment must align such large substrings (first periodic and then non-periodic) that have synchronized occurrences in edit distance instances, and so, we do not have to worry about most of these large substrings when calculating edit distance. See Fig. 1 and Fig. 2 for examples of these two edit distance equivalent steps.

Lemma 2.9. Let $k \in \mathbb{Z}_+$, let Q be a string, and let $e, e' \in \mathbb{Z}_{\geq 4k}$. Then, Q^e and $Q^{e'}$ are $\text{ed}_{\leq k}^w$ -equivalent for every weight function w .

Proof. We assume without loss of generality that Q is primitive. (If $Q = R^m$ for $m \in \mathbb{Z}_{\geq 2}$, then $Q^e = R^{me}$ and $Q^{e'} = R^{me'}$ can be interpreted as powers of R rather than powers of Q .) Suppose that Q^e occurs in strings X and Y at positions p_X and p_Y , respectively, satisfying $|p_X - p_Y| \leq k$. Denote $X' = X[0..p_X] \cdot Q^{e'} \cdot X[p_X + |Q^e|..|X|]$ and $Y' = Y[0..p_Y] \cdot Q^{e'} \cdot Y[p_Y + |Q^e|..|Y|]$. Moreover, let $q = |Q|$ and let $\mathcal{A} \in \mathcal{A}(X, Y)$ be an alignment such that $\text{ed}^w(X, Y) = \text{ed}_{\mathcal{A}}^w(X, Y) \leq k$.

Claim 2.10. There exist $i_X, i_Y \in [0..3k]$ such that

$$X[p_X + i_X \cdot q..p_X + (i_X + 1) \cdot q] \simeq_{\mathcal{A}} Y[p_Y + i_Y \cdot q..p_Y + (i_Y + 1) \cdot q].$$

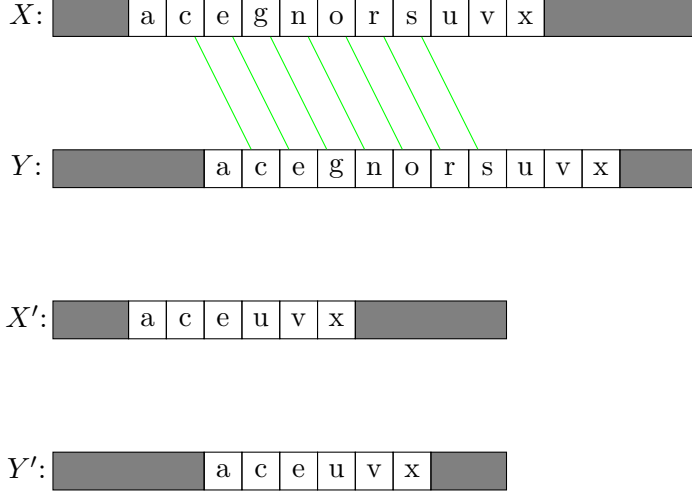


Figure 2: For any synchronized occurrences of a substring P that avoids k -periodicity, any optimal alignment (depicted by lines connecting characters of the two strings) must match most of the inner characters of P (see green lines). We can construct strings X', Y' removing these matched characters such that $\text{ed}(X, Y) = \text{ed}(X', Y')$.

Proof. Let $(t_X, t_Y) \in \mathcal{A}$ be the leftmost element of \mathcal{A} such that $t_X \geq p_X$ and $t_Y \geq p_Y$. By symmetry between X and Y , we assume without loss of generality that $t_X = p_X$. Consider the $k+1$ occurrences of Q in X starting at positions $p_X + i \cdot q$ for $i \in [0..k]$. The alignment \mathcal{A} matches at least one of them exactly; we can thus define $i_X \in [0..k]$ so that \mathcal{A} matches $X[p_X + i_X \cdot q..p_X + (i_X + 1) \cdot q]$ exactly to some fragment $Y[s_Y..s_Y + q]$. Due to $(t_X, t_Y) \in \mathcal{A}$, the non-crossing property of \mathcal{A} implies that $s_Y \geq t_Y \geq p_Y$. Moreover, since $\text{ed}_{\mathcal{A}}(X, Y) \leq k$ and $X[p_X + i_X \cdot q] \simeq_{\mathcal{A}} Y[s_Y]$, we have $s_Y \leq (p_X + i_X \cdot q) + k \leq p_X + kq + k \leq p_Y + kq + 2k \leq p_Y + 3kq$. Furthermore, since Q is primitive (i.e., distinct from all its non-trivial cyclic rotations), we conclude that $s_Y = p_Y + i_Y \cdot q$ for some $i_Y \in [0..3k]$. \square

Now, if $Q^e = X[p_X..p_X + e \cdot q] = Y[p_Y..p_Y + e \cdot q]$ is replaced with $Q^{e'}$ for $e' \geq e - 1$, we can interpret this as replacing $Q = X[p_X + i_X \cdot q..p_X + (i_X + 1) \cdot q] = Y[p_Y + i_Y \cdot q..p_Y + (i_Y + 1) \cdot q]$ with $Q^{1+e'-e}$. By Claim 2.10, \mathcal{A} can be trivially adapted without modifying its cost, and hence $\text{ed}^w(X', Y') \leq \text{ed}_{\mathcal{A}}^w(X, Y) = \text{ed}^w(X, Y)$. If $e' < e - 1$, we repeat the above argument to decrement the exponent e one step at a time, still concluding that $\text{ed}^w(X', Y') \leq \text{ed}^w(X, Y)$. In either case, the converse inequality follows by symmetry between (X, Y, e) and (X', Y', e') . \square

We say that a string avoids k -periodicity if it does not contain any substring of the form Q^{4k+1} with $|Q| \in [1..2k]$.

Lemma 2.11. *Let $k \in \mathbb{Z}_+$ and let P, P' be strings of lengths at least $42k^3$ such that $P[0..21k^3] = P'[0..21k^3]$ and $P[|P| - 21k^3..|P|] = P'[|P'| - 21k^3..|P'|]$ avoid k -periodicity. Then, P and P' are $\text{ed}_{\leq k}^w$ -equivalent for every weight function w .*

Proof. Suppose that P occurs in strings X and Y at positions p_X and p_Y , respectively, satisfying $|p_X - p_Y| \leq k$. Denote $X' = X[0..p_X] \cdot P' \cdot X[p_X + |P|..|X|]$ and $Y' = Y[0..p_Y] \cdot P' \cdot Y[p_Y + |P|..|Y|]$. Moreover, let $\mathcal{A} \in \mathbf{A}(X, Y)$ be an alignment such that $\text{ed}^w(X, Y) = \text{ed}_{\mathcal{A}}^w(X, Y) \leq k$.

Claim 2.12. *There exist $d, e \in [0..21k^3]$ such that*

$$X[p_X + d..p_X + |P| - e] \sim_{\mathcal{A}} Y[p_Y + d..p_Y + |P| - e].$$

Proof. Let us partition $X[p_X \dots p_X + 21k^3]$ into individual characters representing deletions or substitutions of \mathcal{A} and maximal fragments that \mathcal{A} matches perfectly (to fragments of Y). By Fact 2.7, the number of such maximal fragments is at most $k + 1$ and their total length is at least $21k^3 - k \geq 20k^3$. Hence, one of these fragments is of length at least $\frac{20k^3}{k+1} \geq 10k^2$. Thus, let $R := X[r_X \dots r_X + |R|]$ be a fragment of length at least $10k^2$ contained in $X[p_X \dots p_X + 21k^3]$ that \mathcal{A} matches perfectly to $Y[r_Y \dots r_Y + |R|]$. Moreover, let $r'_Y := r_X + p_Y - p_X$. If $r_Y = r'_Y$, then we set $d := r_X - p_X = r_Y - p_Y$ so that $(p_X + d, p_Y + d) \in \mathcal{A}$. Otherwise, both $Y[r_Y \dots r_Y + |R|]$ and $Y[r'_Y \dots r'_Y + |R|]$ are occurrences of R in Y . Moreover, $0 < |r_Y - r'_Y| \leq |r_Y - r_X| + |r'_Y - r_X| \leq \text{ed}_{\mathcal{A}}^w(X, Y) + |p_Y - p_X| \leq 2k$. Hence, $\text{per}(R) \leq |r_Y - r'_Y| \leq 2k$ and $\text{exp}(R) \geq \frac{|R|}{2k} \geq 4k + 1$. Since $Y[r'_Y \dots r'_Y + |R|]$ is contained in $Y[p_Y \dots p_Y + 21k^3] = P[0 \dots 21k^3]$, this contradicts the assumption about $P[0 \dots 21k^3]$ avoiding k -periodicity.

A symmetric argument shows that $(p_X + |P| - e, p_Y + |P| - e)$ holds for some $e \in [0 \dots 21k^3]$, which lets us conclude that $X[p_X + d \dots p_X + |P| - e] \sim_{\mathcal{A}} Y[p_Y + d \dots p_Y + |P| - e]$. \square

By Claim 2.12, we have $X[p_X + d \dots p_X + |P| - e] \sim_{\mathcal{A}} Y[p_Y + d \dots p_Y + |P| - e]$. Both fragments match $P[d \dots |P| - e]$, so the optimality of \mathcal{A} guarantees $X[p_X + d \dots p_X + |P| - e] \simeq_{\mathcal{A}} Y[p_Y + d \dots p_Y + |P| - e]$. Hence, if $P = X[p_X \dots p_X + |P|] = Y[p_Y \dots p_Y + |P|]$ is replaced with P' , we can interpret this as $P[d \dots |P| - e] = X[p_X + d \dots p_X + |P| - e] = Y[p_Y + d \dots p_Y + |P| - e]$ with $P'[d \dots |P| - e]$. Since $X[p_X + d \dots p_X + |P| - e] \simeq_{\mathcal{A}} Y[p_Y + d \dots p_Y + |P| - e]$, the alignment \mathcal{A} can be trivially adapted without modifying its cost, and therefore $\text{ed}^w(X', Y') \leq \text{ed}_{\mathcal{A}}^w(X, Y) = \text{ed}^w(X, Y)$. The converse inequality follows by symmetry between (X, Y, P) and (X', Y', P') . \square

2.4 Algorithm

The following lemma lets us transform any string P to a string P' that avoids k -periodicity and is $\text{ed}_{\leq k}^w$ -equivalent to P for every weight function w . It is stated in a general form so that it can be reused in subsequent sections.

Lemma 2.13. *Let $e \in \mathbb{Z}_+$ and let \mathcal{Q} be a family of primitive strings of length at most e . There is an algorithm that repeatedly transforms an input string P by replacing an occurrence of Q^{e+1} (for some $Q \in \mathcal{Q}$) with an occurrence of Q^e , arriving at a string P' that does not contain any occurrence of Q^{e+1} (for any $Q \in \mathcal{Q}$). Moreover, this algorithm can be implemented in linear time using a constant-time oracle that tests whether a given primitive fragment of P belongs to \mathcal{Q} .*

Proof. At preprocessing, we construct data structures for LCE and 2-Period queries in P ; see Theorems 2.1 and 2.2. In the main phase, our algorithm scans the string P from left to right maintaining a string R and an index $r \in [0 \dots |P|]$ such that $R \cdot P[r \dots |P|]$:

- is obtained from P by repeatedly replacing an occurrence of Q^{e+1} (for some $Q \in \mathcal{Q}$) with an occurrence of Q^e ,
- does not contain any occurrence of Q^{e+1} (for $Q \in \mathcal{Q}$) starting at position smaller than $|R|$.

We initialize the process with $R := \varepsilon$ and $r := 0$. At each step, we test if $P[r \dots r + 2q]$ is periodic; if so, we retrieve its shortest period q ; otherwise, we set $q := 1$. Then, we further check whether $P[r \dots r + q] \in \mathcal{Q}$ and $P[r \dots r + eq] = P[r + q \dots r + q + eq]$. If both tests are successful, we move the index r to position $r + q$. Otherwise, we append $P[r]$ to R and increment r .

Let us analyze the correctness of this algorithm. First, suppose that $P[r \dots |P|]$ does not have a prefix of the form Q^{e+1} for any $Q \in \mathcal{Q}$. In particular, $P[r \dots r + q] \notin \mathcal{Q}$ or $m < eq$. Thus, our algorithm appends $P[r]$ to R and increments r . The invariant remains satisfied because $R \cdot P[r \dots |P|]$ did not change and $P[r \dots |P|]$ had no prefix of the form Q^{e+1} for any $Q \in \mathcal{Q}$.

Algorithm 1: Caps the exponent of every power of $Q \in \mathcal{Q}$ occurring in P to at most e .

```

1 PeriodicityReduction( $P, e, \mathcal{Q}$ ):
2    $R \leftarrow \varepsilon$ ;
3    $r \leftarrow 0$ ;
4   while  $r < |P|$  do
5     if  $r + 2e \leq |P|$  and  $P[r..r + 2e]$  is periodic then  $q \leftarrow \text{per}(P[r..r + 2e])$ ;
6     else  $q \leftarrow 1$ ;
7      $m \leftarrow \max\{\ell : P[r..r + \ell] = P[r + q..r + q + \ell]\}$ ;
8     if  $m \geq eq$  and  $P[r..r + q] \in \mathcal{Q}$  then
9        $r \leftarrow r + q$ ;
10    else
11       $R \leftarrow R \cdot P[r]$ ;
12       $r \leftarrow r + 1$ ;
13  return  $R$ ;
```

Next, suppose that $P[r..|P|)$ has a prefix of the form Q^{e+1} for some $Q \in \mathcal{Q}$. If $|Q| \neq 1$, then $|Q|$ is the shortest period of $P[r..r + 2e)$ because Q is primitive and $|Q| \leq e$. If $|Q| = 1$, on the other hand, then $P[r..r + 2e)$ either has period 1 or at least $e + 2$. In all cases, the algorithm correctly identifies $q = |Q|$. Moreover, the subsequent tests whether $P[r..r + q)$ belongs to \mathcal{Q} and $m \geq eq$ are successful. Hence, the algorithm transforms $R \cdot P[r..|P|)$ into $R \cdot P[r + q..|P|)$, which is a valid operation because the prefix Q^{e+1} of $P[r..|P|)$ is replaced with the prefix Q^e of $P[r + q..|P|)$. Thus, it remains to prove that $R \cdot P[r + q..|P|)$ does not contain any occurrence of \hat{Q}^{e+1} (for any $\hat{Q} \in \mathcal{Q}$) starting at position smaller than $|R|$. Since $R \cdot P[r..|P|)$ did not contain such an occurrence, the occurrence of \hat{Q}^{e+1} would need to end at position $|R| + m$ or larger. The fragment $P[r + q..r + q + m)$ thus has periods q and $\hat{q} := |\hat{Q}|$. Moreover, by primitivity of Q and \hat{Q} , the Periodicity Lemma [FW65] implies $q = \hat{q}$ due to $m \geq eq \geq e + q - 1 \geq \hat{q} + q - 1$. However, this means that q is a period of $P[r..r + q + m)$, contradicting the definition of m .

The overall running time is linear, including the preprocessing and the query time of the data structures of Theorems 2.1 and 2.2, because each iteration of the **while** loop costs constant time. \square

Corollary 2.14. *There exists a linear-time algorithm that, given a string P and an integer $k \in \mathbb{Z}_+$, constructs a string of length at most $42k^3$ that is $\text{ed}_{\leq k}^w$ -equivalent to P for every weight function w .*

Algorithm 2: Construct a string of length at most $42k^3$ that is $\text{ed}_{\leq k}^w$ -equivalent to P .

```

1 StringReduction( $P, k$ ):
2    $P' \leftarrow \text{PeriodicityReduction}(P, 4k, \{Q \in \Sigma^+ : |Q| \leq 2k \text{ and } Q \text{ is primitive}\})$ ;
3   if  $|P'| \geq 42k^3$  then return  $P'[0..21k^3) \cdot P'[(|P'| - 21k^3)..|P'|)$ ;
4   else return  $P'$ ;
```

Proof. We set $P' := \text{PeriodicityReduction}(P, 4k, \mathcal{Q})$ with \mathcal{Q} consisting of all primitive strings of length in $[1..2k]$. We return $P'' := P'[0..21k^3) \cdot P'[(|P'| - 21k^3)..|P'|)$ or P' depending on whether $|P'| \geq 42k^3$ or not. By Lemmas 2.9 and 2.13, the string P' is $\text{ed}_{\leq k}^w$ -equivalent to P and avoids k -periodicity. Thus, if $|P'| \leq 42k^3$, then the algorithm is correct. Otherwise, Lemma 2.11 implies that P'' is $\text{ed}_{\leq k}^w$ -equivalent to P' (and, by transitivity, to P) because $P'[0..21k^3)$ and

$P'(|P'| - 21k^3 \dots |P'|)$ avoid k -periodicity. Due to Lemma 2.13, the running time is linear (a primitive fragment belongs to \mathcal{Q} if and only if its length does not exceed $2k$, which takes $\mathcal{O}(1)$ time to test). \square

Theorem 2.15. *There exists a linear-time algorithm that, given strings X, Y and an integer $k \in \mathbb{Z}_+$, constructs strings X', Y' of lengths at most $85k^4$ such that $\text{ed}_{\leq k}^w(X, Y) = \text{ed}_{\leq k}^w(X', Y')$ holds for every weight function w .*

Algorithm 3: Construct strings X', Y' of length at most $85k^4$ such that $\text{ed}_{\leq k}^w(X, Y) = \text{ed}_{\leq k}^w(X', Y')$

```

1 StringKernel( $X, Y, k$ ):
2   if  $|X| \leq 85k^4$  and  $|Y| \leq 85k^4$  then return  $(X, Y)$ ;
3   if  $\text{ed}(X, Y) > k$  then return  $(a^{k+1}, \varepsilon)$  for some  $a \in \Sigma$ ;
4   Let  $(x_t, y_t)_{t=0}^m \in \mathcal{A}(X, Y)$  be an alignment satisfying  $\text{ed}_{\mathcal{A}}(X, Y) \leq k$ ;
5    $X', Y', P \leftarrow \varepsilon$ ;
6   for  $t \leftarrow 0$  to  $m$  do
7     if  $t < m$  and  $x_{t+1} > x_t$  and  $y_{t+1} > y_t$  and  $X[x_t] = Y[y_t]$  then
8       |  $P \leftarrow P \cdot X[x_t]$ 
9     else
10      |  $P \leftarrow \text{StringReduction}(P, k)$ ;
11      |  $X' \leftarrow X' \cdot P$ ;
12      |  $Y' \leftarrow Y' \cdot P$ ;
13      |  $P \leftarrow \varepsilon$ ;
14      | if  $t < m$  and  $x_{t+1} > x_t$  then  $X' \leftarrow X' \cdot X[x_t]$ ;
15      | if  $t < m$  and  $y_{t+1} > y_t$  then  $Y' \leftarrow Y' \cdot Y[y_t]$ ;
16   return  $(X', Y')$ 

```

Proof. Our procedure is implemented as Algorithm 3. First, if X and Y are already of length at most $85k^4$, then we return X and Y unchanged. If $\text{ed}(X, Y) > k$, we return strings a^{k+1} and ε , where $a \in \Sigma$ is an arbitrary character. If $\text{ed}(X, Y) \leq k$, we construct an alignment $\mathcal{A} := (x_t, y_t)_{t=0}^m \in \mathcal{A}(X, Y)$ of (unweighted) cost at most k . We then build the output strings X' and Y' during a left-to-right scan of the alignment \mathcal{A} : We append to X' and Y' every character of X and Y (respectively) that \mathcal{A} edits. Moreover, for every pair of maximal fragments in X and Y that \mathcal{A} matches perfectly, we apply the reduction of Corollary 2.14 and append the resulting string to both X' and Y' .

Let us now prove that the resulting instance (X', Y') satisfies $\text{ed}_{\leq k}^w(X, Y) = \text{ed}_{\leq k}^w(X', Y')$. This is trivial when the algorithm returns (X, Y) in Line 2. If $\text{ed}(X, Y) > k$, then $\text{ed}_{\leq k}(X, Y) = \infty = \text{ed}_{\leq k}(a^{k+1}, \varepsilon)$ and thus also $\text{ed}_{\leq k}^w(X, Y) = \infty = \text{ed}_{\leq k}^w(a^{k+1}, \varepsilon)$ because the weighted edit distance with a normalized weight function is at least as large as the unweighted edit distance. In the remaining case of $\text{ed}(X, Y) \leq k$, we maintain an invariant that $|X'| - |Y'| = x_t - y_t$ and $\text{ed}_{\leq k}^w(X, Y) = \text{ed}_{\leq k}^w(X' \cdot P \cdot X[x_t \dots x_m], Y' \cdot P \cdot Y[y_t \dots y_m])$ hold at the beginning of every iteration of the **for** loop as well as after every execution of Line 10 and Line 13. It is easy to see that the strings $X' \cdot P \cdot X[x_t \dots x_m]$ and $Y' \cdot P \cdot Y[y_t \dots y_m]$ change only at Line 10, when P is replaced with $\text{StringReduction}(P, k)$. The correctness of this step follows directly from the definition of $\text{ed}_{\leq k}^w$ -equivalence (Definition 2.8) since $\text{StringReduction}(P, k)$ is $\text{ed}_{\leq k}^w$ -equivalent to P .

Next, we show that the returned strings are of length at most $85k^4$. This is clear when the algorithm terminates at Line 2 or 3. Otherwise, we apply Fact 2.7 to observe that X is decomposed

into at most k characters that \mathcal{A} deletes or substitutes (which are copied to X') and at most $k + 1$ maximal fragments that \mathcal{A} matches perfectly to fragments of Y (which are copied to X' after applying `StringReduction`). By the guarantee of Corollary 2.14, we conclude that $|X'| \leq k + (k + 1) \cdot 42k^3 \leq 85k^4$. Symmetrically, we have $|Y'| \leq 85k^4$.

It remains to analyze the time complexity of our procedure. We use the Landau–Vishkin algorithm [LV88] to check whether $\text{ed}(X, Y) \leq k$ and, if so, construct the alignment \mathcal{A} . This costs $\mathcal{O}(n + k^2)$ time, which is $\mathcal{O}(n)$ because we perform this step only if $n \geq k^4 \geq k^2$. The scan of the alignment \mathcal{A} takes $\mathcal{O}(m) = \mathcal{O}(n)$ time, including the applications of Corollary 2.14, which operate on strings of total length at most n . \square

Having reduced the string lengths to $\mathcal{O}(k^4)$, we can use the classic dynamic programming [WF74] to compute $\text{ed}_{\leq k}^w(X, Y)$ in $\mathcal{O}(k^8)$ time. However, since w is a normalized, the running of [WF74] can be reduced to $\mathcal{O}(nk)$. For completeness, we describe this improvement below.

Proposition 2.16. *Given strings X, Y of length at most n , an integer $k \in \mathbb{Z}_+$, and a weight function w , the value $\text{ed}_{\leq k}^w(X, Y)$ can be computed in $\mathcal{O}(nk)$ time.*

Proof. Recall that the algorithm of [WF74] maintains a table $D[0..|X|, 0..|Y|]$ such that $D[i, j] = \text{ed}^w(X[0..i], Y[0..j])$ holds for each $i \in [0..|X|]$ and $j \in [0..|Y|]$. We have $D[0, 0] = 0$, whereas the remaining entries are constructed in $\mathcal{O}(1)$ time each using the following formula:

$$D[i, j] = \min \begin{cases} D[i-1, j] + w(X[i-1], \varepsilon) & \text{if } i > 0, \\ D[i, j-1] + w(\varepsilon, Y[j-1]) & \text{if } j > 0, \\ D[i-1, j-1] + w(X[i-1], Y[j-1]) & \text{if } i, j > 0. \end{cases} \quad (1)$$

In order to compute $\text{ed}_{\leq k}^w(X, Y)$, we use a modified table $D'[0..|X|, 0..|Y|]$ such that $D'[0, 0] = 0$, $D'[i, j] = \infty$ if $|i-j| > k$, whereas the remaining entries are computed using (1) (with D replaced by D'). A straightforward inductive argument shows that $D'[i, j] \geq D[i, j]$ holds for all $i \in [0..|X|]$ and $j \in [0..|Y|]$ and, moreover, $D[i, j] \leq k$ implies $D'[i, j] = D[i, j]$. For $|i-j| > k$, this is true because w is normalized and thus $D[i, j] = \text{ed}^w(X[0..i], Y[0..j]) \geq \text{ed}(X[0..i], Y[0..j]) \geq |i-j| > k$. For $|i-j| \leq k$, on the other hand, the argument is based on the inductive hypothesis and the fact that the weight function w has non-negative values. The entries $D'[i, j] = \infty$ for $|i-j| > k$ can be set implicitly, which reduces the running time to $\mathcal{O}(nk)$. \square

Theorem 1.1. *Given strings X, Y of length at most n , an integer $k \in \mathbb{Z}_+$, and a weight function w , the value $\text{ed}_{\leq k}^w(X, Y)$ can be computed in $\mathcal{O}(n + k^5)$ time.*

Proof. We first apply Theorem 2.15 to build strings X', Y' of length $\mathcal{O}(k^4)$ such that $\text{ed}_{\leq k}^w(X', Y') = \text{ed}_{\leq k}^w(X, Y)$. Then, we compute $\text{ed}_{\leq k}^w(X', Y')$ using Proposition 2.16. The running times of these two steps are $\mathcal{O}(n)$ and $\mathcal{O}(k^4 \cdot k) = \mathcal{O}(k^5)$, respectively, for a total of $\mathcal{O}(n + k^5)$. \square

3 Tree Edit Distance

3.1 Preliminaries

For an alphabet Σ , we define a set $\mathsf{P}_\Sigma := \bigcup_{a \in \Sigma} \{(a,)_a\}$ of parentheses with labels over Σ . A *forest* with node labels over Σ is a *balanced* string of parentheses over Σ . Formally, the set of forests with labels over Σ is defined as the smallest subset $\mathcal{F}_\Sigma \subseteq \mathsf{P}_\Sigma^*$ satisfying the following conditions:

- $\varepsilon \in \mathcal{F}_\Sigma$,

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

For a forest F , we define the set of *nodes* V_F as the set of pairs $(i, j) \in [0..|F|]$ such that $F[i]$ is an opening parenthesis, $F[j]$ is a closing parenthesis, and $F[i..j]$ is balanced. For a node $u = (i, j) \in V_F$, we denote the positions of the opening and the closing parenthesis by $o(u) := i$ and $c(u) := j$. A forest F is a tree if $(0, |F| - 1) \in V_F$.

Fact 3.1. *A forest F can be preprocessed in linear time so that one can test in constant time whether any given fragment $F[i..j]$ is balanced.*

Proof. Let us define the height function $H : [0..|F|] \rightarrow \mathbb{Z}$ so that $H(i)$ equals the number of opening parentheses in $F[0..i]$ minus the number of closing parentheses in $F[0..i]$. Since F is balanced, the fragment $F[i..j]$ is balanced if and only if $H(i) = H(j) = \min_{m \in [i..j]} H(m)$. This condition can be tested in $\mathcal{O}(1)$ time after linear-time preprocessing using range minimum queries (RMQ) [HT84]. \square

A context with node labels over Σ is a pair $C = \langle C_L; C_R \rangle \in \mathcal{P}_\Sigma \times \mathcal{P}_\Sigma$ such that $C_L \cdot C_R$ is a tree. The node set V_C of a context C is identified with the node set of the underlying tree $C_L \cdot C_R$. The *depth* of a context C is the number of nodes $u \in V_C$ whose opening parenthesis belongs to C_L and closing parenthesis belongs to C_R , that is, $o(u) < |C_L| < c(u)$.

The (vertical) composition of contexts C, D results in a context $C \star D := \langle C_L \cdot D_L; D_R \cdot C_R \rangle$. Moreover, vertical composition of a context C and a forest F results in a tree $C \star F := C_L \cdot F \cdot C_R$. A context C is primitive if it cannot be expressed as vertical composition of $e \geq 2$ copies of the same context.

A context C *occurs* in a forest F at node $u \in V_F$ if $C_L = F[o(u)..o(u) + |C_L|]$ and $C_R = F[c(u) - |C_R|..c(u)]$, or equivalently, $F[o(u)..c(u)] = C \star G$ for some forest G

3.2 Forest Alignments and Weighted Forest Edit Distance

We begin our discussion of weighted tree edit distance by formally defining forest alignments, which are similar to alignments on strings with just a few additional restrictions to make sure the alignments make valid edits on forests.

Definition 3.2. We say that an alignment $\mathcal{A} \in \mathbf{A}(F, G)$ is a *forest alignment* of forests F and G if the following *consistency* conditions are satisfied for each $u \in V_F$:

- either \mathcal{A} deletes both $F[o(u)]$ and $F[c(u)]$, or
- there exists $v \in V_G$ such that $F[o(u)] \sim_{\mathcal{A}} G[o(v)]$ and $F[c(u)] \sim_{\mathcal{A}} G[c(v)]$.

The set of all forests alignments of F onto G is denoted with $\mathbf{TA}(F, G) \subseteq \mathbf{A}(F, G)$.

Define $\overline{\mathcal{P}}_\Sigma = \mathcal{P}_\Sigma \cup \varepsilon$ and a mapping $\lambda : \overline{\mathcal{P}}_\Sigma \rightarrow \overline{\Sigma}$ such that $\lambda((a)) = \lambda(\varepsilon) = a$ for each $a \in \Sigma$, and $\lambda(\varepsilon) = \varepsilon$. For a weight function $w : \overline{\Sigma} \times \overline{\Sigma} \rightarrow \mathbb{R}_{\geq 0}$, we define a corresponding weight function $\tilde{w} : \overline{\mathcal{P}}_\Sigma \times \overline{\mathcal{P}}_\Sigma \rightarrow \mathbb{R}_{\geq 0}$ so that $\tilde{w}(p, q) = w(\lambda(p), \lambda(q))$ for all $p, q \in \overline{\mathcal{P}}_\Sigma$. The *cost* of a forest alignment $\mathcal{A} \in \mathbf{TA}(F, G)$ with respect to a weight function w is defined as $\text{ted}_{\mathcal{A}}^w(F, G) := \frac{1}{2} \text{ed}_{\mathcal{A}}^{\tilde{w}}(F, G)$. Moreover, for any two forests F, G , we define the *weighted tree edit distance* $\text{ted}^w(F, G) = \min_{\mathcal{A} \in \mathbf{TA}(F, G)} \text{ted}_{\mathcal{A}}^w(F, G)$, and for a threshold $k \in \mathbb{R}_{\geq 0}$, we set

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

The superscript is omitted if w is the discrete metric over $\overline{\Sigma}$.

Fact 3.3. *If w is a quasimetric on $\bar{\Sigma}$, then ted^w is a quasimetric on \mathcal{F}_Σ . In that case, $\text{ted}^w(F, G)$ can be equivalently defined as the minimum cost of a sequence of edits transforming F into G , where inserting a node with label b costs $w(\varepsilon, b)$, deleting a node with label a costs $w(a, \varepsilon)$, and changing a node label from a to b costs $w(a, b)$.*

Proof. Consider arbitrary forests $F, G, H \in \mathcal{F}_\Sigma$ as well as alignments $A = (x_t, y_t)_{t=0}^m \in \text{TA}(F, G)$ and $B = (\hat{y}_t, \hat{z}_t)_{t=0}^m \in \text{TA}(G, H)$. We can construct the product alignment $\mathcal{A} \otimes \mathcal{B}$ as in the proof of Fact 2.5, which has $\text{ed}_{\mathcal{A} \otimes \mathcal{B}}^w(F, H) \leq \text{ed}_A^w(F, G) + \text{ed}_B^w(G, H)$. Therefore, it remains to prove that $\mathcal{A} \otimes \mathcal{B}$ is a tree alignment.

Consider an arbitrary node $u_F \in V_F$. If \mathcal{A} deletes u_F (that is, it deletes both characters $F[o(u_F)]$ and $F[c(u_F)]$), then $\mathcal{A} \otimes \mathcal{B}$ also deletes u_F ; see Case 2 in the recursive definition of $\mathcal{A} \otimes \mathcal{B}$. The other possibility is that \mathcal{A} aligns u_F with some node $u_G \in V_G$ (that is, it aligns $F[o(u_F)]$ with $G[o(u_G)]$ and $F[c(u_F)]$ with $G[c(u_G)]$). If \mathcal{B} deletes u_G , then $\mathcal{A} \otimes \mathcal{B}$ deletes u_F ; see Case 6 in the recursive definition of $\mathcal{A} \otimes \mathcal{B}$. Finally, if \mathcal{B} aligns u_G with some node $u_H \in V_H$, then $\mathcal{A} \otimes \mathcal{B}$ aligns u_F with u_H ; see Case 7 in the recursive definition of $\mathcal{A} \otimes \mathcal{B}$. \square

3.3 Combinatorial Foundations

3.3.1 Forests

Similar to our discussion of weighted string edit distance, before giving our tree edit distance algorithms we prove the existence of small edit distance equivalent forests for synchronized occurrences of large subforests in the input instance forests.

Definition 3.4. For $k \in \mathbb{Z}_{\geq 0}$ and a weight function w , forests P, P' are called $\text{ted}_{\leq k}^w$ -equivalent if

$$\text{ted}_{\leq k}^w(F, G) = \text{ted}_{\leq k}^w(F[0..p_F] \cdot P' \cdot F[p_F + |P|..|F|], G[0..p_G] \cdot P' \cdot G[p_G + |P|..|G|])$$

holds for all forests F and G in which P occurs at positions p_F and p_G , respectively, satisfying $|p_F - p_G| \leq 2k$.

Lemma 3.5. *Let $k \in \mathbb{Z}_+$, let Q be a forest, and let $e, e' \in \mathbb{Z}_{\geq 4k}$. Then, Q^e and $Q^{e'}$ are $\text{ted}_{\leq k}^w$ -equivalent for every normalized weight function w .*

Proof. We assume without loss of generality that Q is primitive. (If $Q = R^m$ for $m \in \mathbb{Z}_{\geq 2}$, then $Q^e = R^{me}$ and $Q^{e'} = R^{me'}$ can be interpreted as powers of R rather than powers of Q .) Suppose that Q^e occurs in forests F and G at positions p_F and p_G , respectively, satisfying $|p_F - p_G| \leq 2k$. Denote $F' = F[0..p_F] \cdot Q^{e'} \cdot F[p_F + |Q^e|..|F|]$ and $G' = G[0..p_G] \cdot Q^{e'} \cdot G[p_G + |Q^e|..|G|]$. Moreover, let $q = |Q|$ and let \mathcal{A} be a forest alignment such that $\text{ted}^w(F, G) = \text{ted}_{\mathcal{A}}^w(F, G) \leq k$.

Claim 3.6. *There exist $i_F, i_G \in [0..3k]$ such that*

$$F[p_F + i_F \cdot q..p_F + (i_F + 1) \cdot q] \simeq_{\mathcal{A}} G[p_G + i_G \cdot q..p_G + (i_G + 1) \cdot q].$$

Proof. Let $(f_b, g_b) \in \mathcal{A}$ be the leftmost element of \mathcal{A} such that $f_b \geq p_F$ and $g_b \geq p_G$. By symmetry between F and G , we assume without loss of generality that $f_b = p_F$. Consider the $k+1$ occurrences of Q in F starting at positions $p_F + i \cdot q$ for $i \in [0..k]$. Since Q is balanced, the alignment \mathcal{A} (of unweighted 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 $F[p_F + i_F \cdot q..p_F + (i_F + 1) \cdot q]$ exactly to some fragment $G[g_a..g_a + q]$. By definition of b , we have $a \geq b$ and thus $g_a \geq g_b \geq p_G$. Moreover, since $\text{ted}_{\mathcal{A}}(F, G) \leq k$ and $F[p_F + i_F \cdot q] \sim_{\mathcal{A}} G[g_a]$, we have $g_a \leq (p_F + i_F \cdot q) + 2k \leq p_F + kq + 2k \leq p_G + kq + 4k \leq p_G + 3kq$, where the last inequality follows from $q \geq 2$ (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 $g_a = p_G + i_G \cdot q$ for some $i_G \in [0..3k]$. \square

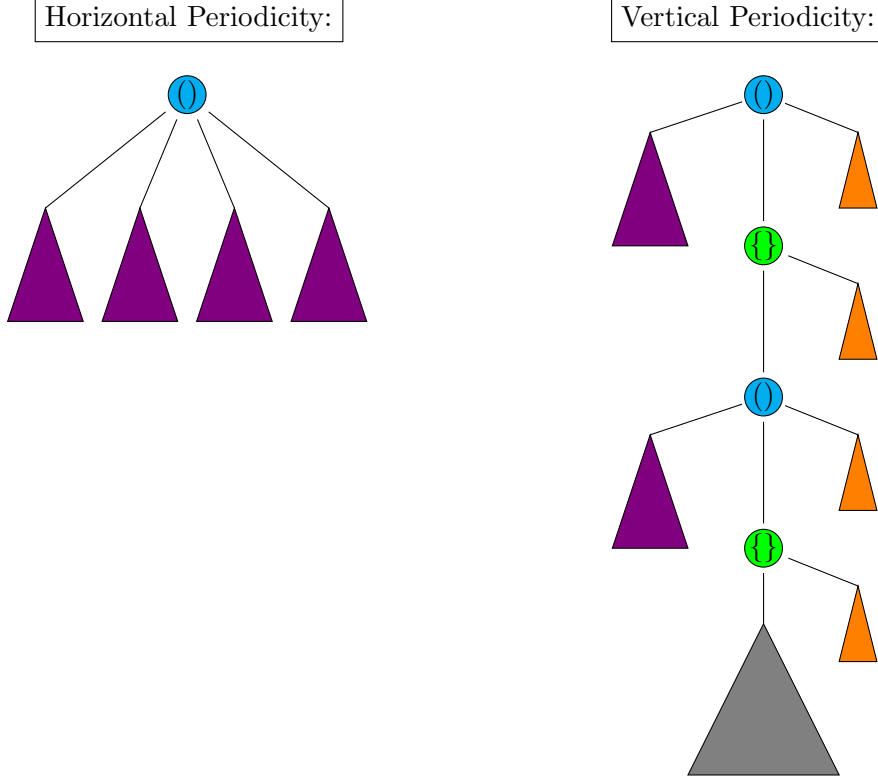


Figure 3: Pictured left: horizontal periodicity with string representation “ $([\dots][\dots][\dots][\dots])$ ”. Pictured right: vertical periodicity with string representation “ $([\dots]\{([\dots]\{[\dots][\dots]\}[\dots])[\dots]\}[\dots])$ ”.

Now, if $Q^e = F[p_F \dots p_F + e \cdot q] = G[p_G \dots p_G + e \cdot q]$ is replaced with $Q^{e'}$ for $e' \geq e - 1$, we can interpret this as replacing $Q = F[p_F + i_F \cdot q \dots p_F + (i_F + 1) \cdot q] = G[p_G + i_G \cdot q \dots p_G + (i_G + 1) \cdot q]$ with $Q^{1+e'-e}$. By Claim 3.6, \mathcal{A} can be trivially adapted without modifying its cost, and hence $\text{ted}^w(F', G') \leq \text{ted}_{\mathcal{A}}^w(F, G) = \text{ted}^w(F, G)$. If $e' < e - 1$, we repeat the above argument to decrement the exponent one step at a time, still concluding that $\text{ted}^w(F', G') \leq \text{ted}^w(F, G)$. In either case, the converse inequality follows by symmetry between (F, G, e) and (F', G', e') . \square

We say that a forest F avoids horizontal k -periodicity if there is no forest Q of length $|Q| \in [1 \dots 4k]$ such that Q^{4k+1} occurs in F .

Lemma 3.7. *Let $k \in \mathbb{Z}_+$ and let P, P' be forests of length $|P|, |P'| \geq 74k^3$ avoiding horizontal k -periodicity. Then, P and P' are $\text{ted}_{\leq k}^w$ -equivalent for every normalized quasimetric w .*

Proof. Suppose that P occurs in forests F and G at positions p_F and p_G , respectively, satisfying $|p_F - p_G| \leq 2k$. Denote $F' = F[0 \dots p_F] \cdot P' \cdot F[p_F + |P| \dots |F|]$ and $G' = G[0 \dots p_G] \cdot P' \cdot G[p_G + |P| \dots |G|]$.

Let $\mathcal{A} = (f_t, g_t)_{t=0}^m$ be an alignment such that $\text{ted}^w(F, G) = \text{ted}_{\mathcal{A}}^w(F, G) \leq k$. Moreover, let $(f_a, g_a) \in \mathcal{A}$ be the leftmost element of \mathcal{A} such that $f_a \geq p_F$ or $g_a \geq p_G$, and let $(f_b, g_b) \in \mathcal{A}$ be the leftmost element of \mathcal{A} such that $f_b \geq p_F + |P|$ and $g_b \geq p_G + |P|$. We construct an alignment \mathcal{A}' so that it:

- aligns $F[0 \dots f_a)$ with $G[0 \dots g_a)$ in the same way as \mathcal{A} does;
- deletes $F[f_a \dots p_F)$ and inserts $G[g_a \dots p_G)$ (at least one of these fragments is empty);
- matches $F[p_F \dots p_F + |P|) = P$ with $G[p_G \dots p_G + |P|) = P$;

- deletes $F[p_F + |P| \dots f_b)$ and inserts $G[p_G + |P| \dots g_b)$ (at least one of these fragments is empty);
- aligns $F[f_b \dots |F|)$ with $G[g_b \dots |G|)$ in the same way as \mathcal{A} does.

To prove that \mathcal{A}' is a forest alignment, let us consider several possibilities for a node u in F .

- If u is inside $P = F[p_F \dots p_F + |P|)$, then \mathcal{A}' matches u to the corresponding node inside $P = G[p_G \dots p_G + |P|)$.
- If u is outside $P = F[p_F \dots p_F + |P|)$ and \mathcal{A} aligns u to a node v of G inside $P = G[p_G \dots p_G + |P|)$, then $o(u), c(u) \in [f_a \dots p_F) \cup [p_F + |P| \dots f_b)$ because of the non-crossing property of $\mathcal{A} \ni (f_a, g_a), (f_b, g_b)$. Hence, \mathcal{A}' deletes u .
- If u is outside $P = F[p_F \dots p_F + |P|)$ and \mathcal{A} aligns u to a node v of G outside $P = G[p_G \dots p_G + |P|)$, then $o(u), c(u) \in [0 \dots f_a) \cup [f_b \dots |F|)$ because of the non-crossing property of $\mathcal{A} \ni (f_a, g_a), (f_b, g_b)$. Hence, \mathcal{A}' also aligns u to v .
- If u is outside $P = F[p_F \dots p_F + |P|)$ and \mathcal{A} deletes u , then \mathcal{A}' also deletes u .

Our next goal is to prove that $\text{ted}_{\mathcal{A}'}^w(F, G) \leq \text{ted}_{\mathcal{A}}^w(F, G)$. This relies on the following claim.

Claim 3.8. *There exists $t \in [a \dots b]$ such that $f_t - g_t = p_F - p_G$.*

Proof. Let us partition $P = F[p_F \dots p_F + |P|)$ into individual characters representing deletions or substitutions of \mathcal{A} and maximal fragments that \mathcal{A} matches perfectly (to fragments of G). By Fact 2.7, the number of such fragments is at most $2k + 1$ and their total length is at least $|P| - 2k$. Hence, one of these fragments, denoted $R = F[r_F \dots r_F + |R|)$, is of length at least $\frac{|P| - 2k}{2k + 1} \geq 24k^2$. Suppose that the fragment of G matched perfectly to R is $G[r_G \dots r_G + |R|)$. If $r_F - p_F = r_G - p_G$, the claim holds for t such that $(f_t, g_t) = (r_F, r_G)$. Otherwise, we note that R has period $q := |(r_F - p_F) - (r_G - p_G)| \in [1 \dots 4k]$. Let $q = R[0 \dots q)$ and observe that $\frac{|R|}{q} \geq \frac{24k^2}{4k} \geq 4k + 2$. Hence, Q^{4k+2} is a substring of P ; since P avoids horizontal k -periodicity, we conclude that no cyclic rotation of Q is balanced.

As Q is a substring of a balanced string P , this means that the number of opening parentheses in Q does not match the number of closing parentheses in Q . By symmetry (up to reversal), we assume without loss of generality that Q has more opening than closing parentheses. Thus, there exists a node u in F such that $o(u) \in [r_F \dots r_F + q)$ yet $c(u) \geq r_F + |R|$. In particular, $c(u) - o(u) \geq |R| - q \geq 24k^2 - 4k > 8k$. Let v, v' be the nodes in G matched with u by \mathcal{A} and \mathcal{A}' , respectively. Note that $|o(v) - o(v')| \leq 4k$ and $|c(v) - c(v')| \leq 4k$. Due to $c(v') - o(v') = c(u) - o(u) > 8k$, we conclude that v is ancestor of v' or vice versa. In either case, we have $0 \geq (o(v') - o(v)) \cdot (c(v') - c(v)) = ((o(u) - o(v)) - (p_F - p_G)) \cdot ((c(u) - c(v)) - (p_F - p_G))$. The value $(f_t - g_t) - (p_F - p_G)$ can change by at most one for subsequent indices t . The sign of this value is different when $(f_t, g_t) = (o(u), o(v))$ and $(f_t, g_t) = (c(u), c(v))$, so it must be equal to 0 at some intermediate index t . \square

The alignments \mathcal{A} and \mathcal{A}' only differ in how they align $F[f_a \dots f_t)$ with $G[g_a \dots g_t)$ and $F[f_t \dots f_b)$ with $G[g_t \dots g_b)$, and, by Fact 2.6, \mathcal{A}' provides an optimum alignment of these fragments. Now, if $P = F[p_F \dots p_F + |P|) = G[p_G \dots p_G + |P|)$ is modified to P' , then \mathcal{A}' can be trivially adapted without modifying its cost and hence $\text{ted}^w(F', G') \leq \text{ted}_{\mathcal{A}'}^w(F, G) = \text{ted}^w(F, G)$. The converse inequality follows by symmetry between (F, G, P) and (F', G', P') . \square

3.3.2 Contexts

Definition 3.9. For $k \in \mathbb{Z}_{\geq 0}$ and a weight function w , contexts $P = \langle P_L; P_R \rangle$ and $P' = \langle P'_L; P'_R \rangle$ are called $\text{ted}_{\leq k}^w$ -equivalent if

$$\begin{aligned} \text{ted}_{\leq k}^w(F, G) = \text{ted}_{\leq k}^w(F[0..o(u)] \cdot P'_L \cdot F[o(u) + |P_L|..c(u) - |P_R|] \cdot P'_R \cdot F(c(u)..|F|), \\ G[0..o(v)] \cdot P'_L \cdot G[o(v) + |P_L|..c(u) - |P_R|] \cdot P'_R \cdot G(c(v)..|G|)) \end{aligned}$$

holds for all forests F and G in which P occurs at nodes u and v , respectively, satisfying $|o(u) - o(v)| \leq 2k$ and $|c(u) - c(v)| \leq 2k$.

Lemma 3.10. Let $k \in \mathbb{Z}_+$, let Q be a context, and let $e, e' \in \mathbb{Z}_{\geq 6k}$. Then, Q^e and $Q^{e'}$ are $\text{ted}_{\leq k}^w$ -equivalent for every normalized weight function w .

Proof. We assume without loss of generality that Q is primitive. (If $Q = R^m$ for $m \in \mathbb{Z}_{\geq 2}$, then $Q^e = R^{me}$ and $Q^{e'} = R^{me'}$ can be interpreted as powers of R rather than powers of Q .) Let $Q = \langle Q_L; Q_R \rangle$ with $q_L = |Q_L|$ and $q_R = |Q_R|$. Suppose that Q^e occurs in forests F and G at nodes u and v , respectively, satisfying $|o(u) - o(v)| \leq 2k$ and $|c(u) - c(v)| \leq 2k$. Denote

$$\begin{aligned} F' &= F[0..o(u)] \cdot Q_L^{e'} \cdot F[o(u) + |Q_L^e|..c(u) - |Q_R^e|] \cdot Q_R^{e'} \cdot F(c(u)..|F|), \\ G' &= G[0..o(v)] \cdot Q_L^{e'} \cdot G[o(v) + |Q_L^e|..c(u) - |Q_R^e|] \cdot Q_R^{e'} \cdot G(c(v)..|G|). \end{aligned}$$

For $i \in [0..e]$, let u_i be the node of F with $o(u_i) = o(u) + i \cdot q_L$ (and $c(u_i) = c(u) - i \cdot q_R$) and let v_i be the node of G with $o(v_i) = o(v) + i \cdot q_L$ (and $c(v_i) = c(v) - i \cdot q_R$). Moreover, let \mathcal{A} be an optimal forest alignment such that $\text{ted}(F, G) = \text{ted}_{\mathcal{A}}(F, G) \leq k$.

Claim 3.11. There exist $i_F, i_G \in [0..5k]$ such that

$$\begin{aligned} F[o(u) + i_F \cdot q_L..o(u) + (i_F + 1) \cdot q_L] &\simeq_{\mathcal{A}} G[o(v) + i_G \cdot q_L..o(v) + (i_G + 1) \cdot q_L], \\ F(c(u) - (i_F + 1) \cdot q_R..c(u) - i_F \cdot q_R) &\simeq_{\mathcal{A}} G(c(v) - (i_G + 1) \cdot q_R..c(v) - i_G \cdot q_R). \end{aligned}$$

Proof. Let $(f_b, g_b) \in \mathcal{A}$ be the leftmost element of \mathcal{A} such that $f_b \geq o(u)$ and $g_b \geq o(v)$. By symmetry between F and G , we may assume without loss of generality that $f_b = o(u)$. Consider the $k+1$ disjoint occurrences of C in F at positions $(o(u) + i \cdot q_L, c(u) + i \cdot q_R)$ for $i \in [0..k]$. The alignment \mathcal{A} (of unweighted cost at most k) must match one of these occurrences perfectly to a context within G . We pick the index $i_F \in [0..k]$ of one such perfectly matched occurrence and suppose that it occurs at a node v' of G .

In particular,

$$\begin{aligned} F[o(u_{i_F})..o(u_{i_F}) + q_L] &\simeq_{\mathcal{A}} G[o(v')..o(v') + q_L], \\ F(c(u_{i_F}) - q_R..c(u_{i_F})) &\simeq_{\mathcal{A}} G(c(v') - q_R..c(v')). \end{aligned}$$

Since $(f_b, g_b) \in \mathcal{A}$, we must have $o(v') \geq g_b \geq o(v)$ by the non-crossing property of \mathcal{A} . At the same time, since the unweighted cost of \mathcal{A} does not exceed k , we have $o(v') \leq o(u_{i_F}) + 2k \leq o(u) + kq_L + 2k \leq o(v) + kq_L + 4k \leq o(v) + 5kq_L$. Similarly, $c(v') \geq c(v) - 5kq_R$, which also implies $c(v') \leq c(v)$.

Our next goal is to show that $v' = v_{i_G}$ for some $i_G \in [0..5k]$. For a proof by contradiction, suppose that $o(v_i) < o(v') < o(v_{i+1})$ for some $i \in [0..5k]$. Due to $c(v') > c(v) - 5kq_R$, this also implies that $c(v_i) > c(v') > c(v_{i+1})$, i.e., that v' 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 v' is at distance ℓ' from v_i . Hence, $G[o(v_i)..o(v')]$ has

ℓ' unmatched opening parentheses out of the ℓ unmatched opening parentheses in Q_L . Moreover, $G[o(v_i) \dots o(v')] \cdot G[o(v') \dots o(v_{i+1})] = Q_L = G[o(v') \dots o(v_{i+1})] \cdot G[o(v_i) \dots o(v')]$, and thus there is a primitive string Q_L such that $G[o(v') \dots o(v_{i+1})]$ and $G[o(v_i) \dots o(v')]$ are both powers of Q_L . The number of unmatched opening parentheses in Q_L must be a common divisor of ℓ and ℓ' , i.e., Q_L can be expressed as a string power with exponent $\ell / \gcd(\ell, \ell')$. A symmetric argument shows that Q_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, $v' = v_{i_G}$ for some $i_G \in [0 \dots 5k]$ holds as claimed and, in particular, $o(v') = o(v) + i_G q_L$ and $c(v') = c(v) - i_G q_R$. \square

Now, if the occurrences of Q^e at nodes u, v are replaced with $Q^{e'}$ for $e' \geq e - 1$, we can interpret this as replacing the occurrences of Q at nodes u_{i_F}, v_{i_G} with $Q^{1+e'-e}$. By Claim 3.11, \mathcal{A} can be trivially adapted without modifying its cost, and hence $\text{ted}^w(F', G') \leq \text{ted}_{\mathcal{A}}^w(F, G) = \text{ted}^w(F, G)$. If $e' < e - 1$, we repeat the above argument to decrement the exponent one step at a time, still concluding that $\text{ted}^w(F', G') \leq \text{ted}^w(F, G)$. In either case, the converse inequality follows by symmetry between (F, G, e) and (F', G', e') . \square

We say that a context $P = \langle P_L; P_R \rangle$ avoids vertical k -periodicity if it cannot be expressed as $P = C \star Q^{6k+1} \star D$ for some contexts C, Q, D satisfying $|Q| \in [1 \dots 8k]$.

Lemma 3.12. *Let $k \in \mathbb{Z}_+$, let $P = \langle P_L; P_R \rangle, P' = \langle P'_L; P'_R \rangle$ be contexts of length $|P_L| + |P_R|, |P'_L| + |P'_R| \geq 578k^4$ that avoid vertical k -periodicity and whose halves do not contain any balanced substring of length more than $74k^3$. Then, P and P' are $\text{ted}_{\leq k}^w$ -equivalent for every normalized weight function w .*

Proof. Suppose that P occurs in forests F and G at nodes u and v , respectively, satisfying $|o(u) - o(v)| \leq 2k$ and $|c(u) - c(v)| \leq 2k$. Denote

$$\begin{aligned} F' &= F[0 \dots o(u)] \cdot P'_L \cdot F[o(u) + |P_L| \dots c(u) - |P_R|] \cdot P'_R \cdot F(c(u) \dots |F|), \\ G' &= G[0 \dots o(v)] \cdot P'_R \cdot G[o(v) + |P_L| \dots c(v) - |P_R|] \cdot P'_L \cdot G(c(v) \dots |G|). \end{aligned}$$

Let $\mathcal{A} = (f_t, g_t)_{t=0}^m$ be an optimal forest alignment such that $\text{ted}(F, G) = \text{ted}_{\mathcal{A}}(F, G) \leq k$. Moreover, let $(f_a, g_a) \in \mathcal{A}$ be the leftmost element of \mathcal{A} such that $f_a \geq o(u)$ or $g_a \geq o(v)$, $(f_b, g_b) \in \mathcal{A}$ be the leftmost element of \mathcal{A} such that $f_b \geq o(u) + |P_L|$ and $g_b \geq o(v) + |P_L|$, $(f_c, g_c) \in \mathcal{A}$ be the leftmost element of \mathcal{A} such that $f_c > c(u) - |P_R|$ or $g_c > c(v) - |P_R|$, and let (f_d, g_d) be the leftmost element of \mathcal{A} such that $f_d > c(u)$ and $g_d > c(v)$. We construct an alignment \mathcal{A}' so that it:

- aligns $F[0 \dots f_a]$ with $G[0 \dots g_a]$ in the same way as \mathcal{A} does;
- deletes $F[f_a \dots o(u)]$ and inserts $G[g_a \dots o(v)]$ (at least one of these fragments is empty);
- matches $F[o(u) \dots o(u) + |P_L|] = P_L$ with $G[o(v) \dots o(v) + |P_L|] = P_L$;
- if $b > c$, deletes $F[o(u) + |P_L| \dots c(u) - |P_R|]$ and inserts $G[o(v) + |P_L| \dots c(v) - |P_R|]$;
- if $b \leq c$, deletes $F[o(u) + |P_L| \dots f_b]$ and inserts $G[o(v) + |P_L| \dots g_b]$ (at least one of these fragments is empty);
- if $b \leq c$, aligns $F[f_b \dots f_c]$ with $G[g_b \dots g_c]$ in the same way as \mathcal{A} does;
- if $b \leq c$, deletes $F[f_c \dots c(u) - |P_R|]$ and inserts $G[g_c \dots c(v) - |P_R|]$ (at least one of these fragments is empty);
- matches $F(c(u) - |P_R| \dots c(u)) = P_R$ with $G(c(v) - |P_R| \dots c(v)) = P_R$;
- deletes $F(c(u) \dots f_d)$ and inserts $G(c(v) \dots g_d)$ (at least one of these fragments is empty);
- aligns $F[f_d \dots |F|]$ with $G[g_d \dots |G|]$ in the same way as \mathcal{A} does.

To prove that \mathcal{A}' is a forest alignment, let us consider several possibilities for a node u' in F .

- If u' belongs to $P = \langle F[o(u) \dots o(u) + |P_L|]; F(c(u) - |P_R| \dots c(u)) \rangle$, then \mathcal{A}' matches u' to the corresponding node that belongs to $P = \langle G[o(v) \dots o(v) + |P_L|]; G(c(v) - |P_R| \dots c(v)) \rangle$.
- If u' is outside $F[o(u) \dots c(u)]$ and \mathcal{A} aligns u' to a node v' of G inside $G[o(v) \dots c(v)]$, then $o(u'), c(u') \in [f_a \dots o(u)] \cup (c(u) \dots f_d)$ because of the non-crossing property of $\mathcal{A} \ni (f_a, g_a), (f_d, g_d)$. Hence, \mathcal{A}' deletes u' .
- If u' is outside $F[o(u) \dots c(u)]$ and \mathcal{A} aligns u' to a node v' of G outside $G[o(v) \dots c(v)]$, then $o(u'), c(u') \in [0 \dots f_a] \cup [f_d \dots |F|)$ because of the non-crossing property of $\mathcal{A} \ni (f_a, g_a), (f_d, g_d)$. Hence, \mathcal{A}' also aligns u' to v' .
- If u' is outside $F[o(u) \dots c(u)]$ and \mathcal{A} deletes u' , then \mathcal{A}' also deletes u' .
- If $b > c$ and u' is inside $F[o(u) + |P_L| \dots c(u) - |P_R|]$, then \mathcal{A}' deletes u' .
- If $b \leq c$, u' is inside $F[o(u) + |P_L| \dots c(u) - |P_R|]$, and \mathcal{A} aligns u' to a node v' of G outside $G[o(v) + |P_L| \dots c(v) - |P_R|]$, then $o(u'), c(u') \in [o(u) + |P_L| \dots f_b] \cup [f_c \dots c(u) - |P_R|]$ because of the non-crossing property of $\mathcal{A} \ni (f_b, g_b), (f_c, g_c)$. Hence, \mathcal{A}' deletes u' .
- If $b \leq c$, u' is inside $F[o(u) + |P_L| \dots c(u) - |P_R|]$, and \mathcal{A} aligns u' to a node v' of G inside $G[o(v) + |P_L| \dots c(v) - |P_R|]$, then $o(u'), c(u') \in [f_b \dots f_c]$ because of the non-crossing property of $\mathcal{A} \ni (f_b, g_b), (f_c, g_c)$. Hence, \mathcal{A}' also aligns u' to v' .
- If $b \leq c$, u' is inside $F[o(u) + |P_L| \dots c(u) - |P_R|]$, and \mathcal{A} deletes u' , then \mathcal{A}' also deletes u' .

Let us now prove that $\text{ted}_{\mathcal{A}'}^w(F, G) \leq \text{ted}_{\mathcal{A}}^w(F, G)$. This relies on the following claim.

Claim 3.13. *There exist $t_L \in [a \dots b]$ such that $f_{t_L} - g_{t_L} = o(u) - o(v)$ and $t_R \in [c \dots d]$ such that $f_{t_R} - g_{t_R} = c(u) - c(v)$.*

Proof. By symmetry (up to reversal), we can focus without loss of generality on the first claim. Moreover, by symmetry between F and G , we can assume without loss of generality that $f_a = o(u)$; in particular, this implies $f_a - g_a \geq o(u) - o(v)$. If there exists $t \in [a \dots b]$ such that $f_t - g_t \leq o(u) - o(v)$, then, since $f_t - g_t$ may change by at most one for subsequent positions, there is also $t_L \in [a \dots b]$ such that $f_{t_L} - g_{t_L} = o(u) - o(v)$. Consequently, it remains to consider the case when $f_t - g_t > o(u) - o(v)$ holds for all $t \in [a \dots b]$.

Let us express P as a vertical composition of e contexts $P = P_0 \star \dots \star P_{e-1}$, where e is the depth of P . Observe that the occurrences of P at node u in F and v in G , for each $i \in [0 \dots e)$, induce occurrences of P_i at some nodes u_i in F and v_i in G . Since $F(o(u_i) \dots o(u_i) + |P_{i,L}|)$ and $F(c(u_i) - |P_{i,R}| \dots c(u_i))$ are balanced, we conclude that $|P_i| \leq 2 \cdot (74k^3 + 1) \leq 150k^3$. We can decompose $[0 \dots e)$ into at most k individual indices i such that \mathcal{A} does not match perfectly the occurrence of P_i at v_i and at most $k + 1$ intervals $[i \dots i')$ such that \mathcal{A} matches the occurrence $P_i \star \dots \star P_{i'-1}$ at v_i perfectly to a context in F . Let us choose such an interval $[i \dots i')$ maximizing $|P_i \star \dots \star P_{i'-1}|$; this length is at least $\frac{578k^4 - k \cdot 150k^3}{k+1} \geq 214k^3$. Let $i'' \in [i \dots i')$ be the maximum index such that $|P_{i''} \star \dots \star P_{i''-1}| > 8k$; note that $|P_i \star \dots \star P_{i''-1}| \geq 214k^3 - (150k^3 + 8k) \geq 56k^2$.

For each $j \in [i \dots i'')$, denote by u'_j be the node of F matched with v_j by \mathcal{A} . Note that $|o(u'_j) - o(u_j)| \leq 4k$ and $|c(u'_j) - c(u_j)| \leq 4k$. Moreover, $o(u'_j) - o(v_j) > o(u) - o(v) = o(u_j) - o(v_j)$ implies $o(u'_j) > o(u_j)$. Since $|P_j \star \dots \star P_{i''-1}| > 8k$, we conclude that $u'_j = u_{j'}$ for some $j' \in [j \dots i')$. Moreover, if $j > i$, then u'_j must be a child of u'_{j-1} . Hence, there exists $\delta > 0$ such that $u'_j = u_{j+\delta}$ holds for all $j \in [i \dots i'')$. For $j \in [i \dots i'')$, this implies $P_j = P_{j+\delta}$ and that both halves of $P_j \star \dots \star P_{j+\delta-1}$ are of length at most $4k$. In particular, if we define $Q = P_i \star \dots \star P_{i+\delta-1}$, then, due to $\frac{|P_i \star \dots \star P_{i+\delta-1}|}{8k} \geq \frac{56k^2}{8k} \geq 6k + 1$, we conclude that Q^{6k+1} occurs in F and G at positions u_i and v_i , respectively. This contradicts the assumption that P avoids vertical periodicity. \square

The alignments \mathcal{A} and \mathcal{A}' only differ in how they align the following fragments:

- $F[f_a \dots f_{t_L}]$ with $G[g_a \dots g_{t_L}]$: here, \mathcal{A}' matches one fragment perfectly with a suffix of the other; by Fact 2.6, this is optimal.
- $F[f_{t_L} \dots f_{t_R}]$ with $G[g_{t_L} \dots g_{t_R}]$ if $b > c$: here, the cost of \mathcal{A}' is equal to the cost of deleting $F[o(u) + |P_L| \dots c(u) - |P_R|]$ and inserting $G[o(v) + |P_L| \dots c(v) - |P_R|]$. By Fact 2.6, these two costs do not exceed the cost of \mathcal{A} aligning $F[f_{t_L} \dots f_c]$ with $G[g_{t_L} \dots g_c]$ and aligning $F[f_b \dots f_{t_R}]$ with $G[g_b \dots g_{t_R}]$.
- $F[f_{t_L} \dots f_b]$ with $G[g_{t_L} \dots g_b]$ if $b \leq c$: here, \mathcal{A}' matches one fragment perfectly with a prefix of the other; by Fact 2.6, this is optimal.
- $F[f_c \dots f_{t_R}]$ with $G[g_c \dots g_{t_R}]$ if $b \leq c$: here, \mathcal{A}' matches one fragment perfectly with a suffix of the other; by Fact 2.6, this is optimal.
- $F[f_{t_R} \dots f_d]$ with $G[g_{t_R} \dots g_d]$: here, \mathcal{A}' matches one fragment perfectly with a prefix of the other; by Fact 2.6, this is optimal.

If the occurrences of P at nodes u in F and v in G are modified to occurrences of P' , then \mathcal{A}' can be trivially adapted without modifying its cost and hence $\text{ted}^w(F', G') \leq \text{ted}_{\mathcal{A}'}^w(F, G) = \text{ted}^w(F, G)$. The converse inequality follows by symmetry between (F, G, P) and (F', G', P') . \square

3.4 Algorithms

We say a *piece* of a forest F is a balanced fragment $F[i \dots j]$ or a pair of fragments $\langle F[i \dots i']; F[j' \dots j] \rangle$ that form a context, that is, $F[i \dots j]$ is a tree and $F[i' \dots j']$ is balanced. For a fragment $F[i \dots j]$, we denote the set of pieces contained in $F[i \dots j]$ by $\mathcal{P}(F[i \dots j])$. Moreover, let $\mathcal{P}(F) = \mathcal{P}(F[0 \dots |F|])$.

Definition 3.14. A set $D \subseteq \mathcal{P}(F[i \dots j])$ is a *piece decomposition* of a balanced fragment $F[i \dots j]$ of a forest F if it satisfies one of the following conditions:

- $D = \emptyset$ and $i = j$;
- $D = \{F[i \dots j]\}$ and $i < j$;
- $D = D_L \cup D_R$ for some piece decompositions D_L of $F[i \dots m]$ and D_R of $F[m \dots j]$, where $m \in (i \dots j)$.
- $D = \{\langle F[i \dots i']; F[j' \dots j] \rangle\} \cup D'$ for a context $\langle F[i \dots i']; F[j' \dots j] \rangle \in \mathcal{P}(F)$ and a piece decomposition D' of $F[i' \dots j']$.

Lemma 3.15. *There exists a linear-time algorithm that, given a forest F and an integer $t \geq 2$, constructs a piece decomposition D of F consisting of at most $\max(1, \frac{6|F|}{t} - 1)$ pieces of length at most t each.*

Algorithm 4: $\mathcal{D}(i, j)$: Construct a decomposition of a balanced fragment $F[i \dots j]$.

```

1 if  $j = i$  then return  $\emptyset$ ;
2 if  $j \leq i + t$  then return  $\{F[i \dots j]\}$ ;
3  $i' \leftarrow i; j' \leftarrow j$ ;
4 while true do //  $F[i' \dots j']$  is balanced and  $(i' - i) + (j - j') \leq t$ 
5   Let  $m \in [i' \dots j']$  be such that  $F[i' \dots m]$  is a tree;
6   if  $(m - i) + (j - j') \leq t$  then  $i' \leftarrow m$ ;
7   else if  $m < j'$  and  $(i' - i) + (j - m) \leq t$  then  $j' \leftarrow m$ ;
8   else if  $F[i \dots j]$  is not a tree then return  $\mathcal{D}(i, i') \cup \mathcal{D}(i', m) \cup \mathcal{D}(m, j') \cup \mathcal{D}(j', j)$ ;
9   else if  $m = j'$  and  $(i' + 1 - i) \leq (j - j' + 1) \leq t$  then  $i' \leftarrow i' + 1; j' \leftarrow j' - 1$ ;
10  else return  $\{\langle F[i \dots i']; F[j' \dots j] \rangle\} \cup \mathcal{D}(i', m) \cup \mathcal{D}(m, j')$ ;

```

Proof. Algorithm 4 provides a recursive procedure that, for every balanced fragment $F[i..j]$ of F , constructs a piece decomposition $\mathcal{D}(i, j)$ of $F[i..j]$ consisting of pieces of size at most t . In the corner cases of $j = i$ and $j \in (i..i+t]$, we return $\mathcal{D}(i, j) = \emptyset$ and $\mathcal{D}(i, j) = \{F[i..j]\}$, respectively. Otherwise, we iteratively grow fragments $F[i..i']$ and $F[j'..j]$ (initially empty) maintaining the following invariants:

- (a) $F[i'..j']$ is balanced;
- (b) $|F[i..i']| + |F[j'..j]| \leq t$;
- (c) if $F[i..j]$ is a tree, then $F[i..i'] = F[j'..j] = \varepsilon$ or $\langle F[i..i']; F[j'..j] \rangle$ is a context;
- (d) if $F[i..j]$ is not a tree, then $F[i..i']$ and $F[j'..j]$ are balanced.

At each iteration, we identify a position $m \in (i'..j']$ such that $F[i'..m]$ is a tree (such a position always exists due to $j - i > t$ and by invariants (a), (b)).

- (1) We set $i' \leftarrow m$ as long as it would not violate invariant (b).
- (2) If $m \neq j'$, we set $j \leftarrow m$ as long as it would not violate invariant (b).
- (3) If $m = j'$ and $F[i..j]$ is a tree, we set $(i', j') \leftarrow (i' + 1, j' - 1)$ as long as it would not violate invariant (b).
- (4) Otherwise, we return $\mathcal{D}(i, j) := \{\langle F[i..i']; F[j'..j] \rangle\} \cup \mathcal{D}(i', m) \cup \mathcal{D}(m, j')$ if $F[i..j]$ is a tree and $\mathcal{D}(i, j) := \mathcal{D}(i, i') \cup \mathcal{D}(i', m) \cup \mathcal{D}(m, j') \cup \mathcal{D}(j', j)$ if $F[i..j]$ is not a tree.

It is easy to see that cases (1)–(3) preserve the invariants and hence case (4) results in a valid piece decomposition with pieces of size at most t .

Next, we prove that the number of pieces is at most $\max(1, \frac{6(j-i)}{t} - 3)$ if $F[i..j]$ is a tree and at most $\max(1, \frac{6(j-i)}{t} - 1)$ otherwise. This holds trivially if $j \leq i + t$, where Algorithm 4 terminates at Line 1 or 2. If $F[i..j]$ is a tree of size $j - i > t$, then Algorithm 4 terminates at Line 10. We consider several sub-cases:

- 1. If $m - i' \leq \frac{2t}{3}$ and $j' - m \leq \frac{t}{3}$, then $|\mathcal{D}(i, j)| \leq 3 < \frac{6(j-i)}{t} - 3$ because $j - i > t$.
- 2. If $m - i' \leq \frac{2t}{3}$ and $j' - m > \frac{t}{3}$, then $(m - i) + (j - j') > t$ because the test in Line 6 failed. Hence, $j' - m < j - i - t$ and $|\mathcal{D}(i, j)| \leq 2 + |\mathcal{D}(m, j')| \leq 2 + \frac{6(j'-m)}{t} - 1 < \frac{6(j-i-t)}{t} + 1 < \frac{6(j-i)}{t} - 3$.
- 3. If $m - i' > \frac{2t}{3}$ and $j' - m = 0$, then $(i' + 1 - i) + (j - j' + 1) > t$ because the test in Line 9 failed. Hence, $j' - i' \leq j - i - t + 1$ and $|\mathcal{D}(i, j)| \leq 1 + |\mathcal{D}(i', j')| \leq 1 + \frac{6(j'-i')}{t} - 3 \leq \frac{6(j-i-t+1)}{t} - 2 < \frac{6(j-i)}{t} - 3$.
- 4. If $m - i' > \frac{2t}{3}$ and $0 < j' - m \leq \frac{t}{3}$, then $(i' - i) + (j - m) > t$ because the test in Line 7 failed. Hence, $m - i' < j - i - t$ and $|\mathcal{D}(i, j)| \leq 2 + |\mathcal{D}(i', m)| \leq 2 + \frac{6(m-i')}{t} - 3 < \frac{6(j-i-t)}{t} - 1 < \frac{6(j-i)}{t} - 3$.
- 5. If $m - i' > \frac{2t}{3}$ and $j' - m > \frac{t}{3}$, then $|\mathcal{D}(i, j)| \leq 1 + |\mathcal{D}(i', m)| + |\mathcal{D}(m, j')| \leq 1 + \frac{6(m-i')}{t} - 3 + \frac{6(j'-m)}{t} - 1 \leq \frac{6(j-i)}{t} - 3$.

If $F[i..j]$ is not a tree, then Algorithm 4 terminates at Line 8. We consider several sub-cases:

- 1. If $m - i' \leq \frac{2t}{3}$ and $j' - m \leq \frac{t}{3}$, then $|\mathcal{D}(i, j)| \leq 4 < \frac{6(j-i)}{t} - 1$ because $j - i > t$.
- 2. If $m - i' \leq \frac{2t}{3}$ and $j' - m > \frac{t}{3}$, then $(m - i) + (j - j') > t$ because the test in Line 6 failed. Hence, $j' - m < j - i - t$ and $|\mathcal{D}(i, j)| \leq 3 + |\mathcal{D}(m, j')| \leq 3 + \frac{6(j'-m)}{t} - 1 < \frac{6(j-i-t)}{t} + 2 < \frac{6(j-i)}{t} - 1$.
- 3. If $m - i' > \frac{2t}{3}$ and $j' - m = 0$, then $|\mathcal{D}(i, j)| \leq 2 + |\mathcal{D}(i', j')| \leq 2 + \frac{6(j'-i')}{t} - 3 \leq \frac{6(j-i)}{t} - 1$.
- 4. If $m - i' > \frac{2t}{3}$ and $0 < j' - m \leq \frac{t}{3}$, then $(i' - i) + (j - m) > t$ because the test in Line 7 failed. Hence, $m - i' < j - i - t$ and $|\mathcal{D}(i, j)| \leq 3 + |\mathcal{D}(i', m)| \leq 3 + \frac{6(m-i')}{t} - 3 < \frac{6(j-i-t)}{t} < \frac{6(j-i)}{t} - 1$.
- 5. If $m - i' > \frac{2t}{3}$ and $j' - m > \frac{t}{3}$, then $|\mathcal{D}(i, j)| \leq 2 + |\mathcal{D}(i', m)| + |\mathcal{D}(m, j')| \leq 2 + \frac{6(m-i')}{t} - 3 + \frac{6(j'-m)}{t} - 1 \leq \frac{6(j-i)}{t} - 2 < \frac{6(j-i)}{t} - 1$.

It remains to provide a linear-time implementation of our algorithm. We assume that there are bidirectional pointers between the opening and the closing parentheses representing the same node. Such pointers can be constructed using a linear-time stack-based preprocessing of the input forest F . Each iteration of the **while** loop increases $j - j' + i' - i$ (except for the final one), so a single call to the $\mathcal{D}(i, j)$ function costs $\mathcal{O}(t)$ due to invariant (b). The total number of calls is $\mathcal{O}(|\mathcal{D}(0, |F|)|) = \mathcal{O}(\frac{1}{t} \cdot |F|)$, so the overall running time, including preprocessing, is $\mathcal{O}(|F|)$. \square

Lemma 3.16. *Given forests F and G of total size n , a piece decomposition \mathbf{D} of F , and an integer $s \in \mathbb{Z}_+$, one can find in $\mathcal{O}(n + |\mathbf{D}|s^3)$ time a maximum-size set $S \subseteq \mathbf{D} \times \mathcal{P}(G)$ that, for some alignment $\mathcal{A} \in \text{TA}(F, G)$ of width at most s , contains only pairs of pieces that \mathcal{A} matches perfectly.*

Algorithm 5: Compute a maximum-size element of $\mathcal{S}(D_{i,j}, G[i'..j'])$.

```

1 Pairs( $D_{i,j}, G[i'..j']$ ):
2    $S \leftarrow \emptyset$ ;
3   if  $i' < \min(j', i + s)$  then  $S \stackrel{\max}{\leftarrow} \text{Pairs}(D_{i,j}, G[i' + 1..j'])$ ;
4   if  $j' > \max(i', j - s)$  then  $S \stackrel{\max}{\leftarrow} \text{Pairs}(D_{i,j}, G[i'..j' - 1])$ ;
5   if  $D_{i,j} = \{F[i..j]\}$  and  $F[i..j] = G[i'..j']$  then
6      $S \stackrel{\max}{\leftarrow} \{(F[i..j], G[i'..j'])\}$ ;
7   if  $D_{i,j} = D_{i,m} \cup D_{m,j}$  for some  $m \in (i..j)$  then
8     foreach  $m' \in [m - s..m + s] \cap [i'..j']$  do
9        $S \stackrel{\max}{\leftarrow} \text{Pairs}(D_{i,m}, G[i'..m']) \cup \text{Pairs}(D_{m,j}, G[m'..j'])$ ;
10  if  $D_{i,j} = \{ \langle F[i..i + \ell]; F[j - r..j] \rangle \} \cup D_{i+\ell, j-r}$  then
11    if  $F[i..i + \ell] = G[i'..i' + \ell]$  and  $F[j - r..j] = G[j' - r..j']$  and  $G[i' + \ell..j' - r]$ 
12      is balanced then
13         $S \stackrel{\max}{\leftarrow} \{ \langle \langle F[i..i + \ell]; F[j - r..j] \rangle, \langle G[i'..i' + \ell]; G[j' - r..j'] \rangle \} \cup$ 
14           $\text{Pairs}(D_{i+\ell, j-r}, G[i' + \ell..j' - r])$ 
15     $S \stackrel{\max}{\leftarrow} \text{Pairs}(D_{i+\ell, j-r}, G[\max(i + \ell - s, i').. \min(j - r + s, j')])$ ;
16  return  $S$ ;
```

Proof. For a piece decomposition $D_{i,j}$ of a balanced fragment $F[i..j]$ and a fragment $G[i'..j']$, let $\mathcal{S}(D_{i,j}, G[i'..j'])$ be the family of all subsets of $D_{i,j} \times \mathcal{P}(G[i'..j'])$ that, for some alignment $\mathcal{A} \in \text{A}(F[i..j], G[i'..j'])$ of width at most s , contain only pairs of pieces that \mathcal{A} matches perfectly. Algorithm 5 implements a recursive procedure $\text{Pairs}(D_{i,j}, G[i'..j'])$ that computes a maximum-size element of $\mathcal{S}(D_{i,j}, G[i'..j'])$ assuming that $i' \in [i - s..i + s]$ and $j' \in [j - s..j + s]$. It uses an $S \stackrel{\max}{\leftarrow} S'$ operator that assigns $S \leftarrow S'$ if $|S'| > |S|$. The algorithm returns the largest of the following candidates:

1. \emptyset . This is trivially valid because every alignment $\mathcal{A} \in \text{A}(F[i..j], G[i'..j'])$ of width at most s witnesses $\emptyset \in \mathcal{S}(D_{i,j}, G[i'..j'])$.
2. $\text{Pairs}(D_{i,j}, G[i' + 1..j'])$ if $i' < \min(j', i + s)$. Let $S = \text{Pairs}(D_{i,j}, G[i' + 1..j'])$ with a witness alignment $\mathcal{A}' \in \text{A}(F[i..j], G[i' + 1..j'])$. An alignment obtained from \mathcal{A}' by prepending (i, i') , which corresponds to inserting $Y[i']$, witnesses $S \in \mathcal{S}(D_{i,j}, G[i'..j'])$.
3. $\text{Pairs}(D_{i,j}, G[i'..j' - 1])$ if $j' > \max(i', j - s)$. Let $S = \text{Pairs}(D_{i,j}, G[i'..j' - 1])$ with a witness alignment $\mathcal{A}' \in \text{A}(F[i..j], G[i'..j' - 1])$. An alignment obtained from \mathcal{A}' by appending (j, j') , which corresponds to inserting $Y[j' - 1]$, witnesses $S \in \mathcal{S}(D_{i,j}, G[i'..j'])$.

4. $\{(F[i..j], G[i'..j'])\}$ if $D_{i,j} = \{F[i..j]\}$ and $F[i..j] = G[i'..j']$. The alignment $(i+t, j+t)_{t=0}^{i'-i} \in \mathcal{A}(F[i..j], G[i'..j'])$ witnesses $\{(F[i..j], G[i'..j'])\} \in \mathcal{S}(D_{i,j}, G[i'..j'])$.
5. $\text{Pairs}(D_{i,m}, G[i'..m']) \cup \text{Pairs}(D_{m,j}, G[m'..j'])$ if $D_{i,j} = D_{i,m} \cup D_{m,j}$ for some $m \in (i..j)$ and $m' \in [m-s..m+s] \cap [i'..j']$. Denote $S_L = \text{Pairs}(D_{i,m}, G[i'..m'])$ and $S_R = \text{Pairs}(D_{m,j}, G[m'..j'])$ with witness alignments $\mathcal{A}_L \in \mathcal{A}(F[i..m], G[i'..m'])$ and $\mathcal{A}_R \in \mathcal{A}(F[m..j], G[m'..j'])$, respectively. Stitching \mathcal{A}_L and \mathcal{A}_R at the common endpoint (m, m') yields an alignment witnessing $S_L \cup S_R \in \mathcal{S}(D_{i,j}, G[i'..j'])$.
6. $\{(\langle F[i..i+\ell]; F[j-r..j] \rangle, \langle G[i'..i'+\ell]; G[j'-r..j'] \rangle)\} \cup \text{Pairs}(D_{i+\ell, j-r}, G[i'+\ell..j'-r])$ if $D_{i,j} = \{\langle F[i..i+\ell]; F[j-r..j] \rangle\} \cup D_{i+\ell, j-r}$ and $\langle G[i'..i'+\ell]; G[j'-r..j'] \rangle$ is a context in G matching $\langle F[i..i+\ell]; F[j-r..j] \rangle$. Consider a set $S' = \text{Pairs}(D_{i+\ell, j-r}, G[i'+\ell..j'-r])$ and a witness alignment $\mathcal{A}' \in \mathcal{A}(F[i+\ell..j-r], G[i'+\ell..j'-r])$. Stitching $(i+t, i'+t)_{t=0}^\ell$, \mathcal{A}' , and $(j+t, j'+t)_{t=-r}^0$ at the common endpoints yields an alignment witnessing $S' \cup \{(\langle F[i..i+\ell]; F[j-r..j] \rangle, \langle G[i'..i'+\ell]; G[j'-r..j'] \rangle)\} \in \mathcal{S}(D_{i,j}, G[i'..j'])$.
7. $\text{Pairs}(D_{i+\ell, j-r}, G[\max(i+\ell-s, i').. \min(j-r+s, j')])$ if $D_{i,j} = \{\langle F[i..i+\ell]; F[j-r..j] \rangle\} \cup D_{i+\ell, j-r}$. Let $S = \text{Pairs}(D_{i+\ell, j-r}, G[i'+\ell'..j'-r'])$, where $\ell' = \max(0, i+\ell-s-i')$ and $r' = \max(0, j'-j+r-s)$, with a witness alignment $\mathcal{A}' \in \mathcal{A}(F[i+\ell, j-r], G[i'+\ell', j'-r'])$. Stitching $(i+t, i'+t)_{t=0}^{\ell'}$, $(i+t, i'+\ell')_{t=\ell'}^\ell$, \mathcal{A}' , $(j+t, j'-r')_{t=-r}^{-r'}$, and $(j+t, j'+t)_{t=-r'}^0$ yields an alignment witnessing $S \in \mathcal{S}(D_{i,j}, G[i'..j'])$.

Next, consider a maximum-size element $S \in \mathcal{S}(D_{i,j}, G[i'..j'])$ and a witness alignment $\mathcal{A} \in \mathcal{A}(F[i..j], G[i'..j'])$ of width at most s .

- (a) If $D_{i,j} = \emptyset$, we must have $S = \emptyset$, which is covered by candidate 1.
- (b) Suppose that $D_{i,j} = \{F[i..j]\}$. The case of $S = \emptyset$ is covered by candidate 1. Otherwise, $S = \{(F[i..j], G[i''..j''])\}$ for some $i'' \in [i'..i+s]$ and $j'' \in [j-s..j']$. This is covered by $i''-i'$ applications of candidate 2, $j'-j''$ applications of candidate 3, and finally an application of candidate 4.
- (c) Suppose that $D_{i,j} = D_{i,m} \cup D_{m,j}$ for some $m \in (i..j)$. Since the width of \mathcal{A} does not exceed s , we must have $(m, m') \in \mathcal{A}$ for some $m' \in [m-s..m+s] \cap [i'..j']$. Consequently, S can be expressed as a union of an element of $\mathcal{S}(D_{i,m}, G[i'..m'])$ and an element of $\mathcal{S}(D_{m,j}, G[m'..j'])$. This case is thus covered by candidate 5.
- (d) Suppose that $D_{i,j} = \{\langle F[i..i+\ell]; F[j-r..j] \rangle\} \cup D_{i+\ell, j-r}$. If S does not contain any pair of the form $(\langle F[i..i+\ell]; F[j-r..j] \rangle, \langle G[i''..i''+\ell]; G[j''-r..j''] \rangle)$, then $S \in \mathcal{S}(D_{i+\ell, j-r}, G[\max(i+\ell-s, i').. \min(j-r+s, j')])$, and this case is covered by candidate 7. Otherwise, we must have $i'' \in [\max(i', i-s)..i+s]$ and $j'' \in [j-s.. \min(j', j+s)]$. This is covered by $i''-i'$ applications of candidate 2, $j'-j''$ applications of candidate 3, and finally an application of candidate 4 because $S \setminus \{(\langle F[i..i+\ell]; F[j-r..j] \rangle, \langle G[i''..i''+\ell]; G[j''-r..j''] \rangle)\} \in \mathcal{S}(D_{i+\ell, j-r}, G[i'+\ell..j'-r])$.

This completes the proof that Algorithm 5 is correct. The sought set S is obtained via a call $\text{Pairs}(D, G[0..|G|])$ which is valid as long as $s \geq ||F| - |G||$. Otherwise, there is no alignment $\mathcal{A} \in \text{TA}(F, G)$ of width at most s , and thus we return $S = \emptyset$.

As for the efficient implementation, we use memoization to make sure that each call to Pairs is executed at most once. The number of calls is $\mathcal{O}(|D| \cdot s^2)$ and each one performs $\mathcal{O}(s)$ instructions. In order to implement every instruction in $\mathcal{O}(1)$ time, we implement sets as persistent linked lists augmented with their size (this is valid because the arguments of every union operation are guaranteed to be disjoint). Moreover, we use Theorem 2.1 (for checking whether fragments of F match fragments of G) and Fact 3.1 (for checking whether fragments of G are balanced). Including the necessary preprocessing, the overall runtime is $\mathcal{O}(n + |D|s^3)$. \square

Lemma 3.17. *There exists a linear-time algorithm that, given a forest P and an integer $k \in \mathbb{Z}_+$, constructs a forest of length at most $74k^3$ that is $\text{ted}_{\leq k}^w$ -equivalent to P for every normalized quasimetric w .*

Algorithm 6: Construct a forest of length at most $74k^3$ that is $\text{ted}_{\leq k}^w$ -equivalent to P .

```

1 HorizontalReduction( $P, k$ ):
2    $P' \leftarrow \text{PeriodicityReduction}(P, 4k, \{Q \in \Sigma^+ : |Q| \leq 4k \text{ and } Q \text{ is a primitive forest}\});$ 
3   if  $|P'| \geq 74k^3$  then return  $\binom{37k^3}{a} \binom{37k^3}{a}$  for some  $a \in \Sigma$ ;
4   else return  $P'$ ;

```

Proof. We apply Lemma 2.13 with $e = 4k$ and \mathcal{Q} consisting of all primitive forests of length at most $4k$. We return $P'' := \binom{37k^3}{a} \binom{37k^3}{a}$ (for an arbitrary label $a \in \Sigma$) or P' depending on whether $|P'| \leq 74k^3$ or not.

Observe that \mathcal{Q} is chosen so that P' avoids horizontal k -periodicity and, by Lemma 3.5, P' is $\text{ted}_{\leq k}^w$ -equivalent to P for every normalized quasimetric w . Thus, the algorithm is correct if $|P'| < 74k^3$. Otherwise, Lemma 3.7 implies that P' and P'' are $\text{ted}_{\leq k}^w$ -equivalent for every normalized quasimetric w (both avoid horizontal k -periodicity and are of length at least $74k^3$).

The oracle testing in constant time whether a given fragment of P belongs to \mathcal{Q} can be implemented using Fact 3.1. Thus, by Lemma 2.13, the overall running time is linear. \square

Lemma 3.18. *There exists a linear-time algorithm that, given a context P and an integer $k \in \mathbb{Z}_+$, constructs a context of length at most $578k^4$ that is $\text{ted}_{\leq k}^w$ -equivalent to P for every normalized quasimetric w .*

Algorithm 7: Construct a context of length at most $578k^4$ that is $\text{ted}_{\leq k}^w$ -equivalent to P .

```

1 VerticalReduction( $P, k$ ):
2   Let  $P = P_0 \star \dots \star P_{e-1}$ , where each  $P_i$  is a context of depth 1;
3   for  $i \leftarrow 0$  to  $e$  do
4     Let  $P_i = \langle (a_i F_i; G_i)_{a_i} \rangle$ ;
5      $\mathbf{P} \leftarrow \mathbf{P} \cdot \langle (a_i \cdot \text{HorizontalReduction}(F_i, k); \text{HorizontalReduction}(G_i, k) \cdot)_{a_i} \rangle$ ;
6    $\mathcal{Q} \leftarrow \{\mathbf{Q} : \star_{i=0}^{|\mathbf{Q}|-1} \mathbf{Q}[i] \text{ is a primitive context of length at most } 8k\}$ ;
7    $\mathbf{P}' \leftarrow \text{PeriodicityReduction}(\mathbf{P}, 6k, \mathcal{Q})$ ;
8    $P' \leftarrow \star_{i=0}^{|\mathbf{P}'|-1} \mathbf{P}'[i]$ ;
9   if  $|P'| \geq 578k^4$  then return  $\star_{i=0}^{17k^2-1} \langle (a((a)_a)^i; ((a)_a)^{17k^2-1-i} \rangle$  for some  $a \in \Sigma$ ;
10  else return  $P'$ ;

```

Proof. Let $P = P_0 \star \dots \star P_{e-1}$, where each P_i is a context of depth 1, that is, $P_i = \langle (a_i F_i; G_i)_{a_i} \rangle$ for some label $a_i \in \Sigma$ and forests F_i, G_i . As the first step, our algorithm constructs a string $\mathbf{P}[0..e)$ whose characters are depth-1 contexts defined so that $\mathbf{P}[i] = \langle (a_i F'_i; G'_i)_{a_i} \rangle$, where forests $F'_i = \text{HorizontalReduction}(F_i, k)$ and $G'_i = \text{HorizontalReduction}(G_i, k)$ are constructed using Lemma 3.17. Next, we transform \mathbf{P} using Lemma 2.13 with $e = 6k$ and a family \mathcal{Q} defined so that $\mathbf{P}[i..j) \in \mathcal{Q}$ if and only if $\mathbf{P}[i] \star \dots \star \mathbf{P}[j-1)$ is a primitive context of length at most $8k$ (this implies $j - i \leq 4k$). In order to apply Lemma 2.13 to \mathbf{P} , we use linear-time string

sorting [PT87, AN94] to map characters of \mathbf{P} (depth-1 contexts) to integer identifiers. By composing the contexts corresponding to the resulting string \mathbf{P}' , we obtain a context P' . We return $P'' := \star_{i=0}^{17k^2-1} \langle (a)_a^i; (a)_a^{17k^2-1-i} \rangle$ (for an arbitrary label $a \in \Sigma$) or P' depending on whether $|P'| \geq 578k^4$ or not.

Note that $|P''| = \sum_{i=0}^{17k^2-1} (1 + 2 \cdot i + 2 \cdot (17k^2 - 1 - i) + 1) = 17k^2 \cdot 2 \cdot 17k^2 = 578k^4$. Thus, the resulting context (either P' or P'') is guaranteed to be of length at most $578k^4$. Let us now argue that it is $\text{ted}_{\leq k}^w$ -equivalent to P for every normalized quasimetric w . By Lemma 3.17, the forests F'_i and G'_i are $\text{ted}_{\leq k}^w$ -equivalent to F_i and G_i , respectively, and thus $\star_{i=0}^{e-1} \mathbf{P}[i]$ is $\text{ted}_{\leq k}^w$ -equivalent to P . By Lemma 2.13, the context P' is obtained from $\star_{i=0}^{e-1} \mathbf{P}[i]$ by repeatedly replacing Q^{6k+1} with Q^{6k} for primitive contexts Q of length at most $8k$. By Lemma 3.10, Q^{6k+1} is then $\text{ted}_{\leq k}^w$ -equivalent to Q^{6k} , so this operation preserves $\text{ted}_{\leq k}^w$ -equivalence, i.e., P' is also $\text{ted}_{\leq k}^w$ -equivalent to P . Moreover, each depth-1 context in \mathbf{P}' originates from \mathbf{P} , so each forest occurring in (either half of) P' is of length at most $74k^3$. Furthermore, Lemma 2.13 guarantees that P' is not of the form $C \star Q^{6k+1} \star D$ for any context Q of length at most $8k$, and thus P' avoids vertical k -periodicity. By construction, P'' avoids vertical k -periodicity and its halves contain only forests of lengths at most $74k^3$ (in fact, at most $34k^2$). Consequently, Lemma 3.12 implies that P'' is $\text{ted}_{\leq k}^w$ -equivalent to P' (and, by transitivity, to P) provided that $|P''| \geq 578k^4$.

As for the running time analysis, we note that all applications of Lemma 3.17 concern disjoint fragments of P , so the total cost of the calls to `HorizontalReduction` is linear. Assigning integer identifiers to contexts $\mathbf{P}[i]$ and applying Lemma 2.13 also takes linear time. Finally, P'' is constructed only if $|P'| \geq 578k^4$, so the cost of this step is also bounded by $\mathcal{O}(k^4) = \mathcal{O}(|P|)$. \square

Theorem 3.19. *There exists an $\mathcal{O}(n)$ -time algorithm that, given forests F, G of size at most $n \geq 12716k^5$ and an integer $k \in \mathbb{Z}_+$, constructs forests F', G' of lengths at most $\frac{n}{2} + 6358k^5$ such that $\text{ted}_{\leq k}^w(F, G) = \text{ted}_{\leq k}^w(F', G')$ holds for every normalized quasimetric w .*

Proof. By symmetry, we assume without loss of generality that $|F| \geq |G|$. We start by applying Lemma 3.15 to construct a piece decomposition D of F consisting of at most $12k - 1$ pieces of length at most $\lceil \frac{n}{2k} \rceil$ each. Next, we use Lemma 3.16 to identify a maximum-size set $S \subseteq D \times \mathcal{P}(G)$ that, for some alignment $\mathcal{A} \in A(F, G)$ of width at most $2k$, contains only pairs of pieces that \mathcal{A} matches perfectly. If $|S| < |D| - k$, we return $F' = ((a)_a)^{k+1}$ and $G' = \varepsilon$ for some $a \in \Sigma$. Otherwise, for each pair of matching forests $F[i..j] = P = G[i'..j']$ in S , we use Lemma 3.17 to construct a forest P' of length at most $74k^3$ that is $\text{ted}_{\leq k}^w$ -equivalent to P for every normalized quasimetric w . We replace the occurrences of P at $F[i..j]$ and $G[i'..j']$ by occurrences of P' . Similarly, for every pair of matching contexts $\langle F[i..i+\ell]; F[j-r..j] \rangle = P = \langle G[i'..i'+\ell]; G[j'-r..j'] \rangle$ in S , we use Lemma 3.18 to construct a context P' of length at most $578k^4$ that is $\text{ted}_{\leq k}^w$ -equivalent to P for every normalized quasimetric w . We replace the occurrences of P at $\langle F[i..i+\ell]; F[j-r..j] \rangle$ and $\langle G[i'..i'+\ell]; G[j'-r..j'] \rangle$ by occurrences of P' .

If $\text{ted}^w(F, G) \leq k$ holds for any normalized weight function w , then the unweighted cost of the underlying optimal alignment does not exceed k . Thus, its width is at most $2k$, and it matches perfectly all but at most k pieces of D . Consequently, in that case, $|S| \geq |D| - k$. In particular, if we return $F' = ((a)_a)^{k+1}$ and $G' = \varepsilon$ for some $a \in \Sigma$, then $\text{ted}_{\leq k}^w(F, G) = \infty = \text{ted}_{\leq k}^w(F', G')$ holds as claimed. In that case, $|F'|, |G'| \leq 2k + 2 \leq 4k < 6358k^5$. Otherwise, by definition of $\text{ted}_{\leq k}^w$ -equivalence, the resulting forests F' and G' satisfy $\text{ted}_{\leq k}^w(F, G) = \text{ted}_{\leq k}^w(F', G')$. Moreover, $|F'| \leq k \cdot \lceil \frac{n}{2k} \rceil + (11k - 1) \cdot 578k^4 \leq \frac{n}{2} + k + (11k - 1) \cdot 578k^4 \leq \frac{n}{2} + 11 \cdot 578 \cdot k^5 = \frac{n}{2} + 6358k^5$ and $|G'| = |F'| + |G| - |F| \leq |F'| \leq \frac{n}{2} + 6358k^5$.

The applications of Lemmas 3.15 and 3.16 cost $\mathcal{O}(n)$ and $\mathcal{O}(n + k^4) = \mathcal{O}(n)$ time, respectively. The calls to `HorizontalReduction` and `VerticalReduction` concern disjoint pieces of F , so their

total cost is $\mathcal{O}(n)$ by Lemmas 3.17 and 3.18, respectively. \square

Corollary 3.20. *There exists a linear-time algorithm that, given forests F, G and an integer $k \in \mathbb{Z}_+$, constructs forests F', G' of lengths at most $12717k^5$ such that $\text{ted}_{\leq k}^w(F, G) = \text{ted}_{\leq k}^w(F', G')$ holds for every normalized quasimetric w .*

Proof. We iteratively apply Theorem 3.19 as long as $\max(|F|, |G|) > 12717k^5$ and return the resulting pair of forests. Formally, we construct a sequence $(F_i, G_i)_{i=0}^t$ such that $(F_0, G_0) = (F, G)$ and $\text{ted}_{\leq k}^w(F_{i+1}, G_{i+1}) = \text{ted}_{\leq k}^w(F_i, G_i)$ holds for every $i \in [0..t)$. Consider forests (F_i, G_i) at iteration i . If $n_i := \max(|F_i|, |G_i|) \leq 12717k^5$, we set $t := i$ and return $(F', G') := (F_i, G_i)$. Otherwise, we apply Theorem 3.19 to derive forests F_{i+1} and G_{i+1} of lengths at most $n_{i+1} := \max(|F_{i+1}|, |G_{i+1}|) \leq \frac{1}{2}n_i + 6358k^5$ such that $\text{ted}_{\leq k}^w(F_{i+1}, G_{i+1}) = \text{ted}_{\leq k}^w(F_i, G_i)$. Since $n_{i+1} - 12716k^5 \leq \frac{1}{2}(n_i - 12716k^5)$, the value n_i strictly decreases at each iteration and thus the process terminates. Moreover, the running time of each iteration is $\mathcal{O}(n_i) = \mathcal{O}(n_i - 12716k^5)$. The latter values form a geometric series dominated by the leading term at $i = 0$. Hence, the total running time is linear in the input size. \square

Theorem 1.2. *Given forests F, G of length at most n , an integer $k \in \mathbb{Z}_+$, and a quasimetric w , the value $\text{ted}_{\leq k}^w(F, G)$ can be computed in $\mathcal{O}(n + k^{15})$ time. Moreover, $\text{ted}_{\leq k}(F, G)$ can be computed in $\mathcal{O}(n + k^7 \log k)$ time.*

Proof. We first apply Corollary 3.20 to build forests F', G' of length $\mathcal{O}(k^5)$ such that $\text{ted}_{\leq k}^w(F, G) = \text{ted}_{\leq k}^w(F', G')$. Then, we compute $\text{ted}_{\leq k}^w(F', G')$ using the algorithm of Demaine, Mozes, Rossman, and Weimann [DMRW10]. The running times of these two steps are $\mathcal{O}(n)$ and $\mathcal{O}((k^5)^3)$, respectively, for a total of $\mathcal{O}(n + k^{15})$. If w is the discrete metric (the unweighted case), then we compute $\text{ted}_{\leq k}(F', G')$ using the algorithm of Akmal and Jin [AJ21], which costs $\mathcal{O}(k^5 \cdot k^2 \cdot \log(k^5)) = \mathcal{O}(k^7 \log k)$ time. \square

4 Dyck Edit Distance

In this section we give a deterministic algorithm that computes weighted Dyck edit distance of a given input string. Formally we show the following.

Theorem 1.3. *Given a string X of length n , an integer $k \in \mathbb{Z}_+$, and a skewmetric w , the value $\text{dyck}_{\leq k}^w(X)$ can be computed in $\mathcal{O}(n + k^{12})$ time.*

4.1 Preliminaries

In Dyck Language, the alphabet Σ consists of two disjoint sets T and \bar{T} of *opening* and *closing* parentheses, respectively, with a bijection $f : T \rightarrow \bar{T}$ mapping each opening parenthesis to the corresponding closing parenthesis. We extend this mapping to an involution $f : T \cup \bar{T} \rightarrow T \cup \bar{T}$ and then to an involution $f : \Sigma^* \rightarrow \Sigma^*$ mapping each string $X[0]X[1] \cdots X[|X| - 1]$ to its reverse complement $\overline{X[|X| - 1] \cdots X[1]X[0]}$. Given two strings X, Y , we denote their concatenation by XY or $X \cdot Y$.

The *Dyck* language $\text{Dyck}(\Sigma) \subseteq \Sigma^*$ consists of all well-parenthesized expression over Σ ; formally, it can be defined using a context-free grammar whose only non-terminal S admits productions $S \rightarrow SS, S \rightarrow \emptyset$ (empty string), and $S \rightarrow aS\bar{a}$ for all $a \in T$.

Definition 4.1 (Heights). Given an alphabet set Σ , define the function $h : \Sigma \rightarrow \{-1, 1\}$ where $h(a) = 1$ if $a \in \Sigma$ is an opening parenthesis and $h(a) = -1$ otherwise. Given a string $X \in \Sigma^n$, define the height of a position i where $0 \leq i \leq n$, as $H(i) = \sum_{j=0}^{i-1} h(X[j])$.

Here $H(i)$ is the difference between the number of opening parentheses and the number of closing parentheses in $X[0..i]$.

Definition 4.2 (Peaks and valleys). Given a string $X \in \Sigma$, an index $i \in [1..n]$ is called a peak if $H(i-1) < H(i) > H(i+1)$ and a valley if $H(i-1) > H(i) < H(i+1)$.

4.2 Dyck Language Alignments and Weighted Dyck Edit Distance

We say that $\mathcal{M} \subseteq \{(i, j) \subseteq \mathbb{Z}^2 : i < j\}$ is a *non-crossing matching* if any two distinct pairs $(i, j), (i', j') \in \mathcal{M}$ satisfy $i < j < i' < j'$ or $i < i' < j' < j$. Such a matching can also be interpreted as a function $\mathcal{M} : \mathbb{Z} \rightarrow \mathbb{Z} \cup \{\perp\}$ with $\mathcal{M}(i) = j$ if $(i, j) \in \mathcal{M}$ or $(j, i) \in \mathcal{M}$ for some $j \in \mathbb{Z}$, and $\mathcal{M}(i) = \perp$ otherwise. For a string $X \in \Sigma^*$ we define its *Dyck language alignment* to be a matching function \mathcal{M} as defined above.

For two fragments $X[p..q]$ and $X[p'..q']$ of X , we write $X[p..q] \simeq_{\mathcal{M}} \overline{X[p'..q']}$ if $X[p..q] = \overline{X[p'..q']}$ and $(r, q' - r - 1)$ holds for every $r \in [p..q]$.

Similar to Section 2.2 we define a weight function w on $\bar{\Sigma} := \Sigma \cup \{\varepsilon\}$. We call this weight function a *skewmetric* if it satisfies the triangle inequality, that is, $w(a, b) + w(b, c) \geq w(a, c)$ holds for every $a, b, c \in \bar{\Sigma}$ and skew-symmetry, that is, $w(a, b) = w(\bar{b}, \bar{a})$ holds for every $a, b \in \bar{\Sigma}$. In the rest of this section we assume the weight function w to be skewmetric unless stated otherwise.

Definition 4.3. The weighted Dyck edit distance of a string $X \in \Sigma^*$ with respect to a weight function w is the minimum edit distance $\text{ed}^w(X, Y)$ between X and a string $Y \in \text{Dyck}(\Sigma)$. Formally,

$$\text{dyck}^w(X) = \min_{Y \in \text{Dyck}(\Sigma)} \text{ed}^w(X, Y).$$

For $k \in \mathbb{R}_{\geq 0}$, we also denote

$$\text{dyck}_{\leq k}^w(X) = \begin{cases} \text{dyck}^w(X) & \text{if } \text{dyck}^w(X) \leq k, \\ \infty & \text{otherwise.} \end{cases}$$

The *cost* of an alignment $\mathcal{M} \in \mathbf{M}(X)$ with respect to a *weight function* w , denoted $\text{dyck}_{\mathcal{M}}^w(X)$, is defined as

$$\text{dyck}_{\mathcal{M}}^w(X) = \sum_{(i,j) \in \mathcal{M}} \text{dyck}^w(X[i]X[j]) + \sum_{i \in [0..|X|]: \mathcal{M}(i) = \perp} \text{dyck}^w(X[i]).$$

Fact 4.4. For every string X and weight function w , we have $\text{dyck}^w(X) = \min_{\mathcal{M} \in \mathbf{M}(X)} \text{dyck}_{\mathcal{M}}^w(X)$.

Proof. We first show by induction on $|X|$ that $\text{dyck}^w(X) \leq \text{dyck}_{\mathcal{M}}^w(X)$ holds for every $\mathcal{M} \in \mathbf{M}(X)$. The claim is trivial if $|X| = 0$. If $\mathcal{M}(0) = \perp$, then we construct $\mathcal{M}' := \{(i-1, j-1) : (i, j) \in \mathcal{M}\}$ and $X' := X[1..|X|]$. By the inductive assumption, $\text{dyck}^w(X) \leq \text{dyck}^w(X') + \text{dyck}^w(X[0]) \leq \text{dyck}_{\mathcal{M}'}^w(X') + \text{dyck}^w(X[0]) = \text{dyck}_{\mathcal{M}}^w(X)$. If $\mathcal{M}(0) = |X| - 1$, then we construct $\mathcal{M}' := \{(i-1, j-1) : (i, j) \in \mathcal{M} \setminus \{(0, |X| - 1)\}\}$ and $X' = [1..|X| - 1]$. By the inductive assumption, $\text{dyck}^w(X) \leq \text{dyck}^w(X') + \text{dyck}^w(X[0]X[|X| - 1]) \leq \text{dyck}_{\mathcal{M}'}^w(X') + \text{dyck}^w(X[0]X[|X| - 1]) = \text{dyck}_{\mathcal{M}}^w(X)$. Otherwise, we have $(0, p) \in \mathcal{M}$ for some $p \in [1..|X| - 1]$. In this case, we construct $\mathcal{M}' := \{(i, j) \in \mathcal{M} : j \leq p\}$ and $X' := X[0..p]$, as well as $\mathcal{M}'' := \{(i-p-1, j-p-1) : (i, j) \in \mathcal{M} \text{ and } i > p\}$ and $X'' := X[p+1..|X|]$. By the inductive assumption, $\text{dyck}^w(X) \leq \text{dyck}^w(X') + \text{dyck}^w(X'') \leq \text{dyck}_{\mathcal{M}'}^w(X') + \text{dyck}_{\mathcal{M}''}^w(X'') = \text{dyck}_{\mathcal{M}}^w(X)$; here, the last equality follows from the fact that $|\mathcal{M}| = |\mathcal{M}'| + |\mathcal{M}''|$: any $(i, j) \in \mathcal{M}$ with $i \leq p$ and $j > p$ would violate the non-crossing property of \mathcal{M} .

Next, we show by induction on $|X|$ that there exists $\mathcal{M} \in \mathcal{M}(X)$ such that $\text{dyck}_{\mathcal{M}}^w(X) \leq \text{dyck}^w(X)$; again, the claim is trivial for $|X| = 0$. Let us fix $Y \in \text{Dyck}(\Sigma)$ and $\mathcal{A} \in \mathcal{A}(X, Y)$ such that $\text{dyck}^w(X) = \text{ed}_{\mathcal{A}}^w(X, Y)$. If \mathcal{A} deletes $X[0]$, we consider $X' := X[1..|X|]$. The inductive assumption yields a matching \mathcal{M}' such that $\text{dyck}_{\mathcal{M}'}^w(X') \leq \text{dyck}^w(X')$. In this case, we set $\mathcal{M} := \{(i+1, j+1) : (i, j) \in \mathcal{M}'\}$ so that $\text{dyck}_{\mathcal{M}}^w(X) = \text{dyck}^w(X[0]) + \text{dyck}_{\mathcal{M}'}^w(X') \leq \text{ed}^w(X[0], \varepsilon) + \text{dyck}^w(X') \leq w(X[0], \varepsilon) + \text{ed}^w(X', Y') \leq \text{ed}_{\mathcal{A}}^w(X, Y) = \text{dyck}^w(X)$. The case when \mathcal{A} deletes $X[|X|-1]$ is symmetric, so we may assume that \mathcal{A} deletes neither $X[0]$ nor $X[|X|-1]$; in particular, $Y \neq \varepsilon$.

Suppose that $Y = Y' \cdot Y''$ for some non-empty strings $Y', Y'' \in \text{Dyck}(\Sigma)$. This yields a decomposition $X = X' \cdot X''$ such that $\text{ed}^w(X, Y) = \text{ed}^w(X', Y') + \text{ed}^w(X'', Y'')$. Moreover, the optimality of Y guarantees that X' and X'' are both non-empty. The inductive assumption yields matchings $\mathcal{M}', \mathcal{M}''$ such that $\text{dyck}_{\mathcal{M}'}^w(X') \leq \text{dyck}^w(X')$ and $\text{dyck}_{\mathcal{M}''}^w(X'') \leq \text{dyck}^w(X'')$. In this case, we set $\mathcal{M} := \mathcal{M}' \cup \{(i+|X'|, j+|X''|) : (i, j) \in \mathcal{M}''\}$ so that $\text{dyck}_{\mathcal{M}}^w(X) = \text{dyck}_{\mathcal{M}'}^w(X') + \text{dyck}_{\mathcal{M}''}^w(X'') \leq \text{dyck}^w(X') + \text{dyck}^w(X'') \leq \text{ed}^w(X', Y') + \text{ed}^w(X'', Y'') \leq \text{ed}_{\mathcal{A}}^w(X, Y) = \text{dyck}^w(X)$.

In the remaining case, we must have $Y = aY'\bar{a}$ for $a \in T$ and $Y' \in \text{Dyck}(\Sigma)$. Let us first suppose that \mathcal{A} aligns $Y[0] = a$ with $X[0]$ and $Y[|Y|-1] = \bar{a}$ with $X[|X|-1]$. In this case, \mathcal{A} aligns $Y[1..|Y|-1] = Y'$ with $X' := X[1..|X|-1]$. The inductive assumption yields a matching \mathcal{M}' such that $\text{dyck}_{\mathcal{M}'}^w(X') \leq \text{dyck}^w(X')$. In this case, we set $\mathcal{M} := \{(0, |X|-1)\} \cup \{(i+1, j+1) : (i, j) \in \mathcal{M}'\}$ so that $\text{dyck}_{\mathcal{M}}^w(X) = \text{dyck}^w(X[0]X[|X|-1]) + \text{dyck}_{\mathcal{M}'}^w(X') \leq \text{ed}^w(X[0]X[|X|-1], a\bar{a}) + \text{dyck}^w(X') \leq w(X[0], a) + w(X[|X|-1], \bar{a}) + \text{ed}^w(X', Y') \leq \text{ed}_{\mathcal{A}}^w(X, Y) = \text{dyck}^w(X)$. Next, suppose that \mathcal{A} aligns $Y[0] = a$ with $X[0]$ but inserts $Y[|Y|-1] = \bar{a}$. In this case, \mathcal{A} aligns $Y[1..|Y|-1] = Y'$ with $X' := X[1..|X|]$. The inductive assumption yields a matching \mathcal{M}' such that $\text{dyck}_{\mathcal{M}'}^w(X') \leq \text{dyck}^w(X')$. In this case, we set $\mathcal{M} := \{(i+1, j+1) : (i, j) \in \mathcal{M}'\}$ so that $\text{dyck}_{\mathcal{M}}^w(X) = \text{dyck}^w(X[0]) + \text{dyck}_{\mathcal{M}'}^w(X') \leq \text{ed}^w(X[0], a\bar{a}) + \text{dyck}^w(X') \leq w(X[0], a) + w(\varepsilon, \bar{a}) + \text{ed}^w(X', Y') \leq \text{ed}_{\mathcal{A}}^w(X, Y) = \text{dyck}^w(X)$. The case when \mathcal{A} inserts $Y[0] = a$ and aligns $Y[|Y|-1] = \bar{a}$ with $X[|X|-1]$ is symmetric. The case when \mathcal{A} inserts both $Y[0] = a$ and $Y[|Y|-1] = \bar{a}$ is impossible by optimality of Y . Finally, we note that, since \mathcal{A} deletes neither $X[0]$ nor $X[|X|-1]$, the alignment \mathcal{A} cannot align $Y[0]$ to any character other than $X[0]$ and $Y[|Y|-1]$ to any character other than $Y[|Y|-1]$. Thus, the case analysis above is complete. \square

Claim 4.5. For every $x \in \Sigma$ and skewmetric weight function w , $\text{dyck}^w(x) = w(x, \varepsilon) = w(\varepsilon, \bar{x})$.

Proof. We consider the following three different cases.

Case 1: x is deleted. In this case $\text{dyck}^w(x) = w(x, \varepsilon)$.

Case 2: \bar{x} is inserted after x if $x \in T$ and before x if $x \in \bar{T}$. In this case $\text{dyck}^w(x) = w(\varepsilon, \bar{x}) = w(x, \varepsilon)$. The last equality follows as w is skew-symmetric. Thus an insertion can be replaced with a deletion. From now on wards we assume that only allowed edits are deletion and substitutions.

Case 3: x is substituted by some $y \in \Sigma$. Here we also need to insert \bar{y} . Thus $\text{dyck}^w(x) = w(x, y) + w(\varepsilon, \bar{y}) = w(x, y) + w(y, \varepsilon) \geq w(x, \varepsilon)$. The second equality follows as w is skew-symmetric and the last inequality follows as w obeys triangle inequality. Trivially $\text{dyck}^w(x) \leq w(x, \varepsilon)$. Thus the claim follows. \square

Claim 4.6. For every $x, y \in \bar{\Sigma}$ and skewmetric weight function w , $\text{dyck}^w(xy) = \min_{z \in T \cup \{\varepsilon\}} w(x, z) + w(y, \bar{z})$.

Proof. Let $z \in T \cup \{\varepsilon\}$ minimizes $w(x, z) + w(y, \bar{z})$. It is straight forward to argue that $\text{dyck}^w(xy) \leq w(x, z) + w(y, \bar{z})$ as x, y can be substituted by z, \bar{z} respectively. Next we argue the converse. Following Claim 4.5 we assume the only allowed edits are deletions and substitutions.

Case 1: Both x and y are deleted. Here $\text{dyck}^w(xy) \geq w(x, \epsilon) + w(y, \epsilon)$. The claim follows as $\epsilon \in T \cup \{\epsilon\}$ and $\bar{\epsilon} = \epsilon$.

Case 2: x is substituted by \bar{y} . Here $\text{dyck}^w(xy) \geq w(x, \bar{y}) = w(x, \bar{y}) + w(y, y)$. Thus the claim follows as $\bar{y} \in T$.

Case 3: y is substituted by \bar{x} . Here $\text{dyck}^w(xy) \geq w(y, \bar{x}) = w(x, x) + w(y, \bar{x})$. Thus the claim follows as $x \in T$.

Case 3: x is substituted by z and y is substituted by \bar{z} . Here $\text{dyck}^w(xy) \geq w(x, z) + w(y, \bar{z})$. Thus the claim follows as $z \in T$. \square

From now on wards we assume w to be skew-symmetric.

4.2.1 Preprocessing.

Given the input string $X \in \Sigma^n$, preprocess X as follows. As long as there are two neighboring indices $i, i+1$ such that $X[i+1] = \overline{X[i]}$ and $X[i] \in T$ remove them. Let the resulting string be X' . We make the following claim.

Claim 4.7. $\text{dyck}^w(X) = \text{dyck}^w(X')$.

Proof. Let \mathcal{M} be an optimal alignment of X . For contradiction assume for two consecutive indices $i, i+1$, $X[i+1] = \overline{X[i]}$, $X[i] \in T$ but $(i, i+1) \notin \mathcal{M}$. Next depending on the matching indices of $i, i+1$, we consider the following three cases.

Case 1: Let $(j, i), (i+1, k) \in \mathcal{M}$ where $j \in [0 \dots i] \cup \{\perp\}$ and $k \in (i+1 \dots |X|) \cup \{\perp\}$. In this case we create another alignment $\mathcal{M}' = \mathcal{M} \setminus \{(j, i), (i+1, k)\} \cup \{(i, i+1), (j, k)\}$. We argue $\text{dyck}^w(X[j]X[k]) \leq \text{dyck}^w(X[j]X[i]) + \text{dyck}^w(X[i+1]X[k])$, thus proving $\text{dyck}_{\mathcal{M}'}^w(X) \leq \text{dyck}_{\mathcal{M}}^w(X)$. Following Claim 4.6, let $a, b \in T \cup \{\epsilon\}$ be such that $\text{dyck}^w(X[j]X[i]) = w(X[j], a) + w(X[i], \bar{a})$ and $\text{dyck}^w(X[i+1]X[k]) = w(X[i+1], b) + w(X[k], \bar{b})$. Thus,

$$\begin{aligned} \text{dyck}^w(X[j]X[i]) + \text{dyck}^w(X[i+1]X[k]) &= w(X[j], a) + w(X[i], \bar{a}) + w(X[i+1], b) + w(X[k], \bar{b}) \\ &= w(X[j], a) + w(a, \overline{X[i]}) + w(X[i+1], b) + w(X[k], \bar{b}) \\ &\geq w(X[j], \overline{X[i]}) + w(X[i+1], b) + w(X[k], \bar{b}) \\ &\geq w(X[j], b) + w(X[k], \bar{b}) \\ &\geq \text{dyck}^w(X[j], X[k]) \end{aligned}$$

The second equality follows as w is skew-symmetric; thus $w(X[i], \bar{a}) = w(a, \overline{X[i]})$. The third and fourth inequality follows as w follows triangle inequality and $\overline{X[i]} = X[i+1]$. The last inequality follows from Claim 4.6.

Case 2: Let $(k, i), (j, i+1) \in \mathcal{M}$ where $k, j \in [0 \dots i] \cup \{\perp\}$ and $j < k$. In this case we create another alignment $\mathcal{M}' = \mathcal{M} \setminus \{(k, i), (j, i+1)\} \cup \{(i, i+1), (j, k)\}$. We argue $\text{dyck}^w(X[j]X[k]) \leq \text{dyck}^w(X[k]X[i]) + \text{dyck}^w(X[j]X[i+1])$, thus proving $\text{dyck}_{\mathcal{M}'}^w(X) \leq \text{dyck}_{\mathcal{M}}^w(X)$. Following Claim 4.6, let $a, b \in T \cup \{\epsilon\}$ be such that $\text{dyck}^w(X[k]X[i]) = w(X[k], a) + w(X[i], \bar{a})$ and $\text{dyck}^w(X[j]X[i+1]) =$

$w(X[j], b) + w(X[i+1], \bar{b})$. Thus,

$$\begin{aligned}
\text{dyck}^w(X[k]X[i]) + \text{dyck}^w(X[j]X[i+1]) &= w(X[k], a) + w(X[i], \bar{a}) + w(X[j], b) + w(X[i+1], \bar{b}) \\
&= w(X[k], a) + w(a, \overline{X[i]}) + w(X[j], b) + w(X[i+1], \bar{b}) \\
&\geq w(X[k], \overline{X[i]}) + w(X[i+1], \bar{b}) + w(X[j], b) \\
&\geq w(X[k], \bar{b}) + w(X[j], b) \\
&\geq \text{dyck}^w(X[j], X[k])
\end{aligned}$$

Case 3: Let $(i, k), (i+1, j) \in \mathcal{M}$ where $k, j \in (i+1 \dots |X|) \cup \{\perp\}$ and $j < k$. In this case we create another alignment $\mathcal{M}' = \mathcal{M} \setminus \{(i, k), (i+1, j)\} \cup \{(i, i+1), (j, k)\}$. We argue $\text{dyck}^w(X[j]X[k]) \leq \text{dyck}^w(X[i]X[k]) + \text{dyck}^w(X[i+1]X[j])$, thus proving $\text{dyck}_{\mathcal{M}'}^w(X) \leq \text{dyck}_{\mathcal{M}}^w(X)$. Following Claim 4.6, let $a, b \in T \cup \{\epsilon\}$ be such that $\text{dyck}^w(X[i]X[k]) = w(X[i], a) + w(X[k], \bar{a})$ and $\text{dyck}^w(X[i+1]X[j]) = w(X[i+1], b) + w(X[j], \bar{b})$. Thus,

$$\begin{aligned}
\text{dyck}^w(X[i]X[k]) + \text{dyck}^w(X[i+1]X[j]) &= w(X[i], a) + w(X[k], \bar{a}) + w(X[i+1], b) + w(X[j], \bar{b}) \\
&= w(X[i], a) + w(X[k], \bar{a}) + w(X[j], \bar{b}) + w(\bar{b}, \overline{X[i+1]}) \\
&\geq w(X[i], a) + w(X[k], \bar{a}) + w(X[j], \overline{X[i+1]}) \\
&\geq w(X[j], a) + w(X[k], \bar{a}) \\
&\geq \text{dyck}^w(X[j], X[k])
\end{aligned}$$

□

The preprocessing can be done in time $O(n)$. Also, we can assume that in the preprocessed string no two neighbouring symbols can be aligned. Following this and Claim 35 from [], we can make the following claim.

Claim 4.8. *Let $X \in \Sigma^n$. There exists an algorithm that preprocesses X in $O(n)$ time, and either declares $\text{dyck}^w(X) > k$, or outputs a string X' of length at most n such that $\text{dyck}^w(X) = \text{dyck}^w(X')$ and X' has at most $2k$ valleys.*

Thus from now on wards we assume X to be preprocessed and has at most $2k$ valleys.

4.3 Periodicity Reduction

Definition 4.9. For $k \in \mathbb{Z}_{>0}$ a fragments $X[a..b]$ and $X[c..d]$ of a string X are k -synchronized if $X[a..b] \in T^*$, $X[c..d] \in \overline{T}^*$, $b - a = d - c$, $b \leq c$, and $H(b) + H(c) - 2 \min_{m \in [b..c]} H(m) \leq 2k$.

Note that $X[a..b]$ and $X[c..d]$ are 0-synchronized if and only if (a, b, c, d) is a trapezoid.

Definition 4.10. For $k \in \mathbb{Z}_{\geq 0}$ and a skewmetric weight function w , strings $P, P' \in T^*$ are called $\text{dyck}_{\leq k}^w$ -equivalent if

$$\text{dyck}_{\leq k}^w(X) = \text{dyck}_{\leq k}^w(X[0..a] \cdot P' \cdot X[b..c] \cdot \overline{P'} \cdot X[d..|X|])$$

holds for every string X with k -synchronized fragments $X[a..b] = P$ and $X[c..d] = \overline{P}$.

Fact 4.11 (Fact 36, [BO16]). *Let \mathcal{M} be an alignment such that $\text{dyck}_{\mathcal{M}}^w(X) \leq k$. If $X[a..b] \simeq_{\mathcal{M}} X[c..d]$, then the fragments $X[a..b]$ and $X[c..d]$ are k -synchronized.*

Fact 4.12. Consider a string X and an alignment $\mathcal{M} \in \mathbf{M}(X)$ such that $\text{dyck}_{\mathcal{M}}(X) \leq k$ for some $k \in \mathbb{Z}_{\geq 0}$. Moreover, let $X[a..b]$ and $X[c..d]$ be k -synchronized fragments of length $\ell > 6k$. Then, there exist k -synchronized fragments $X[a'..b']$ and $X[c'..d']$ of length $\ell' \geq \frac{\ell-6k}{k+1}$, such that $X[a'..b'] \simeq_{\mathcal{M}} \overline{X[c'..d']}$ and $a \leq a' \leq b' \leq b \leq c \leq c' \leq d' \leq d$. Furthermore, we then have $|(a+d) - (a'+d')| \leq 4k$.

Proof. Since \mathcal{M} is non-crossing, it is disjoint with $[a..b] \times [d..|X|]$ or $[0..a] \times [c..d]$. By symmetry (up to the reverse complement), let us assume that \mathcal{M} is disjoint with $[a..b] \times [d..|X|]$. Consider $x \in [a..b-4k]$ such that $X[x] \simeq_{\mathcal{M}} \overline{X[y]}$. The assumption implies that $y < d$. Moreover, $b-x = H(b) - H(x) > 4k$, so $H(x) < H(b) - 4k$. At the same time, $|H(y+1) - H(x)| \leq 2k$, so $H(y+1) < H(b) - 2k$. Since $X[a..b]$ and $X[c..d]$ are k -synchronized, this means that $y+1 \notin [b..c]$, i.e., $y \in [c..d]$. Consider the fragment $X[a..b-4k]$ and the minimal subfragment of $X[c..d]$ containing positions that \mathcal{M} matches perfectly to positions $X[x]$ with $x \in [a..b-4k]$. These two fragments contain at most $2k$ positions that are deleted or matched imperfectly. The remaining positions constitute a common subsequence of $X[a..b-4k]$ and $X[c..d]$; this subsequence can be interrupted at most k times, so there is a contiguous subsequence $X[a'..b'] \simeq_{\mathcal{M}} X[c'..d']$ of length at least $\frac{\ell-6k}{k+1}$. Due to $|H(a) - H(d)| \leq 2k$ and $|H(a') - H(d')| \leq 2k$, we have $4k \geq |H(a) - H(d) - H(a') + H(d')| = |a - a' + d - d'|$. \square

Lemma 4.13. Let $k \in \mathbb{Z}_+$, let $Q \in T^*$ be a string, and let $e, e' \in \mathbb{Z}_{\geq 8k}$. Then Q^e and $Q^{e'}$ are $\text{dyck}_{\leq k}^w$ -equivalent for every skewmetric weight function w .

Proof. We assume without loss of generality that Q is primitive. (If $Q = R^m$ for $m \in \mathbb{Z}_{\geq 2}$, then $Q^e = R^{me}$ and $Q^{e'} = R^{me'}$ can be interpreted as powers of R rather than powers of Q .) Let $q = |Q|$. Consider a string X and positions $p_T, p_{\overline{T}}$ such that $Q^e = X[p_T..p_T + e \cdot q]$ and $\overline{Q^e} = X[p_{\overline{T}} - e \cdot q..p_{\overline{T}}]$ are k -synchronized fragments. Denote $X[0..p_T] \cdot Q^{e'} \cdot X[p_T + e \cdot q..p_T - e \cdot q] \cdot \overline{Q^{e'}} \cdot X[p_{\overline{T}}..|X|]$. Moreover, let $\mathcal{M} \in \mathbf{M}(X)$ be an alignment such that $\text{dyck}^w(X, Y) = \text{dyck}_{\mathcal{M}}^w(X, Y) \leq k$.

Claim 4.14. There exist $i_T, i_{\overline{T}} \in [0..7k]$ such that

$$X[p_T + i_T \cdot q..p_T + (i_T + 1) \cdot q] \simeq_{\mathcal{M}} \overline{X[p_{\overline{T}} - (i_{\overline{T}} + 1) \cdot q..p_{\overline{T}} - i_{\overline{T}} \cdot q]}.$$

Proof. Consider the $8k$ occurrences of Q starting at positions $p_T + i \cdot q$ for $i \in [0..7k]$ (let this fragment be P) and $8k$ occurrences of \overline{Q} ending at positions $p_{\overline{T}} - i \cdot q$ for $i \in [0..7k]$ (let this fragment be \overline{P}). Note P, \overline{P} are also k -synchronized fragments. Thus following Fact 4.12, there exists at least one occurrence of Q in P , starting at index ℓ such that \mathcal{M} matches it exactly with a fragment in \overline{P} . We can thus define $i_T \in [0..7k]$ so that \mathcal{M} matches $X[p_T + i_T \cdot q..p_T + (i_T + 1) \cdot q]$ exactly to some fragment $X[s_{\overline{T}} - q..s_{\overline{T}}] \in \overline{P}$. By definition of \overline{P} , we have $s_{\overline{T}} \geq p_{\overline{T}} - 7kq$. Furthermore, since Q is primitive (i.e., distinct from all its non-trivial cyclic rotations), we conclude that $s_{\overline{T}} = p_{\overline{T}} - i_{\overline{T}} \cdot q$ for some $i_{\overline{T}} \in [0..7k]$. \square

Now, if $Q^e = X[p_T..p_T + e \cdot q]$ is replaced with $Q^{e'}$ and $\overline{Q^e} = X[p_{\overline{T}} - e \cdot q..p_{\overline{T}}]$ is replaced with $\overline{Q^{e'}}$ for $e' \geq e - 1$, we can interpret this as replacing $Q = X[p_T + i_T \cdot q..p_T + (i_T + 1) \cdot q]$ with $Q^{1+e'-e}$ and $\overline{Q} = X[p_{\overline{T}} - (i_{\overline{T}} + 1) \cdot q..p_{\overline{T}} - i_{\overline{T}} \cdot q]$ with $\overline{Q}^{1+e'-e}$. By Claim 4.14, \mathcal{M} can be trivially adapted without modifying its cost, and hence $\text{dyck}^w(X') \leq \text{dyck}_{\mathcal{M}}^w(X) = \text{dyck}^w(X)$. If $e' < e - 1$, we repeat the above argument to decrement the exponent e one step at a time, still concluding that $\text{dyck}^w(X') \leq \text{dyck}^w(X)$. In either case, the converse inequality follows by symmetry between (X, e) and (X', e') . \square

We say that a string $P \in T^*$ avoids k -periodicity if it does not contain any substring Q^{8k+1} with $|Q| \in [1..4k]$.

Lemma 4.15. *Let $k \in \mathbb{Z}_+$ and let $P, P' \in T^*$ be strings of lengths at least $156k^3$ such that $P[0..78k^3] = P'[0..78k^3]$ and $P[|P| - 78k^3..|P|] = P'[|P'| - 78k^3..|P'|]$ avoid k -periodicity. Then, P and P' are $\text{dyck}_{\leq k}^w$ -equivalent for every skewmetric weight function w .*

Proof. Consider a string X and positions $p_T, p_{\overline{T}}$ such that $P = X[p_T..p_T + |P|]$, $\overline{P} = X(p_{\overline{T}} - |P|..p_{\overline{T}}]$ are k -synchronized fragments. Denote $X' = X[0..p_T] \cdot P' \cdot X[p_T + |P|..p_{\overline{T}} - |P|] \cdot \overline{P}' \cdot X(p_{\overline{T}}..|X|)$. Moreover, let $\mathcal{M} \in \mathcal{M}(X)$ be an alignment such that $\text{dyck}^w(X) = \text{dyck}_{\mathcal{M}}^w(X) \leq k$.

Claim 4.16. *There exist $d, e \in [0..78k^3]$ such that $(p_T + d, p_{\overline{T}} - d) \in \mathcal{M}$ and $(p_T + |P| - e, p_{\overline{T}} - |P| + e) \in \mathcal{M}$.*

Proof. By Fact 4.12, $X[p_T..p_T + 78k^3]$ contains a fragment of length at least $\frac{78k^3 - 6k}{k+1} \geq 36k^2$ that \mathcal{M} matches perfectly to a fragment of $X(p_{\overline{T}} - 78k^3..p_{\overline{T}}]$. Thus, let $R := X[r_T..r_T + |R|]$ be a fragment of length at least $36k^2$ contained in $X[p_T..p_T + |P|]$ that \mathcal{M} matches perfectly to $X[r_{\overline{T}} - |R|..r_{\overline{T}}] = \overline{R}$. Moreover, let $r'_T := p_T + p_{\overline{T}} - r_T$. If $r_{\overline{T}} = r'_T$, then the claim is satisfied for $d = r_T - p_T = p_{\overline{T}} - r_{\overline{T}}$. Otherwise, both $X[r_{\overline{T}} - |R|..r_{\overline{T}}]$ and $X[r'_T - |R|..r'_T]$ are occurrences of \overline{R} in X . Moreover, $0 < |r_{\overline{T}} - r'_T| \leq |(p_{\overline{T}} - r'_T) - (p_{\overline{T}} - r_{\overline{T}})| \leq |(r_T - p_T) - (p_{\overline{T}} - r_{\overline{T}})| + |(r_T - p_T) - (p_{\overline{T}} - r'_T)| \leq 2\text{dyck}_{\mathcal{M}}^w(X) + 2k \leq 4k$. Hence, $\text{per}(\overline{R}) \leq |r_{\overline{T}} - r'_T| \leq 4k$ and $\exp(\overline{R}) \geq \frac{|R|}{4k} \geq 9k$. Since $X[r'_T - |R|..r'_T]$ is contained in $X(p_{\overline{T}} - |P|..p_{\overline{T}}] = \overline{P[0..78k^3]}$, this contradicts the assumption that $\overline{P[0..78k^3]}$ and thus $P[0..78k^3]$ avoids k -periodicity.

The second part of the claim is proved analogously. \square

As $X[p_T + d..p_T + |P| - e] \in T$ and $X[p_T + d..p_T + |P| - e] = \overline{X(p_{\overline{T}} - |P| + e..p_{\overline{T}} - d)}$, the optimality of \mathcal{M} guarantees that $X[p_T + d..p_T + |P| - e] \simeq_{\mathcal{M}} \overline{X(p_{\overline{T}} - |P| + e..p_{\overline{T}} - d)}$. Hence, if $P = X[p_T..p_T + |P|] = \overline{X(p_{\overline{T}} - |P|..p_{\overline{T}}]}$ is replaced with P' , we can interpret this as $P[d..|P| - e] = X[p_T + d..p_T + |P| - e] = \overline{X(p_{\overline{T}} - |P| + e..p_{\overline{T}} - d)}$ with $P'[d..|P'| - e]$. Since $X[p_T + d..p_T + |P| - e] \simeq_{\mathcal{M}} \overline{X(p_{\overline{T}} - |P| + e..p_{\overline{T}} - d)}$, the alignment \mathcal{M} can be trivially adapted without modifying its cost, and therefore $\text{dyck}^w(X') \leq \text{dyck}_{\mathcal{M}}^w(X) = \text{dyck}^w(X)$. The converse inequality follows by symmetry between (X, P) and (X', P') . \square

Corollary 4.17. *Let $k \in \mathbb{Z}_+$. For every string $P \in T^*$, there exists a string of length at most $156k^3$ that is $\text{dyck}_{\leq k}^w$ -equivalent to P for every skewmetric weight function w .*

Proof. We proceed by induction on $|P|$ with the trivial base case of $|P| \leq 156k^3$. If $|P| \geq 156k^3$ and P avoids k -periodicity, then Lemma 4.15 implies that P is equivalent to a string $P' := P[0..78k^3] \cdot P[|P| - 78k^3..|P|]$ of length $156k^3$. Thus, suppose that P contains a fragment $P[i..j] = Q^{8k+1}$ and $|Q| \in [1..4k]$. By Lemma 4.13, Q^{8k+1} is equivalent to Q^{8k} , and thus P is equivalent to a string $P' := P[0..i] \cdot P[i + |Q|..|P|]$. By the inductive assumption, P' is equivalent to some string P'' of length at most $156k^3$, and, by transitivity of the considered equivalence, P is also equivalent to P'' . \square

4.4 Algorithm

Lemma 4.18. *There exists a linear-time algorithm that, given a string P and an integer $k \in \mathbb{Z}_+$, constructs a string P' of length at most $156k^3$ that is $\text{dyck}_{\leq k}^w$ -equivalent to P for every skewmetric weight function w . Moreover P' avoids k -periodicity.*

Algorithm 8: Construct a string of length at most $156k^3$ that is $\text{dyck}_{\leq k}^w$ -equivalent to P .

```

1 DyckReduction( $P, k$ ):
2    $P' \leftarrow \text{PeriodicityReduction}(P, 8k, \{Q \in T^+ : |Q| \leq 4k \text{ and } Q \text{ is primitive}\});$ 
3   if  $|P'| \geq 156k^3$  then return  $P'[0..78k^3] \cdot P'[|P'| - 78k^3..|P'|];$ 
4   else return  $P'$ ;

```

Proof. We apply Algorithm 1 with $e = 8k$ and \mathcal{Q} consisting of all primitive strings in T^* of length in $[1..4k]$. If the resulting string P' satisfies $|P'| < 156k^3$, we return P' . By Lemmas 4.13 and 2.13, the string P' is $\text{dyck}_{\leq k}^w$ -equivalent to P and avoids k -periodicity. Thus, if $|P'| \leq 156k^3$, then the algorithm is correct. Otherwise, we return $P'[0..78k^3] \cdot P'[|P'| - 78k^3..|P'|]$. $P'[0..78k^3]$ and $P'[|P'| - 78k^3..|P'|]$ both avoid k -periodicity, so $P'[0..78k^3] \cdot P'[|P'| - 78k^3..|P'|]$ is $\text{dyck}_{\leq k}^w$ -equivalent to P by Lemma 4.15. Due to Lemma 2.13, the running time is linear (testing whether a primitive fragment belongs to \mathcal{Q} simplifies to checking if its length does not exceed $4k$). \square

Theorem 4.19. *There exists a $O(n+k^5)$ -time algorithm that, given a preprocessed string X and an integer $k \in \mathbb{Z}_+$, constructs strings X' of lengths at most $630k^4$ such that $\text{dyck}_{\leq k}^w(X) = \text{dyck}_{\leq k}^w(X')$ holds for every skewmetric weight function w .*

Proof. Our procedure is implemented as Algorithm 9. First, if X is already of length at most $630k^4$, then we return X as it is. If $\text{dyck}(X) > k$, we return strings a^{k+1} , where $a \in \Sigma$ is an arbitrary character. If $\text{dyck}(X) \leq k$, we construct a Dyck language alignment $\mathcal{M} \in \mathcal{M}(X)$ of (unweighted) cost at most k . We then build the output string X' using \mathcal{M} as follows: scan X from left to right, if the scanned character $X[i]$ is edited by \mathcal{M} we append it to X' (here $\text{Order}(i, j) = 1$ if $j > i$ otherwise it is 0). Otherwise $X[i]$ is matched under \mathcal{M} . If $X[i] \in T$, we proceed with scanning the following characters to identify the maximal fragment $P = X[i..j] \in T^*$ such that there is a fragment $X(i'..j')$ where $\text{dyck}(X[i..j]X(i'..j')) = 0$ and \mathcal{M} matches $X[i..j]$ with $X(i'..j')$. Next we apply the reduction of Lemma 4.18 on P and append the reduced string to X' . Otherwise $X[i] \in \bar{T}$. Here also we proceed with scanning the following characters to identify the maximal fragment $Q = X[i..j] \in \bar{T}^*$ such that there is a fragment $X(i'..j')$ where $\text{dyck}(X(i'..j')X[i..j]) = 0$ and \mathcal{M} matches $X[i..j]$ with $X(i'..j')$. Next we consider the \bar{Q} (note $\bar{Q} \in T^*$), apply the reduction of Lemma 4.18 on \bar{Q} and append the reverse complement of the reduced string to X' .

Let us now prove that the resulting string X' satisfies $\text{dyck}_{\leq k}^w(X) = \text{dyck}_{\leq k}^w(X')$. This is trivial when the algorithm returns X in Line 2. If $\text{dyck}(X) > k$, then $\text{dyck}_{\leq k}(X) = \text{dyck}_{\leq k}(a^{k+1}) = \infty$ and thus also $\text{dyck}_{\leq k}^w(X) = \text{dyck}_{\leq k}^w(a^{k+1}) = \infty$ because the weighted Dyck edit distance with a normalized weight function is at least as large as the unweighted Dyck edit distance. In the remaining case we assume $\text{dyck}(X) \leq k$. Let $S = \{P_1, \dots, P_\ell\}$ be the set of fragments from T^* that are ever generated and processed using $\text{DyckReduction}()$ routine at Line 13. Similarly let $\bar{S} = \{Q_1, \dots, Q_{\ell'}\}$ be the set of fragments from \bar{T}^* that are ever generated and whose reverse complements are processed using $\text{DyckReduction}()$ routine at Line 20. By construction it is trivial to follow that (i) the fragments are disjoint; (ii) for all $i \in [0..n]$, if $X[i] \in T$ and is not edited by \mathcal{M} , then there exist some P_j such that $X[i] \in P_j$ and if $X[i] \in \bar{T}$ and is not edited by \mathcal{M} , then there exist some Q_j such that $X[i] \in Q_j$; (iii) The fragments are maximal in a sense that if $P_i = X[a..b] \in S$, then either $\mathcal{M}(b) \neq \mathcal{M}(b-1) - 1$ or $X[b] \neq \overline{X[\mathcal{M}(b)]}$ and same holds for the fragments in \bar{S} . Next we prove for each $P_i = X[a..b] \in S$, $\exists Q_j = X[c..d] \in \bar{S}$, such that $|P_i| = |Q_j|$ and for all $k \in [0..|P_i|]$, $(a+k, d-k) \in \mathcal{M}$ and $X[a+k] = \overline{X[d-k]}$. For this we first claim that $\mathcal{M}(b-1)$

Algorithm 9: Construct strings X' of length at most $630k^4$ such that $\text{dyck}_{\leq k}^w(X) = \text{dyck}_{\leq k}^w(X')$

```

1 DyckKernel( $X, k$ ):
2   if  $|X| \leq 630k^4$  then return ( $X$ );
3   if  $\text{dyck}(X) > k$  then return ( $a^{k+1}$ ) for some  $a \in \Sigma$ ;
4   Let  $\mathcal{M} \in \mathcal{M}(X)$  be a dyck language alignment satisfying  $\text{dyck}_{\mathcal{M}}(X) \leq k$ ;
5    $X', P, Q \leftarrow \varepsilon$ ;
6   for  $i \leftarrow 0$  to  $n - 1$  do
7     if  $\mathcal{M}(i) = \perp$  or  $\text{dyck}(X[i]X[\mathcal{M}(i)]) \cdot \text{Order}(i, \mathcal{M}(i)) = 1$  or
       $\text{dyck}(X[\mathcal{M}(i)]X[i]) \cdot \text{Order}(\mathcal{M}(i), i) = 1$  then
8        $X' \leftarrow X' \cdot X[i]$ 
9     else if  $X[i] \in T$  and  $\mathcal{M}(i+1) = \mathcal{M}(i) - 1$  and  $\text{dyck}(X[i+1]X[\mathcal{M}(i) - 1]) = 0$ 
      then
10       $P \leftarrow P \cdot X[i]$ 
11    else if  $X[i] \in \bar{T}$  then
12       $P \leftarrow P \cdot X[i]$ ;
13       $P \leftarrow \text{DyckReduction}(P, k)$ ;
14       $X' \leftarrow X' \cdot P$ ;
15       $P \leftarrow \varepsilon$ ;
16    else if  $X[i] \in \bar{T}$  and  $\mathcal{M}(i+1) = \mathcal{M}(i) - 1$  and  $\text{dyck}(X[\mathcal{M}(i) - 1]X[i+1]) = 0$ 
      then
17       $Q \leftarrow Q \cdot X[i]$ 
18    else
19       $Q \leftarrow Q \cdot X[i]$ ;
20       $Q \leftarrow \text{DyckReduction}(\overline{Q}, k)$ ;
21       $X' \leftarrow X' \cdot Q$ ;
22       $Q \leftarrow \varepsilon$ ;
23   return ( $X'$ )

```

is a starting index of some $Q_j \in \overline{S}$. As otherwise $\mathcal{M}(b) = \mathcal{M}(b-1) - 1$ and $X[b] \neq \overline{X[\mathcal{M}(b)]}$; this contradicts the maximality of P_i . Further by construction for all $k \in [a..b)$, $X[\mathcal{M}(k)] \in Q_j$. Finally we argue $\mathcal{M}(a)$ is an ending index of Q_j . As otherwise $\mathcal{M}(a-1) = \mathcal{M}(a) + 1$ and this contradicts the fact that a is the starting index of some segment from S . Similarly we can show for each $Q_j \in \overline{S}$ there is a corresponding match $P_i \in S$ and this provides an one to one correspondence between a pair of fragments from S and \overline{S} . Thus for a fragment $P_i \in S$ let $\mathcal{M}(P_i)$ represents the corresponding matched fragments from \overline{S} and we can represent $S \cup \overline{S} = \cup_{i \in [q]} (P_i, \mathcal{M}(P_i))$. Following Fact 4.11, $P_i, \mathcal{M}(P_i)$ are k -synchronized. Next in the algorithm for each pair $(P_i, \mathcal{M}(P_i))$ we add strings $\text{DyckReduction}(P_i)$ representing P_i and $\overline{\text{DyckReduction}(P_i)}$ (note $\overline{\mathcal{M}(P_i)} = P_i$) representing $\mathcal{M}(P_i)$ to X' . Following the fact that every character that is not contained in a fragment from $S \cup \overline{S}$ is edited by \mathcal{M} and thus copied to X' directly, by applying Lemma 4.18 repeatedly for every pair $(P_i, \mathcal{M}(P_i))$, we claim $\text{dyck}_{\leq k}^w(X) = \text{dyck}_{\leq k}^w(X')$.

Next, we show that the returned string is of length at most $630k^4$. This is clear when the algorithm terminates at Line 2 or 3. Otherwise, we create a string X' , to which we directly copy the characters that are edited by \mathcal{M} . However there are at most $2k$ characters that \mathcal{M} deletes or substitutes. Next we identify maximal fragments $P = X[i..j] \in T^*$ such that there is another fragment $X'[i'..j'] \in \overline{T^*}$ that is matched with P by \mathcal{M} . The maximality of P and the preprocessing of X ensure that at least one of $X[j]$ and $X'[i']$ is edited by \mathcal{M} . We call these characters the boundary characters for P . Notice for any two distinct fragments $P, P' \in T^*$, the boundary characters are different and by construction P, P' are disjoint. As there are at most $2k$ characters that \mathcal{M} edits, we conclude there can be at most $2k$ fragments over T^* , that our algorithm can construct. For each such fragment following the reduction of Lemma 4.18, we add a substring of length $156k^3$ to X' . Thus the total length of all the substrings is $312k^4$. Similarly we can argue for the fragments $Q \in \overline{T^*}$. Thus we can bound the total length of X' by $2 \cdot 312k^4 + 2k < 630k^4$.

It remains to analyze the complexity of our procedure. We use the algorithm [FGK+22a] to check whether $\text{dyck}(X) \leq k$ and, if so, construct the alignment \mathcal{M} . This costs $\mathcal{O}(n+k^5)$ time. Next we perform a single left to right scan of X . Throughout, all the conditions in the *if/else* statements can be checked in $\mathcal{O}(1)$ time. Moreover any character is passed to the $\text{DyckReduction}()$ routine at most twice. Thus following Lemma 4.18, given X and \mathcal{M} , X' can be constructed in linear time. \square

Proof of Theorem 1.3. We first preprocess X in linear time following the steps described in Section 4.2.1 to build strings X' such that $\text{dyck}_{\leq k}^w(X') = \text{dyck}_{\leq k}^w(X)$. Next we apply Theorem 4.19 on X' , to build strings X'' of length $\mathcal{O}(k^4)$ such that $\text{dyck}_{\leq k}^w(X'') = \text{dyck}_{\leq k}^w(X')$. This takes time $\mathcal{O}(n+k^5)$. Lastly if $X = a^{k+1}$ (this can be checked in time $\mathcal{O}(k)$) output the distance is $> k$. Otherwise we compute $\text{dyck}_{\leq k}^w(X'')$ using the dynamic program algorithm from [Mye95] in time $\mathcal{O}(k^{12})$. Thus the total running time is $\mathcal{O}(n+k^{12})$. \square

A Deferred Proofs from Section 2

In the following, we give the missing proofs of facts from Section 2.

Fact 2.5. *If w is a quasimetric on $\overline{\Sigma}$, then ed^w is a quasimetric on Σ^* . In this case, $\text{ed}^w(X, Y)$ can be equivalently defined as the minimum cost of a sequence of edits transforming X into Y .*

Proof. Consider arbitrary strings $X, Y, Z \in \Sigma^*$ as well as alignments $\mathcal{A} = (x_t, y_t)_{t=0}^m \in \mathbf{A}(X, Y)$ and $\mathcal{B} = (\hat{y}_t, \hat{z}_t)_{t=0}^{\hat{m}} \in \mathbf{A}(Y, Z)$. We construct a *product alignment* $\mathcal{A} \otimes \mathcal{B} \in \mathbf{A}(X, Z)$ such that $\text{ed}_{\mathcal{A} \otimes \mathcal{B}}^w(X, Z) \leq \text{ed}_{\mathcal{A}}^w(X, Y) + \text{ed}_{\mathcal{B}}^w(Y, Z)$. Let us denote $\mathcal{A}' = (x_t, y_t)_{t=0}^{m-1}$ and $\mathcal{B}' = (\hat{y}_t, \hat{z}_t)_{t=0}^{\hat{m}-1}$, as well as $X' = X[0..|X|-1]$ if $X \neq \varepsilon$, $Y' = Y[0..|Y|-1]$ if $Y \neq \varepsilon$, and $Z' = Z[0..|Z|-1]$ if $Z \neq \varepsilon$.

We proceed by induction on $m + \hat{m}$ and consider several cases based on how \mathcal{A} and \mathcal{B} handle the trailing characters of X , Y , and Z .

1. $m = \hat{m} = 0$. In this case, $X = Y = Z = \varepsilon$, and we define $\mathcal{A} \otimes \mathcal{B} := (0, 0)$. Trivially, $\text{ed}_{\mathcal{A} \otimes \mathcal{B}}^w(X, Z) = 0 = \text{ed}_{\mathcal{A}}^w(X, Y) + \text{ed}_{\mathcal{B}}^w(Y, Z)$.
2. $(x_{m-1}, y_{m-1}) = (|X| - 1, |Y|)$, that is, \mathcal{A} deletes $X[|X| - 1]$. In this case, $\mathcal{A}' \in \mathbf{A}(X', Y)$, and we define $\mathcal{A} \otimes \mathcal{B} := (\mathcal{A}' \otimes \mathcal{B}) \odot (|X|, |Z|)$, where \odot denotes concatenation, so that $\mathcal{A} \otimes \mathcal{B}$ deletes $X[|X| - 1]$. By the induction hypothesis, $\text{ed}_{\mathcal{A} \otimes \mathcal{B}}^w(X, Z) = \text{ed}_{\mathcal{A}' \otimes \mathcal{B}}^w(X', Z) + w(X[|X| - 1], \varepsilon) \leq \text{ed}_{\mathcal{A}'}^w(X', Y) + \text{ed}_{\mathcal{B}}^w(Y, Z) + w(X[|X| - 1], \varepsilon) = \text{ed}_{\mathcal{A}}^w(X, Y) + \text{ed}_{\mathcal{B}}^w(Y, Z)$.
3. $(\hat{y}_{\hat{m}-1}, \hat{z}_{\hat{m}-1}) = (|Y| - 1, |Z|)$, that is, \mathcal{B} inserts $Z[|Z| - 1]$. In this case, $\mathcal{B}' \in \mathbf{A}(Y, Z')$, and we define $\mathcal{A} \otimes \mathcal{B} := (\mathcal{A} \otimes \mathcal{B}') \odot (|X|, |Z|)$ so that $\mathcal{A} \otimes \mathcal{B}$ inserts $Z[|Z| - 1]$. By the induction hypothesis, $\text{ed}_{\mathcal{A} \otimes \mathcal{B}}^w(X, Z) = \text{ed}_{\mathcal{A} \otimes \mathcal{B}'}^w(X, Z') + w(\varepsilon, Z[|Z| - 1]) \leq \text{ed}_{\mathcal{A}}^w(X, Y) + \text{ed}_{\mathcal{B}'}^w(Y, Z') + w(\varepsilon, Z[|Z| - 1]) = \text{ed}_{\mathcal{A}}^w(X, Y) + \text{ed}_{\mathcal{B}}^w(Y, Z)$.
4. $(x_{m-1}, y_{m-1}) = (|X|, |Y| - 1)$ and $(\hat{y}_{\hat{m}-1}, \hat{z}_{\hat{m}-1}) = (|Y| - 1, |Z|)$, that is, \mathcal{A} inserts $Y[|Y| - 1]$ and \mathcal{B} deletes $Y[|Y| - 1]$. In this case, $\mathcal{A}' \in \mathbf{A}(X, Y')$ and $\mathcal{B}' \in \mathbf{A}(Y', Z)$, and we define $\mathcal{A} \otimes \mathcal{B} := \mathcal{A}' \otimes \mathcal{B}'$. By the induction hypothesis, $\text{ed}_{\mathcal{A} \otimes \mathcal{B}}^w(X, Z) = \text{ed}_{\mathcal{A}' \otimes \mathcal{B}'}^w(X, Z) \leq \text{ed}_{\mathcal{A}'}^w(X, Y') + \text{ed}_{\mathcal{B}'}^w(Y', Z) \leq \text{ed}_{\mathcal{A}}^w(X, Y) + \text{ed}_{\mathcal{B}}^w(Y, Z)$.
5. $(x_{m-1}, y_{m-1}) = (|X|, |Y| - 1)$ and $(\hat{y}_{\hat{m}-1}, \hat{z}_{\hat{m}-1}) = (|Y| - 1, |Z| - 1)$, that is, \mathcal{A} inserts $Y[|Y| - 1]$ and \mathcal{B} aligns $Y[|Y| - 1]$ with $Z[|Z| - 1]$. In this case, $\mathcal{A}' \in \mathbf{A}(X, Y')$ and $\mathcal{B}' \in \mathbf{A}(Y', Z')$, and we define $\mathcal{A} \otimes \mathcal{B} := (\mathcal{A}' \otimes \mathcal{B}') \odot (|X|, |Z|)$ so that $\mathcal{A} \otimes \mathcal{B}$ inserts $Z[|Z| - 1]$. By the induction hypothesis, $\text{ed}_{\mathcal{A} \otimes \mathcal{B}}^w(X, Z) = \text{ed}_{\mathcal{A}' \otimes \mathcal{B}'}^w(X, Z') + w(\varepsilon, Z[|Z| - 1]) \leq \text{ed}_{\mathcal{A}'}^w(X, Y') + \text{ed}_{\mathcal{B}'}^w(Y', Z') + w(\varepsilon, Y[|Y| - 1]) + w(Y[|Y| - 1], Z[|Z| - 1]) = \text{ed}_{\mathcal{A}}^w(X, Y) + \text{ed}_{\mathcal{B}}^w(Y, Z)$.
6. $(x_{m-1}, y_{m-1}) = (|X| - 1, |Y| - 1)$ and $(\hat{y}_{\hat{m}-1}, \hat{z}_{\hat{m}-1}) = (|Y| - 1, |Z|)$, that is, \mathcal{A} aligns $X[|X| - 1]$ with $Y[|Y| - 1]$ and \mathcal{B} deletes $Y[|Y| - 1]$. In this case, $\mathcal{A}' \in \mathbf{A}(X', Y')$ and $\mathcal{B}' \in \mathbf{A}(Y', Z)$, and we define $\mathcal{A} \otimes \mathcal{B} := (\mathcal{A}' \otimes \mathcal{B}') \odot (|X|, |Z|)$ so that $\mathcal{A} \otimes \mathcal{B}$ deletes $X[|X| - 1]$. By the induction hypothesis, $\text{ed}_{\mathcal{A} \otimes \mathcal{B}}^w(X, Z) = \text{ed}_{\mathcal{A}' \otimes \mathcal{B}'}^w(X', Z) + w(X[|X| - 1], \varepsilon) \leq \text{ed}_{\mathcal{A}'}^w(X', Y') + \text{ed}_{\mathcal{B}'}^w(Y', Z) + w(X[|X| - 1], Y[|Y| - 1]) + w(Y[|Y| - 1], \varepsilon) = \text{ed}_{\mathcal{A}}^w(X, Y) + \text{ed}_{\mathcal{B}}^w(Y, Z)$.
7. $(x_{m-1}, y_{m-1}) = (|X| - 1, |Y| - 1)$ and $(\hat{y}_{\hat{m}-1}, \hat{z}_{\hat{m}-1}) = (|Y| - 1, |Z| - 1)$, that is, \mathcal{A} aligns $X[|X| - 1]$ with $Y[|Y| - 1]$ and \mathcal{B} aligns $Y[|Y| - 1]$ with $Z[|Z| - 1]$. In this case, $\mathcal{A}' \in \mathbf{A}(X', Y')$ and $\mathcal{B}' \in \mathbf{A}(Y', Z')$, and we define $\mathcal{A} \otimes \mathcal{B} := (\mathcal{A}' \otimes \mathcal{B}') \odot (|X|, |Z|)$ so that $\mathcal{A} \otimes \mathcal{B}$ aligns $X[|X| - 1]$ with $Z[|Z| - 1]$. By the induction hypothesis, $\text{ed}_{\mathcal{A} \otimes \mathcal{B}}^w(X, Z) = \text{ed}_{\mathcal{A}' \otimes \mathcal{B}'}^w(X', Z') + w(X[|X| - 1], Z[|Z| - 1]) \leq \text{ed}_{\mathcal{A}'}^w(X', Y') + \text{ed}_{\mathcal{B}'}^w(Y', Z) + w(X[|X| - 1], Y[|Y| - 1]) + w(Y[|Y| - 1], Z[|Z| - 1]) = \text{ed}_{\mathcal{A}}^w(X, Y) + \text{ed}_{\mathcal{B}}^w(Y, Z)$.

It is easy to check that the above cases cover all the possibilities. In particular, Case 2 covers the case of $\hat{m} = 0 < m$ whereas Case 3 covers the case of $m = 0 < \hat{m}$. We also remark that Cases 2 and 3 are sometimes both applicable; by convention, we then follow Case 2. Finally, we note that Cases 5–7 rely on the assumption that w satisfies the triangle inequality. This completes the proof of the first part of the fact.

To show that $\text{ed}^w(X, Y)$ can be equivalently defined as the minimum cost of a sequence of edits transforming X into Y , we first consider each of the operations in a minimum alignment \mathcal{A} of X and Y individually to build a sequence of edits S from \mathcal{A} . We iterate through all pairs (x_t, y_t) of \mathcal{A} from right to left starting with $t = m - 1$, stopping after $t = 0$ has been handled, and building S according to the definition of alignments:

1. If $(x_t, y_t) = (x_{t+1} - 1, y_{t+1} - 1)$ and $X[x_t] = Y[y_t]$, we add nothing to S .
2. If $(x_t, y_t) = (x_{t+1} - 1, y_{t+1} - 1)$ and $X[x_t] \neq Y[y_t]$, we add a substitution of $X[x_t]$ with $Y[y_t]$ to S .
3. If $(x_t, y_t) = (x_{t+1} - 1, y_{t+1})$, we add a deletion of $X[x_t]$ to S .

4. If $(x_t, y_t) = (x_{t+1}, y_{t+1} - 1)$, we add an insertion of $Y[y_t]$ at position x_t in X to S .

In all cases, we decrement t by 1. Clearly the resulting sequence of edits has the same cost as \mathcal{A} , and by the definition of alignments, S transforms X into Y . We now consider a minimum sequence of edits S that transforms X to Y and build an alignment $\mathcal{A} \in \mathbf{A}(X, Y)$ from S such that $\text{ed}_{\mathcal{A}}^w(X, Y) \leq \text{cost}(S)$ (we let $\text{cost}(S)$ denote the total cost of edits by S). We use notation \mathcal{A}', X', Y' as before, and proceed by induction to construct \mathcal{A} :

1. If $X[|X| - 1]$ is deleted by S and a character is inserted at the end of X , then $\mathcal{A}' \in \mathbf{A}(X', Y')$ and we set $\mathcal{A} = \mathcal{A}' \odot (|X|, |Y|)$. We note that the inserted character c may be substituted to $Y[|Y| - 1]$. We let S' be the sequence S without the insertion, deletion, and if possible substitution on the last character of X . By the induction hypothesis and triangle inequality, $\text{ed}_{\mathcal{A}}(X, Y) = \text{ed}_{\mathcal{A}'}(X', Y') + w(X[|X| - 1], Y[|Y| - 1]) \leq \text{ed}_{\mathcal{A}'}(X', Y') + w(X[|X| - 1], \varepsilon) + w(\varepsilon, Y[|Y| - 1]) \leq \text{ed}_{\mathcal{A}'}(X', Y') + w(X[|X| - 1], \varepsilon) + w(\varepsilon, c) + w(c, Y[|Y| - 1]) \leq \text{cost}(S') + w(X[|X| - 1], \varepsilon) + w(\varepsilon, c) + w(c, Y[|Y| - 1]) = \text{cost}(S)$.
2. If $X[|X| - 1]$ is deleted by S and no character is inserted at the end of X , then $\mathcal{A}' \in \mathbf{A}(X', Y)$ and we set $\mathcal{A} = \mathcal{A}' \odot (|X|, |Y|)$. We let S' be the sequence S without the deletion of $X[|X| - 1]$. By the induction hypothesis, $\text{ed}_{\mathcal{A}}(X, Y) = \text{ed}_{\mathcal{A}'}(X', Y) + w(X[|X| - 1], \varepsilon) \leq \text{cost}(S') + w(X[|X| - 1], \varepsilon) = \text{cost}(S)$.
3. If a character is inserted at the end of X , then $\mathcal{A}' \in \mathbf{A}(X, Y')$ and we set $\mathcal{A} = \mathcal{A}' \odot (|X|, |Y|)$. We let S' be the sequence S without this insertion. By the induction hypothesis, $\text{ed}_{\mathcal{A}}(X, Y) = \text{ed}_{\mathcal{A}'}(X, Y') + w(\varepsilon, Y[|Y| - 1], \varepsilon) \leq \text{cost}(S') + w(\varepsilon, Y[|Y| - 1]) = \text{cost}(S)$.
4. If $X[|X| - 1]$ is substituted by S , then $\mathcal{A}' \in \mathbf{A}(X', Y')$ and we set $\mathcal{A} = \mathcal{A}' \odot (|X|, |Y|)$. We let S' be the sequence of S without any substitutions of $X[|X| - 1]$ and let C be an ordered list of characters substituted by S at $X[|X| - 1]$. Then, by the induction hypothesis, $\text{ed}_{\mathcal{A}}(X, Y) = \text{ed}_{\mathcal{A}'}(X', Y') + w(X[|X| - 1], Y[|Y| - 1]) \leq \text{ed}_{\mathcal{A}'}(X', Y') + \sum_{i=0}^{|C|-1} w(c_i, c_{i+1}) \leq \text{cost}(S') + \sum_{i=0}^{|C|-1} w(c_i, c_{i+1}) = \text{cost}(S)$.

By induction, we can see that there exists an alignment \mathcal{A} with cost at most that of S . □

Fact 2.6. *Consider a string X and its fragment $X[i..j]$. Then, for every quasimetric w , we have $\text{ed}^w(X, X[i..j]) = \text{ed}^w(X[0..i] \cdot X[j..|X|), \varepsilon)$.*

Proof. The unique alignment in $\mathbf{A}(X[0..i] \cdot X[j..|X|), \varepsilon)$ deletes all characters, and therefore $\text{ed}^w(X[0..i] \cdot X[j..|X|), \varepsilon) = \sum_{u \in [0..i] \cup [j..|X|)} w(X[u], \varepsilon)$.

Next, consider an alignment $\mathcal{A} \in \mathbf{A}(X, X[i..j])$ that deletes $X[0..i]$, matches $X[i..j]$ perfectly, and deletes $X[j..|X|)$. Hence, $\text{ed}^w(X, X[i..j]) \leq \text{ed}_{\mathcal{A}}^w(X, X[i..j]) = \sum_{u \in [0..i] \cup [j..|X|)} w(X[u], \varepsilon) = \text{ed}^w(X[0..i] \cdot X[j..|X|), \varepsilon)$.

Now, let $Y = X[i..j]$ and consider an arbitrary alignment $\mathcal{B} \in \mathbf{A}(X, Y)$. For each $u \in [0..i] \cup [j..|X|)$, we recursively define a sequence $(c_{u,t})_{t=0}^{m_u}$ so that $c_{u,0} = u$, the alignment \mathcal{B} aligns $X[c_{u,t}]$ to $Y[c_{u,t+1} - i] = X[c_{u,t+1}]$ for $t \in [0..m_u)$, and \mathcal{B} deletes $X[c_{u,m_u}]$. By construction, the sequences $(c_{u,t})_{t=0}^{m_u}$ are finite and each position in X belongs to at most one such sequence. Moreover, since w is quasimetric, $w(X[c_{u,t}], \varepsilon) \leq w(X[c_{u,t}], X[c_{u,t+1}]) + w(X[c_{u,t+1}], \varepsilon)$ holds for every $t \in [0..m_u)$, and thus $w(X[c_{u,0}], \varepsilon) \leq w(X[c_{u,m_u}], \varepsilon) + \sum_{t \in [0..m_u)} w(X[c_{u,t}], X[c_{u,t+1}])$. Since

\mathcal{B} deletes $X[c_{u,m_u}]$ and $X[c_{u,t}] \sim_{\mathcal{B}} Y[c_{u,t+1} - i] = X[c_{u,t+1}]$ holds for $t \in [0..m_u)$, this yields

$$\begin{aligned} \text{ed}_{\mathcal{B}}^w(X, Y) &\geq \sum_{u \in [0..i) \cup [j..|X|)} \left(w(X[c_{u,m_u}], \varepsilon) + \sum_{t \in [0..m_u)} w(X[c_{u,t}], X[c_{u,t+1}]) \right) \\ &\geq \sum_{u \in [0..i) \cup [j..|X|)} w(X[u], \varepsilon) \\ &= \text{ed}^w(X[0..i) \cdot X[j..|X|), \varepsilon). \end{aligned}$$

Since \mathcal{B} was chosen arbitrarily, we conclude that $\text{ed}^w(X, Y) \geq \text{ed}^w(X[0..i) \cdot X[j..|X|), \varepsilon)$. \square

References

- [Abb14] Amir Abboud. Hardness for easy problems, 2014. Presented at Satellite Workshop of ICALP (YR-ICALP). URL: <https://www.dropbox.com/s/jt9uz1jjmormkb7/EasyHardness.pdf>.
- [ABW18] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant’s parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.
- [AJ21] Shyan Akmal and Ce Jin. Faster algorithms for bounded tree edit distance. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPICs*, pages 12:1–12:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.12.
- [AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, FOCS ’10*, page 377–386, USA, 2010. IEEE Computer Society. doi:10.1109/FOCS.2010.43.
- [AN94] Arne Andersson and Stefan Nilsson. A new efficient radix sort. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 714–721. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365721.
- [AN20] Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it’s a constant factor. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 990–1001. IEEE, 2020. doi:10.1109/FOCS46700.2020.00096.
- [AO09] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, STOC ’09*, page 199–204, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1536414.1536444.
- [AP72] Alfred V Aho and Thomas G Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM Journal on Computing*, 1(4):305–312, 1972.

- [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 *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, SODA '06*, page 792–801, USA, 2006. Society for Industrial and Applied Mathematics. doi:10.1145/1109557.1109644.
- [BGHS19] Mahdi Boroujeni, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. $1+\epsilon$ approximation of tree edit distance in quadratic time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 709–720. ACM, 2019. doi:10.1145/3313276.3316388.
- [BGK03] Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed xml. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29, VLDB '03*, page 141–152. VLDB Endowment, 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), jul 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 Journal on Computing*, 48(2):481–512, 2019. doi:10.1137/17M112720X.
- [BI18] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.
- [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, jun 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 Rubinfeld. 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, mar 1998. doi:10.1016/S0167-8655(97)00179-7.

- [BW08] Mikolaj Bojanczyk and Igor Walukiewicz. Forest algebras. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 107–132. Amsterdam University Press, 2008.
- [BYJKK04] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, FOCS '04*, page 550–559, USA, 2004. IEEE Computer Society. 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), oct 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. Society for Industrial and Applied Mathematics, jan 2022. doi:10.1137/1.9781611977073.60.
- [Cha99] Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In Malcolm P. Atkinson, Maria E. Orłowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 90–101. Morgan Kaufmann, 1999. URL: <http://www.vldb.org/conf/1999/P8.pdf>.
- [CIG22] Kateryna Chumachenko, Alexandros Iosifidis, and Moncef Gabbouj. Weighted edit distance for country code recognition in license plates. In *30th European Signal Processing Conference, EUSIPCO 2022*, pages 1111–1115. IEEE, 2022. URL: <https://ieeexplore.ieee.org/document/9909869>.
- [DGH⁺22] Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, Barna Saha, and Hamed Saleh. $\tilde{O}(n + \text{poly}(k))$ -time algorithm for bounded tree edit distance. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*. IEEE, 2022. arXiv:2209.07524.
- [DKS22] Debarati Das, Tomasz Kociumaka, and Barna Saha. Improved approximation algorithms for dyck edit distance and RNA folding. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *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), dec 2010. doi:10.1145/1644015.1644017.
- [Dür22] Anita Dürr. Improved bounds for rectangular monotone min-plus product, 2022. doi:10.48550/arXiv.2208.02862.
- [FFF⁺16] Lionel Fontan, Isabelle Ferrané, Jérôme Farinas, Julien Piquier, and Xavier Aumont. Using phonologically weighted levenshtein distances for the prediction

- of microscopic intelligibility. In *17th Annual Conference of the International Speech Communication Association, Interspeech 2016*, pages 650–654. ISCA, 2016. doi:10.21437/Interspeech.2016-431.
- [FFM00] Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000. doi:10.1145/355541.355547.
- [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 Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 3650–3669. 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), nov 2009. doi:10.1145/1613676.1613680.
- [FW65] Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proc. Am. Math. Soc.*, 16(1):109–114, 1965. doi:10.2307/2034009.
- [GKS19] Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1101–1120. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00070.
- [GRS20] Elazar Goldenberg, Aviad Rubinfeld, 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. ACM, 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.
- [GWK21] Andrew Gerlach, Adam Wiemerslage, and Katharina Kann. Paradigm clustering with weighted edit distance. In *18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 107–114. Association for Computational Linguistics, 2021. doi:10.18653/v1/2021.sigmorphon-1.12.
- [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 Rubinfeld, and Amirbehshad Shahrashbi. Near-linear time insertion-deletion codes and $(1 + \epsilon)$ -approximating edit distance via indexing. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, page 697–708, New York, NY, USA, 2019. Association for Computing Machinery. 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, may 1984. doi:10.1137/0213024.
- [Ind01] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, FOCS '01*, page 10, USA, 2001. IEEE Computer Society. doi:10.1109/sfcs.2001.959878.
- [JM09] Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009. URL: <https://www.worldcat.org/oclc/315913020>.
- [Kle98] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms, ESA '98*, page 91–102, Berlin, Heidelberg, 1998. Springer-Verlag. doi:10.1007/3-540-68530-8_8.
- [Koz97] Dexter C. Kozen. *Automata and Computability*. Springer New York, 1997. doi:10.1007/978-1-4612-1844-9.
- [KRRW15] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In Piotr Indyk, editor, *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.
- [KS20a] Tomasz Kociumaka and Barna Saha. Sublinear-time algorithms for computing & embedding gap edit distance. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1168–1179. IEEE, 2020. doi:10.1109/FOCS46700.2020.00112.
- [KS20b] 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.
- [Kur96] Stefan Kurtz. Approximate string searching under weighted edit distance. In *3rd South American Workshop on String Processing, WSP 1996*, pages 156–170. Carleton University Press, 1996.
- [Kus19] William Kuszmaul. Dynamic time warping in strongly subquadratic time: Algorithms for the low-distance regime and approximate evaluation. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPICs*, pages 80:1–80:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.80.
- [KXI20] Satoshi Koide, Chuan Xiao, and Yoshiharu Ishikawa. Fast subtrajectory similarity search in road networks under weighted edit distance constraints. *Proceedings of the VLDB Endowment*, 13(12):2188–2201, 2020. doi:10.14778/3407790.3407818.
- [Lev65] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710, 1965.

- [LMS98] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, apr 1998. doi:10.1137/s0097539794264810.
- [LV88] Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.
- [Mao21] Xiao Mao. Breaking the cubic barrier for (unweighted) tree edit distance. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 792–803. IEEE, 2021. doi:10.1109/FOCS52979.2021.00082.
- [Mye86] Eugene W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446.
- [Mye95] Gene Myers. Approximately matching context-free languages. *Information Processing Letters*, 54(2):85–92, 1995. doi:10.1016/0020-0190(95)00007-y.
- [NW70] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, mar 1970. doi:10.1016/0022-2836(70)90057-4.
- [PM02] Guillermo Peris and Andrés Marzal. Fast cyclic edit distance computation with weighted edit costs in classification. In *16th International Conference on Pattern Recognition, ICPR 2002*, pages 184–187. IEEE, IEEE Computer Society, 2002. doi:10.1109/ICPR.2002.1047428.
- [PR05] Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM Journal on Computing*, 34(6):1398–1431, 2005. doi:10.1137/s0097539702419650.
- [PT87] Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987. doi:10.1137/0216062.
- [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.
- [Sah17] Barna Saha. Fast & space-efficient approximations of language edit distance and rna folding: An amnesic dynamic programming approach. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 295–306. IEEE, 2017.
- [Sel77] Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977. doi:10.1016/0020-0190(77)90064-3.
- [Ski20] Steven Skiena. *The Algorithm Design Manual, Third Edition*. Texts in Computer Science. Springer, 2020. doi:10.1007/978-3-030-54256-6.
- [SS22] Masoud Seddighin and Saeed Seddighin. $3+\epsilon$ approximation of tree edit distance in truly subquadratic time. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022*,

Berkeley, CA, USA, volume 215 of *LIPICs*, pages 115:1–115:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:[10.4230/LIPICs.ITCS.2022.115](https://doi.org/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](https://doi.org/10.1093/bioinformatics/6.4.309).
- [Tai79] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, jul 1979. doi:[10.1145/322139.322143](https://doi.org/10.1145/322139.322143).
- [Tou05] Hélène Touzet. A linear tree edit distance algorithm for similar ordered trees. In *Proceedings of the 16th Annual Conference on Combinatorial Pattern Matching, CPM'05*, page 334–345, Berlin, Heidelberg, 2005. Springer-Verlag. doi:[10.1007/11496656_29](https://doi.org/10.1007/11496656_29).
- [WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974. doi:[10.1145/321796.321811](https://doi.org/10.1145/321796.321811).
- [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](https://doi.org/10.1137/0218082).