Almost-Optimal Sublinear-Time Edit Distance in the Low Distance Regime*

Karl Bringmann
Saarland University
Max Planck Institute for Informatics
Saarbrücken, Germany

Alejandro Cassis
Saarland University
Max Planck Institute for Informatics
Saarbrücken, Germany

Nick Fischer
Saarland University
Max Planck Institute for Informatics
Saarbrücken, Germany

Vasileios Nakos
RelationalAI
Berkeley, United States of America

ABSTRACT

We revisit the task of computing the edit distance in sublinear time. In the (k, K)-gap edit distance problem we are given oracle access to two strings of length n and the task is to distinguish whether their edit distance is at most k or at least K. It has been established by Goldenberg, Krauthgamer and Saha (FOCS '19), with improvements by Kociumaka and Saha (FOCS '20), that the (k, k^2)-gap problem can be solved in time $\widetilde{O}(n/k + \mathrm{poly}(k))$. One of the most natural questions in this line of research is whether the (k, k^2)-gap is best-possible for the running time $\widetilde{O}(n/k + \mathrm{poly}(k))$.

In this work we answer this question by significantly improving the gap. Specifically, we show that in time $O(n/k + \mathrm{poly}(k))$ we can even solve the $(k, k^{1+o(1)})$-gap problem. This is the first algorithm that breaks the (k, k^2)-gap in this running time. Our algorithm is almost optimal in the following sense: In the low distance regime ($k \le n^{0.19}$) our running time becomes $O(n/k)$, which matches a known $n/k^{1+o(1)}$ lower bound for the $(k, k^{1+o(1)})$-gap problem up to lower order factors.

Our result also reveals a surprising similarity of Hamming distance and edit distance in the low distance regime: For both, the $(k, k^{1+o(1)})$-gap problem has time complexity $n/k^{1\pm o(1)}$ for small k.

In contrast to previous work, which employed a subsampled variant of the Landau-Vishkin algorithm, we instead build upon the algorithm of Andoni, Krauthgamer and Onak (FOCS '10) which approximates the edit distance in almost-linear time $O(n^{1+\varepsilon})$ within a polylogarithmic factor. We first simplify their approach and then show how to to effectively prune their computation tree in order to obtain a sublinear-time algorithm in the given time bound. Towards that, we use a variety of structural insights on the (local and global) patterns that can emerge during this process and design appropriate property testers to effectively detect these patterns.

*This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

STOC '22, June 20–24, 2022, Rome, Italy
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9264-8/22/06.
https://doi.org/10.1145/3519935.3519990

CCS CONCEPTS

• **Theory of computation → Streaming, sublinear and near linear time algorithms**.

KEYWORDS

Edit Distance, Sublinear Algorithms, Precision Sampling

ACM Reference Format:
Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. 2022. Almost-Optimal Sublinear-Time Edit Distance in the Low Distance Regime. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3519935.3519990

1 INTRODUCTION

The *edit distance* (also called *Levenshtein distance*) [24] between two strings X and Y is the minimum number of character insertions, deletions and substitutions required to transform X into Y. It constitutes a fundamental string similarity measure with applications across several disciplines, including computational biology, text processing and information retrieval.

Computational problems involving the edit distance have been studied extensively. A textbook dynamic programming algorithm computes the edit distance of two strings of length n in time $O(n^2)$ [26, 27]. It is known that beating this quadratic time by a polynomial improvement would violate the Strong Exponential Time Hypothesis [1, 2, 9, 14], one of the cornerstones of fine-grained complexity theory. For faster algorithms, we therefore have to resort to *approximating* the edit distance.[1] A long line of research (starting even before the hardness result emerged) lead to successively improved approximation algorithms: The first result established that in linear time the edit distance can be $O(\sqrt{n})$-approximated [22]. The approximation ratio was improved to $O(n^{3/7})$ in [10] and to $n^{1/3+o(1)}$ in [12]. Making use of the Ostrovsky-Rabani technique for embedding edit distance into the ℓ_1-metric [25], Andoni and Onak [8] gave an $n^{o(1)}$-approximation algorithm which runs in time $n^{1+o(1)}$. Later, Andoni, Krauthgamer and Onak achieved an algorithm in time $O(n^{1+\varepsilon})$ computing a $(\log n)^{O(1/\varepsilon)}$-approximation [4]. A breakthrough result by Chakraborty, Das, Goldenberg, Koucký and Saks

[1]Throughout the paper, by "algorithm" we mean "randomized algorithm with constant error probability".

showed that it is even possible to compute a constant-factor approximation in strongly subquadratic time [15]. Subsequent work [13, 21] improved the running time to close-to-linear for the regime of near-linear edit distance. Very recently, Andoni and Nosatzki [7] extended this to the general case, showing that in time $O(n^{1+\varepsilon})$ one can compute a $f(1/\varepsilon)$-approximation for some function f depending solely on ε.

While for exact algorithms (almost-)linear-time algorithms are the gold standard, for approximation algorithms it is not clear whether a running time of $O(n^{1+\varepsilon})$ is desired, as in fact one might hope for *sublinear-time* algorithms. Indeed, another line of research explored the edit distance in the sublinear setting, where one has random access to the strings X and Y, and the goal is to compute the edit distance between X and Y without even reading the whole input. Formally, in the (k, K)-gap edit distance problem the task is to distinguish whether the edit distance is at most k or at least K. The running time is analyzed in terms of the string length n and the gap parameters k and K. Initiating the study of this problem, Batu et al. [11] showed how to solve the $(k, \Omega(n))$-gap problem in time $O(k^2/n + \sqrt{k})$, assuming $k \leq n^{1-\Omega(1)}$. The aforementioned algorithm by Andoni and Onak [8] can be viewed in the sublinear setting and solves the (k, K)-gap problem in time $n^{2+o(1)} \cdot k/K^2$, assuming $K/k = n^{\Omega(1)}$. In a major contribution, Goldenberg, Krauthgamer and Saha [17] showed how to solve the (k, K)-gap problem in time[2] $\widetilde{O}(nk/K + k^3)$. Subsequently, Kociumaka and Saha [20] improved the running time to $\widetilde{O}(nk/K + k^2 + \sqrt{nk^5}/K)$. They focus their presentation on the (k, k^2)-gap problem, where they achieve time $\widetilde{O}(n/k + k^2)$.

How far from optimal are these algorithms? It is well-known that (k, K)-gap edit distance requires time $\Omega(n/K)$ (this follows from the same bound for Hamming distance). Batu et al. [11] additionally proved that $(k, \Omega(n))$-gap edit distance requires time $\Omega(\sqrt{k})$. Together, (k, K)-gap edit distance requires time $\Omega(n/K + \sqrt{k})$, but this leaves a big gap to the known algorithms. In particular, in the low distance regime (where, say, $K \leq n^{0.01}$) the lower bound is $\Omega(n/K)$ and the upper bound is $\widetilde{O}(nk/K)$. Closing this gap has been raised as an open problem by the authors of the previous sublinear-time algorithms.[3] In particular, a natural question is to determine the smallest possible gap which can be distinguished within the time budget of the previous algorithms, say $O(n/k + \mathrm{poly}(k))$. Is the currently-best (k, k^2)-gap barrier penetrable or can one prove a lower bound?

Our Contribution. We show that $(k, k^{1+o(1)})$-gap edit distance can be solved in time $O(n/k + k^{4+o(1)})$. In the low distance regime ($k \leq n^{0.19}$) this runs in the same time as the previous algorithms for the (k, k^2)-gap problem, so we significantly improve the gap [17, 20]. We stress that the quadratic gap seems to be the limit of the techniques of previous work. Formally, we obtain the following results.

Theorem 1 (Main Theorem). *Let $2 \leq B \leq k$ be a parameter. The $(k, \Theta(k \log_B(k) \cdot B))$-gap edit distance problem can be solved in time*

$$\frac{n}{k} \cdot (\log k)^{O(\log_B(k))} + \widetilde{O}(k^4 \operatorname{poly}(B)).$$

Corollary 2 (Subpolynomial Gap). *In time $O(n/k + k^{4+o(1)})$ we can solve $(k, k \cdot 2^{\widetilde{O}(\sqrt{\log k})})$-gap edit distance.*

Corollary 3 (Polylogarithmic Gap). *For any $\varepsilon \in (0, 1)$, in time $O(n/k^{1-\varepsilon} + k^{4+o(1)})$ we can solve $(k, k \cdot (\log k)^{O(1/\varepsilon)})$-gap edit distance.*

Note that one can solve the (k, k^2)-gap edit distance problem by running our algorithm from Corollary 2 for $\bar{k} := k^{2-o(1)}$. This runs in time $O(n/k^{2-o(1)} + \mathrm{poly}(k))$, which improves the previously best running time of $\widetilde{O}(n/k + \mathrm{poly}(k))$ for the (k, k^2)-gap edit distance problem [17, 20] by a factor $k^{1-o(1)}$ in the low distance regime.

Edit Distance versus Hamming Distance. It is interesting to compare our result against the best-possible sublinear-time algorithms for approximating the *Hamming distance.* For Hamming distance, it is well known that the (k, K)-gap problem has complexity $\Theta(n/K)$, with matching upper and (unconditional) lower bounds, assuming that $K/k = 1 + \Omega(1)$. In the large distance regime (where, say, $k \geq n^{0.51}$), Hamming distance and edit distance have been *separated* by the $\Omega(\sqrt{n})$ lower bound for $(k, O(k))$-gap edit distance [6, 11], because $(k, O(k))$-gap Hamming distance can be solved in time $O(n/k)$.

Our results show a surprising *similarity* of Hamming distance and edit distance: In the low distance regime ($k \leq n^{0.19}$) the complexity of the $(k, k^{1+o(1)})$-gap problem is $n/k^{1\pm o(1)}$ for both Hamming distance and edit distance. Thus, up to $k^{o(1)}$-factors in the gap and running time, their complexity is the same in the low distance regime.[4]

Our Techniques. To achieve our result, we depart from the framework of subsampling the Landau-Vishkin algorithm [22, 23], which has been developed by the state-of-the-art algorithms for sublinear edit distance [17, 20]. Instead, we pick up the thread from the *almost-linear-time* algorithm by Andoni, Krauthgamer and Onak [4]: First, we give (what we believe to be) a more accessible view on that algorithm. Subsequently, we design a sublinear version of it by *pruning* certain branches in its recursion tree and thereby avoid spending time on "cheap" subproblems. To this end, we use a variety of structural insights on the (local and global) patterns that can emerge during the algorithm and design property testers to effectively detect these patterns.

Comparison to Goldenberg, Kociumaka, Krauthgamer, Saha [16]. In a recent arXiv paper, Goldenberg et al. studied the complexity of the (k, k^2)-gap edit distance problem in terms of *non-adaptive* algorithms [16]. Their main result is an $O(n/k^{3/2})$-time algorithm, and a matching query complexity lower bound. We remark that our results are incomparable: For the (k, k^2)-gap problem they obtain a non-adaptive algorithm (which is faster for large k), whereas we

[2]We write $\widetilde{O}(\cdot)$ to hide polylogarithmic factors, that is, $\widetilde{O}(T) = \bigcup_{c \geq 0} O(T \log^c T)$.
[3]See https://youtu.be/WFxk3JAOC84?t=1104 for the open problems raised in the conference talk on [17], and similarly https://youtu.be/3jfHHEFNRU4?t=1159 for [20].

[4]We remark that this similarity does not yet follow from previous sublinear algorithms, since they solve the (k, k^2)-gap edit distance in time $\widetilde{O}(n/k + \mathrm{poly}(k))$ [17, 20], while (k, k^2)-gap Hamming distance can be solved much faster, namely in time $O(n/k^2)$.

present an adaptive algorithm (which is faster for small k). Moreover, we can improve the gap to $(k, k^{1+o(1)})$, while still running in sublinear time $O(n/k)$ when k is small. Our techniques also differ substantially: Goldenberg et al. build on the work of Andoni and Onak [8] and Batu et al. [11], whereas our algorithm borrows from [4].

Outline. The rest of this paper is structured as follows. In Section 2 we introduce the necessary preliminaries. In Section 3 we give a high-level overview of our algorithm, starting with the necessary ingredients from previous work. We omit the proof details in this section and give the detailed proofs in Section 4.

2 PRELIMINARIES

For integers i, j we write $[\,i\,..\,j\,] = \{\,i, i+1, \dots, j-1\,\}$ and $[\,j\,] = [\,0\,..\,j\,]$. We also set $\mathrm{poly}(n) = n^{O(1)}$, $\mathrm{polylog}(n) = (\log n)^{O(1)}$ and $\widetilde{O}(n) = n \,\mathrm{polylog}(n)$.

Strings. We usually denote strings by capital letters X, Y, Z. The length of a string X is denoted by $|X|$ and we write $X \circ Y$ to denote the *concatenation* of X and Y. For a nonnegative integer i, we denote by $X[\,i\,]$ the i-th character in X (starting with index zero). For integers i, j we denote by $X[\,i\,..\,j\,]$ the substring of X with indices in $[\,i\,..\,j\,]$. In particular, if the indices are out-of-bounds, then we set $X[\,i\,..\,j\,] = X[\,\max(i,0)\,..\,\min(j,|X|)\,]$.

For two strings X, Y with equal length, X is a *rotation* of Y if $X[\,i\,] = Y[\,(i+s) \bmod |Y|\,]$ for some integer shift s. We say that X is *primitive* if all of the nontrivial rotations of X are not equal to X. For a string P, we denote by P^* the infinite-length string obtained by repeating P. We say that X is *periodic with period P* if $X = P^*[\,0\,..\,|X|\,]$. We also say that X is *p-periodic* if it is periodic with some period of length at most p.

Hamming and Edit Distance. For two equal-length strings X, Y, we define their *Hamming distance* $\mathrm{HD}(X,Y)$ as the number of non-equal characters: $\mathrm{HD}(X,Y) = |\{\,i : X[\,i\,] \neq Y[\,i\,]\,\}|$. For two strings X, Y (with possibly different lengths), we define their *edit distance* $\mathrm{ED}(X,Y)$ as the smallest number of edit operations necessary to transform X into Y; here, an edit operation means *inserting*, *deleting* or *substituting* a character.

We also define an *optimal alignment*, which is a basic object in several of the forthcoming proofs. For two strings X, Y, an *alignment* between X and Y is a monotonically non-decreasing function $A : \{\,0, \dots, |X|\,\} \to \{\,0, \dots, |Y|\,\}$ such that $A(0) = 0$ and $A(|X|) = |Y|$. We say that A is an *optimal alignment* if additionally

$$\mathrm{ED}(X,Y) = \sum_{i=0}^{|X|-1} \mathrm{ED}(X[\,i\,], Y[\,A(i)\,..\,A(i+1)\,]).$$

This definition is slightly non-standard (compare for instance to the definition in [18]), but more convenient for our purposes. Note that the alignments between X and Y correspond to the paths through the standard edit distance dynamic program. In that correspondence, an optimal alignment corresponds to a minimum-cost path.

Trees. In the following we will implicitly refer to trees T where each node has an ordered list of children. A node is a *leaf* if it has no children and otherwise an *internal* node. The *depth* of a node v is defined as the number of ancestors of v, and the depth of a tree T is the length of the longest root-leaf path. We refer to the subset of nodes with depth i as the *i-th level* in T.

Approximations. We often deal with additive *and* multiplicative approximations in this paper. To simplify the notation, we say that $\widetilde{x}$ is a (a,b)-approximation of x whenever $x/a - b \leq \widetilde{x} \leq ax + b$.

3 OVERVIEW

We start by reinterpreting the algorithm of Andoni, Krauthgamer and Onak [4] as a *framework* consisting of a few fundamental ingredients (Section 3.1). We then quickly show how to instantiate this framework to recover their original algorithm (Section 3.2), and then develop further refinements to obtain our main result (Section 3.3). The goal of this section is to outline the main pieces of our algorithm; we defer the formal proofs to Section 4 and the Appendix.

3.1 The Andoni-Krauthgamer-Onak Framework

First Ingredient: Tree Distance. The first crucial ingredient for the framework is a way to split the computation of the edit distance into smaller, independent subtasks. A natural approach would be to divide the two strings into equally sized blocks, compute the edit distances of the smaller blocks recursively, and combine the results. The difficulty in doing this is that the edit distance might depend on a *global alignment*, which determines how the blocks should align and therefore the subproblems are not independent (e.g. the optimal alignment of one block might affect the optimal alignment of the next block). However, this can be overcome by computing the edit distances of one block in one string with several shifts of its corresponding block in the other string, and combining the results smartly. This type of *hierarchical decomposition* appeared in previous algorithms for approximating edit distance [4, 8, 11, 25]. In particular, Andoni, Krauthgamer and Onak [4] define a string similarity measure called the *tree distance*[5] which gives a good approximation of the edit distance and cleanly splits the computation into independent subproblems.

We will define the tree distance for an underlying tree T which we sometimes refer to as the *partition tree*.

Definition 4 (Partition Tree). *Let T be a tree where each node v is labeled with a non-empty interval I_v. We call T a* partition tree *if*

- *for the root node v we have $I_v = [\,n\,]$, and*
- *for any node v with children $v_0, \dots, v_{B-1}$, I_v is the concatenation of $I_{v_0}, \dots, I_{v_{B-1}}$.*

For the original Andoni-Krauthgamer-Onak algorithm, we will use a complete B-ary partition tree T with n leaves. In particular, the depth of T is bounded by $\log_B(n)$. There is a unique way to label T with intervals I_v: For the i-th leaf (ordered from left to right) we set $I_v = \{\,i\,\}$; this choice determines the intervals for all internal nodes. For our algorithm we will later focus on the subtree of T with depth bounded by $O(\log_B(k))$ (this is again a partition tree, as can easily be checked).

The purpose of the partition tree is that it determines a decomposition of two length-n strings X and Y: For a node labeled with

[5]In fact, Andoni et al. call the measure the $\mathcal{E}$-*distance*. However, in a talk by Robert Krauthgamer he recoined the name to *tree distance*, and we decided to stick to this more descriptive name.

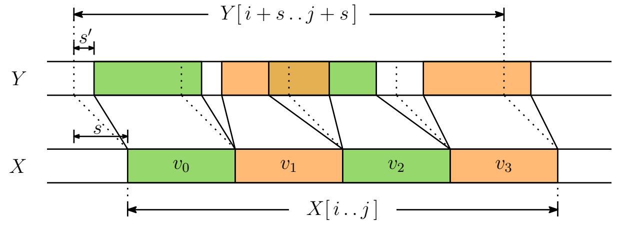

Figure 1: Illustrates the tree distance $\mathrm{TD}_{v,s}(X,Y)$ at a node v with interval $I_v = [\,i\,..\,j\,]$, shift s and children $v_0,\dots,v_3$. The dashed lines denote the shift given by s. The bold lines show the "local" shifts s' for each of the children (explicitly labeled for v_0).

interval $I_v = [\,i\,..\,j\,]$ we focus on the substrings $X[\,i\,..\,j\,]$ and $Y[\,i\,..\,j\,]$. In particular, the leaf labels in the partition tree determine a partition into consecutive substrings. With this definition in hand, we can define the tree distance:

Definition 5 (Tree Distance). *Let X, Y be length-n strings and let T be a partition tree. For any node v in T and any shift $s \in \mathbf{Z}$, we set:*

- *If v is a leaf with $I_v = [\,i\,..\,j\,]$, then*
$$\mathrm{TD}_{v,s}(X,Y) = \mathrm{ED}(X[\,i\,..\,j\,], Y[\,i+s\,..\,j+s\,]).$$
- *If v is an internal node with children $v_0,\dots,v_{B-1}$, then*
$$\mathrm{TD}_{v,s}(X,Y) = \sum_{i\in[\,B\,]} \min_{s'\in\mathbf{Z}} (\mathrm{TD}_{v_i,s'}(X,Y) + 2\cdot|s-s'|). \quad (1)$$

We write $\mathrm{TD}_T(X,Y) = \mathrm{TD}_{v,0}(X,Y)$ where v is the root node in T, and we may omit the subscript T when it is clear from the context.

Figure 1 gives an illustration of this definition. The following lemma (which slightly generalizes the analogous result by [4]) shows that the tree distance is a useful measure to approximate the edit distance of two strings. We include a proof in the full version.

Lemma 6 (Equivalence of Edit Distance and Tree Distance). *Let X, Y be strings and let T be a partition tree with degree at most B and depth at most D. Then $\mathrm{ED}(X,Y) \le \mathrm{TD}_T(X,Y) \le 2BD\cdot\mathrm{ED}(X,Y)$.*

In light of this lemma, we now focus on approximating the tree distance of two strings. The idea behind the Andoni-Krauthgamer-Onak algorithm is to approximately evaluate Definition 5 for all nodes v in the partition tree: For the leaves we directly evaluate the edit distance and for the internal nodes we use Equation (1) to combine the recursive computations. However, notice that in Equation (1) we minimize over an *infinite* number of shifts $s' \in \mathbf{Z}$. To remedy this situation, recall that we anyways only want to solve a gap problem: Whenever the tree distance exceeds some value K we will immediately report *far*. We therefore restrict our attention to approximating the *capped tree distance:*

Definition 7 (Capped Distances). *Let K be a threshold. We define the K-capped edit distance $\mathrm{ED}^{\le K}(X,Y) = \min(\mathrm{ED}(X,Y), K)$ and the K-capped tree distance $\mathrm{TD}^{\le K}_T(X,Y) = \min(\mathrm{TD}_T(X,Y), K)$.*

It is easy to observe that for computing the K-capped tree distance $\mathrm{TD}^{\le K}(X,Y)$, it suffices to let s' range through $\{-K,\dots,K\}$ in Equation (1), that is:

$$\mathrm{TD}^{\le K}_{v,s}(X,Y) = \min\left(\sum_{i\in[\,B\,]} \min_{-K\le s'\le K} (\mathrm{TD}^{\le K}_{v_i,s'}(X,Y) + 2\cdot|s-s'|),\ K\right). \quad (2)$$

The Formal Setup. At this point we are ready to define precisely what we want to compute. We will associate a computational problem to each node in the partition tree T. Roughly speaking, the goal for a node v is to approximate the tree distance $\mathrm{TD}^{\le K}_{v,s}(X,Y)$. We now make the details of this approximation precise. Let us first introduce the following notational shortcuts:

- $X_v = X[\,i\,..\,j\,]$ (the substring of X relevant at v),
- $Y_{v,s} = Y[\,i+s\,..\,j+s\,]$ (the substring of Y relevant at v for one specific shift s),
- $Y_v = Y[\,i-K\,..\,j+K\,]$ (the entire substring of Y relevant at v).

Moreover, for each node v in the partition tree we also define:

- A *rate* (or *additive accuracy*) r_v. This parameter serves two purposes. First, it gives a budget for the number of characters that can be read at v (i.e., we are allowed to read $r_v|X_v|$ symbols). Second, it determines the *additive* approximation guarantee at v, in the sense that the output at v is allowed to have additive error $1/r_v$. For the root node we set $r_v = 1000/K$.
- The *multiplicative accuracy* $\alpha_v > 1$. This parameter determines the multiplicative approximation ratio at v. For the root node we set $\alpha_v = 10$.

We are now ready to define the precise computational task for each node:

Definition 8 (Tree Distance Problem). *Given a node v in the partition tree T, compute a list of numbers $\Delta_{v,-K},\dots,\Delta_{v,K}$ such that*

$$\frac{1}{\alpha_v}\,\mathrm{ED}^{\le K}(X_v, Y_{v,s}) - \frac{1}{r_v} \le \Delta_{v,s} \le \alpha_v\,\mathrm{TD}^{\le K}_{v,s}(X,Y) + \frac{1}{r_v}. \quad (3)$$

Given a node v of the partition tree, we will sometimes refer to solving the Tree Distance Problem at v as simply *solving* v.

As a sanity check, let us confirm that an algorithm for the Tree Distance Problem can distinguish edit distances with a polylogarithmic gap. Assume that we solve the Tree Distance Problem for the root node v, and let $\Delta = \Delta_{v,0}$. Then, Equation (3) implies that $0.1\cdot\mathrm{ED}^{\le K}(X,Y) - 0.001K \le \Delta \le 10\cdot\mathrm{TD}^{\le K}(X,Y) + 0.001K$. Consequently, we can distinguish whether the edit distance $\mathrm{ED}(X,Y)$ is at most $k = K/(1000B\log_B(n))$ or at least K: On the one hand, if $\mathrm{ED}(X,Y) \le k$ then Lemma 6 implies that $\Delta \le 0.02K + 0.001K = 0.021K$. On the other hand, if $\mathrm{ED}(X,Y) \ge K$ then $\Delta \ge 0.099K$.

Second Ingredient: The Precision Sampling Lemma. The next step is to assign appropriate sampling rates r_v to every node in the partition tree. The rate at any node indicates that we can afford to read an r_v-fraction of the characters in the computation of that entire subtree (at the root, we will set $r_v = 1000/k$ so we read $O(n/k)$ characters in total).

Focus on some node with rate r_v. Due to the sampling rate, we are bound to incur an *additive* error of at least $1/r_v$. The challenge is now the following: We want to assign rates to the B children of v

in such a way that we can obtain a good approximation of the tree distance at v after combining the results from the children. Naively assigning the same rate to all children incurs an additive error of at least B/r_v, which is too large. To decrease the error we can increase the rate, but this results in reading more than the $r_v \cdot |X_v|$ characters we are aiming for.

Roughly speaking we have an instance of the following problem: There are unknown numbers $A_1, \dots, A_n \in \mathbf{R}$. We can specify *precisions* $u_1, \dots, u_n$ and obtain estimates $\widetilde{A}_i$ such that $|A_i - \widetilde{A}_i| \le u_i$, where the *cost* of each estimate is $1/u_i$ (in our setting the cost corresponds to the number of characters we read). The goal is to set the precisions appropriately to be able to distinguish whether $\sum_i A_i < 0.1$ or $\sum_i A_i > 10$, say, and minimize the total cost $\sum_i 1/u_i$. If we set the precisions equally, we would need to have $u_i < 10/n$ (otherwise we cannot distinguish the case where $A_i = 10/n$ for all i from the case $A_i = 0$ for all i), which incurs in total cost $\Omega(n^2)$. Andoni, Krauthgamer and Onak [4] give a very elegant randomized solution to this problem with total cost $\widetilde{O}(n)$ and good error probability.

Lemma 9. *Let $\varepsilon, \delta > 0$. There is a distribution $\mathcal{D} = \mathcal{D}(\varepsilon, \delta)$ supported over the real interval $(0, 1]$ with the following properties:*

- **Accuracy:** *Let $A_1, \dots, A_n \in \mathbf{R}$ and let $u_1, \dots, u_n \sim \mathcal{D}$ be sampled independently. There is a recovery algorithm with the following guarantee: Given $(a, b \cdot u_i)$-approximations $\widetilde{A}_i$ of A_i, the algorithm $((1+\varepsilon) \cdot a, b)$-approximates $\sum_i A_i$ in time $O(n \cdot \varepsilon^{-2} \log(\delta^{-1}))$, with probability at least $1 - \delta$ and for any parameters $a \ge 1$ and $b \ge 0$.*
- **Efficiency:** *Sample $u \sim \mathcal{D}$. Then, for any $N \ge 1$ there is an event $E = E(u)$ happening with probability at least $1 - 1/N$, such that $\mathbf{E}_{u \sim \mathcal{D}}[\, 1/u \mid E \,] \le \widetilde{O}(\varepsilon^{-2} \log(\delta^{-1}) \log N)$.*

The Precision Sampling Lemma was first shown in [4] and later refined and simplified in [3, 5]. In the full version we include a proof.

Third Ingredient: A Range Minimum Problem. The final ingredient is an efficient algorithm to combine the recursively computed tree distances. Specifically, the following subproblem can be solved efficiently:

Lemma 10 (A Range Minimum Problem). *There is an $O(K)$-time algorithm for the following problem: Given integers $A_{-K}, \dots, A_K$, compute for all $s \in \{-K, \dots, K\}$:*

$$B_s = \min_{-K \le s' \le K} A_{s'} + 2 \cdot |s - s'|.$$

In [4], the authors use efficient Range Minimum queries (for instance implemented by segment trees) to *approximately* solve this problem with a polylogarithmic overhead. We present a simpler, faster and *exact* algorithm in the full version of this paper.

The Framework. We are ready to assemble the three ingredients in a baseline algorithm which solves the Tree Distance Problem; see Algorithm 1 for the pseudocode. We will first sketch the correctness of this algorithm. Later (in Sections 3.2 and 3.3) we will improve this algorithm to first obtain the Andoni-Krauthgamer-Onak algorithm, and second our sublinear-time version.

We quickly discuss the correctness of Algorithm 1. In Lines 1 and 2 we test whether the node v is a leaf,

Algorithm 1 The framework

```
Input: Strings $X, Y$, a node $v$ in the partition tree $T$ and a sampling
    rate $r_v > 0$
Output: $\Delta_{v,s}$ for all shifts $s \in \{-K, \dots, K\}$
1 if $v$ is a leaf then
2     return $\Delta_{v,s} = \mathrm{ED}(X_v, Y_{v,s})$ for all $s \in \{-K, \dots, K\}$
3 for each $i \in [\,B\,]$ do
4     Let $v_i$ be the $i$-th child of $v$ and sample $u_{v_i} \sim \mathcal{D}$
      (with appropriate parameters)
5     Recursively compute $\Delta_{v_i,s}$ with rate $r_v/u_{v_i}$ for
      all $s \in \{-K, \dots, K\}$
6     Compute $\widetilde{A}_{i,s} = \min_{s' \in \{-K,\dots,K\}} \Delta_{v_i,s'} + 2 \cdot |s - s'|$
      using Lemma 10
7 for each $s \in \{-K, \dots, K\}$ do
8     Let $\Delta_{v,s}$ be the result of the recovery algorithm (Lemma 9)
      applied to $\widetilde{A}_{0,s}, \dots, \widetilde{A}_{B-1,s}$ with precisions $u_{v_0}, \dots, u_{v_{B-1}}$
9 return $\min(\Delta_{v,s}, K)$ for all $s \in \{-K, \dots, K\}$
```

then we have reached the base case of Definition 5 and computing the tree distances boils down to computing the edit distance. In Lines 3 to 9 we use the approximations recursively computed by v's children to solve v. The idea is to approximately evaluate the following expression for the tree distance (which is equivalent to Equation (2)):

$$\mathrm{TD}^{\le K}_{v,s}(X, Y) = \min\left(\sum_{i \in [\,B\,]} A_{i,s},\ K\right),$$
$$A_{i,s} = \min_{-K \le s' \le K} (\mathrm{TD}^{\le K}_{v_i,s'}(X, Y) + 2 \cdot |s - s'|).$$

For each child v_i of v the algorithm first recursively computes an approximation $\Delta_{v_i,s'}$ of $\mathrm{TD}^{\le K}_{v_i,s'}(X, Y)$ in Line 5. Then, in Line 6, we exactly evaluate

$$\widetilde{A}_{i,s} = \min_{-K \le s' \le K} \Delta_{v_i,s'} + 2 \cdot |s - s'|$$

for all s using Lemma 10. In the next step, the Precision Sampling Lemma comes into play. The recursive call returns approximations $\Delta_{v_i,s'}$ with multiplicative error α_{v_i} and additive error $1/r_{v_i} = u_{v_i}/r_v$. Hence, $\widetilde{A}_{i,s}$ is also an approximation of $A_{i,s}$ with multiplicative error α_{v_i} and additive error u_{v_i}/r_v. In this situation, the Precision Sampling Lemma (Lemma 9) allows to approximate the sum $\sum_i A_{i,s}$ (for some fixed s) with multiplicative error $(1+\varepsilon) \cdot \alpha_{v_i}$ and additive error $1/r_v$—just as required if we set $\alpha_v > (1+\varepsilon) \cdot \alpha_{v_i}$. Hence, the values $\Delta_{v,s}$ computed in Line 8 are as claimed (up to taking the minimum with K in order to obtain estimates for the capped tree distance, see Line 9).

In the next Sections 3.2 and 3.3 we will instantiate this framework by adding *pruning rules* to Algorithm 1. That is, we design rules to directly solve certain nodes v without having to recur on v's children. In this way we will reduce the number of recursive calls and thereby the running time of Algorithm 1.

3.2 The Andoni-Krauthgamer-Onak Algorithm

For completeness (and also to showcase how to instantiate the previous framework), we quickly demonstrate how a single pruning rule leads to the algorithm by Andoni, Krauthgamer and Onak [4]. Specifically, we add the following pruning rule to Algorithm 1: *If* $|X_v| \le 1/r_v$, *then return* $\Delta_{v,s} = 0$ *for all* $s \in \{-K, \dots, K\}$. This rule is correct, since the edit distance $\mathrm{ED}(X_v, Y_{v,s})$ is at most $|X_v|$. Hence, returning zero satisfies the condition in Equation (3).

It remains to analyze the running time. In this regard the analysis differs quite substantially from our new sublinear-time algorithm. We will therefore be brief here and refer to the full version for the complete proof. A single execution of Algorithm 1 (ignoring the recursive calls and lower-order factors such as B) takes roughly time $O(K)$. However, note that the recursive calls do not necessarily reach every node in the partition tree: Some nodes v are trivially solved by the new rule and thus their children are never explored. Let us call a node v *active* if the recursive computation reaches v. One can bound the number of active nodes by $n/K \cdot (\log n)^{O(\log_B(n))}$ hence obtaining the time bound $n \cdot (\log n)^{O(\log_B(n))}$.

3.3 Our Algorithm

We are finally ready to describe the pruning rules leading to our sublinear-time algorithm. In contrast to the previous section, our pruning rules will allow us to bound the number of active nodes by $\mathrm{poly}(K)$. We will however spend more time for each active node: In the Andoni-Krauthgamer-Onak algorithm the running time per node is essentially K, whereas in our version we run some more elaborate tests per node spending time proportional to $r_v|X_v| + \mathrm{poly}(K)$. We will now progressively develop the pruning rules; the pseudocode is given at the end of this section.

3.3.1 *The Big Picture.*

First Insight: Matching Substrings. The first insight is that if $\mathrm{ED}(X, Y) \le K$, then we can assume that for almost all nodes v there exists a shift $s^* \in \{-K, \dots, K\}$ for which $X_v = Y_{v,s^*}$. In this case, we say that the node v is *matched.* The benefit is that if we know that v is matched with shift s^*, then instead of approximating the edit distances between X_v and all shifts $Y_{v,s}$, we can instead approximate the edit distance between Y_{v,s^*} and all shifts $Y_{v,s}$. That is, it suffices to compute the edit distances between a string and *a shift of itself.*

To see that almost all nodes are matched, consider an optimal alignment between X and Y and recall that each level of the partition tree induces a partition of X into substrings X_v. The number of misaligned characters is bounded by $\mathrm{ED}(X, Y)$, thus there are at most $\mathrm{ED}(X, Y)$ parts X_v containing a misalignment. For all other parts, the complete part X_v is perfectly matched to some substring $Y_{v,s}$. We prove the claim formally in Lemma 23. In light of this insight, we can assume that there are only K unmatched nodes—otherwise, the edit distance and thereby also the tree distance between X and Y exceeds K and we can stop the algorithm. In the following we will therefore focus on matched nodes only.

For now we assume that we can efficiently test whether a node is matched. We justify this assumption soon (see the *Matching Test* in Lemma 16) by giving a property tester for this problem in sublinear time.

Second Insight: Structure versus Randomness. The second idea is to exploit a structure versus randomness dichotomy on strings: As the two extreme cases, a string is either periodic or random-like. The hope is that whenever Y_v falls into one of these extreme categories, then we efficiently approximate $\mathrm{ED}(Y_{v,s^*}, Y_{v,s})$ (without expanding v's children). Concretely, we use the following measure to interpolate between periodic and random-like:

Definition 11 (Block Periodicity). *Let* Y *be a string. The* K*-block periodicity* $\mathrm{BP}_K(Y)$ *of* Y *is the smallest integer* L *such that* Y *can be partitioned into* $Y = \bigcirc_{\ell=1}^{L} Y_\ell$, *where each substring* Y_ℓ *is* K*-periodic (i.e.,* Y_ℓ *is periodic with period length at most* K*).*

Suppose for the moment that we could efficiently compute the block periodicity of a string Y. Under this assumption, we first compute $\mathrm{BP}_{4K}(Y_v)$ for each matched node v in the tree and distinguish three regimes depending on whether the block periodicity is small, large or intermediate. In the following section we discuss the pruning rules that we apply in these regimes.

3.3.2 *Pruning Rules.*

The Periodic Regime: $\mathrm{BP}_{4K}(Y_v) = 1$. For this case, we can approximate the edit distances via the following lemma, where you can think of $Y = Y_v$ and $Y_s = Y_{v,s}$.

Lemma 12 (Periodic Rule). *Let* Y *be a string and write* $Y_s = Y[\,K + s \,..\, |Y| - K + s\,]$. *If* Y *is periodic with primitive period* P *and* $|Y| \ge |P|^2 + 2K$, *then for all* $s, s' \in \{-K, \dots, K\}$*:*

$$\mathrm{ED}(Y_s, Y_{s'}) = 2 \cdot \min_{j \in \mathbb{Z}} \bigl|\, s - s' + j|P| \,\bigr|.$$

Note that if some node v that is matched and we know that Y_v is periodic, then Lemma 12 allows us to return $\Delta_{v,s} = 2 \cdot \min_{j \in \mathbb{Z}} \bigl| s - s^* + j|P| \bigr|$ as the desired estimates for each shift s. In this way we do not recur on v's children and thereby prune the entire subtree below v.

To get some intuition for why Lemma 12 holds, observe that $2 \cdot \min_j \bigl| s - s' + j|P| \bigr|$ is exactly the cost of aligning both Y_s and $Y_{s'}$ in such a way that all occurrences of the period P match (i.e., we shift both strings to the closest-possible occurrence of P). The interesting part is to prove that this alignment is best-possible. We give the complete proof in Section 4.2.

The Random-Like Regime: $\mathrm{BP}_{4K}(Y_v) > 10K$. Next, we give the analogous pruning rule for the case when the block periodicity of Y_v is large. We show that in this case, the best possible way to align any two shifts $Y_{v,s}$ and $Y_{v,s'}$ is to insert and delete $|s - s'|$ many characters. In this sense, Y_v behaves like a *random* string.

Lemma 13 (Random-Like Rule). *Let* Y *be a string and write* $Y_s = Y[\,K + s \,..\, |Y| - K + s\,]$. *If* $\mathrm{BP}_{4K}(Y) > 10K$, *then for all* $s, s' \in \{-K, \dots, K\}$*:*

$$\mathrm{ED}(Y_s, Y_{s'}) = 2 \cdot |s - s'|.$$

Again, this rule can be used to solve a matched node v directly, pruning the subtree below v. We give the proof in Section 4.2.

The Intermediate Regime: $1 < \mathrm{BP}_{4K}(Y_v) \le 10K$. We cannot directly approximate the edit distance $\mathrm{ED}(Y_{v,s^*}, Y_{v,s})$ in this case. Instead, we exploit the following lemma to argue that the branching procedure below v is computationally cheap:

Lemma 14 (Intermediate). *Let v be a node in the partition tree. Then, in any level in the subtree below v, for all but at most $2\,\mathrm{BP}_{4K}(Y_v)$ nodes w the string Y_w is $4K$-periodic.*

Indeed, since any node v for which Y_w is $4K$-periodic can be solved by the Periodic Rule, this lemma implies that there are at most $20K$ active nodes on any level in the partition subtree below any matched node v.

3.3.3 String Property Testers. Next, we describe how to remove the assumptions that we can efficiently test whether a node is matched and that we can compute block periodicities. We start with the second task.

Computing Block Periodicities? The most obvious approach would be to show how to compute (or appropriately approximate) the block periodicity. This is indeed possible, but leads to a more complicated and slower algorithm (in terms of $\mathrm{poly}(K)$).

Instead, we twist the previous argument: We first show how to detect whether a string is periodic (in the straightforward way, see the following Lemma 15). For any matched node v we then run the following procedure: If Y_v is $4K$-periodic, then we solve v according to Lemma 12. Otherwise, we continue to explore v's children—with the following constraint: If at some point there are more than $20K$ active nodes which are not $4K$-periodic on any level in the recursion tree below v, then we interrupt the computation of this subtree and immediately solve v according to Lemma 13. This approach is correct, since by Lemma 14 witnessing more than $20K$ active nodes which are not $4K$-periodic on any level serves as a certificate that $\mathrm{BP}_{4K}(Y_v)$ is large. To test whether Y_v is $4K$-periodic, we use the following tester:

Lemma 15 (Periodicity Test). *Let X be a string, and let $r > 0$ be a sampling rate. There is an algorithm which returns one of the following two outputs:*

- *Close(P), where P is a primitive string of length $\leq K$ with $\mathrm{HD}(X, P^*[\,0\,..\,|X|\,]) \leq 1/r$.*
- *Far, in which case X is not K-periodic.*

The algorithm runs in time $O(r|X|\log(\delta^{-1}) + K)$ and is correct with probability $1-\delta$.

Note that as we are shooting for a sublinear-time algorithm we have to resort to a property tester which can only distinguish between *close* and *far* properties (in this case: periodic or *far from periodic*). The proof of Lemma 15 is simple, see Section 4.3 for details.

Testing for Matched Nodes. We need another algorithmic primitive to test whether a node is matched. Again, since our goal is to design a sublinear-time algorithm, we settle for the following algorithm which distinguishes whether v is matched or *far from matched.*

Lemma 16 (Matching Test). *Let X, Y be strings such that $|Y| = |X| + 2K$, and let $r > 0$ be a sampling rate. There is an algorithm which returns one of the following two outputs:*

- *Close(s^*), where $s^* \in \{-K, \dots, K\}$ satisfies $\mathrm{HD}(X, Y[\,K + s^*\,..\,|X| + K + s^*\,]) \leq 1/r$.*
- *Far, in which case there is no $s^* \in \{-K, \dots, K\}$ with $X = Y[\,K + s^*\,..\,|X| + K + s^*\,]$.*

Algorithm 2

```
Input: Strings $X, Y$, a node $v$ in the partition tree $T$ and a rate $r_v$
Output: $\Delta_{v,s}$ for all shifts $s \in \{-K, \dots, K\}$
1  if $v$ is a leaf or $|X_v| \leq 100K^2$ then
2      return $\Delta_{v,s} = \mathrm{ED}^{\leq K}(X_v, Y_{v,s})$
       (or compute 2-approximations using Theorem 17)
3  Run the Matching Test (Lemma 16) for $X_v, Y_v$
   (with parameters $r = 3r_v$ and $\delta = 0.01 \cdot K^{-100}$)
4  Run the $4K$-Periodicity Test (Lemma 15) for $Y_v$
   (with parameters $r = 3r_v$ and $\delta = 0.01 \cdot K^{-100}$)
5  if the Matching Test returns Close($s^*$) then
6      if the Periodicity Test returns Close($P$) then
7          return $\Delta_{v,s} = 2 \cdot \min_{j \in \mathbb{Z}} |\,s - s^* + j|P|\,|$
8      else
9          Continue in Line 10 with the following exception:
           If at some point during the recursive computation there
           is some level containing more than $20K$ active nodes
           below $v$ for which the Periodicity Test (in Line 4)
           reports Far, then interrupt the recursive computation
           and return $\Delta_{v,s} = 2 \cdot |s - s^*|$
10 for each $i \in [\,B\,]$ do
11     Let $v_i$ be the $i$-th child of $v$ and sample
       $u_{v_i} \sim \mathcal{D}((200 \log K)^{-1}, 0.01 \cdot K^{-101})$
12     Recursively compute $\Delta_{v_i,s}$ with rate $r_v/u_{v_i}$ for
       all $s \in \{-K, \dots, K\}$
13     Compute $\widetilde{A}_{i,s} = \min_{s' \in \{-K,\dots,K\}} \Delta_{v_i,s'} + 2 \cdot |s - s'|$
       using Lemma 10
14 for each $s \in \{-K, \dots, K\}$ do
15     Let $\Delta_{v,s}$ be the result of the recovery algorithm (Lemma 9)
       applied to $\widetilde{A}_{0,s}, \dots, \widetilde{A}_{B-1,s}$ with precisions $u_{v_0}, \dots, u_{v_{B-1}}$
16 return $\min(\Delta_{v,s}, K)$ for all $s \in \{-K, \dots, K\}$
```

The algorithm runs time $O(r|X|\log(\delta^{-1}) + K\log|X|)$ and is correct with probability $1-\delta$.

The proof of Lemma 16 is non-trivial. The test involves checking whether X and Y follow a common period—in this case we can simply return any shift respecting the periodic pattern. If instead we witness errors to the periodic pattern, then we try to identify a shift under which also the errors align. See Section 4.3 for the details.

3.3.4 Putting the Pieces Together. We finally assemble our complete algorithm. The pseudocode is given in Algorithm 2. In this section we will sketch that Algorithm 2 correctly and efficiently solves the Tree Distance Problem. The formal analysis is deferred to Section 4.

Correctness. We first sketch the correctness of Algorithm 2. The recursive calls in Lines 10 to 16 are essentially copied from Algorithm 1 (except for differences in the parameters which are not important here) and correct by the same argument as before (using the Precision Sampling Lemma as the main ingredient). The interesting part happens in Lines 1 to 9. In Lines 1 and 2 we test whether the strings are short enough so that we can afford to compute the edit distances $\mathrm{ED}(X_v, Y_{v,s})$ by brute-force. If not, we continue to run the Matching Test for X_v and Y_v and the Periodicity Test for Y_v.

There are two interesting cases—both assume that the Matching Test reports $\text{Close}(s^*)$ and therefore $X_v \approx Y_{v,s^*}$ where $\approx$ denotes equality up to $1/(3r_v)$ Hamming errors. If also the Periodicity Test returns $\text{Close}(P)$ then we are in the situation that $X_v \approx Y_{v,s^*} \approx P^*$. Moreover $Y_{v,s}$ is $\approx$-approximately equal to a shift of P^*. Lemma 12 implies that the edit distance between X_v and $Y_{v,s}$ is approximately $2 \cdot \min_{j\in\mathbb{Z}} |\, s - s^* + j|P|\, |$. We lose an additive error of $1/(3r_v)$ for each of the three $\approx$ relations, hence the total additive error is $1/r_v$ as hoped, and we do not introduce any multiplicative error.

Next, assume that the Periodicity Test reports Far. Then, as stated in Line 9 we continue the recursive computation, but with an exception: If at some point during the recursive computation there are more than $20K$ active nodes on any level for which the Periodicity Test reports Far, then we interrupt the computation and return $\Delta_{v,s} = 2 \cdot |s - s^*|$. Suppose that indeed this exception occurs. Then there are more than $20K$ nodes w on one level of the partition tree below v for which Y_w is *not* $4K$-periodic. Using Lemma 14 we conclude that $\text{BP}_{4K}(Y_v) > 10K$, and therefore returning $\Delta_{v,s} = 2 \cdot |s - s^*|$ is correct by Lemma 13.

Running Time. We will first think of running Algorithm 2 with T being a balanced B-ary partition tree with n leaves, just as as in the Andoni-Krauthgamer-Onak algorithm (we will soon explain why we need to modify this). To bound the running time of Algorithm 2 we first prove that the number of active nodes in the partition tree is bounded by $\text{poly}(K)$. One can show that there are only $\text{poly}(K)$ unmatched nodes in the tree (see Lemma 23), so we may only focus on the matched nodes. Each matched node, however, is either solved directly (in Line 2 or in Line 7) or continues the recursive computation with at most $\text{poly}(K)$ active nodes (in Line 9). In Section 4 we give more details.

Knowing that the number of active nodes is small, we continue to bound the total expected running time of Algorithm 2. It is easy to check that a single execution of Algorithm 2 (ignoring the cost of recursive calls) is roughly in time $r_v|X_v| + \text{poly}(K)$. Using the Precision Sampling Lemma, we first bound r_v by $1/K \cdot (\log n)^{O(\log_B(n))}$ in expectation, for any node v. Therefore, the total running time can be bounded by

$$\begin{aligned}\sum_{v \text{ active}} (r_v|X_v| + \text{poly}(K)) \\ &\leq \frac{1}{K} \cdot (\log n)^{O(\log_B(n))} \cdot \sum_v |X_v| + \text{poly}(K) \\ &\leq \frac{n}{K} \cdot (\log n)^{O(\log_B(n))} + \text{poly}(K).\end{aligned}$$

Here we used that $\sum_w |X_w| = n$ where the sum is over all nodes w on any fixed level in the partition tree. It follows that $\sum_v |X_v| \leq n \log n$, summing over *all* nodes v.

Optimizing the Lower-Order Terms. This running time bound does not match the claimed bound in Theorem 1: The overhead $(\log n)^{O(\log_B(n))}$ should rather be $(\log K)^{O(\log_B(K))}$ and not depend on n. If $n \leq K^{100}$, say, then both terms match. But we can also reduce the running time in the general case by "cutting" the partition tree at depth $\log_B(K^{100})$. That is, we delete all nodes below that depth from the partition tree and treat the nodes at depth $\log_B(K^{100})$ as leaves in the algorithm. The remaining tree is still a partition tree, according to Definition 4. The correctness argument remains valid, but we have to prove that the running time of Line 2 does not explode. For each leaf at depth $\log_B(K^{100})$ we have that $|X_v| \leq n/K^{100}$ and for that reason computing $\text{ED}^{\leq K}(X_v, Y_{v,s})$ for all shifts s (say with the Landau-Vishkin algorithm) takes time $O(n/K^{99} + K^3)$. Since there are much less than K^{99} active nodes, the total contribution can again be bounded by $O(n/K + \text{poly}(K))$.

Optimizing the Polynomial Dependence on K. Finally, let us pinpoint the exponent of the polynomial dependence on K. In the current algorithm we can bound the number of active nodes by roughly K^2 (there are at most K nodes per level for which the matching test fails, and the subtrees rooted at these nodes contain at most K active nodes per level). The most expensive step in Algorithm 2 turns out to be the previously mentioned edit distance computation in Line 2. Using the Landau-Vishkin algorithm (with cap K) for $O(K)$ shifts s, the running time of Line 2 incurs a cubic dependence on K, and therefore the total dependence on K becomes roughly K^5.

We give a simple improvement to lower the dependence to K^4 and leave further optimizations of the $\text{poly}(K)$ dependence as future work. The idea is to use the following result on approximating the edit distances *for many shifts s:*

Theorem 17 (Edit Distance for Many Shifts). *Let X, Y be strings with $|Y| = |X| + 2K$. We can compute a (multiplicative) 2-approximation of $\text{ED}^{\leq K}(X, Y[\,K + s\,..\,|X| + K + s\,])$, for all shifts $s \in \{-K, \dots, K\}$, in time $O(|X| + K^2)$.*

This result can be proven by a modification of the Landau-Vishkin algorithm [23]; we provide the details in the full version. It remains to argue that computing a 2-approximation in Line 2 does not mess up the correctness proof. This involves the multiplicative approximation guarantee of our algorithm which we entirely skipped in this overview, and for this reason we defer the details to Section 4.

Implementation Details. We finally describe how to implement the interrupt condition in Line 9. Note that the order of the recursive calls in Line 12 is irrelevant. In the current form the algorithm explores the partition tree in a depth-first search manner, but we might as well use breadth-first search. For any node v which reaches Line 9 we may therefore continue to compute all recursive computations using breadth-first search. If at some point we encounter one search level containing more than $200K$ active nodes for which the Periodicity Test reports Far, we stop the breadth-first search and jump back to Line 9. With this modification the algorithm maintains one additional counter which does not increase the asymptotic time complexity.

4 OUR ALGORITHM IN DETAIL

In this section we give the formal analysis of Algorithm 2. We split the proof into the following parts: In Section 4.1 we prove some lemmas about periodic and block-periodic strings. In Section 4.2 we give the structural lemmas about edit distances in special cases. In Section 4.3 we prove the correctness of the string property testers (the "Matching Test" and "Periodicity Test"). In Section 4.4 we finally carry out the correctness and running time analyses for

Algorithm 2, and in Section 4.5 we give a formal proof of our main theorem.

In the following proofs we will often use the following simple proposition.

Proposition 18 (Alignments Have Small Stretch). *Let X, Y be strings of equal length. If A is an optimal alignment between X and Y, then $|i - A(i)| \le \frac{1}{2}\,\mathrm{ED}(X, Y)$ for all $0 \le i \le |X|$.*

Proof. Since A is an optimal alignment between X and Y, we can write the edit distance $\mathrm{ED}(X, Y)$ as $\mathrm{ED}(X[\,0\,..\,i\,], Y[\,0\,..\,A(i)\,]) + \mathrm{ED}(X[\,i\,..\,|X|\,], Y[\,A(i)\,..\,|Y|\,])$. Both edit distances are at least $|i - A(i)|$ which is the length difference of these strings, respectively. It follows that $\mathrm{ED}(X, Y) \ge 2 \cdot |i - A(i)|$, as claimed. □

4.1 Some Facts about Periodicity

We prove the following two lemmas, both stating roughly that *if a string X closely matches a shift of itself, then X is close to periodic.* The first lemma is easy and well-known. The second lemma is new.

Lemma 19 (Self-Alignment Implies Periodicity). *Let X be a string. For any shift $s > 0$, if $X[\,0\,..\,|X| - s\,] = X[\,s\,..\,|X|\,]$ then X is s-periodic (with period $X[\,0\,..\,s\,]$).*

Proof. By assumption we have that $X[j] = X[s + j]$ for each $j \in [\,|X| - s\,]$. It follows that for $P = X[\,0\,..\,s\,]$ we have $X[\,j\,] = P[\,j \bmod s\,]$ for all indices $j \in [\,|X|\,]$ and thus $X = P^*[\,0\,..\,|X|\,]$. □

Lemma 20 (Self-Alignment Implies Small Block Periodicity). *Let X be a string. For any shift $s > 0$, if*

$$\mathrm{ED}(X[\,0\,..\,|X| - s\,], X[\,s\,..\,|X|\,]) < 2s$$

then $\mathrm{BP}_{2s}(X) \le 4s$.

Proof. Let $Y = X[\,0\,..\,|X|-s\,]$, $Z = X[\,s\,..\,|X|\,]$ and let A denote an optimal alignment between Y and Z. We will greedily construct a sequence of indices $0 = i_0 < \cdots < i_L = |Y|$ as follows: Start with $i_0 = 0$. Then, having assigned i_ℓ we next pick the smallest index $i_{\ell+1} > i_\ell$ for which $Y[\,i_\ell\,..\,i_{\ell+1}\,] \ne Z[\,A(i_\ell)\,..\,A(i_{\ell+1})\,]$. Using that A is an optimal alignment, we have constructed a sequence of $L < 2s$ indices, since

$$2s > \mathrm{ED}(Y, Z) = \sum_{\ell=0}^{L-1} \mathrm{ED}(Y[\,i_\ell\,..\,i_{\ell+1}\,], Z[\,A(i_\ell)\,..\,A(i_{\ell+1})\,]) \ge L.$$

Moreover, by Proposition 18 we have that $|i - A(i)| < s$ for all i. The greedy construction guarantees that $Y[\,i_\ell\,..\,i_{\ell+1} - 1\,] = Z[\,A(i_\ell)\,..\,A(i_{\ell+1} - 1)\,]$. Therefore, and since Y, Z are substrings of X, we have $X[\,i_\ell\,..\,i_{\ell+1} - 1\,] = X[\,s + A(i_\ell)\,..\,s + A(i_{\ell+1} - 1)\,]$. We will now apply Lemma 19 to these substrings of X with shift $s' = s + A(i_\ell) - i_\ell$. Note that $0 < s' < 2s$ (which satisfies the precondition of Lemma 19) and thus $X[\,i_\ell\,..\,i_{\ell+1} - 1\,]$ is $2s$-periodic.

Finally, consider the following partition of X into $2L+1$ substrings

$$X = \left(\bigcirc_{\ell=0}^{L-1} X[\,i_\ell\,..\,i_{\ell+1} - 1\,] \circ X[\,i_{\ell+1} - 1\,]\right) \circ X[\,|X| - s\,..\,|X|\,].$$

We claim that these substrings are $2s$-periodic: For $X[\,i_\ell\,..\,i_{\ell+1} - 1\,]$ we have proved this in the previous paragraph, and the strings $X[\,i_{\ell+1} - 1\,]$ and $X[\,|X| - s\,..\,|X|\,]$ have length less than $2s$ and are thus trivially $2s$-periodic. This decomposition certifies that $\mathrm{BP}_{2s}(X) \le 2L + 1 \le 4s$. □

4.2 Edit Distances between Periodic and Random-Like Strings

The goal of this section is to prove the structural Lemmas 12 and 13 which determine the edit distance between certain structured strings.

Lemma 12 (Periodic Rule). *Let Y be a string and write $Y_s = Y[\,K + s\,..\,|Y| - K + s\,]$. If Y is periodic with primitive period P and $|Y| \ge |P|^2 + 2K$, then for all $s, s' \in \{\,-K, \dots, K\,\}$:*

$$\mathrm{ED}(Y_s, Y_{s'}) = 2 \cdot \min_{j \in \mathbb{Z}} \big|\, s - s' + j|P| \,\big|.$$

Proof. For simplicity set $\Delta = 2 \cdot \min_{j\in\mathbb{Z}} \big|s - s' + j|P|\big|$ and $p = |P|$. We will argue that Δ is both an upper bound and lower bound for $\mathrm{ED}(Y_s, Y_{s'})$. The upper bound is simple: Due to the periodicity of Y, we can transform $Y_{s'}$ into Y_s by deleting and inserting $\min_{j\in\mathbb{Z}} \big|\, s - s' + j|P| \,\big|$ many characters. Thus, $\mathrm{ED}(Y_s, Y_{s'}) \le \Delta$.

Next, we prove the lower bound. For contradiction suppose that $\mathrm{ED}(Y_s, Y_{s'}) < \Delta$. We assumed that $|Y_s| = |Y| - 2K \ge p^2$, and we can therefore split Y_s into p parts of length p plus some rest. Let $i_\ell = \ell \cdot p$ for all $i \in \{\,0, \dots, p\,\}$ and let $i_{p+1} = |Y_s|$; we treat $Y_s[\,i_\ell\,..\,i_{\ell+1}\,]$ as the ℓ-th part. Let A denote an optimal alignment between Y_s and $Y_{s'}$; we have

$$\mathrm{ED}(Y_s, Y_{s'}) = \sum_{\ell=0}^{p} \mathrm{ED}(Y_s[\,i_\ell\,..\,i_{\ell+1}\,], Y_{s'}[\,A(i_\ell)\,..\,A(i_{\ell+1})\,]).$$

We assumed that $\mathrm{ED}(Y_s, Y_{s'}) < \Delta \le |P|$ (the latter inequality is by the definition of Δ) and therefore at least one of the first p summands must be zero, say the ℓ-th one, $\ell < p$. It follows that the two strings $Y_s[\,i_\ell\,..\,i_{\ell+1}\,]$ and $Y_{s'}[\,A(i_\ell)\,..\,A(i_{\ell+1})\,]$ are equal. Both are length-p substrings of Y and thus rotations of the global period P. We assumed that P is primitive (i.e., P is not equal to any of its non-trivial rotations) and therefore $s + i_\ell \equiv s' + A(i_\ell) \pmod{p}$. By the definition of Δ we must have that $|A(i_\ell) - i_\ell| \ge \Delta/2$. But this contradicts Proposition 18 which states that $|A(i_\ell) - i_\ell| \le \mathrm{ED}(Y_s, Y_{s'})/2 < \Delta/2$. □

Lemma 13 (Random-Like Rule). *Let Y be a string and write $Y_s = Y[\,K + s\,..\,|Y| - K + s\,]$. If $\mathrm{BP}_{4K}(Y) > 10K$, then for all $s, s' \in \{\,-K, \dots, K\,\}$:*

$$\mathrm{ED}(Y_s, Y_{s'}) = 2 \cdot |s - s'|.$$

Proof. Let $\Delta = 2 \cdot |s - s'|$. First note that $\mathrm{ED}(Y_s, Y_{s'}) \le \Delta$ since we can transform Y_s into $Y_{s'}$ by simply inserting and deleting $|s - s'|$ symbols. For the lower bound, suppose that $\mathrm{ED}(Y_s, Y_{s'}) < \Delta$. Therefore, we can apply Lemma 20 for an appropriate substring of Y: Assume without loss of generality that $s \le s'$ and let $Z = Y[\,K + s\,..\,|Y| - K + s'\,]$. Then clearly $Y_s = Z[\,0\,..\,|Z| - s' + s\,]$ and $Y_{s'} = Z[\,s' - s\,..\,|Z|\,]$, so Lemma 20 applied to Z with shift $s' - s$ yields that $\mathrm{BP}_{2(s'-s)}(Y') \le 4 \cdot (s' - s)$. Using that $s' - s \le 2K$ we conclude that $\mathrm{BP}_{4K}(Y') \le 8K$. We can obtain Y by adding at most K characters to the start and end of Z. It follows that $\mathrm{BP}_{4K}(Y) \le \mathrm{BP}_{4K}(Z) + 2 \le 10K$. This contradicts the assumption in the lemma statement, and therefore $\mathrm{ED}(Y_s, Y_{s'}) \ge \Delta$. □

Lemma 14 (Intermediate). *Let v be a node in the partition tree. Then, in any level in the subtree below v, for all but at most $2\,\mathrm{BP}_{4K}(Y_v)$ nodes w the string Y_w is $4K$-periodic.*

Proof. Focus on some level of the computation subtree below v. We will bound the number of nodes w in this level for which Y_w is not $4K$-periodic. Note that if Y_v was partitioned into $Y_v = \bigcirc_w Y_w$, then by the definition of block periodicity we would immediately conclude that at most $\mathrm{BP}_{4K}(Y_v)$ many parts Y_w are not $4K$-periodic. However, recall that for a node w with associated interval $I_w = [\,i \,..\, j\,]$, we defined Y_w as $Y_w = Y[\,i-K \,..\, j+K\,]$. This means that the substrings Y_w overlap with each other and therefore do *not* partition Y_v.

To deal with this, note that we can assume that $|Y_w| > 4K$, since otherwise Y_w is trivially $4K$-periodic. Hence, each Y_w can overlap with at most two neighboring nodes (since the intervals I_w are disjoint). Therefore, we can divide the w's in two groups such that the Y_w's in each group do not not overlap with each other. For each group, we apply the argument from above to derive that there are at most $\mathrm{BP}_{4K}(Y_v)$ many Y_w's which are not $4K$-periodic. In this way, we conclude that there are at most $2\,\mathrm{BP}_{4K}(Y_v)$ nodes in the level which are not $4K$-periodic, as desired. □

4.3 Some String Property Testers

The main goal of this section is to formally prove Lemmas 15 and 16, that is, the Matching Test and Periodicity Test. As a first step, we need the following simple lemma about testing equality of strings.

Lemma 21 (Equality Test). *Let X, Y be two strings of the same length, and let $r > 0$ be a sampling rate. There is an algorithm which returns of the following two outputs:*

- *Close, in which case $\mathrm{HD}(X, Y) \le 1/r$.*
- *Far(i), in which case $X[\,i\,] \ne Y[\,i\,]$.*

The algorithm runs in time $O(r|X|\log(\delta^{-1}))$ and is correct with probability $1-\delta$.

Proof. The idea is standard: For $r|X|\ln(\delta^{-1})$ many random positions $i \in [\,|X|\,]$, test whether $X[\,i\,] = Y[\,i\,]$. If no error is found, then we report Close. This equality test is clearly sound: If $X = Y$, then it will never fail. It remains to argue that if $\mathrm{HD}(X, Y) > 1/r$ then the test fails with probability at least $1-\delta$. Indeed, each individual sample finds a Hamming error with probability $(r|X|)^{-1}$. Hence, the probability of not finding any Hamming error across all samples is at most

$$\left(1 - \frac{1}{r|X|}\right)^{r|X|\ln(\delta^{-1})} < \exp(-\ln(\delta^{-1})) = \delta.$$

The running time is bounded by $O(r|X|\log(\delta^{-1}))$. □

Lemma 15 (Periodicity Test). *Let X be a string, and let $r > 0$ be a sampling rate. There is an algorithm which returns one of the following two outputs:*

- *Close(P), where P is a primitive string of length $\le K$ with $\mathrm{HD}(X, P^*[\,0 \,..\, |X|\,]) \le 1/r$.*
- *Far, in which case X is not K-periodic.*

The algorithm runs in time $O(r|X|\log(\delta^{-1}) + K)$ and is correct with probability $1-\delta$.

Proof. We start analyzing the length-$2K$ prefix $Y = X[\,0 \,..\, 2K\,]$. In time $O(K)$ we can compute the smallest period P such that $Y = P^*[\,0 \,..\, |Y|\,]$ by searching for the first match of Y in $Y \circ Y$, e.g. using the Knuth-Morris-Pratt pattern matching algorithm [19]. If no such match exists, we can immediately report Far. So suppose that we find a period P. It must be primitive (since it is the smallest such period) and it remains to test whether X globally follows the period. For this task we use the Equality Test (Lemma 21) with inputs X and P^* (of course, we cannot write down the infinite string P^*, but we provide oracle access to P^* which is sufficient here). On the one hand, if X is indeed periodic with period P, then the Equality Test reports Close. On the other hand, if X is $1/r$-far from any periodic string, then it particular $\mathrm{HD}(X, P^*) > 1/r$ and therefore the Equality Test reports Far. The only randomized step is the Equality Test. We therefore set the error probability of the Equality Test to δ and achieve total running time $O(r|X|\log(\delta^{-1}) + K)$. □

Lemma 16 (Matching Test). *Let X, Y be strings such that $|Y| = |X| + 2K$, and let $r > 0$ be a sampling rate. There is an algorithm which returns one of the following two outputs:*

- *Close(s^*), where $s^* \in \{-K, \dots, K\}$ satisfies $\mathrm{HD}(X, Y[\,K+s^* \,..\, |X|+K+s^*\,]) \le 1/r$.*
- *Far, in which case there is no $s^* \in \{-K, \dots, K\}$ with $X = Y[\,K+s^* \,..\, |X|+K+s^*\,]$.*

The algorithm runs time $O(r|X|\log(\delta^{-1}) + K\log|X|)$ and is correct with probability $1-\delta$.

Proof. For convenience, we write $Y_s = Y[\,K+s \,..\, |X|+K+s\,]$. Our goal is to obtain a single *candidate shift* s^* (that is, knowing s^* we can exclude all other shifts from consideration). Having obtained a candidate shift, we can use the Equality Test (Lemma 21 with parameters r and $\delta/3$) to verify whether we indeed have $X = Y_{s^*}$. In the positive case, Lemma 21 implies that $\mathrm{HD}(X, Y_{s^*}) \le 1/r$, hence returning s^* is valid. The difficulty lies in obtaining the candidate shift. Our algorithm proceeds in three steps:

(1) **Aligning the Prefixes:** We start by computing the set S consisting of all shifts s for which $X[\,0 \,..\, 2K\,] = Y_s[\,0 \,..\, 2K\,]$. One way to compute this set in linear time $O(K)$ is by using a pattern matching algorithm with pattern $X[\,0 \,..\, 2K\,]$ and text $Y[\,0 \,..\, 4K\,]$ (like the Knuth-Morris-Pratt algorithm [19]). It is clear that S must contain any shift s for which globally $X = Y_s$. For that reason we can stop if $|S| = 0$ (in which case we return Far) or if $|S| = 1$ (in which case we test the unique candidate shift $s^* \in S$ and report accordingly).

(2) **Testing for Periodicity:** After running the previous step we can assume that $|S| \ge 2$. Take any elements $s < s'$ from S; we have that $X[\,0 \,..\, 2K\,] = Y_s[\,0 \,..\, 2K\,] = Y_{s'}[\,0 \,..\, 2K\,]$. It follows that $X[\,0 \,..\, 2K - s' + s\,] = X[\,s' - s \,..\, 2K\,]$, and thus by Lemma 19 we conclude that $X[\,0 \,..\, 2K\,]$ is periodic with period $P = X[\,0 \,..\, s' - s\,]$, where $|P| \le s' - s \le 2K$. Obviously the same holds for $Y_s[\,0 \,..\, 2K\,]$ and $Y_{s'}[\,0 \,..\, 2K\,]$.
We will now test whether X and Y_s are also globally periodic with this period P. To this end, we apply the Equality Test two times (each time with parameters $2r$ and $\delta/3$) to check whether $X = P^*[\,0 \,..\, |X|\,]$ and $Y_s = P^*[\,0 \,..\, |Y_s|\,]$. If both tests return Close, then we have $\mathrm{HD}(X, P^*[\,0 \,..\, |X|\,]) \le 1/(2r)$ and $\mathrm{HD}(Y_s, P^*[\,0 \,..\, |Y_s|\,]) \le 1/(2r)$ by Lemma 21 and

hence, by the triangle inequality, $\mathrm{HD}(X, Y_s) \le 1/r$. Note that we have witnessed a matching shift $s^* = s$.

(3) **Aligning the Leading Mismatches:** Assuming that the previous step did not succeed, one of the Equality Tests returned $\textsc{Far}(i_0)$ for some position $i_0 > 2K$ with $X[\,i_0\,] \neq P^*[\,i_0\,]$ or $Y_s[\,i_0\,] \neq P^*[\,i_0\,]$. Let us refer to these indices as *mismatches.* Moreover, we call a mismatch i a *leading mismatch* if the $2K$ positions to the left of i are not mismatches. We continue in two steps: First, we find a leading mismatch. Second, we turn this leading mismatch into a candidate shift.

(a) **Finding a Leading Mismatch:** To find a leading mismatch, we use the following binary search-style algorithm: Initialize $L \leftarrow 0$ and $R \leftarrow i_0$. We maintain the following two invariants: (i) All positions in $[\,L\,..\,L+2K\,]$ are not mismatches, and (ii) R is a mismatch. Both properties are initially true. We will now iterate as follows: Let $M \leftarrow \lceil (L+R)/2 \rceil$ and test whether there is a mismatch $i \in [\,M\,..\,M+2K\,]$. If there is such a mismatch i, we update $R \leftarrow i$. Otherwise, we update $L \leftarrow M$. It is easy to see that in both cases both invariants are maintained. Moreover, this procedure is guaranteed to make progress as long as $L + 4K < R$. If at some point $R \le L + 4K$, then we can simply check all positions in $[\,L\,..\,R\,]$—one of these positions must be a leading mismatch i.

(b) **Finding a Candidate Shift:** Assume that the previous step succeeded in finding a leading mismatch i. Then we can produce a single candidate shift as follows: Assume without loss of generality that $X[\,i\,] \neq P^*[\,i\,]$, and let $i \le j$ be the smallest position such that $Y_s[\,j\,] \neq P^*[\,j\,]$. Then $s^* = s + j - i$ is the only candidate shift (if it happens to fall into the range $\{-K, \dots, K\}$).
Indeed, for any $s'' > s^*$ we can find a position where X and $Y_{s''}$ differ. To see this, we should assume that s'' respects the period (i.e., $P^* = P^*[\,K + s''\,..\,\infty\,]$), since otherwise we find a mismatch in the length-$2K$ prefix. But then

$$\begin{aligned} Y_{s''}[\,j + s - s''\,] &= Y_s[\,j\,] && (4)\\ &\neq P^*[\,j\,] && (5)\\ &= P^*[\,j + s - s''\,] && (6)\\ &= X[\,j + s - s''\,], && (7)\end{aligned}$$

which proves that $X \neq Y_{s''}$ and thereby disqualifies s'' as a feasible shift. Here we used (4) the definition of Y_s, (5) the assumption that $Y_s[\,j\,] \neq P^*[\,j\,]$, (6) the fact that both s and s'' respect the period P and (7) the assumption that i was a leading mismatch which implies that X matches P^* at the position $j + s - s'' < i$.
A similar argument works for any shift $s'' < s^*$. In this case one can show that $X[\,i\,] \neq P^*[\,i\,] = Y_{s''}[\,i\,]$ which also disqualifies s'' as a candidate shift.

We finally bound the error probability and running time of this algorithm. We only use randomness when calling the Equality Test which runs at most three times. Since each time we set the error parameter to $\delta/3$, the total error probability is δ as claimed. The running time of the Equality Tests is bounded by $O(r|X|\log(\delta^{-1}))$ by Lemma 21. In addition, steps 1 and 2 take time $O(K)$. Step 3 iterates at most $\log|X|$ times and each iteration takes time $O(K)$. Thus, the total running time is $O(r|X|\log(\delta^{-1}) + K\log|X|)$. □

4.4 Putting the Pieces Together

Setting the Parameters. Throughout this section we assume that T is a balanced B-ary partition tree with $\min(n, K^{100})$ leaves, where each leaf v is labeled with an interval I_v of length $|I_v| \approx \max(1, \frac{n}{K^{100}})$. In particular there are at most $2 \cdot K^{100}$ nodes in the tree and its depth is bounded by $\lceil \log_B \min\left(n, K^{100}\right) \rceil$. We also specify the following parameters for every node v in the partition tree:

- Rate r_v: If v is the root then we set $r_v = 1000/K$. Otherwise, if v is a child of w then sample $u_v \sim \mathcal{D}((200\log K)^{-1}, K^{-200})$ independently and set $r_v = r_w/u_v$. (This assignment matches the values in Algorithm 2.)
- Multiplicative accuracy $\alpha_v = 10 \cdot (1 - (200\log K)^{-1})^d$ (where d is the depth of v). Note that $\alpha_v \ge 5$, since $d \le \log(K^{100})$.

For these parameters our goal is to compute $\Delta_{v,-K}, \dots, \Delta_{v,K}$ in the sense of Definition 8.

Correctness. We start with the correctness proof.

Lemma 22 (Correctness of Algorithm 2). *Let X, Y be strings. Given any node v in the partition tree, Algorithm 2 correctly solves the Tree Distance Problem.*

Proof. The analysis of Lines 10 to 16 (that is, combining the recursive computations) is precisely as in the analysis of the unmodified Andoni-Krauthgamer-Onak algorithm (see the full version). We therefore omit the details an assume that these steps succeed. In this proof we show that Lines 1 to 9 are correct as well. (We postpone the error analysis to the end of the proof.) There are three possible cases:

- **The Strings are Short:** First assume that $|X_v| \le 100K^2$ or that v is a leaf node, in which case the condition in Line 1 triggers. The algorithm computes and returns a multiplicative 2-approximation $\Delta_{v,s}$ of $\mathrm{ED}(X_v, Y_{v,s})$ for all shifts s using Theorem 17. We need to justify that $2 \le \alpha_v$ so that $\Delta_{v,s}$ is a valid approximation in the sense of Equation (3). Indeed, using the parameter setting in the previous paragraph we have $\alpha_v \ge 5$.

If the algorithm does not terminate in this first case, we may assume that $|X_v| \ge 100K^2$. The algorithm continues running and applies the Matching Test (Lemma 16) to X_v, Y_v and the $4K$-Periodicity Test (Lemma 15) to Y_v, both with rate parameter $r = 3r_v$ and error parameter $\delta = 0.01 \cdot K^{-100}$. We again postpone the error analysis and assume that both tests returned a correct answer. We continue analyzing the remaining two cases:

- **The Strings are Periodic:** Assume that the Matching Test reports $\textsc{Close}(s^*)$ and that the $4K$-Periodicity Test reports $\textsc{Close}(P)$, where P is a primitive string with length $|P| \le 4K$. The algorithm returns $\Delta_{v,s} = 2 \cdot \min_{j \in \mathbb{Z}} \left| s - s^* - j|P| \right|$ in Line 7. We argue that this approximation is valid using Lemma 12 and by applying the triangle inequality three times. To this end we define $Z = P^*[\,0\,..\,|Y_v|\,]$ (that is, Z is equal to Y_v after "correcting" the periodicity errors) and $Z_s = Z[\,K + s\,..\,|Z| -$

$K + s$]. By Lemma 12 we have that

$$\mathrm{ED}(Z_{s^*}, Z_s) = 2 \cdot \min_{j \in \mathbf{Z}} \left| s - s^* - j|P| \right| = \Delta_{v,s},$$

for all shifts s. Here we use the assumption that $|X_v| \geq 100K^2 \geq |P|^2$ and its consequence $|Z| = |Y_v| \geq |P|^2 + 2K$ to satisfy the precondition of Lemma 12. Using that the Periodicity Test reported $\textsc{Close}(P)$, we infer that $\mathrm{HD}(Y_{v,s}, Z_s) \leq \frac{1}{3r_v}$ for all shifts s, and using that the Matching Test reported $\textsc{Close}(s^*)$ we obtain $\mathrm{HD}(X_v, Y_{v,s^*}) \leq 1/(3r_v)$. By applying the triangle inequality three times we conclude that

$$\begin{aligned}\Delta_{v,s} &= \mathrm{ED}(Z_{s^*}, Z_s) \\ &\leq \mathrm{ED}(Z_{s^*}, Y_{v,s^*}) + \mathrm{ED}(Y_{v,s^*}, X_v) \\ &\quad + \mathrm{ED}(X_v, Y_{v,s}) + \mathrm{ED}(Y_{v,s}, Z_s) \\ &\leq \mathrm{ED}(X_v, Y_{v,s}) + 1/r_v,\end{aligned}$$

and similarly $\Delta_{v,s} \geq \mathrm{ED}(X_v, Y_{v,s}) - 1/r_v$. It follows that $\Delta_{v,s}$ is an additive $1/r_v$-approximation of $\mathrm{ED}(X_v, Y_{v,s})$, as required in Equation (3). (Here, we do not suffer any multiplicative error.)

- **The Strings are Random-Like:** Finally assume that the Matching Test reports $\textsc{Close}(s^*)$, but the $4K$-Periodicity Test reports $\textsc{Far}$. In this case the algorithm reaches Line 9 and continues with the recursive computation (in Line 10). However, if at any level in the computation subtree rooted at v there are more than $20K$ active nodes for which the Periodicity Test (in Line 4) reports $\textsc{Far}$, then the recursive computation is interrupted and we return $\Delta_{v,s} = 2 \cdot |s - s^*|$. We have already argued that the unrestricted recursive computation is correct, but it remains to justify why interrupting the computation makes sense.
  So suppose that the recursive computation is interrupted, i.e., assume that there are more than $20K$ descendants w of v at some level for which the Periodicity Test reported $\textsc{Far}$. Assuming that all Periodicity Tests computed correct outputs, we conclude that for all these descendants w the strings Y_v are not $4K$-periodic. From Lemma 14 we learn that necessarily $\mathrm{BP}_{4K}(Y_v) > 10K$. Hence Lemma 13 applies and yields that
  $$\mathrm{ED}(Y_{v,s^*}, Y_{v,s}) = 2 \cdot |s - s^*| = \Delta_{v,s}.$$
  Using again the triangle inequality and the assumption that $\mathrm{ED}(Y_{v,s^*}, X_v) \leq 1/r_v$ (by the Matching Test), we derive that
  $$\begin{aligned}\Delta_{v,s} &= \mathrm{ED}(Y_{v,s^*}, Y_{v,s}) \\ &\leq \mathrm{ED}(Y_{v,s^*}, X_v) + \mathrm{ED}(X_v, Y_{v,s}) \\ &\leq \mathrm{ED}(X_v, Y_{v,s}) + 1/r_v.\end{aligned}$$
  The lower bound can be proved similarly and therefore $\Delta_{v,s}$ is an additive $1/r_v$-approximation of $\mathrm{ED}(X_v, Y_{v,s})$.

We finally analyze the error probability of Algorithm 2. There are three sources of randomness in the algorithm: The Matching and Periodicity Tests in Lines 3 and 4 and the application of the Precision Sampling Lemma. For each node, therefore have three error events: With probability at most $2\delta = 0.02 \cdot K^{-100}$ one of the property tests fails. We apply the Precision Sampling Lemma with $\delta = 0.01 \cdot K^{-101}$ for $2K$ shifts in every node, hence the error probability is bounded by $0.02 \cdot K^{-100}$ as well. In summary: The error probability per node is $0.04 \cdot K^{100}$. Recall that there are at most $2K^{100}$ nodes in the partition tree, and thus the total error probability is bounded by $0.08 \leq 0.1$. ◻

Running Time. This concludes the correctness part of the analysis and we continue bounding the running time of Algorithm 2. We proceed in two steps: First, we give an upper bound on the number of active nodes in the partition tree (see Lemmas 23 and 24). Second, we bound the expected running time of a single execution of Algorithm 2 (ignoring the cost of recursive calls). The expected running time is bounded by their product.

Recall that a node v is *matched* if there is some $s \in \{-K, \dots, K\}$ such that $X_v = Y_{v,s}$. Moreover, we say that v is *active* if the recursive computation of Algorithm 2 reaches v.

Lemma 23 (Number of Unmatched Nodes). *If* $\mathrm{ED}(X, Y) \leq K$ *and the partition tree has depth* D*, then there are at most* KD *nodes which are not matched.*

Proof. Focus on any level in the partition tree and let $0 = i_0 < \cdots < i_w = n$ denote the partition induced by that level, i.e., let $[\, i_\ell \,..\, i_{\ell+1} \,] = I_v$ where v is the ℓ-th node in the level (from left to right). Let A be an optimal alignment between X and Y, then:

$$\mathrm{ED}(X, Y) = \sum_{\ell=0}^{w-1} \mathrm{ED}(X[\, i_\ell \,..\, i_{\ell+1} \,], Y[\, A(i_\ell) \,..\, A(i_{\ell+1}) \,]).$$

Since we assumed that $\mathrm{ED}(X, Y) \leq K$, there can be at most K nonzero terms in the sum. For any zero term we have that

$$X[\, i_\ell \,..\, i_{\ell+1} \,] = Y[\, A(i_\ell) \,..\, A(i_{\ell+1}) \,],$$

and therefore the ℓ-th node in the current level is matched with shift $A(i_\ell) - i_\ell$. By Proposition 18 we have that $|A(i_\ell) - i_\ell| \leq \mathrm{ED}(X, Y) \leq K$. This completes the proof. ◻

Lemma 24 (Number of Active Nodes). *Assume that* $\mathrm{ED}(X, Y) \leq K$*. If the partition tree has depth* D*, then there are at most* $O((KDB)^2)$ *active nodes, with probability* 0.98.

Proof. Recall that (unconditionally) there are at most $2K^{100}$ nodes in the partition tree. Hence, by a union bound, all Matching Tests in Line 3 succeed with probability at least $1 - 0.02 = 0.98$. We will condition on this event throughout the proof. We distinguish between three kinds of nodes v:

(1) v itself and all of v's ancestors are not matched,
(2) v itself is matched, but all of v's ancestors are not matched,
(3) some ancestor of v (and therefore also v itself) is matched.

By the previous lemma we know that there are at most KD nodes which are not matched. It follows that there are at most KD nodes of the first kind.

It is also easy to bound the number of nodes v of the second kind: Observe that v's parent is a node of the first kind. Hence, there can be at most $KD \cdot B$ nodes of the second kind.

Finally, we bound the number of nodes of the third kind. Any such node v has a unique ancestor w of the second kind. There are two cases for w: Either the condition in Line 6 succeeds and the algorithm directly solves w. This is a contradiction since we assumed that v (a descendant of w) is active. Or this condition fails, and the algorithm continues branching with the exception that if in the subtree below w there are more than $20K$ active nodes per level for which the Periodicity Test in Line 4 reports Far, then we interrupt the recursive computation. We claim that consequently in the subtree below w (consisting only of nodes of the third kind), there are at most $20KB$ active nodes per level. Indeed, suppose there were more than $20KB$ active nodes on some level. Then consider their parent nodes; there must be more than $20K$ parents. For each such parent u the Matching Test reported Close (since u is a matched node, and we assumed that all Matching Tests succeed) and the Periodicity Test reported Far (since otherwise the condition in Line 4 triggers and solves u directly, but we assumed that u is the parent of some other active node). Note that we have witnessed more than $20K$ nodes on one level below w for which the Periodicity Test reported Far. This is a contradiction.

In total the number of active nodes below w is bounded by $20KB \cdot D$. Recall that there are at most $KD \cdot B$ nodes w of the second kind, hence the total number of active nodes of the third kind is $20(KDB)^2$. Summing over all three kinds, we obtain the claimed bound. □

We are ready to bound the total running time. In the following lemma we prove that the algorithm is efficient *assuming that the edit distance between X and Y is small*. This assumption can be justified by applying Algorithm 2 with the following modification: We run Algorithm 2 with a time budget and interrupt the computation as soon as the budget is depleted. In this case we can immediately infer that the edit distance between X and Y must be large.

Lemma 25 (Running Time of Algorithm 2). *Let X, Y be strings with $\mathrm{ED}(X,Y) \le K$. Then Algorithm 2 runs in time*

$$n/K \cdot (\log K)^{O(\log_B(K))} + \widetilde{O}(K^4B^2),$$

with constant probability 0.9.

Proof. We first bound the expected running time of a single execution of Algorithm 2 where we ignore the cost of recursive calls. Let D denote the depth of the partition tree. We proceed in the order of the pseudocode:

- Lines 1 and 2: If the test in Line 1 succeeds, then Line 2 takes time $O(|X_v| + K^2)$ by Theorem 17, where $|X_v| \le 100K^2$ or $|X_v| \le O(n/K^{100})$. The total time of this step is therefore bounded by $O(K^2 + n/K^{100})$.
- Lines 3 and 4: Running the Matching and Periodicity Tests (Lemmas 15 and 16) takes time $O(r_v|X_v| \log K + K \log |X_v|)$. We want to replace the $\log |X|$ by $\log K$, so assume that the second summand dominates, i.e., $r_v|X_v| \log K \le K \log |X_v|$. At any node v the rate r_v is always at least $1000/K$ (since at the root the rate is exactly $1000/K$ and below the root the rate never decreases), hence $|X_v|/\log |X_v| \le \mathrm{poly}(K)$. It follows that we can bound the total time of this step indeed by $O(r_v|X_v| \log K + K \log K)$.
- Lines 5 to 9: Here we merely produce the output according to some fixed rules. The time of this step is bounded by $O(K)$.
- Lines 10 to 16: This step takes time $O(KB)$ and the analysis is exactly as in the analysis of the Andoni-Krauthgamer-Onak algorithm, see the full version for details.

In total, the time of a single execution is bounded by $O(r_v|X_v| \log K + K^2 + n/K^{100})$. We will simplify this term by plugging in the (expected) rate r_v for any node v (we defer the detailed analysis to the full version of this paper).

Recall that $r_v = 1000 \cdot (K \cdot u_{v_1} \dots u_{v_d})^{-1}$ where $v_0, v_1, \dots, v_d = v$ is the root-to-node path leading to v and each u_i is sampled from $\mathcal{D}(\varepsilon = (200 \log K)^{-1}, \delta = 0.01 \cdot K^{-101})$, independently. Using Lemma 9 there exist events E_w happening each with probability $1 - 1/N$ such that

$$\mathbf{E}(1/u_w \mid E_w) \le \widetilde{O}(\varepsilon^{-2} \log(\delta^{-1}) \log N) \le \mathrm{polylog}(K).$$

In the last step we set $N = 100K^{100}$. Taking a union bound over all active nodes w (there are at most $2K^{100}$ many), the event $E = \bigwedge_w E_w$ happens with probability at least 0.98 and we will condition on E from now on. Under this condition we have:

$$\begin{aligned}\mathbf{E}(r_v \mid E) &= \frac{1000}{K} \prod_{i=1}^{d} \mathbf{E}(1/u_{v_i} \mid E_{v_i}) \\ &\le \frac{(\log K)^{O(d)}}{K} \le \frac{(\log K)^{O(\log_B(K))}}{K}.\end{aligned}$$

Finally, we can bound the total expected running time (conditioned on E) as follows, summing over all active nodes v:

$$\sum_v O\left(|X_v| \cdot \frac{(\log K)^{O(\log_B(K))}}{k} + K^2 + \frac{n}{K^{100}}\right).$$

Using that $\sum_w |X_w| = n$ whenever w ranges over all nodes on a fixed level in the partition tree, and thus $\sum_v |X_v| \le n \cdot D$ where v ranges over all nodes, we can bound the first term in the sum by $n/K \cdot (\log K)^{O(\log_B(K))}$. The second term can be bounded by K^2 times the number of active nodes. By Lemma 24 this becomes $O(K^4D^2B^2) = \widetilde{O}(K^4B^2)$. By the same argument the third term becomes at most n/K^{90} and is therefore negligible.

We conditioned on two events: The event E and the event that the number of active nodes is bounded by $O(K^2D^2B^2)$ (Lemma 24). Both happen with probability at least 0.98, thus the total success probability is $0.96 \ge 0.9$. □

4.5 Main Theorem

We finally recap and formally prove our main theorem and its two corollaries.

Theorem 1 (Main Theorem). *Let $2 \le B \le k$ be a parameter. The $(k, \Theta(k \log_B(k) \cdot B))$-gap edit distance problem can be solved in time*

$$\frac{n}{k} \cdot (\log k)^{O(\log_B(k))} + \widetilde{O}(k^4 \,\mathrm{poly}(B)).$$

Proof. For now we keep K as a parameter and will later set K in terms of k. We run Algorithm 2 to compute an approximation $\Delta = \Delta_{r,0}$ where r is the root node in the partition tree. By the correctness of Algorithm 2 (Lemma 22) we have that $0.1\,\mathrm{ED}^{\le K}(X,Y) - 0.001K \le \Delta \le 10\,\mathrm{TD}^{\le K}(X,Y) + 0.001K$, and using the equivalence of edit distance and tree distance (Lemma 6) we conclude that $\Delta \le 20BD \cdot \mathrm{ED}^{\le K}(X,Y) + 0.001K$, where $D \le \log_B((K)^{100})$ is the depth of the partition tree. It follows that we can distinguish whether the edit distance $\mathrm{ED}(X,Y)$ is at most $K/(1000BD)$ or at least K. Indeed:

- If $\mathrm{ED}(X,Y) \le K/(1000BD)$, then $\Delta \le 0.02K + 0.001K = 0.021K$.

- If $\mathrm{ED}(X, Y) \geq K$, then $\Delta \geq 0.1K - 0.001K = 0.099K$.

To bound the running time, we run the previous algorithm with time budget $n/K \cdot (\log K)^{O(\log_B(K))} + \widetilde{O}(K^4 B^2)$ (with the same constants as in Lemma 25). If the algorithm exceeds the time budget, then we interrupt the computation and report that $\mathrm{ED}(X, Y) \geq K$. This is indeed valid, since Lemma 25 certifies that $\mathrm{ED}(X, Y) > K$ in this case.

To obtain the claimed statement, we have to pick K. We set $K = \Theta(k \log_B(k) \cdot B)$, where the constant is picked in such a way that $K/(1000BD) \geq K/(1000B \cdot \log_B((K)^{100})) \geq k$. Then the algorithm distinguishes edit distances k versus $K = \Theta(k \log_B(k) \cdot B)$. □

Corollary 2 (Subpolynomial Gap). *In time $O(n/k + k^{4+o(1)})$ we can solve $(k, k \cdot 2^{\widetilde{O}(\sqrt{\log k})})$-gap edit distance.*

Proof. Simply plugging $B = 2^{\sqrt{\log k}}$ into Theorem 1 leads to the correct gap, but we suffer a factor $k^{o(1)}$ in the running time. For that reason, let $\bar{k}$ be a parameter to be specified later and apply Theorem 1 with parameter $\bar{k}$ and $B = 2^{\sqrt{\log \bar{k}}}$. In that way we can distinguish the gap $\bar{k}$ versus $\bar{k} \cdot 2^{\Theta(\sqrt{\log \bar{k}})}$ in time

$$O(n/\bar{k} \cdot (\log \bar{k})^{O(\log_B(\bar{k}))} + \bar{k}^{4+o(1)}).$$

This term can be written as $O(n/\bar{k} \cdot 2^{\alpha(\bar{k})} + \bar{k}^{4+o(1)})$, where $\alpha(\bar{k}) = O\left(\sqrt{\log \bar{k}} \log \log \bar{k}\right)$.

Finally, set $\bar{k} = k \cdot 2^{2\alpha(k)}$. For k at least a sufficiently large constant we have that $2^{\alpha(\bar{k})} \leq 2^{2\alpha(k)}$, and therefore the running time becomes $O(n/k + k^{4+o(1)})$ as claimed. (For small constant k, we can exactly compute the k-capped edit distance in linear time $O(n)$ using the Landau-Vishkin algorithm [23].) The algorithm distinguishes the gap $\bar{k}$ versus $\bar{k} \cdot 2^{O(\sqrt{\log k})} = k \cdot 2^{\widetilde{\Theta}(\sqrt{\log k})}$. Since $k \leq \bar{k}$, this is sufficient to prove the claim. □

Corollary 3 (Polylogarithmic Gap). *For any $\varepsilon \in (0, 1)$, in time $O(n/k^{1-\varepsilon} + k^{4+o(1)})$ we can solve $(k, k \cdot (\log k)^{O(1/\varepsilon)})$-gap edit distance.*

Proof. Let c be the constant so that the time bound in Theorem 1 becomes $n/k \cdot (\log k)^{c \log_B(k)} + \widetilde{O}(k^4 \operatorname{poly}(B))$. We apply Theorem 1 with parameter $B = (\log k)^{c/\varepsilon}$. Then the gap is indeed k versus $k \cdot (\log k)^{O(1/\varepsilon)}$ as claimed. Since $(\log k)^{c \log_B(k)} = k^{\varepsilon}$ the running time bound becomes $O(n/k^{1-\varepsilon} + k^{4+o(1)})$. □

REFERENCES

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. 2015. Tight Hardness Results for LCS and Other Sequence Similarity Measures. In *Proceedings of the 56th IEEE Annual Symposium on Foundations of Computer Science (FOCS '15)*. IEEE Computer Society, 59–78.

[2] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. 2016. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC '16)*. ACM, 375–388.

[3] Alexandr Andoni. 2017. High frequency moments via max-stability. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017, New Orleans, LA, USA, March 5-9, 2017*. IEEE, 6364–6368.

[4] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. 2010. Polylogarithmic Approximation for Edit Distance and the Asymmetric Query Complexity. In *Proceedings of the 51st IEEE Annual Symposium on Foundations of Computer Science (FOCS '10)*. IEEE Computer Society, 377–386.

[5] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. 2011. Streaming Algorithms via Precision Sampling. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science (FOCS '11)*. IEEE Computer Society, 363–372.

[6] Alexandr Andoni and Huy L. Nguyen. 2010. Near-Optimal Sublinear Time Algorithms for Ulam Distance. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA '10)*. SIAM, 76–86.

[7] Alexandr Andoni and Negev Shekel Nosatzki. 2020. Edit Distance in Near-Linear Time: it's a Constant Factor. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS '20)*. IEEE Computer Society, 990–1001.

[8] Alexandr Andoni and Krzysztof Onak. 2012. Approximating Edit Distance in Near-Linear Time. *SIAM J. Comput.* 41, 6 (2012), 1635–1648.

[9] Arturs Backurs and Piotr Indyk. 2015. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the 47th ACM Symposium on Theory of Computing (STOC '15)*. ACM, 51–58.

[10] Ziv Bar-Yossef, S. Thathachar Jayram, Robert Krauthgamer, and Ravi Kumar. 2004. Approximating Edit Distance Efficiently. In *Proceedings of the 45th IEEE Annual Symposium on Foundations of Computer Science (FOCS '04)*. IEEE Computer Society, 550–559.

[11] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. 2003. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC '03)*. ACM, 316–324.

[12] Tugkan Batu, Funda Ergun, and Cenk Sahinalp. 2006. Oblivious string embeddings and edit distance approximations. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA '06)*. SIAM, 792–801.

[13] Joshua Brakensiek and Aviad Rubinstein. 2020. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd ACM Symposium on Theory of Computing (STOC '20)*. ACM, 685–698.

[14] Karl Bringmann and Marvin Künnemann. 2015. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *Proceedings of the 56th IEEE Annual Symposium on Foundations of Computer Science (FOCS '15)*. IEEE Computer Society, 79–97.

[15] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael Saks. 2020. Approximating Edit Distance Within Constant Factor in Truly Sub-Quadratic Time. *J. ACM* 67, 6 (2020), 22 pages.

[16] Elazar Goldenberg, Tomasz Kociumaka, Robert Krauthgamer, and Barna Saha. 2021. Gap Edit Distance via Non-Adaptive Queries: Simple and Optimal. *CoRR* abs/2111.12706 (2021).

[17] Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. 2019. Sublinear Algorithms for Gap Edit Distance. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS '20)*. IEEE Computer Society, 1101–1120.

[18] Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press.

[19] Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. 1977. Fast Pattern Matching in Strings. *SIAM J. Comput.* 6, 2 (1977), 323–350.

[20] Tomasz Kociumaka and Barna Saha. 2020. Sublinear-Time Algorithms for Computing & Embedding Gap Edit Distance. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS '20)*. IEEE Computer Society, 1168–1179.

[21] Michal Koucký and Michael E. Saks. 2020. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd ACM Symposium on Theory of Computing (STOC '20)*. ACM, 699–712.

[22] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. 1998. Incremental String Comparison. *SIAM J. Comput.* 27, 2 (1998), 557–582.

[23] Gad M. Landau and Uzi Vishkin. 1988. Fast String Matching with k Differences. *J. Comput. Syst. Sci.* 37, 1 (1988), 63–78.

[24] Vladimir Iosifovich Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady* 10, 8 (1966), 707–710.

[25] Rafail Ostrovsky and Yuval Rabani. 2007. Low distortion embeddings for edit distance. *J. ACM* 54, 5 (2007), 23.

[26] T. K. Vintsyuk. 1968. Speech discrimination by dynamic programming. *Cybernetics* 4, 1 (1968), 52–57. Russian Kibernetika 4(1):81-88 (1968).

[27] Robert A. Wagner and Michael J. Fischer. 1974. The String-to-String Correction Problem. *J. ACM* 21, 1 (1974), 168–173.