Improved Sublinear-Time Edit Distance for Preprocessed Strings

Karl Bringmann
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Alejandro Cassis
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Nick Fischer
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Vasileios Nakos
RelationalAI, Berkeley, CA, USA

Abstract

We study the problem of approximating the edit distance of two strings in sublinear time, in a setting where one or both string(s) are preprocessed, as initiated by Goldenberg, Rubinstein, Saha (STOC '20). Specifically, in the (k, K)-*gap edit distance* problem, the goal is to distinguish whether the edit distance of two strings is at most k or at least K. We obtain the following results:

- After preprocessing *one* string in time $n^{1+o(1)}$, we can solve $(k, k \cdot n^{o(1)})$-gap edit distance in time $(n/k + k) \cdot n^{o(1)}$.
- After preprocessing *both* strings separately in time $n^{1+o(1)}$, we can solve $(k, k \cdot n^{o(1)})$-gap edit distance in time $kn^{o(1)}$.

Both results improve upon some previously best known result, with respect to either the gap or the query time or the preprocessing time.

Our algorithms build on the framework by Andoni, Krauthgamer and Onak (FOCS '10) and the recent sublinear-time algorithm by Bringmann, Cassis, Fischer and Nakos (STOC '22). We replace many complicated parts in their algorithm by faster and simpler solutions which exploit the preprocessing.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Edit Distance, Property Testing, Preprocessing, Precision Sampling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.32

Category Track A: Algorithms, Complexity and Games

Funding 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).

1 Introduction

The *edit distance* (also known as *Levenshtein distance*) is a fundamental measure of similarity between strings. It has numerous applications in several fields such as information retrieval, computational biology and text processing. Given strings X and Y, their edit distance denoted by $\mathsf{ED}(X, Y)$ is defined as the minimum number of character insertions, deletions and substitutions needed to transform X into Y.

Table 1 A comparison of sublinear-time algorithms for the $(k, k \cdot g)$-gap edit distance problem for different gap parameters g. All algorithms in this table are randomized and succeed with high probability. Note that some of these results are subsumed by others.

Source	Gap g	Preprocessing time	Query time
Goldenberg, Krauthgamer, Saha [20]	$O(k)$	no preprocessing	$\widetilde{O}(n/k + k^3)$
Kociumaka, Saha [24]	$O(k)$	no preprocessing	$\widetilde{O}(n/k + k^2)$
Brakensiek, Charikar, Rubinstein [13]	$O(k)$	no preprocessing	$\widetilde{O}(n/\sqrt{k})$
Bringmann, Cassis, Fischer, Nakos [15]	$O(k)$	no preprocessing	$O^*(n/k^2+k^8)$
Goldenberg, Kociumaka, Krauthgamer, Saha [19]	$O(k)$	no preprocessing	$\widetilde{O}(n/k^{3/2})$
Bringmann, Cassis, Fischer, Nakos [15]	$O^*(1)$	no preprocessing	$O^*(n/k + k^4)$
Goldenberg, Rubinstein, Saha [21]	$O(k)$	one-sided, $\widetilde{O}(n)$	$\widetilde{O}(n/k + k^2)$
Brakensiek, Charikar, Rubinstein [13]	g	one-sided, $\widetilde{O}(nk/g)$	$\widetilde{O}(n/g+k^2/g)$
This work, Theorem 1	$O^*(1)$	one-sided, $O^*(n)$	$O^*(n/k + k)$
Chakraborty, Goldenberg, Koucký [18]	$O(k)$	two-sided, $\widetilde{O}(n)$	$O(\log n)$
Brakensiek, Charikar, Rubinstein [13]	g	two-sided, $\widetilde{O}(nk/g)$	$\widetilde{O}(k^2/g)$
Ostrovsky, Rabani [26]	$O^*(1)$	two-sided, $\widetilde{O}(n^2)$	$O(\log n)$
This work, Theorem 2	$O^*(1)$	two-sided, $O^*(n)$	$O^*(k)$
Goldenberg, Rubinstein, Saha [21]	$O(1)$	two-sided, $\widetilde{O}(n^2)$	$O^*(n^{3/2})$
Goldenberg, Rubinstein, Saha [21]	1	two-sided, $\widetilde{O}(n)$	$\widetilde{O}(k^2)$

A textbook dynamic programming algorithm computes the edit distance of two strings of length n in time $O(n^2)$. Popular conjectures such as the *Strong Exponential Time Hypothesis* imply that that this algorithm is essentially optimal, as there is no strongly subquadratic-time algorithm [8, 1, 16, 2]. As for some applications involving enormous strings (such as DNA sequences) quadratic-time algorithms are impractical, a long line of research developed progressively better and faster *approximation* algorithms [9, 11, 26, 7, 4, 17, 25, 14]. The current best approximation guarantee in near-linear time is an algorithm by Andoni and Nosatzki [6] computing an $f(1/\varepsilon)$-approximation in time $O(n^{1+\varepsilon})$.

Another more recent line of research studies edit distance in the *sublinear-time* setting. Here the goal is to approximate the edit distance without reading the entire input strings. More formally, in the (k, K)-gap edit distance problem the goal is to distinguish whether the edit distance between X and Y is at most k or greater than K. The performance of gap algorithms is typically measured in terms of the string length n and the gap parameters k and K. This problem has been studied in several works [10, 7, 20, 13, 24] most of which focus on the (k, k^2)-gap problem. Currently, there are two incomparable best known results: A recent result by Goldenberg, Kociumaka, Krauthgamer and Saha [19] established a non-adaptive algorithm for the (k, k^2)-gap problem in time $\widetilde{O}(n/k^{3/2})$.[1] Another recent result by Bringmann, Cassis, Fischer and Nakos [15] reduces the gap to $O^*(1)$ and solves the $(k, O^*(k))$-gap problem in time $O^*(n/k+k^4)$.[2] See Table 1 for a more detailed comparison.

Our starting point is the work by Goldenberg, Rubinstein and Saha [21] which studies sublinear algorithms for edit distance in the *preprocessing model*. Here, we are allowed to preprocess one or both input strings X and Y *separately*, and then use the precomputed

[1] We write $\widetilde{O}(\cdot)$ to hide polylogarithmic factors $(\log n)^{O(1)}$.
[2] We write $O^*(\cdot)$ to hide subpolynomial factors $n^{o(1)}$ in n.

information to solve the (k, K)-gap edit distance problem. This model is motivated by applications where many long strings are compared against each other. For example, the *string similarity join* problem is to find all pairs of strings in a database (containing e.g. DNA sequences) which are close in edit distance; see [27] for a survey on practically relevant algorithms. Note that in these applications, if we have an algorithm with almost-linear preprocessing time (which is the case for all the algorithms we present in this paper), then the overhead incurred by preprocessing is comparable to the time necessary to read and store the strings in the first place. In [21], the authors pose and investigate the following open question:

> *"What is the complexity of approximate edit distance with preprocessing when $k \ll n$?" [21]*

This question has spawned significant interest in the community [18, 21, 13], and with this paper we also make progress towards this question. We give an overview of results in Table 1. Note that most results are hard to compare to each other (one-sided versus two-sided preprocessing, exact versus $O(1)$-approximate versus $O(k)$-approximate).

In the two-sided model, all known algorithms (with almost-linear preprocessing time[3] and, say, subpolynomial gap $g = n^{o(1)}$) share the common barrier that the query time is $\Omega(k^2)$. Due to this barrier, Goldenberg et al. [21] specifically ask whether there exists an approximation algorithm with sub-k^2 query time. One of our contributions is that we answer this question in the affirmative.

Our Results. We develop sublinear-time algorithms for the $(k, O^*(k))$-gap edit distance problem, in the one-sided and two-sided preprocessing model, respectively.

▶ **Theorem 1** (One-Sided Preprocessing). *Let X, Y be length-n strings. After preprocessing Y in time $O^*(n)$, we can solve the $(k, k \cdot n^{o(1)})$-gap edit distance problem for X and Y in time $O^*(n/k + k)$ with high probability.*

In comparison to the (k, k^2)-gap algorithms from [21, 13] with best query time[4] $\widetilde{O}(n/k+k)$, we contribute the following improvement: Ignoring lower-order factors, we reduce the gap from k to $O^*(1)$ while achieving the same query time $O^*(n/k+k)$ and the same preprocessing time $O^*(n)$. In comparison to the $(k, O^*(k))$-gap algorithm in time $O^*(n/k + k^4)$ from [15], we achieve the same gap but an improved query time for large k, at the cost of preprocessing one of the strings.

In the two-sided model, we obtain an analogous result, where the query time no longer depends on n/k.

▶ **Theorem 2** (Two-Sided Preprocessing). *Let X, Y be length-n strings. After preprocessing both X and Y (separately) in time $O^*(n)$, we can solve the $(k, k \cdot n^{o(1)})$-gap edit distance problem for X and Y in time $O^*(k)$ with high probability.*

We remark that all hidden factors in both theorems are $2^{\widetilde{O}(\sqrt{\log n})}$. For a detailed comparison of this algorithm to the previously known results, see Table 1. We point out that Theorem 2 settles the open question from [21] whether there exists an edit distance approximation algorithm with small gap and sub-k^2 query time.

[3] Here we insist on almost-linear preprocessing time since the celebrated embedding of edit distance into the ℓ_1-metric with distortion $n^{o(1)}$ due to Ostrovsky and Rabani [26] achieves query time $O(\log n)$ but requires preprocessing time $\Omega(n^2)$.

[4] For the (k, k^2)-gap problem, the running time bounds $\widetilde{O}(n/k + k)$ and $\widetilde{O}(n/k)$ can be considered equal, as for $k \geq \sqrt{n}$ the algorithm may return a trivial answer.

Our Techniques. To achieve our results we build on the recent sublinear-time algorithm by Bringmann, Cassis, Fischer and Nakos [15], which itself builds on an almost-linear-time algorithm by Andoni, Krauthgamer and Onak [4]. The basic idea of the original algorithm is to split the strings into several smaller parts and recur on these parts with non-uniform precisions. The idea of [15] is to prune branches in the recursion tree, by detecting and analyzing periodic substructures. Towards that, they designed appropriate property testers to efficiently detect these structures. In our setting, we observe that having preprocessed the string(s), we can prune the computation tree much more easily. Thus, our algorithm proceeds in the same recursive fashion as [15] and uses a similar set of techniques, but due to the preprocessing it turns out to be simpler and faster.

Further Related Work. In the previous comparison about sublinear-time algorithms, we left out streaming and sketching algorithms [12, 18, 9] and document exchange protocols [23, 12, 22].

Future Directions. There are several interesting directions for future work. We specifically mention two open problems.

1. *Constant gap?* As Table 1 shows, so far no constant-gap sublinear-time algorithm is known. Maybe the one-sided preprocessing setting is more approachable for this challenge. We believe that our approach is hopeless to achieve a constant gap, since we borrow from the recursive decomposition introduced in [4] which inherently incurs a polylogarithmic overhead in the approximation factor.
2. *Improving the query time?* The well-known $\Omega(n/K)$ lower bound against the (k, K)-gap *Hamming distance* problem (and therefore against edit distance) continues to hold in the one-sided preprocessing setting. In particular, the most optimistic hope is an algorithm with query time $O^*(n/k)$ for the $(k, O^*(k))$-gap edit distance problem. Can this be achieved or is the extra $+k$ in the query time of Theorem 1 necessary? For two-sided preprocessing, to the best of our knowledge no lower bound is known.

2 Preliminaries

We set $[\,i\,..\,j\,] = \{\,i, i+1, \ldots, j-1\,\}$ (in particular, $[\,i\,..\,i\,] = \emptyset$) and $[\,j\,] = [\,0\,..\,j\,]$. We say that an event happens *with high probability* if it happens with probability at least $1 - 1/\operatorname{poly}(n)$, where the degree of the polynomial can be an arbitrary constant. We write $\operatorname{poly}(n) = n^{O(1)}$ and $\widetilde{O}(n) = n(\log n)^{O(1)}$.

Let X, Y be strings over an alphabet Σ with polynomial size. We denote by $|X|$ the length of X. We denote by $X \circ Y$ the *concatenation* of X and Y. We denote by $X[\,i\,]$ the i-th character in X starting with index zero. We denote by $X[\,i\,..\,j\,]$ the substring of X with indices in $[\,i\,..\,j\,]$, that is, including i and excluding j. For out-of-bounds indices we set $X[\,i\,..\,j\,] = X[\,\max(i,0)\,..\,\min(j,|X|)\,]$. If X and Y have the same length, we define their *Hamming distance* $\mathsf{HD}(X,Y)$ as the number of non-matching characters $\mathsf{HD}(X,Y) = |\{\,i : X[\,i\,] \neq Y[\,i\,]\,\}|$. For two strings X, Y with possibly different lengths, we define their *edit distance* $\mathsf{ED}(X,Y)$ as the smallest number of character *insertions*, *deletions* and *substitutions* necessary to transform X into Y. An *optimal alignment* between X and Y is a monotonically non-decreasing function $A : \{\,0, \ldots, |X|\,\} \to \{\,0, \ldots, |Y|\,\}$ such that $A(0) = 0$, $A(|X|) = |Y|$ and

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

It is easy to see that there is an optimal alignment between any two strings X, Y: Trace an optimal path through the standard dynamic program for edit distance and assign $A(i)$ to the smallest j for which the path crosses (i, j).

Let T be a rooted tree, and let v be a node in T. We denote by $\mathsf{root}(T)$ the root node in T. We denote by $\mathsf{parent}(v)$ the parent node of v. We denote by $\mathsf{depth}(v)$ the length of the root-to-v path, and by $\mathsf{height}(v)$ the length of the longest v-to-leaf path.

3 Overview

3.1 A Linear-Time Algorithm à la Andoni-Krauthgamer-Onak

We start to outline an almost-linear-time algorithm to approximate the edit distance of two strings X, Y following the framework of the Andoni-Krauthgamer-Onak algorithm [4], with some changes as in [15] and some additional modifications (see the novel trick outlined at the end of this subsection).

First Ingredient: A Divide-and-Conquer Scheme. The basic idea of the algorithm is to apply a divide-and-conquer scheme to reduce the approximation of the global edit distance to approximating the edit distance of several smaller strings. The straightforward idea of partitioning both strings X, Y into parts $X_1, \dots, X_m, Y_1, \dots, Y_m$ and computing the edit distances $\mathsf{ED}(X_i, Y_i)$ does not immediately work; instead we need to consider *several shifts* of the string Y_i. We remark that this concept of recurring on smaller strings for several shifts is quite standard in previous work. The following lemma uses the same ideas as the "$\mathcal{E}$-distance" defined in [4]. We give a proof in Appendix A.

▶ **Lemma 3** (Divide and Conquer). *Let X, Y be length-n strings, and let $0 = j_0 < \cdots < j_B = n$. We write $X_i = X[\, j_{i-1} \,..\, j_i \,]$ and $Y_{i,s} = Y[\, j_{i-1} + s \,..\, j_i + s \,]$.*

- *For all shifts $s_1, \dots, s_B$ we have that $\mathsf{ED}(X, Y) \le \sum_i \mathsf{ED}(X_i, Y_{i,s_i}) + 2|s_i|$.*
- *There are shifts $s_1, \dots, s_B$ with $\sum_i \mathsf{ED}(X_i, Y_{i,s}) \le 2\mathsf{ED}(X, Y)$ and $2|s_i| \le \mathsf{ED}(X, Y)$ for all i.*

To explain how to apply Lemma 3, we first specify on which substrings our algorithm is supposed to recur. To this end, let T be a balanced B-ary tree with n leaves. T will act as the "recursion tree" of the algorithm. For a string X of length n, we define a substring X_v for every node v in T as follows: If the subtree below X_v spans from the i-th to the j-th leaf (ordered from left to right), then we set $X_v = X[\, i \,..\, j+1 \,]$. In particular, X_v is a single character for each leaf v, and $X_{\mathsf{root}(T)} = X$. We further define $X_{v,s} = X[\, i+s \,..\, j+s+1 \,]$. For concreteness, we set $B = 2^{\sqrt{\log n \log\log n}} = n^{o(1)}$ throughout the paper.

A Simple Algorithm. Based on Lemma 3, we next present a simple (yet slow) algorithm. Our goal is to compute, for each node v in the tree T, an approximation $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ of $\mathsf{ED}(X_v, Y_{v,s})$ for all shifts s. The result at the root node is returned as the desired approximation of $\mathsf{ED}(X, Y)$. The algorithm works as follows: For each leaf we can cheaply compute $\mathsf{ED}(X_v, Y_{v,s})$ exactly by comparing the single characters X_v and $Y_{v,s}$. For each internal node with children $v_1, \dots, v_B$ we compute

$$\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) = \sum_{i=1}^{B} \min_{s_i \in \mathbf{Z}} \widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i|. \tag{1}$$

A careful application of Lemma 3 shows that if the recursive approximations $\mathsf{ED}(X_{v_i}, Y_{v_i,s_i})$ have multiplicative error at most α, then by approximating $\mathsf{ED}(X_v, Y_{v,s})$ as in (1) the multiplicative error becomes $2\alpha + B$. Since we repeat this argument recursively up to depth $\mathsf{depth}(T) \leq \log_B(n)$, the multiplicative error accumulates to $B \cdot \exp(O(\log_B(n))) = n^{o(1)}$.

This simple algorithm achieves the desired approximation quality, however, it is not fast enough: For every node v we have to compute $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ for too many shifts s (naively speaking, for up to n shifts). As a first step towards dealing with this issue, we first show that at every node v we can in fact tolerate a certain *additive* error (in addition to the multiplicative error discussed before) using a technique called *Precision Sampling*. Then we exploit the freedom of additive errors to run this algorithm for a *restricted* set of shifts s.

Second Ingredient: Precision Sampling. It ultimately suffices to compute an approximation of $\mathsf{ED}(X, Y)$ with *additive* error k in order to solve the constant-gap edit distance problem. We leverage this freedom to also solve the recursive subproblems up to some additive error. Specifically, we will work with the following data structure:

▶ **Definition 4** (Precision Tree). *Let T be a balanced B-ary tree with n leaves. For $t \in \mathbf{N}$, we randomly associate a* tolerance t_v *to every node v in T as follows:*

- *If v is the root, then set $t_v = t$;*
- *otherwise set $t_v = t_{\mathsf{parent}(v)} \cdot u_v/3$, where we sample $u_v \sim \mathrm{Exp}(O(\log n))$ (the exponential distribution with parameter $O(\log n)$).*

We refer to T as a precision tree with initial tolerance t.

The tolerance t_v at a node v determines the additive error which we can tolerate at v. That is, our goal is to approximate $\mathsf{ED}(X_v, Y_{v,s})$ with additive error t_v (and the same multiplicative error as before). The initial tolerance is set to $t = k$. The critical step is how to combine the recursive approximations with additive error $t_{v_1}, \ldots, t_{v_B}$ to an approximation with additive error t_v. The naive solution would incur error $\sum_i t_{v_i} \gg t_v$. Instead, we employ the *Precision Sampling Lemma* [4, 5, 3, 15] (see Lemma 15 in Section 4) to recombine the recursive approximation and avoid this blow-up in the additive error.

An Improved Algorithm. We can now improve the simple algorithm to a near-linear-time algorithm. In the original Andoni-Krauthgamer-Onak algorithm this was achieved by pruning most recursive subproblems (depending on their tolerances t_v). We will follow a different avenue: Our algorithm recurs on every node in the precision tree, and we obtain a linear-time algorithm by bounding the expected running time per node by $n^{o(1)}$.

We achieve this by the following novel trick: We restrict the set of feasible shifts s at each node v with respect to the tolerance t_v. In fact, we require two constraints: First, we restrict s to values smaller than $\approx k$ in absolute value. This first restriction is correct since we only want to maintain edit distances bounded by k; this idea was also used in previous works. Second, we restrict the feasible shifts s at any node v to multiples of $\lfloor t_v/2 \rfloor$. Then, in order to approximate $\mathsf{ED}(X_v, Y_{v,s})$ for any shift s we let $\tilde{s}$ denote the closest multiple of $\lfloor t_v/2 \rfloor$ to s, and approximate $\mathsf{ED}(X_v, Y_{v,s})$ by $\mathsf{ED}(X_v, Y_{v,\tilde{s}})$. Since $|s - \tilde{s}| \leq t_v/2$, both edit distances differ by at most t_v. Recall that we can tolerate this error using the Precision Sampling technique. Let $S_v = \{ -k \cdot n^{o(1)}, \ldots, k \cdot n^{o(1)} \} \cap \lfloor t_v/2 \rfloor \mathbf{Z}$ denote the set of shifts respecting these restrictions (the precise lower-order term $n^{o(1)}$ will be fixed later).

In terms of efficiency, we have improved as follows: At every node the running time is essentially dominated by the number of feasible shifts s. Using our discretization trick, there are only $|S_v| = k \cdot n^{o(1)}/t_v$ such shifts. By the following Lemma 5, we can bound this number in expectation by $k \cdot n^{o(1)}/t_{\mathsf{root}(T)} = n^{o(1)}$.

▶ **Lemma 5** (Expected Precision). *Let T be a B-ary precision tree with initial tolerance t and $B = \exp(\widetilde{\Theta}(\sqrt{\log n}))$, and let v be a node in T. Then, conditioned on a high-probability event E, it holds that*

$$\mathbf{E}\left(\frac{1}{t_v}\,\middle|\,E\right) \leq \frac{(\log n)^{O(\mathsf{depth}(v))}}{t} \leq \frac{n^{o(1)}}{t}.$$

We include a proof in Appendix A. For technical reasons, the lemma is only true conditioned on some high-probability event E. For the remainder of this paper, we implicitly condition on this event E.

3.2 How to Go Sublinear?

Following the idea of [15], our strategy is to turn this algorithm into a sublinear-time algorithm by not exploring the whole precision tree recursively, and instead only exploring a smaller fragment. To achieve this, the goal is to approximate $\mathsf{ED}(X_v, Y_{v,s})$ for many nodes v directly, without the need to explore their children – we say that we *prune* v. The algorithm by [15] uses several structural insights on periodic versus non-periodic strings to implement pruning rules. We can avoid the complicated treatment and follow a much simpler avenue, exploiting that we can preprocess the strings.

In the following, we will assume that we have access to an oracle answering the following two queries. In the next section we will argue how to efficiently implement data structures to answer these queries.

- $\textsc{Matching}(X, Y, v)$: Returns either $\textsc{Close}(s^*)$ where s^* satisfies $|s^*| \leq k \cdot n^{o(1)}$ and $\mathsf{HD}(X_v, Y_{v,s^*}) \leq t_v/2$, or $\textsc{Far}$ in case that there is no shift s^* with $X_v = Y_{v,s^*}$, see Definition 8. (Note that if $1 \leq \min_{s^*} \mathsf{HD}(X_v, Y_{v,s^*}) \leq t_v/2$, the query can return either $\textsc{Close}(s^*)$ or $\textsc{Far}$.)
- $\textsc{ShiftedDistance}(Y, v, s, s')$: For shifts $|s|, |s'| \leq k \cdot n^{o(1)}$, returns an approximation of $\mathsf{ED}(Y_{v,s}, Y_{v,s'})$ with additive error $t_v/2$ (and $n^{o(1)}$-multiplicative error, see Definition 9).

Suppose for the moment that both queries can be answered in constant time. Then we can reduce the running time of the previous linear-time algorithm to time $k \cdot n^{o(1)}$ as follows: We try to prune each node v by querying $\textsc{Matching}(X, Y, v)$. If the $\textsc{Matching}$ query reports $\textsc{Far}$, we simply continue recursively as before (i.e., no pruning takes place). However, if the matching query reports $\textsc{Close}(s^*)$, then we can prune v as follows: Query $\textsc{ShiftedDistance}(Y, v, s^*, s)$ for all shifts $s \in S_v$ and return the outcome as an approximation of $\mathsf{ED}(X_v, Y_{v,s})$ for all shifts s. For the correctness we apply the triangle inequality and argue that the additive error is bounded by $t_v/2 + t_v/2 = t_v$.

It remains to argue that the number of recursive computations is bounded by $k \cdot n^{o(1)}$. The intuitive argument is as follows: Assume that $\mathsf{ED}(X, Y) \leq k$ (i.e., we are in the "close" case) and consider an optimal alignment between X and Y, which contains at most k mismatches. Recall that each level of the precision tree induces a partition of X into consecutive substrings X_v. Thus, there are at most k substrings X_v which contain a mismatch in the optimal alignment. For all *other* substrings, there are no mismatches and hence $X_v = Y_{v,s^*}$ for some shift s^*. It follows that on every level there are at most k nodes for which the $\textsc{Matching}$ test fails, and in total there are only $k \cdot \mathsf{height}(T) = O(k \log n)$ such nodes.

3.3 How to Answer Matching and Shifted-Distance Queries?

It remains to find data structures which answer these queries efficiently. We assume that the precision tree T has been generated in advance and is shared across all precomputations. In particular, this requires "public randomness" for the otherwise independent precomputations.

Matching Queries. The idea is to precompute and store fingerprints (i.e. hashes) of the substrings $Y_{v,s}$ for every node v in the partition tree and all shifts $s \in \{-k \cdot n^{o(1)}, \dots, k \cdot n^{o(1)}\}$. Then, to answer a query we simply compute the fingerprint of X_v and lookup whether it equals one of the precomputed fingerprints. Alas, a naive implementation of this idea is too slow since upon query we might need to read the whole string X. To obtain the desired sublinear query time, we instead subsample the strings X_v and $Y_{v,s}$ with rate $\approx 1/t_v$. In this way we incur additive Hamming error at most $t_v/2$, as desired. Formally, we show the following lemma in Section 4.1:

▶ **Lemma 6** (Matching Queries). *We can preprocess Y in expected time $n^{1+o(1)}$ to answer $\textsc{Matching}(X, Y, v)$ queries in time $\widetilde{O}(|X_v|/t_v)$ with high probability. Moreover, we can separately preprocess both X and Y in expected time $n^{1+o(1)}$ to answer $\textsc{Matching}(X, Y, v)$ queries in time $O(1)$ with high probability.*

The key difference between the one-sided and the two-sided preprocessing is that in the former we need to compute the fingerprint for X_v, which takes time $O(|X_v|/t_v)$ (we shave the factor t_v due to the subsampling), while in the latter we can afford to precompute these fingerprints and answer the queries faster.

Shifted-Distance Queries. We give two ways to answer ShiftedDistance queries. The first one relies in a black-box manner on any almost-linear-time algorithm to compute an edit distance approximation with multiplicative error $n^{o(1)}$ [4, 7, 6]. Using the same trick as before to restrict the set of feasible shifts, it suffices to approximate $\mathsf{ED}(Y_{v,s}, Y_{v,s'})$ for all shifts $s \in S_v$ and $s' \in \{-k \cdot n^{o(1)}, \dots, k \cdot n^{o(1)}\}$. (We could also discretize the range of s', but this does not improve the performance here.) In this way, we incur an additive error of at most $O(t_v)$. The computation per node v takes time $|Y_v| \cdot k \cdot k \cdot n^{o(1)}/t_v$, which becomes $kn^{1+o(1)}$ in expectation by Lemma 5 and by summing over all nodes v.

We then show that we can improve the preprocessing time to $n^{1+o(1)}$ by applying the non-oblivious embedding of edit distance into ℓ_1 by Andoni and Onak [7]. Formally we obtain the following lemma, which we prove in Section 4.3.

▶ **Lemma 7** (Shifted-Distance Queries). *We can preprocess Y in time $n^{1+o(1)}$ to answer $\textsc{ShiftedDistance}(Y, v, s, s')$ queries in time $n^{o(1)}$ with high probability.*

4 Our Algorithm in Detail

In this section we give a detailed proof of our main theorems by analyzing Algorithms 1 and 2 (see Sections 4.4 and 4.5). Algorithm 2 is the previously discussed reformulation of the Andoni-Krauthgamer-Onak algorithm, with the improvement that the recursive computation can be avoided whenever Algorithm 1 succeeds. Algorithm 1 implements the pruning rule which, as we will argue, triggers often enough to improve the running time.

Throughout we fix the initial tolerance of the precision tree T to $t = t_{\mathsf{root}(T)} = k$. Moreover, we set $S = \{-k \cdot 3^{\log_B(n)}, \dots, k \cdot 3^{\log_B(n)}\}$ and $S_v = S \cap \lfloor t_v/2 \rfloor \mathbf{Z}$. Observe that $|S| = k \cdot 3^{\log_B(n)} = k \cdot n^{o(1)}$ for our choice of $B = \exp(\widetilde{O}(\sqrt{\log n}))$.

Outline. Compared to the technical overview, we present the details in the reverse order: Starting with the implementation of the data structures for Matching and Shifted-Distance queries (in Section 4.1), we first analyze Algorithm 1 (in Section 4.4). Then we analyze Algorithm 2 (in Section 4.5) and put the pieces together for our main theorems (in Section 4.6).

4.1 Matching Queries

In this section we show to implement data structures that answer Matching queries. We start with a formal definition.

▶ **Definition 8** (Matching Queries). *Let X, Y be strings of length n, and let v be a node in the precision tree. A Matching(X, Y, v) query is correctly answered by one of two outputs:*

- *Close(s^*) for some shift $s^* \in S$ satisfying $\mathsf{HD}(X_v, Y_{v,s^*}) \le t_v/2$, or*
- *Far if there exists no shift $s^* \in S$ with $X_v = Y_{v,s^*}$.*

▶ **Lemma 6** (Matching Queries). *We can preprocess Y in expected time $n^{1+o(1)}$ to answer Matching(X, Y, v) queries in time $\widetilde{O}(|X_v|/t_v)$ with high probability. Moreover, we can separately preprocess both X and Y in expected time $n^{1+o(1)}$ to answer Matching(X, Y, v) queries in time $O(1)$ with high probability.*

Proof. We first explain the general idea behind the data structure and then point out the specifics for the one-sided and two-sided preprocessing. For each node v in the precision tree, we subsample a set $H_v \subseteq [\,|Y_v|\,]$ with rate $\Theta(\log n/t_v)$ and sample a hash function $h : \Sigma^{|H_v|} \to [\,\mathrm{poly}(n)\,]$ from any universal family of hash functions. (Take for instance the function $h(\sigma_1, \dots, \sigma_{|H_v|}) = \sum_i a_i\sigma_i \bmod p$ for some prime $p = \mathrm{poly}(n)$ and random a_i's).

We associate to a string $A \in \Sigma^{|Y_v|}$ the *fingerprint* $h(A[\,H_v\,])$, where we write $A[\,H_v\,]$ for the subsequence of A with indices in H_v. We claim that these fingerprints can distinguish any two strings A, B with $\mathsf{HD}(A, B) > t_v/2$ with high probability. Indeed, the probability that H_v contains an index i with $A[\,i\,] \neq B[\,i\,]$ is at least

$$1 - \left(1 - \frac{\Omega(\log n)}{t_v}\right)^{\mathsf{HD}(A,B)} \ge 1 - \exp(-\Omega(\log n)) = 1 - \frac{1}{\mathrm{poly}(n)}.$$

Hence, with high probability we have that $A[\,H_v\,] \neq B[\,H_v\,]$. Moreover, since h is sampled from a universal family of hash functions, we also have that $h(A[\,H_v\,]) \neq h(B[\,H_v\,])$ with high probability. On the other hand, if $A = B$ then clearly $h(A[\,H_v\,]) = h(B[\,H_v\,])$. We now turn to the implementation details for one-sided and two-sided preprocessing.

One-Sided Preprocessing. In the preprocessing phase, we prepare for every node v the fingerprints of $Y_{v,s}$ for all shifts $s \in S$, and store these fingerprints in a lookup table. In the query phase, given some string X we compute the fingerprint of X_v and check whether it appears in the lookup table. If so, we return Close(s^*) for the shift s^* corresponding to the precomputed fingerprint. Otherwise, we return Far. The correctness follows from the previous paragraph.

The expected running time of the preprocessing phase is bounded by $\sum_v \widetilde{O}(|Y_v|/t_v \cdot |S|)$ (for every node we have to prepare $|S|$ fingerprints, each taking time $\widetilde{O}(|Y_v|/t_v)$). This becomes $\sum_v O(|Y_v| \cdot n^{o(1)}/t_{\mathsf{root}(T)} \cdot k) = \sum_v O(|Y_v| \cdot n^{o(1)}) = n^{1+o(1)}$ in expectation over the tolerances t_v; see Lemma 5. The query time is dominated by computing the fingerprint of X_v which takes expected time $\widetilde{O}(|X_v|/t_v)$ (even with high probability by an application of the Chernoff bound).

Two-Sided Preprocessing. For two-sided preprocessing, we can also prepare the fingerprints of X_v for all nodes v in the preprocessing phase. The expected preprocessing time is still $n^{1+o(1)}$, but for queries it only takes constant time to perform the lookup. ◀

4.2 Simple Shifted-Distance Queries

We next demonstrate how to deal with ShiftedDistance queries, formally defined as follows:

▶ **Definition 9** (Shifted-Distance Queries). *Let Y be a string of length n, let v be a node in the precision tree and let $s, s' \in S$. A ShiftedDistance(Y, v, s, s') query computes a number satisfying:*

$$\mathsf{ED}(Y_{v,s}, Y_{v,s'}) - t_v/2 \leq \textsc{ShiftedDistance}(Y, v, s, s') \leq n^{o(1)} \cdot (\mathsf{ED}(Y_{v,s}, Y_{v,s'}) + t_v/2).$$

We will first show how to implement a simpler version of Lemma 7 at the cost of worsening the preprocessing time to $kn^{1+o(1)}$ instead of $n^{1+o(1)}$. The benefit of this weaker version is that it is a black-box reduction to any almost-linear time $n^{o(1)}$-approximation algorithm for edit distance, while the improved version crucially relies on the properties of the particular algorithm by Andoni and Onak [7]. In the next section we show how to obtain the speed-up.

▶ **Lemma 10** (Slower Shifted-Distance Queries). *We can preprocess Y in time $kn^{1+o(1)}$ to answer ShiftedDistance(Y, v, s, s') queries in time $n^{o(1)}$ with high probability.*

Proof. We will use the result that an $n^{o(1)}$-approximation for the edit distance of two length-n strings can be computed in time $n^{1+o(1)}$ [4, 7, 6]. In the preprocessing phase, we compute $n^{o(1)}$-factor approximations $\widetilde{\mathsf{ED}}(Y_{v,\tilde{s}}, Y_{v,s'})$ for all nodes v, all $\tilde{s} \in S_v$ and all $s' \in S$. Then, to answer a query ShiftedDistance(Y, v, s, s') we let

$$\tilde{s} := \operatorname*{argmin}_{\tilde{s} \in S_v} |s - \tilde{s}|$$

and output $\widetilde{\mathsf{ED}}(Y_{v,\tilde{s}}, Y_{v,s'})$.

First we argue that this gives a good approximation. Indeed, we have that $|s - \tilde{s}| \leq t_v/4$ by the definition of S_v. Therefore:

$$\begin{aligned}\widetilde{\mathsf{ED}}(Y_{v,\tilde{s}}, Y_{v,s'}) &\leq n^{o(1)} \cdot \mathsf{ED}(Y_{v,\tilde{s}}, Y_{v,s'}) \\ &\leq n^{o(1)} \cdot (\mathsf{ED}(Y_{v,s}, Y_{v,s'}) + \mathsf{ED}(Y_{v,s}, Y_{v,\tilde{s}})) \\ &\leq n^{o(1)} \cdot (\mathsf{ED}(Y_{v,s}, Y_{v,s'}) + t_v/2),\end{aligned}$$

where second inequality is an application of the triangle inequality, and the last inequality follows since we can transform $Y_{v,\tilde{s}}$ into $Y_{v,s}$ by adding and removing $t_v/4$ symbols. A symmetric argument shows the claimed lower bound $\widetilde{\mathsf{ED}}(Y_{v,\tilde{s}}, Y_{v,s'}) \geq \mathsf{ED}(Y_{v,s}, Y_{v,s'}) - t_v/2$.

Next we analyze the running time. For the preprocessing, we compute $|S_v| \cdot |S| = O^*(k^2/t_v)$ many approximations for each node v, each in time $|Y_v|^{1+o(1)}$. We can bound $\mathbf{E}(1/t_v) = n^{o(1)}/k$ by Lemma 5, so the expected total time is $\sum_v |Y_v|^{1+o(1)} n^{o(1)} \cdot k = kn^{1+o(1)}$. Answering a query takes constant time since we only need to compute s^* as stated above and perform a constant-time lookup. ◀

4.3 Faster Shifted-Distance Queries

In this section we show how to improve the preprocessing time for ShiftedDistance queries to $n^{1+o(1)}$. We thank the anonymous reviewer who suggested this improvement. The key technical tool to obtain this improvement is the following result by Andoni and Onak [7].

Algorithm 1 Approximates the edit distance of preprocessed strings, or fails.

Input: Strings X (un- or preprocessed) and Y (preprocessed), a node v in the precision tree
Output: An approximation $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ of $\mathsf{ED}(X_v, Y_{v,s})$ for all $s \in S_v$, or FAIL

1: **if** MATCHING$(X, Y, v) =$ CLOSE(s^*) **then**
2: **return** $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) =$ SHIFTEDDISTANCE(Y, v, s^*, s) for all $s \in S_v$
3: **else**
4: **return** FAIL

▶ **Theorem 11** (Embedding Substrings into ℓ_1 [7]). *Let X be a string of length n. Then, for each integer m of the form $m = \lfloor n/B^i \rfloor$ for $0 \le i \le \log_B(n)$ and $B = 2^{\Theta(\sqrt{\log n \log\log n})}$, we can embed all length-m substrings $X_0, \dots, X_{n-m}$ of X into vectors $v_0^m, \dots, v_{n-m}^m$ of dimension $n^{o(1)}$ such that for every $j, j' \in [\, n - m + 1\,]$, with high probability it holds that*

$$\mathsf{ED}(X_j, X_{j'}) \le \|v_j^m - v_{j'}^m\|_1 \le n^{o(1)} \cdot \mathsf{ED}(X_j, X_{j'}).$$

The time to compute all vectors is $n^{1+o(1)}$.

The main application of this embedding in [7] is an $n^{o(1)}$-approximation for the edit distance of two length-n strings in time $n^{1+o(1)}$ (in [7], Theorem 11 is applied to the concatenation of two strings to compute the ℓ_1-distance between their corresponding vectors). We remark that the guarantee of the embedding in Theorem 11 is *non-oblivious*, in the sense that the algorithm needs to have access to all the substrings it is embedding. In particular, this means that it cannot be directly applied in the two-sided preprocessing setting where we would like to embed the strings separately.

▶ **Lemma 7** (Shifted-Distance Queries). *We can preprocess Y in time $n^{1+o(1)}$ to answer SHIFTEDDISTANCE(Y, v, s, s') queries in time $n^{o(1)}$ with high probability.*

Proof. We assume without loss of generality that $n = |Y| = |X|$ is a power of B (we can pad both X and Y so that this holds, which has no impact on the running time or the approximation guarantee of our algorithms). We apply Theorem 11 to the string Y and store all the $n^{1+o(1)}$ embedded vectors. To answer a query SHIFTEDDISTANCE(Y, v, s, s'), note that $Y_{v,s}$ and $Y_{v,s'}$ are substrings of Y of length $m := n/B^{\mathsf{depth}(v)}$. Thus, the ℓ_1 distance between their corresponding vectors $v^m_{j(v,s)}, v^m_{j(v,s')}$ given by Theorem 11 gives the desired approximation (with no additive error). Since these vectors have dimension $n^{o(1)}$, the time to compute $\|v^m_{j(v,s)} - v^m_{j(v,s')}\|_1$ and answer the query is $n^{o(1)}$, as desired. ◀

4.4 Pruning Rule for Preprocessed Strings

In this section we analyze Algorithm 1. We always assume that either only Y or both X and Y have been preprocessed by Lemmas 6 and 7 to efficiently answer MATCHING and SHIFTEDDISTANCE queries.

▶ **Lemma 12** (Correctness of Algorithm 1). *Whenever Algorithm 1 does not return FAIL, it returns approximations $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ satisfying with high probability*

$$\mathsf{ED}(X_v, Y_{v,s}) - t_v \le \widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \le n^{o(1)} \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v).$$

Proof. Algorithm 1 only succeeds if the MATCHING query in Algorithm 1 successfully identified some shift s^* with $\mathsf{HD}(X_v, Y_{v,s^*}) \leq t_v/2$ by Lemma 6. In this case, we report $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) := \textsc{ShiftedDistance}(Y, v, s^*, s)$. By Lemma 7, this is an approximation of $\mathsf{ED}(Y_{v,s^*}, Y_{v,s})$ with additive error $t_v/2$ and multiplicative error $n^{o(1)}$. Combining both facts, and using the triangle inequality we obtain that

$$\begin{aligned}\widetilde{\mathsf{ED}}(Y_{v,s^*}, Y_{v,s}) &\leq n^{o(1)} \cdot (\mathsf{ED}(Y_{v,s^*}, Y_{v,s}) + t_v/2)\\ &\leq n^{o(1)} \cdot (\mathsf{ED}(X_v, Y_{v,s}) + \mathsf{ED}(X_v, Y_{v,s^*}) + t_v/2)\\ &\leq n^{o(1)} \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v),\end{aligned}$$

and

$$\begin{aligned}\widetilde{\mathsf{ED}}(Y_{v,s^*}, Y_{v,s}) &\geq \mathsf{ED}(Y_{v,s^*}, Y_{v,s}) - t_v/2\\ &\geq \mathsf{ED}(X_v, Y_{v,s}) - \mathsf{ED}(X_v, Y_{v,s^*}) - t_v/2\\ &\geq \mathsf{ED}(X_v, Y_{v,s}) - t_v.\end{aligned}$$

◀

▶ **Lemma 13** (Running Time of Algorithm 1)**.** *If only the string Y is preprocessed, then Algorithm 1 runs in expected time $O^*(|X_v|/t_v + k/t_v)$. If both strings X, Y are preprocessed, then Algorithm 1 runs in expected time $O^*(k/t_v)$.*

Proof. The expected running time of the MATCHING query is bounded by $\widetilde{O}(|X_v|/t_v)$ (for one-sided preprocessing) or by $O(1)$ (for two-sided preprocessing). The running time of a single SHIFTEDDISTANCE query is bounded by $n^{o(1)}$, and we make $|S_v| = \widetilde{O}(k/t_v)$ SHIFTEDDISTANCE queries. Hence, the total expected time is $O^*(|X_v|/t_v + k/t_v)$ (for one-sided preprocessing) or $O^*(k/t_v)$ (for two-sided preprocessing). ◀

▶ **Lemma 14** (Efficiency of Algorithm 1)**.** *For any two strings X, Y, there are at most $O(k \log n)$ many nodes v in the precision tree for which Algorithm 1 fails, assuming that $\mathsf{ED}(X, Y) \leq k$.*

Proof. It suffices to argue that there are at most $O(\mathsf{ED}(X, Y) \log n)$ nodes for which the MATCHING query in Algorithm 1 fails. We start with a fixed level in the precision tree, and enumerate all nodes on that level as $v_1, \ldots, v_m$. By definition we have that $X = \bigcirc_{i=1}^{m} X_{v_i}$, hence there exist indices $0 = j_0 < \cdots < j_m = n$ such that $X_{v_i} = X[\, j_i \,..\, j_{i+1}\,]$. Now consider an optimal alignment $A : \{\, 0, \ldots, n \,\} \to \{\, 0, \ldots, n \,\}$ between X and Y. In particular, A satisfies

$$\mathsf{ED}(X, Y) = \sum_{i=1}^{m} \mathsf{ED}(X[\, j_i \,..\, j_{i+1}\,], Y[\, A(j_i) \,..\, A(j_{i+1})\,]).$$

There can be at most $\mathsf{ED}(X, Y) \leq k$ many nonzero terms in the sum, and we claim that every zero term corresponds to a node v_i for which Algorithm 1 succeeds. Indeed, for any zero term we have $X[\, j_i \,..\, j_{i+1}\,] = Y[\, A(j_i) \,..\, A(j_{i+1})\,]$ and therefore $X_{v_i} = Y_{v_i,s^*}$ where $s^* = A(j_i) - j_i$. It remains to argue that $|s^*| \leq k$ (since otherwise the index s^* would be out-of-bounds and could not be detected by a MATCHING query). To see this, observe that

$$|s^*| \leq \mathsf{ED}(X[\, j_0 \,..\, j_i\,], Y[\, A(j_0) \,..\, A(j_i)\,]) \leq \mathsf{ED}(X, Y) \leq k,$$

exploiting again that A is an optimal alignment.

Finally, recall that there are only $\log_B(n) \leq \log n$ levels in the precision tree, hence the total number of nodes for which Algorithm 1 fails is bounded by $O(k \log n)$. ◀

Algorithm 2 Approximates the edit distance of preprocessed strings.

Input: Strings X (un- or preprocessed), Y (preprocessed), a node v in the precision tree
Output: An approximation $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ of $\mathsf{ED}(X_v, Y_{v,s})$ for all $s \in S_v$

1: **if** Algorithm 1 succeeds and reports $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ for all $s \in S_v$ **then**
2: **return** $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ for all $s \in S_v$
3: **if** v is a leaf **then**
4: **return** $\mathsf{ED}(X_v, Y_{v,s})$ for all $s \in S_v$
5: **for** all children v_i of v **do**
6: Recursively compute $\widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i})$ for all $s_i \in S_{v_i}$
7: Compute $\widetilde{a}_{i,s} = \min_{s_i \in S_{v_i}} \widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i}) + 2 \cdot |s - s_i|$ for all $s \in S_v$
8: **return** $\textsc{Recover}(\widetilde{a}_{1,s}, \dots, \widetilde{a}_{B,s}, u_{v_1}, \dots, u_{v_B})$ for all $s \in S_v$

4.5 The Complete Algorithm

Our complete Algorithm 2 is essentially what we described in Section 4.5 with the additional pruning rule of applying Algorithm 1. We start with the correctness of Algorithm 2. We need the following lemma, which has previously been referred to as a *Precision Sampling Lemma* [4, 5, 3, 15]. The lemma was introduced by Andoni, Krauthgamer and Onak [4], and was refined in [5, 3].

Intuitively, the lemma serves the following purpose: For fixed numbers $a_1, \dots, a_B$, say that we have access to approximations $\widetilde{a}_1, \dots, \widetilde{a}_B$ with multiplicative error α and additive approximation error β. Then we can naively approximate $\sum_i a_i$ by $\sum_i \widetilde{a}_i$ with multiplicative error α and additive error $B \cdot \beta$. The Precision Sampling Lemma states that the blow-up in the additive error can be avoided if the approximations $\widetilde{a}_i$ instead have additive error $\beta \cdot u_i$ for some *non-uniformly sampled precisions* u_i.

▶ **Lemma 15** (Precision Sampling Lemma [3]). *Fix parameters $\delta > 0$, $\alpha \geq 1$ and $\beta \geq 0$. Let $a_1, \dots, a_B \geq 0$ be reals, and independently sample $u_1, \dots, u_B \sim \mathrm{Exp}(O(\log(\delta^{-1})))$ (for some sufficiently large hidden constant). There is an $O(B \log(\delta^{-1}))$-time algorithm $\textsc{Recover}$ satisfying for all $\widetilde{a}_1, \dots, \widetilde{a}_B$, with success probability at least $1 - \delta$:*

- *If $\widetilde{a}_i \geq \frac{1}{\alpha} \cdot a_i - \beta \cdot u_i$ for all i, then $\textsc{Recover}(\widetilde{a}_1, \dots, \widetilde{a}_B, u_1, \dots, u_B) \geq \frac{1}{2\alpha} \cdot \sum_i a_i - \beta$.*
- *If $\widetilde{a}_i \leq \alpha \cdot a_i + \beta \cdot u_i$ for all i, then $\textsc{Recover}(\widetilde{a}_1, \dots, \widetilde{a}_B, u_1, \dots, u_B) \leq 2\alpha \sum_i a_i + \beta$.*

▶ **Lemma 16** (Correctness of Algorithm 2). *Algorithm 2 computes values $\widetilde{\mathsf{ED}}(X_v, Y_{v,s})$ (for all $s \in S_v$) satisfying the following bounds with high probability:*

- $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \geq n^{-o(1)} \cdot \mathsf{ED}(X_v, Y_{v,s}) - t_v$, *and*
- $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \leq n^{o(1)} \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v)$ *assuming that* $\mathsf{ED}(X_v, Y_{v,s}) \leq k$ *and* $2|s| \leq k$.

Proof. The recursion of Algorithm 2 terminates in one of three cases: If v is a leaf, then the output is exact and the claim is obvious. If v is directly solved by Algorithm 1, then by the guarantee of Lemma 12 the claim is true. It remains to analyze the case when the algorithm recurs, so assume that v is an internal node with children $v_1, \dots, v_B$. We prove the lower and upper bounds separately.

Lower Bound. Fix some shift $s \in S_v$, and let $s_1, \dots, s_B$ be the corresponding shifts selected in Algorithm 2 of the algorithm. We prove that $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \geq \mathsf{ED}(X_v, Y_{v,s})/\alpha(\mathsf{height}(v)) - t_v$ by induction, where $\alpha(\mathsf{height}(v)) = 2^{\mathsf{height}(v)}$. To this end, we apply the Precision Sampling Lemma with $a_i = \mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i|$ and the following parameters:

- $\delta = 1/\operatorname{poly}(n)$,
- $\alpha = \alpha(\mathsf{height}(v) - 1)$,
- $\beta = t_v$.

Recall that the algorithm computes $\widetilde{a}_{i,s} = \widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i|$, where the approximation $\widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i})$ is computed recursively. Hence, it satisfies $\widetilde{a}_{i,s} \geq a_i/\alpha - \beta \cdot u_{v_i}$ by the induction hypothesis (recall that $t_{v_i} \leq t_v \cdot u_{v_i}$). Then $\textsc{Recover}(\widetilde{a}_{1,s}, \ldots, \widetilde{a}_{B,s}, u_{v_1}, \ldots, u_{v_B})$ computes, with high probability, a number satisfying

$$\begin{aligned}
\textsc{Recover}(\cdot) &\geq \frac{1}{2\alpha(\mathsf{height}(v)-1)} \cdot \left(\sum_{i=1}^{B} a_i \right) - \beta \\
&= \frac{1}{2\alpha(\mathsf{height}(v)-1)} \cdot \left(\sum_{i=1}^{B} \mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i| \right) - t_v \\
&\geq \frac{1}{2\alpha(\mathsf{height}(v)-1)} \cdot \mathsf{ED}(X_v, Y_{v,s}) - t_v \\
&\geq \frac{1}{\alpha(\mathsf{height}(v))} \cdot \mathsf{ED}(X_v, Y_{v,s}) - t_v,
\end{aligned}$$

where in the third step, we applied the lower bound from Lemma 3 and in the last step we used the definition of $\alpha(\cdot)$. Finally, recall that $\mathsf{height}(T) \leq \log_B(n)$ and $B = \exp(\widetilde{\Theta}(\sqrt{\log n}))$. Hence the total multiplicative error is bounded by $\alpha(\log_B(n)) \leq 2^{\log_B(n)} = n^{o(1)}$.

Upper Bound. This proof is similar to the previous paragraph, but requires a more careful application of Lemma 3. We prove by induction the algorithm computes an approximation $\widetilde{\mathsf{ED}}(X_v, Y_{v,s}) \leq \alpha(\mathsf{height}(v)) \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v)$ where this time the multiplicative error is bounded by $\alpha(\mathsf{height}(v)) \leq O(B) \cdot 2^{O(\mathsf{height}(v))}$, provided that $\mathsf{ED}(X_v, Y_{v,s}) \leq k \cdot 3^{\mathsf{depth}(v)}$ and $2|s| \leq k \cdot 3^{\mathsf{depth}(v)}$. Note that this implies the lemma statement.

Throughout, fix some shift $s \in S_v$. The idea is to first use Lemma 3 to find "optimal" shifts $s_1^*, \ldots, s_B^* \in \mathbf{Z}$, which we use for the recursive computation. Unfortunately these shifts s_i^* may not fall into the restricted set of feasible shifts S_{v_i}. We therefore argue that picking shifts $s_i \in S_{v_i}$ closest possible to s_i^* is sufficient to obtain the claimed guarantee. Formally, by Lemma 3 there exist shifts $s_1^*, \ldots, s_B^*$ satisfying the following two properties:

$$\sum_{i=1}^{B} \mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*}) \leq 2\mathsf{ED}(X_v, Y_{v,s}), \tag{2}$$

$$2|s - s_i^*| \leq \mathsf{ED}(X_v, Y_{v,s}). \tag{3}$$

We now pick $s_1 \in S_{v_1}, \ldots, s_B \in S_{v_B}$ to be the closest values to the optimal shifts $s_1^*, \ldots, s_B^*$. As a first insight, observe that:

$$2|s_i^*| \leq 2|s - s_i^*| + 2|s| \leq \mathsf{ED}(X_v, Y_{v,s}) + 2|s| \leq 2k \cdot 3^{\mathsf{depth}(v)}.$$

Recall that we set $S = \{ -k \cdot 3^{\log_B(n)}, \ldots, k \cdot 3^{\log_B(n)} \}$ and we therefore have $s_i^* \in S$. It follows that $|s_i - s_i^*| \leq t_{v_i}/2$ and thus

$$2|s_i| \leq 2|s_i^*| + t_{v_i} \leq 2|s_i^*| + k \leq 2k \cdot 3^{\mathsf{depth}(v)} + k \leq k \cdot 3^{\mathsf{depth}(v_i)}.$$

Next, we claim that $\mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) \leq k \cdot 3^{\mathsf{depth}(v_i)}$, which we will use to guarantee that the recursive calls of the algorithm succeed. Indeed, we have that

$$\mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) \leq \mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*}) + t_{v_i} \leq 2 \cdot \mathsf{ED}(X_v, Y_{v,s}) + k \leq k \cdot 3^{\mathsf{depth}(v_i)}.$$

We claim that if the algorithm was to choose the shifts s_i specified in the previous paragraph in Algorithm 2, then the output is bounded as claimed. (This is sufficient, since Algorithm 2 in fact minimizes over all possible shifts s_i.) In this case, we inductively have that

$$\begin{aligned}\widetilde{a}_{i,s} &\leq \widetilde{\mathsf{ED}}(X_{v_i}, Y_{v_i,s_i}) + 2|s - s_i| \\ &\leq \alpha(\mathsf{height}(v) - 1) \cdot (\mathsf{ED}(X_{v_i}, Y_{v_i,s_i}) + t_{v_i}) + 2|s - s_i| \\ &\leq \alpha(\mathsf{height}(v) - 1) \cdot (\mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*}) + 3t_{v_i}) + 2|s - s_i^*|.\end{aligned}$$

In the last step we used that $\mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*})$ differs from $\mathsf{ED}(X_{v_i}, Y_{v_i,s_i})$ by an additive error of t_{v_i}, and the same is true for $2|s - s_i^*|$ and $2|s - s_i|$. Next, we apply the Precision Sampling Lemma with $a_i = \mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*}) + 2\alpha^{-1}|s - s_i^*|$ and parameters

- $\delta = 1/\operatorname{poly}(n)$,
- $\alpha = \alpha(\mathsf{height}(v) - 1)$,
- $\beta = \alpha t_v$.

Recall that $t_{v_i} = t_v \cdot u_{v_i}/3$, thus by definition we have that $\widetilde{a}_{i,s} \leq \alpha \cdot a_i + \beta \cdot u_{v_i}$. Therefore, the Precision Sampling Lemma states that with high probability the recovery algorithm returns

$$\begin{aligned}\textsc{Recover}(\cdot) &\leq 2\alpha \cdot \left(\sum_{i=1}^{B} a_i\right) + \beta \\ &\leq 2\alpha \left(\sum_{i=1}^{B} \mathsf{ED}(X_{v_i}, Y_{v_i,s_i^*})\right) + 2\left(\sum_{i=1}^{B} 2|s - s_i^*|\right) + \alpha t_v \\ &\leq 4\alpha \cdot \mathsf{ED}(X_v, Y_{v,s}) + 2B \cdot \mathsf{ED}(X_v, Y_{v,s}) + \alpha t_v \\ &\leq 4(\alpha + B) \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v) \\ &\leq \alpha(\mathsf{height}(v)) \cdot (\mathsf{ED}(X_v, Y_{v,s}) + t_v),\end{aligned}$$

where for the third inequality we applied the bounds in (2) and (3) and in the last step we used the definition of $\alpha(\mathsf{height}(v)) = O(B) \cdot 2^{O(\mathsf{height}(v))}$ (for sufficiently large hidden constants). Since the tree has height $\log_B(n)$ and $B = \exp(\widetilde{\Theta}(\sqrt{\log n}))$, the overall approximation factor is bounded by $\alpha(\log_B(n)) = O(B) \cdot 2^{O(\log_B(n))} = n^{o(1)}$, as claimed. ◀

Next, we analyze the running time of Algorithm 2. It turns out that the bottleneck is the computation in Algorithm 2, and a naive implementation would be too slow for our purposes. For that reason, we use the following lemma for an improved implementation of Algorithm 2. The lemma is a generalization of [15, Lemma 10].

▶ **Lemma 17** (Range Minimum Problem). *Let T, T' be sets of integers (given in sorted order), and let $(b_{s'})_{s' \in T'}$ be given. There is an $O(|T| + |T'|)$-time algorithm to compute $(a_s)_{s \in T}$ defined by*

$$a_s = \min_{s' \in T'} b_{s'} + 2 \cdot |s - s'|.$$

Proof. The idea is to compute auxiliary values

$$a_s^{\leq} = \min_{\substack{s' \in T' \\ s' \leq s}} b_{s'} + 2s - 2s',$$

$$a_s^{\geq} = \min_{\substack{s' \in T' \\ s' \geq s}} b_{s'} - 2s + 2s',$$

as then returning $a_s = \min(a_s^{\leq}, a_s^{\geq})$ is correct. We show how to compute $a_s^{\leq}$ (for all s); the values $a_s^{\geq}$ are symmetric. Let $T = \{ s_1 < \cdots < s_{|T|} \}$. We evaluate the base case $a_{s_1}^{\leq}$ naively. We then compute $a_{s_i}^{\leq}$ for all $i = 2, \ldots, |T|$ as follows:

$$a_{s_i}^{\leq} = \min \left\{ a_{s_{i-1}}^{\leq} + 2s_i - 2s_{i-1}, \min_{s_{i-1} < s' \leq s_i} b_{s'} + 2s_i - 2s' \right\}.$$

For the correctness, we distinguish two cases. Let $s' \leq s_i$ be the index which attains the minimum in the definition of $a_{s_i}^{\leq}$. On the one hand, if $s' \leq s_{i-1}$, then $a_{s_i}^{\leq} = a_{s_{i-1}}^{\leq} + 2s_i - 2s_{i-1}$ and thus the first term in the minimum is correct. On the other hand, if $s' > s_{i-1}$, then the second term in the minimum is correct by definition. In order to compute $(a_s^{\leq})_s$ we sweep from left to right over all values in T and T' exactly once, hence the running time can be bounded by $O(|T| + |T'|)$. ◀

▶ **Lemma 18** (Running Time of Algorithm 2). *Assume that* $\mathsf{ED}(X, Y) \leq k$. *If only the string* Y *is preprocessed, then Algorithm 2 runs in expected time* $n^{1+o(1)}/k + kn^{o(1)}$. *If both strings* X, Y *are preprocessed, then Algorithm 2 runs in expected time* $kn^{o(1)}$.

Proof. We first bound the running time of a single execution of Algorithm 2 (ignoring the cost of recursive calls). The computation in Algorithm 2 merely compares single characters and therefore takes time $|S_v| = O^*(k/t_v)$. By Lemma 13, the call to Algorithm 1 takes expected time $O^*(|X_v|/t_v + k/t_v)$ (for one-sided preprocessing) or $O^*(k/t_v)$ (for two-sided preprocessing). Each iteration of the loop in Algorithms 2–2 is dominated by the computation in Algorithm 2 (ignoring the recursive calls in Algorithm 2). Using Lemma 17 with $T = S_v$ and $T' = S_{v_i}$, this step takes time $O(|S_v| + |S_{v_i}|)$, and thus the loop runs in time $O(B \cdot |S_v| + \sum_i |S_{v_i}|) = O^*(k/t_v + \sum_i k/t_{v_i})$. In all of these bounds we can bound $1/t_v$ by $n^{o(1)}/k$ in expectation, according to Lemma 5, so the total expected time per node becomes $n^{o(1)} \cdot (|X_v|/k + 1)$ (for one-sided preprocessing) or $n^{o(1)}$ (for two-sided preprocessing).

To account for the recursive calls, we first use Lemma 14 to bound the number of recursive calls by $O(k \log n)$. Let $v_1, \ldots, v_m$ (with $m = O(k \log n)$) denote all nodes for which Algorithm 2 is recursively called. Then the expected running time for one-sided preprocessing can be bounded by

$$\begin{aligned} n^{o(1)} \cdot \sum_{i=1}^{m} \left(\frac{|X_{v_i}|}{k} + 1 \right) &\leq kn^{o(1)} + n^{o(1)} \cdot \sum_{d=1}^{\log_B(n)} \sum_{\substack{i=1 \\ \mathsf{depth}(v_i)=d}}^{m} \frac{|X_{v_i}|}{k} \\ &\leq kn^{o(1)} + n^{o(1)} \cdot \sum_{d=1}^{\log_B(n)} \frac{n}{k} \\ &\leq kn^{o(1)} + \frac{n^{1+o(1)}}{k}. \end{aligned}$$

Here we used that across any level in the precision tree, the strings X_v form a partition of X into consecutive substrings. In the same way we can bound the running time for two-sided preprocessing by $kn^{o(1)}$. ◀

4.6 Proof of the Main Theorems

We are finally ready to prove our main theorems.

Proof of Theorems 1 and 2. We assume that either only Y (for Theorem 1) or both X and Y (for Theorem 2) are preprocessed by Lemmas 6 and 7. We will solve the (k, K)-gap edit distance problem, for some parameter K to be picked later, by running Algorithm 2 (with input $v = \mathsf{root}(T)$ and $s = 0$, so in particular $2|s| \leq k$). For the correctness proof, we apply Lemma 16 to the following two cases:

- If $\mathsf{ED}(X,Y) \le k$, then $\widetilde{\mathsf{ED}}(X,Y) \le n^{o(1)} \cdot (\mathsf{ED}(X,Y)+t) \le n^{o(1)} \cdot k$.
- If $\mathsf{ED}(X,Y) \ge K$, then $\widetilde{\mathsf{ED}}(X,Y) \ge n^{-o(1)} \cdot \mathsf{ED}(X,Y) - t \ge n^{-o(1)} \cdot K$.

By setting $K = k \cdot n^{o(1)}$ for a sufficiently large subpolynomial factor, we can distinguish the two cases based on the outcome $\widetilde{\mathsf{ED}}(X,Y)$.

Next, we analyze the running time. If $\mathsf{ED}(X,Y) \le k$, then the algorithm runs in expected time $n^{1+o(1)}/k + kn^{o(1)}$ or $kn^{o(1)}$, respectively; see Lemma 18. By Markov's inequality, the algorithm respects these time bounds with constant probability. We may therefore interrupt the algorithm after it exceeds its time budget and report "$\mathsf{ED}(X,Y) \ge K$" in case of an interruption. We can boost the success probability to $1 - 1/\operatorname{poly}(n)$ by running the algorithm $O(\log n)$ times in parallel and reporting the majority answer. (This also means that the preprocessing is repeated $O(\log n)$ times with independently sampled precision trees.) ◀

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. 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, pages 59–78. IEEE Computer Society, 2015.

2 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. 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, pages 375–388. ACM, 2016.

3 Alexandr Andoni. High frequency moments via max-stability. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, ICASSP '17, pages 6364–6368. IEEE, 2017.

4 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. 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, pages 377–386. IEEE Computer Society, 2010.

5 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 363–372. IEEE Computer Society, 2011.

6 Alexandr Andoni and Negev Shekel Nosatzki. 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, pages 990–1001. IEEE Computer Society, 2020.

7 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012.

8 Arturs Backurs and Piotr Indyk. 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, pages 51–58. ACM, 2015.

9 Ziv Bar-Yossef, S. Thathachar Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the 45th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '04, pages 550–559. IEEE Computer Society, 2004.

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

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

12 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '16, pages 51–60. IEEE Computer Society, 2016.

13 Joshua Brakensiek, Moses Charikar, and Aviad Rubinstein. A simple sublinear algorithm for gap edit distance, 2020. `arXiv:2007.14368`.

14 Joshua Brakensiek and Aviad Rubinstein. 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, pages 685–698. ACM, 2020.

15 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. Almost-optimal sublinear-time edit distance in the low distance regime. In *Proceedings of the 54rd ACM Symposium on Theory of Computing*, STOC '22. ACM, 2022.

16 Karl Bringmann and Marvin Künnemann. 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, pages 79–97. IEEE Computer Society, 2015.

17 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *J. ACM*, 67(6), 2020.

18 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th ACM Symposium on Theory of Computing*, STOC '16, pages 712–725. ACM, 2016.

19 Elazar Goldenberg, Tomasz Kociumaka, Robert Krauthgamer, and Barna Saha. Gap edit distance via non-adaptive queries: Simple and optimal, 2021. `arXiv:2111.12706`.

20 Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, FOCS '20, pages 1101–1120. IEEE Computer Society, 2019.

21 Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. Does preprocessing help in fast sequence comparisons? In *Proceedings of the 52nd ACM Symposium on Theory of Computing*, STOC '20, pages 657–670. ACM, 2020.

22 Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In *Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '19, pages 334–347. IEEE Computer Society, 2019.

23 Hossein Jowhari. Efficient communication protocols for deciding edit distance. In *Proceedings of the 20th Annual European Conference on Algorithms*, ESA '12, pages 648–658. Springer, 2012.

24 Tomasz Kociumaka and Barna Saha. Sublinear-time algorithms for computing & embedding gap edit distance. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science*, FOCS '20, pages 1168–1179. IEEE Computer Society, 2020.

25 Michal Koucký and Michael E. Saks. 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, pages 699–712. ACM, 2020.

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

27 Sebastian Wandelt, Dong Deng, Stefan Gerdjikov, Shashwat Mishra, Petar Mitankin, Manish Patil, Enrico Siragusa, Alexander Tiskin, Wei Wang, Jiaying Wang, and Ulf Leser. State-of-the-art in string similarity search and join. *SIGMOD Rec.*, 43(1):64–76, 2014.

A Proofs of Lemmas 3 and 5

In this section we provide proofs of Lemmas 3 and 5.

▶ **Lemma 3** (Divide and Conquer). *Let X, Y be length-n strings, and let $0 = j_0 < \cdots < j_B = n$. We write $X_i = X[\, j_{i-1} \,..\, j_i \,]$ and $Y_{i,s} = Y[\, j_{i-1} + s \,..\, j_i + s \,]$.*

- *For all shifts $s_1, \ldots, s_B$ we have that $\mathsf{ED}(X,Y) \le \sum_i \mathsf{ED}(X_i, Y_{i,s_i}) + 2|s_i|$.*
- *There are shifts $s_1, \ldots, s_B$ with $\sum_i \mathsf{ED}(X_i, Y_{i,s}) \le 2\mathsf{ED}(X,Y)$ and $2|s_i| \le \mathsf{ED}(X,Y)$ for all i.*

Proof. Lower Bound. Let us write $Y_i = Y_{i,0} = Y[\, j_{i-1} \,..\, j_i \,]$ (in analogy to the notation X_i). It is clear that $\mathsf{ED}(Y_{i,s_i}, Y_i) \le 2|s_i|$ by deleting and inserting at most $|s_i|$ symbols. Therefore, by several applications of the triangle inequality we have

$$\begin{aligned}\sum_{i=1}^{B} \mathsf{ED}(X_i, Y_{i,s_i}) + 2|s_i| &\ge \sum_{i=1}^{B} \mathsf{ED}(X_i, Y_{i,s_i}) + \mathsf{ED}(Y_{i,s_i}, Y_i) \\ &\ge \sum_{i=1}^{B} \mathsf{ED}(X_i, Y_i) \\ &\ge \mathsf{ED}(X,Y).\end{aligned}$$

Upper Bound. For the upper bound, let $A : \{0, \ldots, n\} \to \{0, \ldots, n\}$ denote an optimal alignment between X and Y (as defined in Section 2). Then

$$\mathsf{ED}(X,Y) = \sum_{i=1}^{B} \mathsf{ED}(X[\, j_{i-1} \,..\, j_i \,], Y[\, A(j_{i-1}) \,..\, A(j_i) \,]). \tag{4}$$

We pick $s_i = A(j_{i-1}) - j_{i-1}$. The first step is to prove that $2|s_i| \le \mathsf{ED}(X,Y)$, thereby proving the second item of the upper bound. To see this, we first express $\mathsf{ED}(X,Y)$ as the sum of two edit distances $\mathsf{ED}(X[\, 0 \,..\, j_{i-1} \,], Y[\, 0 \,..\, A(j_{i-1}) \,])$ and $\mathsf{ED}(X[\, j_{i-1} \,..\, n \,], Y[\, A(j_{i-1}) \,..\, n \,])$, using that A is an optimal alignment. Now, since the edit distance of two strings A, B is always at least $|\,|A| - |B|\,|$, we conclude that $\mathsf{ED}(X,Y) \ge 2|s_i|$.

The next and final part is to prove that $\sum_i \mathsf{ED}(X_i, Y_{i,s_i}) \le 2\mathsf{ED}(X,Y)$.

$$\begin{aligned}\mathsf{ED}(X_i, Y_{i,s_i}) &= \mathsf{ED}(X[\, j_{i-1} \,..\, j_i \,], Y[\, A(j_{i-1}) \,..\, A(j_{i-1}) + j_i - j_{i-1} \,]) \\ &\le \mathsf{ED}(X[\, j_{i-1} \,..\, j_i \,], Y[\, A(j_{i-1}) \,..\, A(j_i) \,]) + |(A(j_i) - A(j_{i-1})) - (j_i - j_{i-1})| \\ &\le 2 \cdot \mathsf{ED}(X[\, i_{j-1} \,..\, i_j \,], Y[\, A(j_{i-1}) \,..\, A(j_i) \,]),\end{aligned}$$

where in the last step we again used that the edit distance between two strings is at least their length difference. It follows that

$$\sum_{i=1}^{B} \mathsf{ED}(X_i, Y_{i,s_i}) \le 2 \sum_{i=1}^{B} \mathsf{ED}(X[\, j_{i-1} \,..\, j_i \,], Y[\, A(j_{i-1}) \,..\, A(j_i) \,]) \le 2\mathsf{ED}(X,Y). \qquad \blacktriangleleft$$

▶ **Lemma 5** (Expected Precision). *Let T be a B-ary precision tree with initial tolerance t and $B = \exp(\widetilde{\Theta}(\sqrt{\log n}))$, and let v be a node in T. Then, conditioned on a high-probability event E, it holds that*

$$\mathbf{E}\left(\frac{1}{t_v} \,\middle|\, E \right) \le \frac{(\log n)^{O(\mathsf{depth}(v))}}{t} \le \frac{n^{o(1)}}{t}.$$

Proof. Recall that we assign $t_v = t_{\mathsf{parent}(v)} \cdot u_v/3$ for all non-root nodes in the precision tree, where the samples u_v are independent of each other and sampled from $\mathrm{Exp}(\lambda)$, the exponential distribution with parameter $\lambda = O(\log n)$. Recall that the exponential distribution has probability density function $f(x) = \lambda e^{-\lambda x}$, and thus

$$\mathop{\mathbf{P}}_{u\sim\mathrm{Exp}(\lambda)}(u \le x) = \int_{u=0}^{x} \lambda e^{-\lambda x} du = 1 - e^{-\lambda x} \le \lambda x.$$

We let E denote the event that $u_v \ge 1/\operatorname{poly}(n)$ for all nodes v. For any specific node v we have $u_v \ge 1/\operatorname{poly}(n)$ with high probability, and thus by a union bound E happens with high probability as well.

Next, we prove that conditioned on E, the expectation of $1/u$ for $u \sim \mathrm{Exp}(\lambda)$ is small:

$$\begin{aligned}
\mathop{\mathbf{E}}_{u\sim\mathrm{Exp}(\lambda)}(1/u \mid E) &\le \frac{1}{\mathbf{P}(E)} \cdot \int_{u=1/\operatorname{poly}(n)}^{\infty} \frac{1}{u} \cdot \lambda e^{-\lambda u} du \\
&\le \frac{1}{\mathbf{P}(E)} \cdot \left(\int_{u=1/\operatorname{poly}(n)}^{1} \frac{1}{u} \cdot \lambda e^{-\lambda u} du + \int_{u=1}^{\infty} \frac{1}{u} \cdot \lambda e^{-\lambda u} du \right) \\
&\le \frac{1}{\mathbf{P}(E)} \cdot O(\lambda \log n + 1) \\
&\le O(\log^2 n).
\end{aligned}$$

We now prove the claimed bound. Fix some node v and let $\mathsf{root}(T) = v_1, \ldots, v_{\mathsf{depth}(v)} = v$ denote the root-to-v path in the precision tree. Recall that $t_v = t \cdot (u_{v_1}/3) \cdots (u_{v_{\mathsf{depth}(v)}}/3)$. We have

$$\begin{aligned}
\mathbf{E}\left(\frac{1}{t_v} \,\middle|\, E\right) &= \frac{1}{t} \cdot \left(3 \cdot \mathop{\mathbf{E}}_{u\sim\mathrm{Exp}(\lambda)}\left(\frac{1}{u} \,\middle|\, E\right)\right)^{\mathsf{depth}(v)} \\
&\le \frac{(\log n)^{O(\mathsf{depth}(v))}}{t} \\
&\le \frac{n^{o(1)}}{t}.
\end{aligned}$$

For the last inequality, we used that $\mathsf{depth}(v) \le \log_B(n) = \widetilde{O}(\sqrt{\log n})$, and therefore the overhead becomes $\exp(\widetilde{O}(\sqrt{\log n})) = n^{o(1)}$. ◀