

# Sublinear-Time Algorithms for Computing & Embedding Gap Edit Distance

Tomasz Kociumaka<sup>\*1,2</sup> and Barna Saha<sup>†2</sup>

<sup>1</sup>Bar-Ilan University, Ramat Gan, Israel

<sup>2</sup>University of California, Berkeley, USA  
kociumaka@berkeley.edu, barnas@berkeley.edu

## Abstract

In this paper, we design new sublinear-time algorithms for solving the gap edit distance problem and for embedding edit distance to Hamming distance. For the gap edit distance problem, we give an  $\tilde{O}(\frac{n}{k} + k^2)$ -time greedy algorithm that distinguishes between length- $n$  input strings with edit distance at most  $k$  and those with edit distance exceeding  $(3k + 5)k$ . This is an improvement and a simplification upon the result of Goldenberg, Krauthgamer, and Saha [FOCS 2019], where the  $k$  vs  $\Theta(k^2)$  gap edit distance problem is solved in  $\tilde{O}(\frac{n}{k} + k^3)$  time. We further generalize our result to solve the  $k$  vs  $k'$  gap edit distance problem in time  $\tilde{O}(\frac{nk}{k'} + k^2 + \frac{k^2}{k'}\sqrt{nk})$ , strictly improving upon the previously known bound  $\tilde{O}(\frac{nk}{k'} + k^3)$ . Finally, we show that if the input strings do not have long highly periodic substrings, then already the  $k$  vs  $(1 + \varepsilon)k$  gap edit distance problem can be solved in sublinear time. Specifically, if the strings contain no substring of length  $\ell$  with period at most  $2k$ , then the running time we achieve is  $\tilde{O}(\frac{n}{\varepsilon^2 k} + k^2 \ell)$ .

We further give the first sublinear-time probabilistic embedding of edit distance to Hamming distance. For any parameter  $p$ , our  $\tilde{O}(\frac{n}{p})$ -time procedure yields an embedding with distortion  $\mathcal{O}(kp)$ , where  $k$  is the edit distance of the original strings. Specifically, the Hamming distance of the resultant strings is between  $\frac{k-p+1}{p+1}$  and  $\mathcal{O}(k^2)$  with good probability. This generalizes the linear-time embedding of Chakraborty, Goldenberg, and Koucký [STOC 2016], where the resultant Hamming distance is between  $\frac{k}{2}$  and  $\mathcal{O}(k^2)$ . Our algorithm is based on a random walk over samples, which we believe will find other applications in sublinear-time algorithms.

## 1 Introduction

The edit distance, also known as the Levenshtein distance [29], is a basic measure of sequence similarity. For two strings  $X$  and  $Y$ , the edit distance  $\text{ED}(X, Y)$  is defined as the minimum number of character insertions, deletions and substitutions required to transform  $X$  into  $Y$ . A natural dynamic programming computes the edit distance of two strings of total length  $n$  in  $\mathcal{O}(n^2)$  time. While running a quadratic-time algorithm is prohibitive for many applications, the Strong Exponential Time Hypothesis (SETH) [25] implies that there is no truly subquadratic-time algorithm that computes edit distance exactly [6].

The last two decades have seen a surge of interest in designing fast approximation algorithms for edit distance computation [2, 4, 5, 7, 8, 9, 11, 14, 16, 22, 26]. A breakthrough result of Chakraborty, Das, Goldenberg, Koucký, and Saks provided the first constant-factor

<sup>\*</sup>Supported by ISF grants no. 1278/16 and 1926/19, by a BSF grant no. 2018364, and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (agreement no. 683064).

<sup>†</sup>Supported by NSF grants no. 1652303 (NSF CAREER), 1934846 (NSF HDR TRIPODS), and 1909046, and by an Alfred P. Sloan fellowship award.

approximation of edit distance in truly subquadratic time [16]. Nearly a decade earlier, Andoni, Krauthgamer, and Onak showed a polylogarithmic-factor approximation for edit distance in near-linear time [2]. Recently, the result of [16] was improved to a constant-factor approximation in near-linear time: initially by Brakensiek and Rubinfeld [14] as well as Koucký and Saks [26] for the regime of near-linear edit distance, and then by Andoni and Nosatzki [4] for the general case. Efficient algorithms for edit distance have also been developed in other models, such as in the quantum and massively parallel framework [11], and when independent preprocessing of each string is allowed [23].

In this paper, we focus on sublinear-time algorithms for edit distance, the study of which was initiated by Batu, Ergün, Kilian, Magen, Rashkodnikova, Rubinfeld, and Sami [8], and then continued in [3, 5, 22, 32]. Here, the goal is to distinguish, in time sublinear in  $n$ , whether the edit distance is at most  $k$  or strictly above  $k'$  for some  $k' \geq k$ . This is known as the *( $k$  vs  $k'$ ) gap edit distance problem*. In computational biology, before an in-depth comparison of new sequences is performed, a quick check to eliminate sequences that are not highly similar can save a significant amount of resources [20]. In text corpora, a super-fast detection of plagiarism upon arrival of a new document can save both time and space. In these applications,  $k$  is relatively small and a sublinear-time algorithm with  $k'$  relatively close to  $k$  could be very useful.

## Results on Gap Edit Distance

The algorithm of Batu et al. [8] distinguishes between  $k = n^{1-\Omega(1)}$  and  $k' = \Omega(n)$  in time  $\mathcal{O}(\max(\frac{k^2}{n}, \sqrt{k}))$ . However, their algorithm crucially depends on  $k' = \Omega(n)$  and cannot distinguish between, say,  $k = n^{0.01}$  and  $k' = n^{0.99}$ . A more recent algorithm by Andoni and Onak [5] resolves this issue and can distinguish between  $k$  and  $k' \geq k \cdot n^{\Omega(1)}$  in time  $\mathcal{O}(\frac{n^{2+o(1)}k}{(k')^2})$ . However, if we want to distinguish between  $k$  and  $k' = \Theta(k^2)$ , then the algorithm of [5] achieves sublinear time only when  $k = \omega(n^{1/3})$ . (Setting  $k' = \Theta(k^2)$  yields a natural test case for the gap edit distance problem since the best that one can currently distinguish in linear time is  $k$  vs  $\Theta(k^2)$  [28].) In a recent work, Goldenberg, Krauthgamer, and Saha [22] gave an algorithm solving *quadratic gap edit distance problem* in  $\tilde{\mathcal{O}}(\frac{n}{k} + k^3)$  time<sup>1</sup>, thereby providing a truly sublinear-time algorithm as long as  $n^{\Omega(1)} \leq k \leq n^{1/3-\Omega(1)}$ .

Bar-Yossef, Jayram, Krauthgamer, and Kumar [7] introduced the gap edit distance problem and solved the quadratic gap edit distance problem for non-repetitive strings. Their algorithm computes a constant-size sketch but still requires a linear-time pass over the data. This result was later generalized to arbitrary sequences [17] via embedding edit distance into Hamming distance, but again in linear time. Nevertheless, already the algorithm of Landau and Vishkin [28] computes the edit distance exactly in  $\mathcal{O}(n + k^2)$  time, and thus also solves the quadratic gap edit distance problem in linear time. Given the prior works, Goldenberg et al. [22] raised a question whether it is possible to solve the quadratic gap edit problem in sublinear time for all  $k \geq n^{\Omega(1)}$ . In particular, the running times of the algorithms of Goldenberg et al. [22] and Andoni and Onak [5] algorithms meet at  $k \approx n^{1/3}$ , when they become nearly-linear. In light of the  $\mathcal{O}(n + k^2)$ -time exact algorithm [28], the presence of a  $k^3$  term in the time complexity of [22] is undesirable, and it is natural to ask if the dependency can be reduced. In particular, if the polynomial dependency on  $k$  can be reduced to  $k^2$ , then, for  $k = O(n^{1/3})$ , the contribution of that term is negligible compared to  $\frac{n}{k}$ .

**Quadratic Gap Edit Distance** We give a simple greedy algorithm solving the quadratic gap edit distance problem in  $\tilde{\mathcal{O}}(\frac{n}{k} + k^2)$  time. This resolves an open question posed in [22] as to whether a sublinear-time algorithm for the quadratic gap edit distance is possible for  $k = n^{1/3}$ . Our algorithm improves upon the main result of [22], also providing a conceptual simplification.

---

<sup>1</sup>The  $\tilde{\mathcal{O}}$  notation hides factors polylogarithmic in the input size and, in case of Monte Carlo randomized algorithms, in the inverse error probability.

**$k$  vs  $k'$  Gap Edit Distance** Combining the greedy approach with the structure of computations in [28], we can solve the  $k$  vs  $k'$  gap edit distance problem in  $\tilde{O}(\frac{nk}{k'} + k^2 + \frac{k^2}{k'}\sqrt{kn})$  time. For all values of  $k'$  and  $k$ , this is at least as fast as the  $\tilde{O}(\frac{nk}{k'} + k^3)$  time bound of [22].

**$k$  vs  $(1 + \varepsilon)k$  Gap Edit Distance** We can distinguish edit distance at most  $k$  and at least  $(1 + \varepsilon)k$  in  $\tilde{O}(\frac{n}{\varepsilon^2 k} + \ell k^2)$  time as long as there is no length- $\ell$  substring with period at most  $2k$ . Previously, sublinear-time algorithms for distinguishing  $k$  vs  $(1 + \varepsilon)k$  were only known for the very special case of Ulam distance, where each character appears at most once in each string [3, 32]. Note that not only we can allow character repetition, but we get an  $(1 + \varepsilon)$ -approximation as long as the same repetitive structure does not continue for more than  $\ell$  consecutive positions or has shortest period larger than  $2k$ . This is the case with most text corpora and for biological sequences with interspersed repeats.

## Embedding Edit Distance to Hamming Distance

Along with designing fast approximation algorithms for edit distance, a parallel line of works investigated how edit distance can be embedded into other metric spaces, especially to the Hamming space [1, 9, 17, 18, 34]. Indeed, such embedding results have led to new approximation algorithms for edit distance (e.g., the embedding of [9, 34] applied in [5, 9]), as well as new streaming algorithms and document exchange protocols (e.g., the embedding of [17] applied in [10, 17]). In particular, Chakraborty, Goldenberg, and Koucký [17] provided a probabilistic embedding of edit distance to Hamming distance with linear distortion. Their algorithm runs in linear time, and if the edit distance between two input sequences is  $k$ , then the Hamming distance between the resultant sequences is between  $\frac{k}{2}$  and  $\mathcal{O}(k^2)$  with good probability. The embedding is based on performing an interesting one-dimensional random walk which had also been used previously to design fast approximation algorithms for a more general *language edit distance* problem [35]. So far, we are not aware of any sublinear-time metric embedding algorithm from edit distance to Hamming distance. In this paper, we design one such algorithm.

**Random Walk over Samples** We show that it is possible to perform a random walk similar to [35, 17] over a suitably crafted sequence of samples. This leads to the first sublinear-time algorithm for embedding edit distance to Hamming distance: Given any parameter  $p = \Omega(\log n)$ , our embedding algorithm processes any length- $n$  string in  $\tilde{O}(\frac{n}{p})$  time and guarantees that (with good probability) the Hamming distance of the resultant strings is between  $\frac{k-p+1}{p+1}$  and  $\mathcal{O}(k^2)$ , where  $k$  is the edit distance of the input strings. That is, we maintain the same expansion rate as [17] and allow additional contraction by a factor roughly  $p$ . As the algorithm of [17] has been very influential (see its applications in [10, 13, 24, 36]), we believe the technique of random walk over samples will also find other usages in designing sublinear-time and streaming algorithms.

## Technical Overview

The classic Landau–Vishkin algorithm [28] tests whether  $\text{ED}(X, Y) \leq k$  in  $\mathcal{O}(n + k^2)$  time, where  $n = |X| + |Y|$ . The algorithm fills a dynamic-programming table with cells  $d_{i,j}$  for  $i \in [0..k]$  and  $j \in [-k..k]$ ,<sup>2</sup> aiming at  $d_{i,j} = \max\{x : \text{ED}(X[0..x], Y[0..x+j]) \leq i\}$ . In terms of the table of all distances  $\text{ED}(X[0..x], Y[0..y])$ , each value  $d_{i,j}$  can be interpreted as (the row of) the farthest cell on the  $j$ th *diagonal* with value  $i$  or less. After preprocessing  $X$  and  $Y$  in linear time, the cells  $d_{i,j}$  can be filled in  $\mathcal{O}(1)$  time each, which results in  $\mathcal{O}(n + k^2)$  time in total.

Our algorithm for the quadratic gap edit distance problem follows the basic framework of [28]. However, instead of computing  $\Theta(k^2)$  values  $d_{i,j}$ , it only computes  $\Theta(k)$  values  $d_i$  with  $i \in [0..k]$ . Here,  $d_i$  can be interpreted as a relaxed version of  $\max_{j=-k}^k d_{i,j}$ , allowing for a

<sup>2</sup>For  $\ell, r \in \mathbb{Z}$ , we denote  $[\ell..r) = \{j \in \mathbb{Z} : \ell \leq j < r\}$  and  $[\ell..r] = \{j \in \mathbb{Z} : \ell \leq j \leq r\}$ .

factor- $\mathcal{O}(k)$  underestimation of the number of edits  $i$ . Now, it suffices to test whether  $d_k = |X|$ , because  $d_k = |X|$  holds if  $\text{ED}(X, Y) \leq k$  and, conversely,  $\text{ED}(X, Y) = \mathcal{O}(k^2)$  holds if  $d_k = |X|$ . In addition to uniform sampling at a rate of  $\tilde{\mathcal{O}}(\frac{1}{k+1})$ , identifying each of these  $\mathcal{O}(k)$  values  $d_i$  requires reading  $\tilde{\mathcal{O}}(k)$  extra characters. This yields a total running time of  $\tilde{\mathcal{O}}(\frac{n}{k} + k^2)$ .

Our algorithm not only improves upon the main result of Goldenberg et al. [22], but also provides a conceptual simplification. Indeed, the algorithm of [22] also utilizes the high-level structure of [28], but it identifies all  $\Theta(k^2)$  values  $d_{i,j}$  (relaxed to allow for factor- $\mathcal{O}(k)$  underestimations), paying extra  $\tilde{\Theta}(k)$  time per each value. In order to do so, the algorithm follows a more complex row-by-row approach of an online version [27] of the Landau–Vishkin algorithm.

For the general  $k$  vs  $\Theta(k')$  gap edit distance problem, greedily computing the values  $d_i$  for  $i \in [0..k]$  is sufficient only for  $k' = \Omega(k^2)$ , when the time complexity becomes  $\tilde{\mathcal{O}}(\frac{nk}{k'} + k^2)$  if we simply decrease the sampling rate to  $\tilde{\mathcal{O}}(\frac{k}{k'})$ . Otherwise, each shift between diagonals  $j \in [-k..k]$ , which happens for each  $i \in [1..k]$  as we determine  $d_i$  based on  $d_{i-1}$ , involves up to  $2k$  insertions or deletions, whereas to distinguish edit distance  $k$  and  $\Theta(k')$ , we would like to approximate the number of edits within a factor  $\mathcal{O}(\frac{k'}{k})$ . In order to do so, we decompose the entire set of  $2k+1$  diagonals into  $\Theta(\frac{k^2}{k'})$  groups of  $\Theta(\frac{k'}{k})$  consecutive diagonals. Within each of these *wide diagonals*, we compute the (relaxed) maxima of  $d_{i,j}$  following our greedy algorithm. This approximates the true maxima up to a factor- $\mathcal{O}(\frac{k'}{k})$  underestimation of the number of edits. Computing each of the  $\mathcal{O}(k)$  values for each wide diagonal requires reading  $\tilde{\mathcal{O}}(\frac{k'}{k})$  extra characters. Hence, the running time is  $\tilde{\mathcal{O}}(\frac{nk}{k'} + k')$  per wide diagonal and  $\tilde{\mathcal{O}}(\frac{nk^3}{(k')^2} + k^2)$  in total (across the  $\Theta(\frac{k^2}{k'})$  wide diagonals). This bound is incomparable to  $\tilde{\mathcal{O}}(\frac{nk}{k'} + k^3)$  of [22]: While we pay less on the second term, the uniform sampling rate increases. In order to decrease the first term, instead of sampling over each wide diagonal independently, we provide a synchronization mechanism so that the global uniform sampling rate remains  $\tilde{\mathcal{O}}(\frac{k}{k'})$ . This leads to an implementation with running time  $\tilde{\mathcal{O}}(\frac{nk}{k'} + \frac{k^4}{k'})$ , which already improves upon [22]. However, synchronizing only over appropriate smaller groups of wide diagonals, we can achieve the running time of  $\tilde{\mathcal{O}}(\frac{nk}{k'} + k^2 + \frac{k^2}{k'}\sqrt{kn})$ , which subsumes both  $\tilde{\mathcal{O}}(\frac{nk^3}{(k')^2} + k^2)$  and  $\tilde{\mathcal{O}}(\frac{nk}{k'} + \frac{k^4}{k'})$ .

Our algorithm for the  $k$  vs  $(1+\varepsilon)k$  gap edit distance for strings without length- $\ell$  substrings with period at most  $2k$  follows a very different approach, inspired by the existing solutions for estimating the Ulam distance [3, 32]. This method consists of three ingredients. First, we construct decompositions  $X = X_0 \cdots X_m$  and  $Y = Y_0 \cdots Y_m$  into phrases of length  $\mathcal{O}(\ell k)$  such that, if  $\text{ED}(X, Y) \leq k$ , then  $\sum_{i=0}^m \text{ED}(X_i, Y_i) \leq \text{ED}(X, Y)$  holds with good probability (note that  $\text{ED}(X, Y) \leq \sum_{i=0}^m \text{ED}(X_i, Y_i)$  is always true). The second ingredient estimates  $\text{ED}(X_i, Y_i)$  for any given  $i$ . This subroutine is then applied for a random sample of indices  $i$  by the third ingredient, which distinguishes between  $\sum_{i=0}^m \text{ED}(X_i, Y_i) \leq k$  and  $\sum_{i=0}^m \text{ED}(X_i, Y_i) > (1+\varepsilon)k$  relying on the Chernoff bound. The assumption that  $X$  does not contain long periodic substrings is needed only in the first step: it lets us uniquely determine the beginning of the phrase  $Y_i$  assuming that the initial  $\ell$  positions of the phrase  $X_i$  are aligned without mismatches in the optimal edit distance alignment (which is true with good probability for a random decomposition of  $X$ ). We did not optimize the  $\tilde{\mathcal{O}}(\ell k^2)$  term in our running time  $\tilde{\mathcal{O}}(\frac{n}{\varepsilon^2 k} + \ell k^2)$  to keep our implementation of the other two ingredients much simpler than their counterparts in [3, 32].

A simple *random deletion process*, introduced in [35], solves the quadratic gap edit distance problem in linear time. The algorithm simultaneously scans  $X$  and  $Y$  from left to right. If the two currently processed characters  $X[x]$  and  $Y[y]$  match, they are aligned, and the algorithm proceeds to  $X[x+1]$  and  $Y[y+1]$ . Otherwise, one of the characters is deleted uniformly at random (that is, the algorithm proceeds to  $X[x]$  and  $Y[y+1]$  or to  $X[x+1]$  and  $Y[y]$ ). This process can be interpreted as a one-dimensional random walk, and the hitting time of the random walk provides the necessary upper bound on the edit distance. In order to conduct a similar process in sublinear query complexity, we compare  $X[x]$  and  $Y[y]$  with probability  $\tilde{\mathcal{O}}(\frac{1}{p})$  only; otherwise, we simply align  $X[x]$  and  $Y[y]$ . We show that performing this random walk

over samples is sufficient for the  $k$  vs  $\Theta(k^2p)$  gap edit distance problem. Finally, we observe that this *random walk over samples* can be implemented in  $\tilde{O}(\frac{n}{p})$  time by batching iterations.

In order to derive an embedding, we modify the random deletion process so that, after learning that  $X[x] = Y[y]$ , the algorithm uniformly at random chooses to stay at  $X[x]$  and  $Y[y]$  or move to  $X[x+1]$  and  $Y[y+1]$ . This has no impact on the final outcome, but the decision whether the algorithm stays at  $X[x]$  or moves to  $X[x+1]$  can now be made independently of  $Y$ . This allows for an embedding whose shared randomness consists in the set  $S \subseteq [1..3n]$  of iterations  $i$  when  $X[x]$  is accessed and, for each  $i \in S$ , a random function  $h_i : \Sigma \rightarrow \{0, 1\}$ . For each iteration  $i \in S$ , the embedding outputs  $X[x]$  and proceeds to  $X[x + h_i(X[x])]$ . If  $Y$  is processed using the same shared randomness, the two output strings are, with good probability, at Hamming distance between  $\frac{k-p+1}{p+1}$  and  $\mathcal{O}(k^2)$ .

## Organization

After introducing the main notations in Section 2, we describe and analyze our algorithm for the quadratic gap edit distance problem in Section 3. In Section 4, we solve the more general  $k$  vs  $k'$  gap edit distance problem for  $k' = \mathcal{O}(k^2)$ . The  $k$  vs  $(1 + \varepsilon)k$  gap edit distance problem for strings without long periodic substrings is addressed in Section 5. The random walk over samples process is presented in Section 6. Finally, the embedding result is provided in Section 7.

## Further Remarks

A recent independent work [12] uses a greedy algorithm similar to ours and achieves a running time of  $\tilde{O}(\frac{n}{\sqrt{k}})$  for the quadratic gap edit distance problem. This is in contrast to our bound of  $\tilde{O}(\frac{n}{k} + k^2)$ , which is superior for  $k \leq n^{2/5}$ . At the same time, for  $k \geq n^{2/5+o(1)}$ , the algorithm of Andoni and Onak [5] has a better running time  $\mathcal{O}(\frac{n^{2+o(1)}}{k^3})$ . We also remark here that there exists an even simpler algorithm (by now folklore) that has query complexity  $\tilde{O}(\frac{n}{\sqrt{k}})$ . The algorithm samples both sequences  $X$  and  $Y$  independently with probability  $\tilde{\Theta}(\frac{1}{\sqrt{k}})$  so that  $\mathbb{P}[X[x] \text{ and } Y[y] \text{ are sampled}] = \tilde{\Theta}(\frac{1}{k})$  for all  $x \in [0..|X|)$  and  $y \in [0..|Y|)$ . Then, by running the Landau–Vishkin algorithm [28] suitably over the sampled sequences, one can solve the quadratic gap edit distance problem. Still, it remains open to tightly characterize the time and query complexity of the quadratic gap edit distance problem.

## 2 Preliminaries

A *string*  $X$  is a finite sequence of characters from an *alphabet*  $\Sigma$ . The length of  $X$  is denoted by  $|X|$  and, for  $i \in [0..|X|)$ , the  $i$ th character of  $X$  is denoted by  $X[i]$ . A string  $Y$  is a *substring* of a string  $X$  if  $Y = X[\ell]X[\ell+1]\cdots X[r-1]$  for some  $0 \leq \ell \leq r \leq |X|$ . We then say that  $Y$  *occurs* in  $X$  at position  $\ell$ . The set of positions where  $Y$  occurs in  $X$  is denoted  $\text{Occ}(Y, X)$ . The *occurrence* of  $Y$  at position  $\ell$  in  $X$  is denoted by  $X[\ell..r]$  or  $X[\ell..r-1]$ . Such an occurrence is a *fragment* of  $X$ , and it can be represented by (a pointer to)  $X$  and a pair of indices  $\ell \leq r$ . Two fragments (perhaps of different strings) *match* if they are occurrences of the same substring. A fragment  $X[\ell..r]$  is a *prefix* of  $X$  if  $\ell = 0$  and a *suffix* of  $X$  if  $r = |X|$ .

A positive integer  $p$  is a *period* of a string  $X$  if  $X[i] = X[i+p]$  holds for each  $i \in [0..|X|-p]$ . We define  $\text{per}(X)$  to be the smallest period of  $X$ . The following result relates periods to occurrences:

**Fact 2.1** (Breslaurer and Galil [15, Lemma 3.2]). *If strings  $P, T$  satisfy  $|T| \leq \frac{3}{2}|P|$ , then  $\text{Occ}(P, T)$  forms an arithmetic progression with difference  $\text{per}(P)$ .*

**Hamming distance and edit distance** The *Hamming distance* between two strings  $X, Y$  of the same length is defined as the number of mismatches. Formally,  $\text{HD}(X, Y) = |\{i \in [0 \dots |X|] : X[i] \neq Y[i]\}|$ . The edit distance between two strings  $X$  and  $Y$  is denoted  $\text{ED}(X, Y)$ .

**LCE queries** Let  $X, Y$  be strings and let  $k$  be a non-negative integer. For  $x \in [0 \dots |X|]$  and  $y \in [0 \dots |Y|]$ , we define  $\text{LCE}_k^{X,Y}(x, y)$  as the largest integer  $\ell$  such that  $\text{HD}(X[x \dots x+\ell], Y[y \dots y+\ell]) \leq k$  (in particular,  $\ell \leq \min(|X| - x, |Y| - y)$  so that  $X[x \dots x+\ell]$  and  $Y[y \dots y+\ell]$  are well-defined). We also set  $\text{LCE}_k^{X,Y}(x, y) = 0$  if  $x \notin [0 \dots |X|]$  or  $y \notin [0 \dots |Y|]$ .

Our algorithms rely on two notions of *approximate* LCE queries. The first variant is sufficient for distinguishing between  $\text{ED}(X, Y) \leq k$  and  $\text{ED}(X, Y) > k(3k+5)$  in  $\tilde{\mathcal{O}}(\frac{n}{k+1} + k^2)$  time, while a more general algorithm distinguishing between  $\text{ED}(X, Y) \leq k$  and  $\text{ED}(X, Y) > \alpha k$  is based on the more subtle second variant.

**Definition 2.2.** Let  $X, Y$  be strings and let  $k \geq 0$  be an integer. For integers  $x, y$ , we set  $\text{LCE}_{\leq k}^{X,Y}(x, y)$  as any value satisfying  $\text{LCE}_0^{X,Y}(x, y) \leq \text{LCE}_{\leq k}^{X,Y}(x, y) \leq \text{LCE}_k^{X,Y}(x, y)$ .

**Definition 2.3.** Let  $X, Y$  be strings and let  $r > 0$  be a real parameter. For integers  $x, y$ , we set  $\overline{\text{LCE}}_r^{X,Y}(x, y)$  as any *random variable* satisfying the following conditions:

- $\overline{\text{LCE}}_r^{X,Y}(x, y) \geq \text{LCE}_0^{X,Y}(x, y)$ ,
- $\mathbb{P}[\overline{\text{LCE}}_r^{X,Y}(x, y) > \text{LCE}_k^{X,Y}(x, y)] \leq \exp(-\frac{k+1}{r})$  for every integer  $k \geq 0$ .

Note that  $\overline{\text{LCE}}_r^{X,Y}(x, y)$  for  $r = \frac{k+1}{\ln N}$  satisfies the conditions on  $\text{LCE}_{\leq k}^{X,Y}(x, y)$  with probability  $1 - \frac{1}{N}$ . Thus,  $\overline{\text{LCE}}_r$  queries with sufficiently small  $r = \tilde{\Theta}(k+1)$  yield  $\text{LCE}_{\leq k}$  queries with high probability.

### 3 Quadratic Gap Edit Distance

The classic Landau–Vishkin exact algorithm [28] for testing if  $\text{ED}(X, Y) \leq k$  is given below as Algorithm 1. The key property of this algorithm is that  $d_{i,j} = \max\{x : \text{ED}(X[0 \dots x], Y[0 \dots x+j]) \leq i\}$  holds for each  $i \in [0 \dots k]$  and  $j \in [-k \dots k]$ . Since  $\text{LCE}_0^{X,Y}$  queries can be answered in  $\mathcal{O}(1)$  time after linear-time preprocessing, the running time is  $\mathcal{O}(|X| + k^2)$ .

---

**Algorithm 1:** The Landau–Vishkin algorithm [28]

---

```

1 foreach  $i \in [0 \dots k]$  and  $j \in [-k-1 \dots k+1]$  do  $d'_{i,j} := d_{i,j} := -\infty$ ;
2  $d'_{0,0} := 0$ ;
3 for  $i := 0$  to  $k$  do
4   for  $j := -k$  to  $k$  do
5     if  $d'_{i,j} \neq -\infty$  then  $d_{i,j} := d'_{i,j} + \text{LCE}_0^{X,Y}(d'_{i,j}, d'_{i,j} + j)$  ;
6   for  $j := -k$  to  $k$  do  $d'_{i+1,j} := \min(|X|, \max(d_{i,j-1}, d_{i,j} + 1, d_{i,j+1} + 1))$  ;
7 if  $||X| - |Y|| \leq k$  and  $d_{k,|Y|-|X|} = |X|$  then return YES;
8 else return NO;
```

---

The main idea behind the algorithm of Goldenberg et al. [22] is that if  $\text{LCE}_0$  queries are replaced with  $\text{LCE}_{\leq k}$  queries, then the algorithm is still guaranteed to return YES if  $\text{ED}(X, Y) \leq k$  and NO if  $\text{ED}(X, Y) > k(k+2)$ . The cost of their algorithm is  $\tilde{\mathcal{O}}(\frac{1}{k+1}|X|)$  plus  $\tilde{\mathcal{O}}(k)$  per  $\text{LCE}_{\leq k}$  query, which yields  $\tilde{\mathcal{O}}(\frac{1}{k+1}|X| + k^3)$  in total. Nevertheless, their implementation is tailored to the specific structure of LCE queries in Algorithm 1, and it requires these queries to be asked and answered in a certain order, which makes them use an online variant [27] of the Landau–Vishkin algorithm.

An auxiliary result of this paper is that  $\text{LCE}_{\leq k}^{X,Y}(x, y)$  queries with  $|x - y| \leq k$  can be answered in  $\tilde{\mathcal{O}}(k)$  time after  $\tilde{\mathcal{O}}(\frac{1}{k+1}|X|)$  preprocessing, which immediately yields a more modular implementation of the algorithm of [22]. In fact, we show that  $\tilde{\mathcal{O}}(k)$  time is sufficient to answer *all* queries  $\text{LCE}_{\leq k}^{X,Y}(x, y)$  with fixed  $x$  and arbitrary  $y \in [x - k \dots x + k]$ .

Unfortunately, this does not give a direct speed-up, because the values  $d'_{i,j}$  in Algorithm 1 might all be different. However, given that relaxing  $\text{LCE}_0$  queries to  $\text{LCE}_{\leq k}$  queries yields a cost of up to  $k$  mismatches for every  $\text{LCE}_{\leq k}^{X,Y}(x, y)$  query, the algorithm may as well pay  $\mathcal{O}(k)$  further edits (insertions or deletions) to change the *shift*  $j = y - x$  arbitrarily. As a result, we do not need to consider each shift  $j$  separately. This results in a much simpler Algorithm 2.

---

**Algorithm 2:** Simple algorithm

---

```

1  $d'_0 := 0$ ;
2 for  $i := 0$  to  $k$  do
3    $d_i := d'_i + \max_{\delta=-k}^k \text{LCE}_{\leq k}^{X,Y}(d'_i, d'_i + \delta)$ ;
4    $d'_{i+1} := \min(|X|, d_i + 1)$ ;
5 if  $\|X\| - \|Y\| \leq k$  and  $d_k = |X|$  then return YES;
6 else return NO;
```

---

**Lemma 3.1.** *Algorithm 2 returns YES if  $\text{ED}(X, Y) \leq k$  and NO if  $\text{ED}(X, Y) > (3k + 5)k$ .*

*Proof.* We prove two claims on the values  $d'_i$  and  $d_i$ .

**Claim 3.2.** *Each  $i \in [0 \dots k]$  has the following properties:*

- (a)  $\text{ED}(X[0 \dots d'_i], Y[0 \dots y]) \leq (3k + 1)i + k$  for every  $y \in [d'_i - k \dots d'_i + k] \cap [0 \dots |Y|]$ ;
- (b)  $\text{ED}(X[0 \dots d_i], Y[0 \dots y]) \leq (3k + 1)i + 4k$  for every  $y \in [d_i - k \dots d_i + k] \cap [0 \dots |Y|]$ .

*Proof.* We proceed by induction on  $i$ . Our base case is Property (a) for  $i = 0$ . Since  $d'_0 = 0$ , we have  $\text{ED}(X[0 \dots d'_0], Y[0 \dots y]) = y \leq k$  for  $y \in [d'_0 - k \dots d'_0 + k] \cap [0 \dots |Y|]$ .

Next, we shall prove that Property (b) holds for  $i \geq 0$  assuming that Property (a) is true for  $i$ . By definition of  $\text{LCE}_{\leq k}$  queries, we have  $d_i \leq d'_i + \text{LCE}_k^{X,Y}(d'_i, y')$  for some position  $y' \in [d'_i - k \dots d'_i + k] \cap [0 \dots |Y|]$ , and thus  $\text{HD}(X[d'_i \dots d_i], Y[y' \dots y' + d_i - d'_i]) \leq k$ . The assumption yields  $\text{ED}(X[0 \dots d'_i], Y[0 \dots y']) \leq (3k + 1)i + k$ , so we have  $\text{ED}(X[0 \dots d_i], Y[0 \dots y' + d_i - d'_i]) \leq (3k + 1)i + 2k$ . Due to  $|y' + d_i - d'_i - y| \leq 2k$ , we conclude that  $\text{ED}(X[0 \dots d_i], Y[0 \dots y]) \leq (3k + 1)i + 4k$ .

Finally, we shall prove that Property (a) holds for  $i > 0$  assuming that Property (b) is true for  $i - 1$ . Since  $d'_i \leq d_{i-1} + 1$ , the assumption yields  $\text{ED}(X[0 \dots d'_i - 1], Y[0 \dots y - 1]) \leq (3k + 1)(i - 1) + 4k$ , and thus  $\text{ED}(X[0 \dots d'_i], Y[0 \dots y]) \leq 1 + (3k + 1)(i - 1) + 4k = (3k + 1)i + k$ .  $\square$

Thus,  $\text{ED}(X, Y) \leq (3k + 5)k$  if the algorithm returns YES.

**Claim 3.3.** *If  $\text{ED}(X[0 \dots x], Y[0 \dots y]) = i \in [0 \dots k]$  for  $x \in [0 \dots |X|]$  and  $y \in [0 \dots |Y|]$ , then  $x \leq d_i$ .*

*Proof.* We proceed by induction on  $i$ . Both in the base case of  $i = 0$  and the inductive step of  $i > 0$ , we shall prove that  $x \leq d'_i + \max_{\delta=-k}^k \text{LCE}_0^{X,Y}(d'_i, d'_i + \delta)$ . Since  $d_i \geq d'_i + \max_{\delta=-k}^k \text{LCE}_0^{X,Y}(d'_i, d'_i + j)$  holds by definition of  $\text{LCE}_{\leq k}$  queries, this implies the claim.

In the base case of  $i = 0$ , we have  $X[0 \dots x] = Y[0 \dots y]$  and  $d'_0 = 0$ . Consequently,  $x \leq \text{LCE}_0^{X,Y}(0, 0) \leq d'_0 + \max_{\delta=-k}^k \text{LCE}_0^{X,Y}(d'_0, d'_0 + \delta)$ .

For  $i > 0$ , we consider an optimal alignment between  $X[0 \dots x]$  and  $Y[0 \dots y]$ , and we distinguish its maximum prefix with  $i - 1$  edits. This yields positions  $x', x'' \in [0 \dots x]$  and

$y', y'' \in [0..y]$  with  $x'' - x' \in \{0, 1\}$  and  $y'' - y' \in \{0, 1\}$  such that  $\text{ED}(X[0..x'], Y[0..y']) = i - 1$  and  $X[x''..x] = Y[y''..y]$ . The inductive assumption yields  $x' \leq d_{i-1}$ , which implies  $x'' \leq \min(x, d_{i-1} + 1) \leq d'_i$ . Due to  $X[x''..x] = Y[y''..y]$ , we have  $\text{LCE}_0^{X,Y}(x'', y'') \geq x - x''$ . By  $x'' \leq d'_i$ , this implies  $\text{LCE}_0^{X,Y}(d'_i, d'_i + y - x) \geq x - d'_i$ . Since  $|y - x| \leq k$ , we conclude that  $x = d'_i + (x - d'_i) \leq d'_i + \text{LCE}_0^{X,Y}(d'_i, d'_i + y - x) \leq d'_i + \max_{\delta=-k}^k \text{LCE}_0^{X,Y}(d'_i, d'_i + \delta)$ .  $\square$

Hence, the algorithm returns YES if  $\text{ED}(X, Y) \leq k$ .  $\square$

A data structure computing  $\text{LCE}_{\leq k}^{X,Y}(x, y)$  for a given  $x$  and all  $y \in [x - k..x + k]$  is complicated, but a simpler result stated below and proved in Section 3.1 suffices here.

**Proposition 3.4.** *There exists an algorithm that, given strings  $X$  and  $Y$ , an integer  $k \geq 0$ , an index  $i$ , and a range of indices  $J$ , computes  $\ell := \max_{j \in J} \text{LCE}_{\leq k}^{X,Y}(i, j)$ . With high probability, the algorithm is correct and its running time is  $\tilde{O}(\frac{\ell}{k+1} + |J|)$ .*

**Theorem 3.5.** *There exists an algorithm that, given strings  $X$  and  $Y$ , and an integer  $k \geq 0$ , returns YES if  $\text{ED}(X, Y) \leq k$ , and NO if  $\text{ED}(X, Y) > (3k + 5)k$ . With high probability, the algorithm is correct and its running time is  $\tilde{O}(\frac{1}{k+1}|X| + k^2)$ .*

*Proof.* The pseudocode is given in Algorithm 2. Queries  $\text{LCE}_{\leq k}$  are implemented using Proposition 3.4. With high probability, all the queries are answered correctly. Conditioned on this assumption, Lemma 3.1 yields that Algorithm 2 is correct with high probability. It remains to analyze the running time. The cost of instructions other than  $\text{LCE}_{\leq k}$  queries is  $\mathcal{O}(k)$ . By Proposition 3.4, the cost of computing  $d_i$  is  $\tilde{O}(\frac{1}{k+1}(d_i - d'_i) + k)$ . Due to  $0 \leq d'_0 \leq d_0 \leq d'_1 \leq d_1 \leq \dots \leq d'_k \leq d_k \leq |X|$ , this sums up to  $\tilde{O}(\frac{1}{k+1}|X| + k^2)$  across all queries.  $\square$

### 3.1 Proof of Proposition 3.4

Our implementation of  $\max_{j \in J} \text{LCE}_{\leq k}^{X,Y}(i, j)$  queries heavily borrows from [22]. However, our problem is defined in a more abstract way and we impose stricter conditions on the output value, so we cannot use tools from [22] as black boxes; thus, we opt for a self-contained presentation.

On the highest level, in Lemma 3.7, we develop an oracle that, additionally given a threshold  $\ell$ , must return YES if  $\max_{j \in J} \text{LCE}_0^{X,Y}(i, j) \geq \ell$ , must return NO if  $\max_{j \in J} \text{LCE}_k^{X,Y}(i, j) < \ell$ , and may return an arbitrary answer otherwise. The final algorithm behind Proposition 3.4 is then an exponential search on top of the oracle. This way, we effectively switch to the decision version of the problem, which is conceptually and technically easier to handle.

The oracle can be specified as follows: it must return YES if  $X[i..i + \ell] = Y[j..j + \ell]$  for some  $j \in J$ , and NO if  $\text{HD}(X[i..i + \ell], Y[j..j + \ell]) > k$  for every  $j \in J$ . Now, if  $\ell \leq 3|J|$ , then we can afford running a classic exact pattern matching algorithm [31] to verify the YES-condition. Otherwise, we use the same method to filter *candidate positions*  $j \in J$  satisfying  $X[i..i + 3|J|] = Y[j..j + 3|J|]$ . If there is just one candidate position  $j$ , we can continue checking it by comparing  $X[i + s]$  and  $Y[j + s]$  at shifts  $s$  sampled uniformly at random with rate  $\tilde{\Theta}(\frac{1}{k+1})$ .

If there are many candidate positions, Fact 2.1 implies that  $X[i..i + 3|J|]$  is periodic with period  $p \leq |J|$  and that the candidate positions form an arithmetic progression with difference  $p$ . We then check whether  $p$  remains a period of  $X[i..i + \ell]$ , and of  $Y[j..j + \ell]$  for the leftmost candidate  $j$ . Even if either check misses  $\frac{k}{2}$  mismatches with respect to the period, two positive answers guarantee  $\text{HD}(X[i..i + \ell], Y[j..j + \ell]) \leq k$ , which lets us return YES. Thus, the periodicity check (Lemma 3.6) can be implemented by testing individual positions sampled with rate  $\tilde{\Theta}(\frac{1}{k+1})$ .

A negative answer of the periodicity check is witnessed by a single mismatch with respect to the period. However, further steps of the oracle require richer structure as a leverage. Thus, we augment the periodicity check so that it returns a *break*  $B$  with  $|B| = 2|J|$  and  $\text{per}(B) > |J|$ .



---

**Algorithm 3:** FindBreak( $T, q, k$ )

---

```
1  $p := \text{per}(T[0..2q]);$ 
2 if  $p > q$  then return  $T[0..2q];$ 
3 Let  $S \subseteq [0..|T|)$  with elements sampled independently at sufficiently large rate  $\tilde{\Theta}(\frac{1}{k+1})$ ;
4 foreach  $s \in S$  do
5   if  $T[s] \neq T[s \bmod p]$  then
6      $b := 2q; e := s;$ 
7     while  $b < e$  do
8        $m := \lceil \frac{b+e}{2} \rceil;$ 
9       for  $j := m - 2q$  to  $m - 1$  do
10        if  $T[j] \neq T[j \bmod p]$  then  $e := j;$ 
11        if  $e \geq m$  then  $b := m;$ 
12      return  $T(b - 2q..b]$ 
13 return  $\perp$ 
```

---

For this, we utilize a binary-search-based procedure, which is very similar to finding “*period transitions*” in [22]. Whenever  $X[i..i + \ell) = Y[j..j + \ell)$ , the break  $B$  (contained in either string) must match exactly the corresponding fragment in the other string. Since the break is short, we can afford checking this match for every  $j \in J$  (using exact pattern matching again), and since it is not periodic, at most one candidate position  $j \in J$  passes this test. This brings us back to the case with at most one candidate position.

Compared to the outline above, the algorithm described in Lemma 3.7 handles the two main cases (many candidate positions vs one candidate position) in a uniform way, which simplifies formal analysis and implementation details.

We start with the procedure that certifies (approximate) periodicity or finds a break.

**Lemma 3.6.** *There exists an algorithm that, given a string  $T$  an integer  $k \geq 0$ , and a positive integer  $q \leq \frac{1}{2}|T|$ , returns either*

- a length- $2q$  break  $B$  in  $T$  such that  $\text{per}(B) > q$ , or
- $\perp$ , certifying that  $p := \text{per}(T[0..2q]) \leq q$  and  $|\{i \in [0..|T|) : T[i] \neq T[i \bmod p]\}| \leq k$ .

*With high probability, the algorithm is correct and costs  $\tilde{O}(\frac{1}{k+1}|T| + q)$  time.*

*Proof.* A procedure FindBreak( $T, q, k$ ) implementing Lemma 3.6 is given as Algorithm 3.

First, the algorithm computes the shortest period  $p = \text{per}(T[0..2q])$ . If  $p > q$ , then the algorithm returns  $B := T[0..2q]$ , which is a valid break due to  $\text{per}(T) = p > q$ .

Otherwise, the algorithm tries to check if  $\perp$  can be returned. If we say that a position  $i \in [0..|T|)$  is *compatible* when  $T[i] = T[i \bmod p]$ , then  $\perp$  can be returned provided that there are at most  $k$  incompatible positions. The algorithm samples a subset  $S \subseteq [0..|T|)$  with a sufficiently large rate  $\tilde{O}(\frac{1}{k+1})$ . Such sampling rate guarantees that if there are at least  $k + 1$  incompatible positions, then with high probability at least one of them belongs to  $S$ . Consequently, the algorithm checks whether all positions  $s \in S$  are compatible (Line 5), and, if so, returns  $\perp$  (Line 13); this answer is correct with high probability.

In the remaining case, the algorithm constructs a break  $B$  based on an incompatible position  $s$  (Lines 6–12). The algorithm performs a binary search maintaining positions  $b, e$  with  $2q \leq b \leq e < |T|$  such that  $e$  is incompatible and positions in  $[b - 2q..b)$  are all compatible. The initial choice of  $b := 2q$  and  $e := s$  satisfies the invariant because positions in  $[0..2q)$  are all compatible due to  $p = \text{per}(T[0..2q])$ . While  $b < e$ , the algorithm chooses  $m := \lceil \frac{b+e}{2} \rceil$ . If  $[m - 2q..m)$  contains an incompatible position  $j$ , then  $j \geq b$  (because  $j \geq m - 2q \geq b - 2q$  and

positions in  $[b - 2q .. b)$  are all compatible), so the algorithm maintains the invariant setting  $e := j$  for such a position  $j$  (Line 10). Otherwise, positions in  $[m - 2q .. m)$  are all compatible. Due to  $m \leq e$ , this means that the algorithm maintains the invariant setting  $b := m$  (Line 11). Since  $e - b$  decreases by a factor of at least two in each iteration, after  $\mathcal{O}(\log |T|)$  iterations, the algorithm obtains  $b = e$ . Then, the algorithm returns  $B := T(b - 2q .. b]$ .

We shall prove that this is a valid break. For a proof by contradiction, suppose that  $p' := \text{per}(B) \leq q$ . Then,  $p'$  is also period of  $T(b - 2q .. b)$ . Moreover, the invariant guarantees that positions in  $(b - 2q .. b)$  are all compatible, so also  $p$  is a period of  $T(b - 2q .. b)$ . Since  $p + p' - 1 \leq 2q - 1$ , the periodicity lemma [21] implies that also  $\gcd(p, p')$  is a period of  $X(b - 2q .. b)$ . Consequently,  $T[b] = T[b - p'] = T[b - p] = T[(b - p) \bmod p] = T[b \bmod p]$ , i.e.,  $b$  is compatible. However, the invariant assures that  $b$  is incompatible. This contradiction proves that  $\text{per}(B) > q$ .

It remains to analyze the running time. Determining  $\text{per}(T[0 .. 2q])$  in Line 1 costs  $\mathcal{O}(q)$  time using a classic algorithm [31]. The number of sampled positions is  $|S| = \tilde{\mathcal{O}}(\frac{1}{k+1}|T|)$  with high probability, so the test in Line 5 costs  $\tilde{\mathcal{O}}(\frac{1}{k+1}|T|)$  time in total. Binary search (the loop in Line 7) has  $\mathcal{O}(\log |T|) = \tilde{\mathcal{O}}(1)$  iterations, each implemented in  $\mathcal{O}(q)$  time. The total running time is  $\tilde{\mathcal{O}}(\frac{1}{k+1}|T| + q)$ .  $\square$

Next comes the oracle testing  $\max_{j \in J} \text{LCE}_{\leq k}^{P, T}(i, j) \leq \ell$ .

**Lemma 3.7.** *There exists an algorithm that, given strings  $X$  and  $Y$ , an integer  $k \geq 0$ , an integer  $\ell > 0$ , an integer  $i \in [0 .. |X| - \ell]$ , and a non-empty range  $J \subseteq [0 .. |Y| - \ell]$ , returns YES if  $\exists j \in J : X[i .. i + \ell] = Y[j .. j + \ell]$ , and NO if  $\forall j \in J : \text{HD}(X[i .. i + \ell], Y[j .. j + \ell]) > k$ . With high probability, the algorithm is correct and its running time is  $\tilde{\mathcal{O}}(\frac{\ell}{k+1} + |J|)$ .*

*Proof.* A procedure **Oracle**( $P, T, i, J, k, \ell$ ) implementing Lemma 3.7 is given as Algorithm 4.

**Algorithm** If  $\ell < 3|J|$ , then the algorithm simply returns the answer based on whether  $X[i .. i + \ell] = Y[j .. j + \ell]$  holds for some  $j \in J$ . Otherwise, the algorithm computes a set  $C \subseteq J$  of *candidate positions*  $j$  satisfying  $X[i .. i + 3|J|] = Y[j .. j + 3|J|]$ , and returns NO if  $C = \emptyset$ . In the remaining case, the algorithm applies the procedure **FindBreak** of Lemma 3.6 to  $X[i .. i + \ell]$  and  $Y[\max C .. \min C + \ell]$ , both with  $q = |J|$  and threshold  $\lfloor \frac{k}{2} \rfloor$ . If both strings are certified to have an approximate period, then the algorithm returns YES. Otherwise, the algorithm further filters  $C$  using the breaks returned by **FindBreak**: If a break  $B_X = X[x .. x']$  is found in  $X[i .. i + \ell]$ , then  $C$  is restricted to positions  $j$  satisfying  $B_X = Y[j - i + x .. j - i + x']$ . Similarly, if a break  $B_Y = Y[y .. y']$  is found in  $Y[\max C .. \min C + \ell]$ , then  $C$  is restricted to positions  $j$  satisfying  $B_Y = X[i - j + y .. i - j + y']$ . If this filtering leaves  $C$  empty, then the algorithm returns NO. Otherwise, the algorithm samples a subset  $S \subseteq [0 .. \ell]$  with sufficiently large rate  $\tilde{\mathcal{O}}(\frac{1}{k+1})$ , and returns the answer depending on whether  $X[i + s] = Y[\min C + s]$  holds for all  $s \in S$ .

**Correctness** Denote  $M = \{j \in J : X[i .. i + \ell] = Y[j .. j + \ell]\}$ . Recall that the algorithm must return YES if  $M \neq \emptyset$ , and it may return NO whenever  $M = \emptyset$ .

If  $|J| < 3\ell$ , then the algorithm verifies  $M \neq \emptyset$ , so the answers are correct. Thus, we henceforth assume  $|J| \geq 3\ell$ .

Let us argue that  $M \subseteq C \subseteq J$  holds throughout the execution: indeed, every position  $j \in M$  satisfies  $X[i .. i + 3|J|] = Y[j .. j + 3|J|]$ , as well as  $X[x .. x'] = Y[j - i + x .. j - i + x']$  for every fragment  $X[x .. x']$  contained in  $X[i .. i + \ell]$ , and  $Y[y .. y'] = X[i - j + y .. i - j + y']$  for every fragment  $Y[y .. y']$  contained in  $Y[j .. j + \ell]$ . Moreover, the strings in the two calls to **FindBreak** are chosen so that the breaks, if any, are contained in  $X[i .. i + \ell]$ , and in  $Y[j .. j + \ell]$  for every  $j \in C$ , respectively. Consequently, the NO answers returned in Lines 4 and 10 are correct.

---

**Algorithm 4:** Oracle( $X, Y, i, J, k, \ell$ )

---

```
1 if  $\ell < 3|J|$  then
2   | return  $\exists_{j \in J} : X[i..i + \ell) = Y[j..j + \ell)$ ;
3  $C := \{j \in J : X[i..i + 3|J|) = Y[j..j + 3|J|)\}$ ;
4 if  $C = \emptyset$  then return NO;
5  $B_X := \text{FindBreak}(X[i..i + \ell), |J|, \lfloor \frac{k}{2} \rfloor)$ ;
6  $B_Y := \text{FindBreak}(Y[\max C.. \min C + \ell), |J|, \lfloor \frac{k}{2} \rfloor)$ ;
7 if  $\perp = B_X$  and  $\perp = B_Y$  then return YES;
8 if  $\perp \neq B_X =: X[x..x')$  then  $C := \{j \in C : B_X = Y[j - i + x..j - i + x')\}$ ;
9 if  $\perp \neq B_Y =: Y[y..y')$  then  $C := \{j \in C : B_Y = X[i - j + y..i - j + y')\}$ ;
10 if  $C = \emptyset$  then return NO;
11 Let  $S \subseteq [0.. \ell)$  with elements sampled independently at sufficiently large rate  $\tilde{\Theta}(\frac{1}{k+1})$ ;
12 foreach  $s \in S$  do
13   | if  $X[i + s] \neq Y[\min C + s]$  then return NO;
14 return YES;
```

---

Next, note that the calls to **FindBreak** satisfy the requirements of Lemma 3.6. In particular, the two strings are of length at least  $3|J|$  and  $2|J|$ , respectively. To justify the YES answer in Line 7, we shall prove that  $\text{HD}(X[i..i + \ell), Y[\min C.. \min C + \ell)) \leq k$  holds with high probability in case both calls return  $\perp$ . Denote  $p = \text{per}(X[i..i + 2|J|])$ , let  $P = X[i..i + p)$  be the corresponding string period, and let  $P^\infty$  be the concatenation of infinitely many copies of  $P$ . The outcome  $\perp$  of the first call to **FindBreak** certifies that  $X[i..i + \ell)$  is with high probability at Hamming distance at most  $\frac{k}{2}$  from a prefix of  $P^\infty$ . Due to  $X[i..i + 2|J|) = Y[\max C.. \max C + 2|J|)$ , the outcome  $\perp$  of the second call to **FindBreak** certifies that also  $Y[\max C.. \min C + \ell)$  is with high probability at Hamming distance at most  $\frac{k}{2}$  from a prefix of  $P^\infty$ . Moreover, by Fact 2.1,  $p$  is a divisor of  $\max C - \min C$ , so,  $Y[\min C.. \max C)$  is an integer power of  $P$ . Thus,  $Y[\min C.. \min C + \ell)$  is with high probability at Hamming distance at most  $\frac{k}{2}$  from a prefix of  $P^\infty$ . Now, the triangle inequality yields  $\text{HD}(X[i..i + \ell), Y[\min C.. \min C + \ell)) \leq k$ , as claimed.

It remains to justify the answers returned in Lines 13 and 14. Because the breaks  $B_X$  and  $B_Y$ , if defined, satisfy  $\text{per}(B_X) > |J|$  and  $\text{per}(B_Y) > |J|$ , Lemma 3.6 implies that their exact occurrences must be more than  $|J|$  positions apart. Consequently, applying Line 8 or Line 9 leaves at most one position in  $C$ . Thus, the algorithm correctly returns NO if it detects a mismatch in Line 13 while testing random shifts  $s$  for the unique position  $\min C \in C$ . Finally, note that the sampling rate in the construction of  $S$  guarantees that if there are at least  $k + 1$  mismatches between  $X[i..i + \ell)$  and  $Y[\min C.. \min C + \ell)$ , then with high probability at least one of them is detected. Thus, returning YES in Line 14 is also correct.

**Running time** Lines 2, 3, 8, and 9 can be interpreted as finding exact occurrences of  $X[i..i + \ell)$ ,  $X[i..i + 3|J|)$ ,  $B_X$ , and  $B_Y$ , respectively, starting at up to  $|J|$  consecutive positions of  $X$  or  $Y$ . Since the length of all these patterns is  $\mathcal{O}(|J|)$ , this search can be implemented in  $\mathcal{O}(|J|)$  using a classic pattern matching algorithm [31]. The calls to **FindBreak** from Lemma 3.6 cost  $\tilde{\mathcal{O}}(\frac{\ell}{\lfloor k/2 \rfloor + 1} + |J|)$  time with high probability. Finally, the number of sampled positions is  $|S| = \tilde{\mathcal{O}}(\frac{\ell}{k+1})$  with high probability, and this is also the total cost of Line 13. The total running time is  $\tilde{\mathcal{O}}(\frac{\ell}{k+1} + |J|)$ .  $\square$

Finally, we derive Proposition 3.4 via a simple reduction to Lemma 3.7.

**Proposition 3.4.** *There exists an algorithm that, given strings  $X$  and  $Y$ , an integer  $k \geq 0$ , an index  $i$ , and a range of indices  $J$ , computes  $\ell := \max_{j \in J} \text{LCE}_{\leq k}^{X,Y}(i, j)$ . With high probability, the algorithm is correct and its running time is  $\tilde{\mathcal{O}}(\frac{\ell}{k+1} + |J|)$ .*

*Proof.* Observe that Lemma 3.7 provides an oracle that returns YES if  $\max_{j \in J} \text{LCE}_0^{X,Y}(i, j) \geq \ell$  and NO if  $\max_{j \in J} \text{LCE}_k^{X,Y}(i, j) < \ell$ . However, before calling  $\text{Oracle}(P, T, i, J, k, \ell)$ , we need to make sure that  $\ell > 0$ ,  $i \in [0 \dots |X| - \ell]$ , and  $\emptyset \neq J \subseteq [0 \dots |Y| - \ell]$ . Thus, basic corner cases have to be handled separately: The algorithm returns YES if  $\ell \leq 0$ ; otherwise, it sets  $J := J \cap [0 \dots |Y| - \ell]$ , returns NO if  $i \notin [0 \dots |X| - \ell]$  or  $J = \emptyset$ , and makes a call  $\text{Oracle}(P, T, i, J, k, \ell)$  in the remaining case.

A single call to the oracle costs  $\tilde{O}(\frac{\ell}{k+1} + |J|)$  time. Hence, we need to make sure that the intermediate values of the threshold  $\ell$  are bounded from above by a constant multiple of the final value. For this, the algorithm uses exponential search rather than ordinary binary search.  $\square$

## 4 Improved Approximation Ratio

Goldenberg et al. [22] generalized their algorithm in order to solve the  $k$  vs  $\alpha k$  gap edit distance problem in  $\tilde{O}(\frac{n}{\alpha} + k^3)$  time for any  $\alpha \geq 1$ . This transformation is quite simple, because Algorithm 1 (the Landau–Vishkin algorithm) with  $\text{LCE}_0$  queries replaced by  $\text{LCE}_{\leq \alpha-1}$  queries returns YES if  $\text{ED}(X, Y) \leq k$  and NO if  $\text{ED}(X, Y) > k + (\alpha - 1)(k + 1)$ .

However, if we replace  $\text{LCE}_{\leq k}$  queries with  $\text{LCE}_{\leq \alpha-1}$  queries in Algorithm 2, then we are guaranteed to get a NO answer only if  $\text{ED}(X, Y) > 2k(k + 2) + (\alpha - 1)(k + 1)$ . As a result, with an appropriate adaptation of Proposition 3.4, Algorithm 2 yields an  $\tilde{O}(\frac{n}{\alpha} + k^2)$ -time solution to the  $k$  vs  $\alpha k$  gap edit distance problem only for  $\alpha = \Omega(k)$ . The issue is that Algorithm 2 incurs a cost of up to  $\Theta(k)$  edits for up to  $\Theta(k)$  arbitrary changes of the shift  $y - x$  within queries  $\text{LCE}^{X,Y}(x, y)$ . On the other hand, no such shift changes are performed in Algorithm 1, but this results in  $\text{LCE}^{X,Y}(x, y)$  queries asked for up to  $\Theta(k^2)$  distinct positions  $x$ , which is the reason behind the  $\tilde{O}(k^3)$  term in the running time  $\tilde{O}(\frac{n}{\alpha} + k^3)$  of [22].

Nevertheless, since each  $\text{LCE}_{\leq \alpha-1}^{X,Y}(x, y)$  query incurs a cost of up to  $\alpha - 1$  edits (mismatches) it is still fine to pay  $\mathcal{O}(\alpha - 1)$  further edits (insertions or deletions) to change the shift  $y - x$  by up to  $\alpha - 1$ . Hence, we design Algorithm 5 as a hybrid of Algorithms 1 and 2.

---

### Algorithm 5: Improved algorithm

---

```

1 foreach  $i \in [0 \dots k]$  and  $j \in [\lfloor \frac{-k}{\alpha} \rfloor - 1 \dots \lfloor \frac{k}{\alpha} \rfloor + 1]$  do  $d'_{i,j} := d_{i,j} := -\infty$ ;
2  $d'_{0,0} := 0$ ;
3 for  $i := 0$  to  $k$  do
4   for  $j := \lfloor \frac{-k}{\alpha} \rfloor$  to  $\lfloor \frac{k}{\alpha} \rfloor$  do
5     if  $d'_{i,j} \neq -\infty$  then
6        $d_{i,j} := d'_{i,j} + \max_{\delta=j\alpha}^{(j+1)\alpha-1} \text{LCE}_{\leq \alpha-1}^{X,Y}(d'_{i,j}, d'_{i,j} + \delta)$ ;
7     for  $j := \lfloor \frac{-k}{\alpha} \rfloor$  to  $\lfloor \frac{k}{\alpha} \rfloor$  do
8        $d'_{i+1,j} := \min(|X|, \max(d_{i,j-1}, d_{i,j} + 1, d_{i,j+1} + 1))$ ;
9    $j := \lfloor \frac{1}{\alpha}(|Y| - |X|) \rfloor$ ;
10 if  $\|X\| - |Y| \leq k$  and  $d_{k,j} = |X|$  then return YES;
11 else return NO;
```

---

**Lemma 4.1.** *For any integers  $k \geq 0$  and  $\alpha \geq 1$ , Algorithm 5 returns YES if  $\text{ED}(X, Y) \leq k$  and NO if  $\text{ED}(X, Y) > k + 3(k + 1)(\alpha - 1)$ .*

*Proof.* As in the proof of Lemma 3.1, we characterize the values  $d_{i,j}$  and  $d'_{i,j}$  using two claims.

**Claim 4.2.** *Each  $i \in [0 \dots k]$  and  $j \in [\lfloor \frac{-k}{\alpha} \rfloor \dots \lfloor \frac{k}{\alpha} \rfloor]$  satisfies the following two properties:*

- (a)  $\text{ED}(X[0 \dots d'_{i,j}], Y[0 \dots y]) \leq i + (3i + 1)(\alpha - 1)$  for  $y \in [d'_{i,j} + j\alpha \dots d'_{i,j} + (j + 1)\alpha] \cap [0 \dots |Y|]$ ;
- (b)  $\text{ED}(X[0 \dots d_{i,j}], Y[0 \dots y]) \leq i + 3(i + 1)(\alpha - 1)$  for  $y \in [d_{i,j} + j\alpha \dots d_{i,j} + (j + 1)\alpha] \cap [0 \dots |Y|]$ .

*Proof.* We proceed by induction on  $i$ . Our base case is Property (a) for  $i = 0$ . Due to  $d'_{0,j} = -\infty$  for  $j \neq 0$ , the range for  $y$  is non-empty only for  $j = 0$ , when the range is  $[0 \dots \alpha)$  due to  $d'_{0,0} = 0$ . Moreover, for  $y \in [0 \dots \alpha)$ , we have  $\text{ED}(X[0 \dots d'_{0,0}], Y[0 \dots y]) = y \leq \alpha - 1$ .

Next, we shall prove Property (b) for  $i \geq 0$  assuming that Property (a) is true for  $i$ . By definition of  $\text{LCE}_{\leq \alpha-1}$  queries, we have  $d_{i,j} \leq d'_{i,j} + \text{LCE}_{\alpha-1}^{X,Y}(d'_{i,j}, y')$  for some position  $y' \in [d'_{i,j} + j\alpha \dots d'_{i,j} + (j+1)\alpha) \cap [0 \dots |Y|]$ , and thus  $\text{HD}(X[d'_{i,j} \dots d_{i,j}], Y[y' \dots y' + d_{i,j} - d'_{i,j}]) \leq \alpha - 1$ . The inductive assumption yields  $\text{ED}(X[0 \dots d'_{i,j}], Y[0 \dots y']) \leq i + (3i + 1)(\alpha - 1)$ , so we have  $\text{ED}(X[0 \dots d_{i,j}], Y[0 \dots y' + d_{i,j} - d'_{i,j}]) \leq i + (3i + 2)(\alpha - 1)$ . Due to  $|y' + d_{i,j} - d'_{i,j} - y| \leq \alpha - 1$ , we conclude that  $\text{ED}(X[0 \dots d_{i,j}], Y[0 \dots y]) \leq i + 3(i + 1)(\alpha - 1)$ .

Finally, we shall prove Property (a) for  $i > 0$  assuming that Property (b) is true for  $i - 1$ . We consider three subcases: If  $d'_{i,j} \leq d_{i-1,j-1}$ , then the inductive assumption yields  $\text{ED}(X[0 \dots d'_{i,j}], Y[0 \dots y - \alpha]) \leq (i - 1) + 3i(\alpha - 1)$ , and therefore  $\text{ED}(X[0 \dots d'_{i,j}], Y[0 \dots y]) \leq \alpha + (i - 1) + 3i(\alpha - 1) = i + (3i + 1)(\alpha - 1)$ . If  $d'_{i,j} \leq d_{i-1,j} + 1$ , then the inductive assumption yields  $\text{ED}(X[0 \dots d'_{i,j} - 1], Y[0 \dots y - 1]) \leq (i - 1) + 3i(\alpha - 1)$ , and therefore  $\text{ED}(X[0 \dots d'_{i,j}], Y[0 \dots y]) \leq 1 + (i - 1) + 3i(\alpha - 1) = i + 3i(\alpha - 1)$ . If  $d'_{i,j} \leq d_{i-1,j+1} + 1$ , then the inductive assumption yields  $\text{ED}(X[0 \dots d'_{i,j} - \alpha], Y[0 \dots y]) \leq (i - 1) + 3i(\alpha - 1)$ , and therefore  $\text{ED}(X[0 \dots d'_{i,j}], Y[0 \dots y]) \leq \alpha + (i - 1) + 3i(\alpha - 1) = i + (3i + 1)(\alpha - 1)$ .  $\square$

In particular, if the algorithm returns YES, then  $\text{ED}(X, Y) \leq k + 3(k + 1)(\alpha - 1)$ .

**Claim 4.3.** *If  $\text{ED}(X[0 \dots x], Y[0 \dots y]) = i$  for  $x \in [0 \dots |X|]$ ,  $y \in [0 \dots |Y|]$ , and  $i \in [0 \dots k]$ , then  $x \leq d_{i,j}$  holds for  $j = \lfloor \frac{1}{\alpha}(y - x) \rfloor$ .*

*Proof.* We proceed by induction on  $i$ . Both in the base case of  $i = 0$  and in the inductive step of  $i > 0$ , we prove that  $x \leq d'_{i,j} + \max_{\delta=j\alpha}^{(j+1)\alpha-1} \text{LCE}_0^{X,Y}(d'_{i,j}, d'_{i,j} + \delta)$ . This implies the claim since  $d_{i,j} \geq d'_{i,j} + \max_{\delta=j\alpha}^{(j+1)\alpha-1} \text{LCE}_0^{X,Y}(d'_{i,j}, d'_{i,j} + \delta)$ , holds by definition of  $\text{LCE}_{\leq \alpha-1}$  queries.

In the base case of  $i = 0$ , we have  $X[0 \dots x] = Y[0 \dots y]$ , so  $x = y$  and  $j = 0$ . Consequently, due to  $d'_{0,0} = 0$ , we have  $x \leq \text{LCE}_0^{X,Y}(0, 0) \leq d'_{0,0} + \max_{\delta=0}^{\alpha-1} \text{LCE}_0^{X,Y}(d'_{0,0}, d'_{0,0} + \delta)$ .

For  $i > 0$ , we consider an optimal alignment between  $X[0 \dots x]$  and  $Y[0 \dots y]$ , and we distinguish its maximum prefix with  $i - 1$  edits. This yields positions  $x', x'' \in [0 \dots x]$  and  $y', y'' \in [0 \dots y]$  with  $x'' - x' \in \{0, 1\}$  and  $y'' - y' \in \{0, 1\}$  such that  $\text{ED}(X[0 \dots x'], Y[0 \dots y']) = i - 1$  and  $X[x'' \dots x] = Y[y'' \dots y]$ . The inductive assumption yields  $x' \leq d_{i-1,j'}$ , where  $j' = \lfloor \frac{1}{\alpha}(y' - x') \rfloor$  satisfies  $|j - j'| \leq 1$ . We shall prove that  $x'' \leq d'_{i,j}$  by considering two possibilities. If  $j' \geq j$ , then  $x'' \leq \min(x, x' + 1) \leq \min(|X|, d_{i-1,j'} + 1) \leq d'_{i,j}$ . If  $j' < j$ , on the other hand, then  $y' - x' < y'' - x''$  implies  $x'' = x' \leq d_{i-1,j'} = d_{i-1,j-1} \leq d'_{i,j}$ . Due to  $X[x'' \dots x] = Y[y'' \dots y]$ , we have  $\text{LCE}_0^{X,Y}(x'', y'') \geq x - x''$ . By  $x'' \leq d'_{i,j}$ , this implies  $\text{LCE}_0^{X,Y}(d'_{i,j}, d'_{i,j} + y - x) \geq x - d'_{i,j}$ . By definition of  $j$ , we conclude that  $x = d'_{i,j} + (x - d'_{i,j}) \leq d'_{i,j} + \text{LCE}_0^{X,Y}(d'_{i,j}, d'_{i,j} + y - x) \leq d'_{i,j} + \max_{\delta=j\alpha}^{(j+1)\alpha-1} \text{LCE}_0^{X,Y}(d'_{i,j}, d'_{i,j} + \delta)$ .  $\square$

In particular, if  $\text{ED}(X, Y) \leq k$ , then the algorithm returns YES.  $\square$

If we use Proposition 3.4 to implement  $\text{LCE}_{\leq \alpha-1}$  queries in Algorithm 5, then the cost of computing  $d_{i,j}$  is  $\mathcal{O}(\alpha + \frac{1}{\alpha}(d_{i,j} - d'_{i,j}))$  with high probability. This query is performed only for  $d'_{i,j} \geq 0$ , and it results in  $d_{i,j} \leq |X|$ . As  $d_{i,j} \leq d'_{i+1,j}$ , the total query time for fixed  $j$  sums up to  $\tilde{\mathcal{O}}(\frac{1}{\alpha}|X| + k\alpha)$  across all queries. Over all the  $\mathcal{O}(\frac{k}{\alpha})$  values  $j$ , this gives  $\tilde{\mathcal{O}}(\frac{k}{\alpha^2}|X| + k^2)$  time with high probability, which is not comparable to the running time  $\tilde{\mathcal{O}}(\frac{1}{\alpha}|X| + k^3)$  of [22].

However, we can obtain a faster algorithm using the data structure specified below and described in Section 4.1. In particular, this result dominates Proposition 3.4 and, if we set  $\Delta = [-k \dots k]$ , then  $\text{LCE}_{\leq k}^{X,Y}(x, y)$  queries with  $|x - y| \leq k$  can be answered in  $\tilde{\mathcal{O}}(k)$  time after  $\tilde{\mathcal{O}}(\frac{1}{k+1}|X|)$  preprocessing, as promised in Section 3.

**Proposition 4.4.** *There exists a data structure that, initialized with strings  $X$  and  $Y$ , an integer  $k \geq 0$ , and an integer range  $\Delta$ , answers the following queries: given an integer  $x$ , return  $\text{LCE}_{\leq k}^{X,Y}(x, x + \delta)$  for all  $\delta \in \Delta$ . The initialization costs  $\tilde{O}(\frac{1}{k+1}|X|)$  time with high probability, and the queries cost  $\tilde{O}(|\Delta|)$  time with high probability.*

Since the  $\text{LCE}_{\leq \alpha-1}^{X,Y}(x, y)$  queries in Algorithm 5 are asked for  $\mathcal{O}(\frac{k^2}{\alpha})$  positions  $x$  and for positions  $y$  satisfying  $|y - x| = \mathcal{O}(k)$ , a straightforward application of Proposition 4.4 yields an  $\tilde{O}(\frac{1}{\alpha}|X| + \frac{k^3}{\alpha})$ -time implementation of Algorithm 5, which is already better the running time of [22]. However, the running time of a more subtle solution described below subsumes both  $\tilde{O}(\frac{1}{\alpha}|X| + \frac{k^3}{\alpha})$  and  $\tilde{O}(\frac{k}{\alpha^2}|X| + k^2)$  (obtained using Proposition 3.4).

**Theorem 4.5.** *There exists an algorithm that, given strings  $X$  and  $Y$ , an integer  $k \geq 0$ , and a positive integer  $\alpha = \mathcal{O}(k)$ , returns YES if  $\text{ED}(X, Y) \leq k$ , and NO if  $\text{ED}(X, Y) > k + 3(k + 1)(\alpha - 1)$ . With high probability, the algorithm is correct and its running time is  $\tilde{O}(\frac{1}{\alpha}|X| + k^2 + \frac{k}{\alpha}\sqrt{|X|k})$ .*

*Proof.* We define an integer parameter  $b \in [1 \dots \lceil \frac{k}{\alpha} \rceil]$  (to be fixed later) and initialize  $\mathcal{O}(\frac{k}{\alpha b})$  instances of the data structure of Proposition 4.4 for answering  $\text{LCE}_{\leq \alpha-1}^{X,Y}$  queries. The instances are indexed with  $j' \in [\lfloor \frac{-k}{\alpha b} \rfloor \dots \lfloor \frac{k}{\alpha b} \rfloor]$ , and the  $j'$ 'th instance has interval  $\Delta_{j'} = [j'\alpha b \dots (j' + 1)\alpha b]$ . This way, the value  $d_{i,j}$  can be retrieved from the values  $\text{LCE}_{\leq \alpha-1}(d'_{i,j}, d'_{i,j} + \delta)$  for  $\delta \in \Delta_{\lfloor \frac{j}{b} \rfloor}$ , that is, from a single query to an instance of the data structure of Proposition 4.4.

Correctness follows from Lemma 4.1 since with high probability all  $\text{LCE}_{\leq \alpha-1}$  queries are answered correctly. The total preprocessing cost is  $\tilde{O}(\frac{k}{\alpha b} \cdot \frac{1}{\alpha}|X|) = \tilde{O}(\frac{k}{\alpha^2 b}|X|)$  with high probability, and each value  $d_{i,j}$  is computed in  $\tilde{O}(\alpha b)$  time with high probability. The number of queries is  $\mathcal{O}(\frac{k^2}{\alpha})$ , so the total running time is  $\tilde{O}(\frac{k}{\alpha^2 b}|X| + k^2 b)$  with high probability. Optimizing for  $b$  yields  $\tilde{O}(\frac{k}{\alpha}\sqrt{|X|k})$ . Due to  $b \in [1 \dots \lceil \frac{k}{\alpha} \rceil]$ , we get additional terms  $\tilde{O}(k^2 + \frac{1}{\alpha}|X|)$ .  $\square$

#### 4.1 Proof of Proposition 4.4

While there are many similarities between the proofs of Propositions 3.4 and 4.4, the main difference is that we heavily rely on  $\overline{\text{LCE}}_r$  queries in the proof of Proposition 4.4. The following fact illustrates their main advantage compared to  $\text{LCE}_{\leq k}$  queries: *composability*.

**Fact 4.6.** *Let  $X, X', Y$  be strings, let  $r > 0$  be real parameter, and let  $j \in [0 \dots |Y| - |X|]$ . Suppose that  $\overline{\text{LCE}}_r^{X,Y}(0, j)$  and  $\overline{\text{LCE}}_r^{X',Y}(0, j + |X|)$  are independent random variables, and define*

$$\ell := \begin{cases} \overline{\text{LCE}}_r^{X,Y}(0, j) & \text{if } \overline{\text{LCE}}_r^{X,Y}(0, j) < |X|, \\ |X| + \overline{\text{LCE}}_r^{X',Y}(0, j + |X|) & \text{otherwise.} \end{cases}$$

*Then,  $\ell$  satisfies the conditions for  $\overline{\text{LCE}}_r^{XX',Y}(0, j)$ .*

*Proof.* Define  $d = \text{HD}(X, Y[j \dots j + |X|])$  and note that the following equality holds for  $k \geq 0$ :

$$\text{LCE}_k^{XX',Y}(0, j) = \begin{cases} \text{LCE}_k^{X,Y}(0, j) & \text{if } k < d, \\ |X| + \text{LCE}_{k-d}^{X',Y}(0, j + |X|) & \text{if } k \geq d. \end{cases}$$

Let us first prove that  $\ell \geq \text{LCE}_0^{XX',Y}(0, j)$ . If  $\overline{\text{LCE}}_r^{X,Y}(0, j) < |X|$ , then  $\text{LCE}_0^{X,Y}(0, j) \leq \overline{\text{LCE}}_r^{X,Y}(0, j) < |X|$  implies  $d > 0$ , and thus  $\ell = \overline{\text{LCE}}_r^{X,Y}(0, j) \geq \text{LCE}_0^{X,Y}(0, j) = \text{LCE}_0^{XX',Y}(0, j)$ . Otherwise,  $\ell = |X| + \overline{\text{LCE}}_r^{X',Y}(0, j + |X|) \geq |X| + \text{LCE}_0^{X',Y}(0, j + |X|) \geq \text{LCE}_0^{XX',Y}(0, j)$ . Hence, the claim holds in both cases.

Next, let us bound the probability  $\mathbb{P}[\ell > \text{LCE}_k^{XX',Y}(0, j)]$  for  $k \geq 0$ . We consider two cases. If  $k < d$ , then  $\text{LCE}_k^{XX',Y}(0, j) < |X|$  and

$$\begin{aligned} \mathbb{P}[\ell > \text{LCE}_k^{XX',Y}(0, j)] &\leq \mathbb{P}[\overline{\text{LCE}}_r^{X,Y}(0, j) > \text{LCE}_k^{XX',Y}(0, j)] \\ &= \mathbb{P}[\overline{\text{LCE}}_r^{X,Y}(0, j) > \text{LCE}_k^{X,Y}(0, j)] \\ &\leq \exp(-\frac{k+1}{r}). \end{aligned}$$

On the other hand, if  $k \geq d$ , then  $\text{LCE}_k^{XX',Y}(0, j) = |X| + \text{LCE}_{k-d}^{X',Y}(0, j + |X|) \geq |X| > \text{LCE}_{d-1}^{X,Y}(0, j)$ . Hence, the independence of  $\overline{\text{LCE}}_r^{X,Y}(0, j)$  and  $\overline{\text{LCE}}_r^{X',Y}(0, j + |X|)$  yields

$$\begin{aligned} \mathbb{P}[\ell > \text{LCE}_k^{XX',Y}(0, j)] &\leq \mathbb{P}[\overline{\text{LCE}}_r^{X,Y}(0, j) > |X| \text{ and } \overline{\text{LCE}}_r^{X',Y}(0, j + |X|) > \text{LCE}_{k-d}^{X',Y}(0, j + |X|)] \\ &= \mathbb{P}[\overline{\text{LCE}}_r^{X,Y}(0, j) > |X|] \cdot \mathbb{P}[\overline{\text{LCE}}_r^{X',Y}(0, j + |X|) > \text{LCE}_{k-d}^{X',Y}(0, j + |X|)] \\ &\leq \exp(-\frac{d}{r}) \cdot \exp(-\frac{k-d+1}{r}) \\ &= \exp(-\frac{k+1}{r}). \end{aligned}$$

This completes the proof.  $\square$

Next, we show that a single value  $\overline{\text{LCE}}_r^{X,Y}(0, j)$  can be computed efficiently. We also require that the resulting position  $\ell$  witnesses  $\text{LCE}_0^{X,Y}(0, j) \leq \ell$ .

**Fact 4.7.** *There is an algorithm that, given strings  $X$  and  $Y$ , a real parameter  $r > 0$ , and an integer  $j$ , returns a value  $\ell = \overline{\text{LCE}}_r^{X,Y}(0, j)$  such that  $X[\ell] \neq Y[j + \ell]$  or  $\ell = \min(|X|, |Y| - j)$ . The algorithm takes  $\tilde{\mathcal{O}}(\frac{1}{r}|X|)$  time with high probability.*

*Proof.* If  $r \leq 1$ , then the algorithm returns  $\ell = \text{LCE}_0^{X,Y}(0, j)$  computed naively in  $\mathcal{O}(|X|)$  time. It is easy to see that this value satisfies the required conditions.

If  $r > 1$ , then the algorithm samples a subset  $S \subseteq [0.. \min(|X|, |Y| - j))$  so that the events  $s \in S$  are independent with  $\mathbb{P}[s \in S] = \frac{1}{r}$ . If  $X[s] = Y[j + s]$  for each  $s \in S$ , then the algorithm returns  $\ell = \min(|X|, |Y| - j)$ . Otherwise, the algorithm returns  $\ell = \min\{s \in S : X[s] \neq Y[j + s]\}$ . This way,  $\ell \geq \text{LCE}_0^{X,Y}(0, j)$ , and  $P[\ell \neq Y[j + \ell]]$  or  $\ell = \min(|X|, |Y| - j)$ .

It remains to bound  $\mathbb{P}[\ell > \text{LCE}_k^{X,Y}(0, j)]$  for every  $k \geq 0$ . This event holds only if each of the  $k + 1$  leftmost mismatches (that is, the leftmost positions  $s$  such that  $X[s] \neq Y[j + s]$ ) does not belong to  $S$ . By definition of  $S$ , the probability of this event is  $(1 - \frac{1}{r})^{k+1} \leq \exp(-\frac{k+1}{r})$ .

Since  $|S| = \tilde{\mathcal{O}}(\frac{1}{r} \min(|X|, |Y| - j))$  with high probability, the total running time is  $\tilde{\mathcal{O}}(\frac{1}{r}|X|)$  with high probability.  $\square$

We are now ready to describe a counterpart of Lemma 3.6.

**Lemma 4.8.** *There is an algorithm that, given a string  $T$ , a real parameter  $r > 0$ , and a positive integer  $q \leq \frac{1}{2}|T|$  such that  $p := \text{per}(T[0..2q]) \leq q$ , returns  $\ell \in [2q..|T|]$  such that*

- $\ell = \overline{\text{LCE}}_r^{T,T'}(0, 0)$ , where  $T'$  is an infinite string with  $T'[i] = T[i \bmod p]$  for  $i \geq 0$ , and
- $\ell = |T|$  or  $\text{per}(T(\ell - 2q.. \ell)) > q$ .

*The algorithm takes  $\tilde{\mathcal{O}}(\frac{1}{r}|T| + q)$  time with high probability.*

*Proof.* A procedure  $\text{FindBreak2}(T, r, q)$  implementing Lemma 4.8 is given as Algorithm 6.

First, the algorithm computes the shortest period  $p = \text{per}(T[0..2q])$  (guaranteed to be at most  $q$  by the assumption) and constructs an infinite string  $T'$  with  $T'[i] = T[i \bmod p]$  for each  $i \geq 0$ ; note that random access to  $T'$  can be easily implemented on top of random access to  $T$ .

---

**Algorithm 6:** FindBreak2( $T, r, q$ )

---

```

1  $p := \text{per}(T[0..2q]);$ 
2 Define  $T'[0..\infty)$  with  $T'[i] = T[i \bmod p];$ 
3  $\ell' := \overline{\text{LCE}}_r^{T,T'}(0,0);$   $\triangleright$  computed using Fact 4.7
4 if  $\ell' = |T|$  then return  $|T|;$ 
5  $b := 2q; e := \ell';$ 
6 while  $b < e$  do
7    $m := \lceil \frac{b+e}{2} \rceil;$ 
8   for  $j := m - 2q$  to  $m - 1$  do
9     if  $T[j] \neq T'[j]$  then  $e := j;$ 
10  if  $e \geq m$  then  $b := m;$ 
11 return  $b;$ 

```

---

Next, the algorithm computes  $\ell' := \overline{\text{LCE}}_r^{T,T'}(0,0)$  using Fact 4.7. If  $\ell' = |T|$ , then  $|T|$  satisfies both requirements for the resulting value  $\ell$ , so the algorithm returns  $\ell := |T|$  (Line 1).

Otherwise, the algorithm tries to find a position  $\ell \leq \ell'$  such that  $\text{per}(T(\ell - 2q.. \ell)) > q$  (Lines 5–11). This step is implemented as in the proof of Lemma 3.6. We call a position  $i \in [0..|T|)$  *compatible* if  $T[i] = T'[i]$ . The algorithm performs a binary search maintaining positions  $b, e$ , with  $2q \leq b \leq e < |T|$ , such that  $e$  is incompatible and the positions in  $[b - 2q.. b)$  are all compatible. The initial choice of  $b := 2q$  and  $e := \ell'$  satisfies the invariant because the positions in  $[0..2q)$  are all compatible due to  $p = \text{per}(T[0..2q))$ . While  $b < e$ , the algorithm chooses  $m := \lceil \frac{b+e}{2} \rceil$ . If  $[m - 2q.. m)$  contains an incompatible position  $j$ , then  $j \geq b$  (because  $j \geq m - 2q \geq b - 2q$  and the positions in  $[b - 2q.. b)$  are all compatible), so the algorithm maintains the invariant setting  $e := j$  for such a position  $j$  (Line 9). Otherwise, all the positions in  $[m - 2q.. m)$  are compatible. Due to  $m \leq e$ , this means that the algorithm maintains the invariant setting  $b := m$  (Line 10). Since  $e - b$  decreases at least twofold in each iteration, after  $\mathcal{O}(\log |T|)$  iterations, the algorithm obtains  $b = e$ . Then, the algorithm returns  $\ell := b$ .

We shall prove that this result is correct. For a proof by contradiction, suppose that  $p' := \text{per}(T(b - 2q.. b)) \leq q$ . Then,  $p'$  is also period of  $T(b - 2q.. b)$ . Moreover, the invariant guarantees that positions in  $(b - 2q.. b)$  are all compatible, so also  $p$  is a period of  $T(b - 2q.. b)$ . Since  $p + p' - 1 \leq 2q - 1$ , the periodicity lemma [21] implies that also  $\text{gcd}(p, p')$  is a period of  $T[b - 2q + 1.. b)$ . Consequently,  $T[b] = T[b - p'] = T[b - p] = T'[b - p] = T'[b]$ , i.e.,  $b$  is compatible. However, the invariant assures that  $b$  is incompatible. This contradiction proves that  $\text{per}(T(b - 2q.. b)) > q$ . The incompatibility of  $b$  guarantees that  $\text{LCE}_0^{T,T'}(0,0) \leq b$ . Moreover, since  $b \leq \ell'$ , we have  $\mathbb{P}[b > \text{LCE}_k^{T,T'}(0,0)] \leq \mathbb{P}[\ell' > \text{LCE}_k^{T,T'}(0,0)] \leq \exp(-\frac{k+1}{r})$  for each  $k \geq 0$ . Thus,  $b$  satisfies the requirements for  $\overline{\text{LCE}}_r^{T,T'}(0,0)$ .

It remains to analyze the running time. Determining  $\text{per}(T[0..2q))$  in Line 1 costs  $\mathcal{O}(q)$  time using a classic algorithm [31]. The application of Fact 4.7 costs  $\tilde{\mathcal{O}}(\frac{1}{r}|T|)$  time with high probability. Binary search (the loop in Line 6) has  $\mathcal{O}(\log |T|) = \tilde{\mathcal{O}}(1)$  iterations, each implemented in  $\mathcal{O}(q)$  time. Consequently, the total running time is  $\tilde{\mathcal{O}}(\frac{1}{r}|T| + q)$  with high probability.  $\square$

Next, we develop a counterpart of Lemma 3.7.

**Lemma 4.9.** *There is an algorithm that, given strings  $X$  and  $Y$ , a real parameter  $r > 0$ , and an integer range  $J$ , returns  $\overline{\text{LCE}}_r^{X,Y}(0,j)$  for each  $j \in J$ . The algorithm costs  $\tilde{\mathcal{O}}(\frac{1}{r}|X| + |J|)$  time with high probability.*

*Proof.* A procedure  $\text{Batch}(X, Y, r, J)$  implementing Lemma 4.9 is given as Algorithm 7.

First, the algorithm sets  $\Delta := \max J - \min J$  and computes  $\min(\text{LCE}_0^{X,Y}(0,j), 2\Delta)$  for each  $j \in J$ . Implementation details of this step are discussed later on. Then, the algorithm sets



---

**Algorithm 7:** Batch( $X, Y, r, J$ )

---

```
1  $\Delta := \max J - \min J$ ;
2 foreach  $j \in J$  do  $\ell_j := \min(\text{LCE}_0^{X,Y}(0, j), 2\Delta)$ ;
3  $C := \{j \in J : \ell_j = 2\Delta\}$ ;
4 if  $|C| \leq 1$  then
5   | foreach  $j \in C$  do  $\ell_j := \overline{\text{LCE}}_r^{X,Y}(0, j)$ ; ▷ computed using Fact 4.7
6 else
7   |  $\ell^X := \text{FindBreak2}(X, r, 2\Delta)$ ;
8   |  $\ell^Y := \text{FindBreak2}(Y[\min C .. \min(\max C + |X|, |Y|)], r, 2\Delta)$ ;
9   | foreach  $j \in C$  do
10  |   |  $\ell_j := \min(\ell^X, \ell^Y - j + \min C)$ ;
11  |   | if  $\ell_j < \min(|X|, |Y| - j)$  and  $X(\ell_j - 2\Delta .. \ell_j) = Y(j + \ell_j - 2\Delta .. j + \ell_j)$  then
12  |   |   |  $\ell_j := \overline{\text{LCE}}_r^{X,Y}(0, j)$ ; ▷ computed using Fact 4.7
13 return  $(\ell_j)_{j \in J}$ 
```

---

$C := \{j \in J : \ell_j = 2\Delta\}$  so that  $\ell_j = \text{LCE}_0^{X,Y}(0, j)$  holds for each  $j \in J \setminus C$ . Consequently,  $\ell_j$  can be returned as  $\overline{\text{LCE}}_r^{P,T}(0, j)$  for  $j \in J \setminus C$ , and the algorithm indeed returns these values (the values  $\ell_j$  set in Line 2 are later modified only for  $j \in C$ ).

Thus, the remaining focus is on determining  $\overline{\text{LCE}}_r^{X,Y}(0, j)$  for  $j \in C$ . If  $|C| \leq 1$ , then these values are computed explicitly using Fact 4.7. In the case of  $|C| \geq 2$ , handled in Lines 6–12, the algorithm applies the **FindBreak2** function of Lemma 4.8 for  $X$  and  $\bar{Y} := Y[\min C .. \min(\max C + |X|, |Y|)]$ , both with  $q = 2\Delta$ .

These calls are only valid if  $\text{per}(X[0..2\Delta]) \leq \Delta$  and  $\text{per}(\bar{Y}[0..2\Delta]) \leq \Delta$ , so we shall prove that these conditions are indeed satisfied. First, note that  $C = \{o + \min J : o \in \text{Occ}(X[0..2\Delta], Y[\min J .. \max J + 2\Delta])\}$ . Consequently, Fact 2.1 implies that  $C$  is an arithmetic progression with difference  $p := \text{per}(X[0..2\Delta])$ . Due to  $|C| \geq 2$ , we conclude that  $p \leq \max J - \min J = \Delta$ . Moreover, since  $Y[j .. j + 2\Delta] = X[0..2\Delta]$  for each  $j \in C$ , we have  $\bar{Y}[0..2\Delta] = Y[\min C .. \min C + 2\Delta] = X[0..2\Delta]$ , and thus  $\text{per}(\bar{Y}[0..2\Delta]) \leq \Delta$ . Hence, the calls to **FindBreak2** are indeed valid.

Based on the values  $\ell^X$  and  $\ell^Y$  returned by these two calls, the algorithm sets  $\ell_j := \min(\ell^X, \ell^Y - j + \min C)$  for each  $j \in C$  in Line 10. These values satisfy the following property:

**Claim 4.10.** *For each  $j \in C$ , the value  $\ell_j = \min(\ell^X, \ell^Y - j + \min C)$  set in Line 10 satisfies  $\mathbb{P}[\ell_j > \text{LCE}_k^{X,Y}(0, j)] \leq \exp(-\frac{k+1}{r})$  for every integer  $k \geq 0$ .*

*Proof.* Note that  $\ell_j \leq \min(|X|, |Y| - j)$  due to  $\ell^X \leq |X|$  and  $\ell^Y \leq |\bar{Y}|$ . Consequently, if  $\text{LCE}_k^{X,Y}(0, j) = \min(|X|, |Y| - j)$ , then the claim holds trivially. In the following, we assume that  $d := \text{LCE}_k^{X,Y}(0, j) < \min(|X|, |Y| - j)$  so that  $X[0..d]$  and  $Y[j .. j + d]$  are well-defined fragments with  $\text{HD}(X[0..d], Y[j .. j + d]) = k + 1$ .

Consider an infinite string  $X'$  with  $X'[i] = X[i \bmod p]$  for each  $i \geq 0$ , and define  $k_1 = \text{HD}(X[0..d], X'[0..d])$  as well as  $k_2 = \text{HD}(Y[j .. j + d], X'[0..d])$ , observing that the triangle inequality yields  $k_1 + k_2 \geq k + 1$ . Due to  $\ell^X = \overline{\text{LCE}}_r^{X,X'}(0, 0)$  (by Lemma 4.8), we have  $\mathbb{P}[\ell^X > d] \leq \exp(-\frac{k_1}{r})$ , because  $d \geq \text{LCE}_{k_1-1}^{X,X'}(0, 0)$ .

Next, consider an infinite string  $\bar{Y}'$  with  $\bar{Y}'[i] = \bar{Y}[i \bmod p]$  for  $i \geq 0$ , and observe that  $\bar{Y}' = X'$  due to  $\bar{Y}[0..2\Delta] = X[0..2\Delta]$ . Consequently,  $k_2 = \text{HD}(\bar{Y}[j - \min C .. j - \min C + d], \bar{Y}'[0..d])$ . Since  $C$  forms an arithmetic progression with difference  $p$  and  $j \in C$ , we further have  $k_2 = \text{HD}(\bar{Y}[j - \min C .. j - \min C + d], \bar{Y}'[j - \min C .. j - \min C + d])$ . Moreover,  $p = \text{per}(\bar{Y}[0..2\Delta])$  and  $j - \min C \leq \Delta$  yields  $\bar{Y}[0..j - \min C] = \bar{Y}'[0..j - \min C]$ , and therefore  $k_2 = \text{HD}(\bar{Y}[0..j - \min C + d], \bar{Y}'[0..j - \min C + d])$ . We conclude that  $j - \min C + d \geq$

$\text{LCE}_{k_2-1}^{\bar{Y}, \bar{Y}'}(0, 0)$ . Due to  $\ell^Y = \overline{\text{LCE}}_r^{\bar{Y}, \bar{Y}'}(0, 0)$  (by Lemma 4.8), we thus have  $\mathbb{P}[\ell^Y - j + \min C > d] = \mathbb{P}[\ell^Y > j - \min C + d] \leq \exp(-\frac{k_2}{r})$ .

Finally, since the calls to Lemma 4.8 use independent randomness (and thus  $\ell^X$  and  $\ell^Y$  are independent random variables), we conclude that

$$\begin{aligned} \mathbb{P}[\ell_j > \text{LCE}_k^{X,Y}(0, j)] &= \mathbb{P}[\ell_j > d] = \mathbb{P}[\ell^X > d \text{ and } \ell^Y - j + \min C > d] \\ &= \mathbb{P}[\ell^X > d] \cdot \mathbb{P}[\ell^Y - j + \min C > d] \leq \exp(-\frac{k_1}{r}) \cdot \exp(-\frac{k_2}{r}) = \exp(-\frac{k_1+k_2}{r}) \leq \exp(-\frac{k+1}{r}), \end{aligned}$$

which completes the proof.  $\square$

For each  $j \in C$ , after setting  $\ell_j$  in Line 10, the algorithm performs an additional check in Line 11; its implementation is discussed later on. If the check succeeds, then the algorithm falls back to computing  $\ell_j = \overline{\text{LCE}}_r^{X,Y}(0, j)$ , using Fact 4.7, which results in a correct value by definition. Otherwise,  $\ell_j \geq |X|$ ,  $\ell_j \geq |Y| - j$ , or  $X(\ell_j - 2\Delta \dots \ell_j) \neq Y(j + \ell_j - 2\Delta \dots j + \ell_j)$ . Each of these condition yields  $\text{LCE}_0^{X,Y}(0, j) \leq \ell_j$ . Hence, due to Claim 4.10, returning  $\ell_j$  as  $\overline{\text{LCE}}_r^{X,Y}(0, j)$  is then correct. This completes the proof that the values returned by Algorithm 7 are correct.

However, we still need to describe the implementation of Line 2 and Line 11. The values  $\min(\text{LCE}_0^{X,Y}(0, j), 2\Delta)$  needed in Line 2 are determined using an auxiliary string  $T = X[0 \dots 2\Delta]\$Y[\min J \dots \max J + 2\Delta]$ , where  $\$$  and each out-of-bounds character does not match any other character. The PREF table of  $T$ , with  $\text{PREF}_T[t] = \text{LCE}_0^{T,T}(0, t)$  for  $t \in [0 \dots |T|]$ , can be constructed in  $\mathcal{O}(|T|) = \mathcal{O}(|J|)$  time using a textbook algorithm [19] and satisfies  $\min(\text{LCE}_0^{X,Y}(0, j), 2\Delta) = \text{PREF}_T[2\Delta + 1 + j - \min J]$  for each  $j \in J$ .

Our approach to testing  $X(\ell_j - 2\Delta \dots \ell_j) = Y(j + \ell_j - 2\Delta \dots j + \ell_j)$  in Line 11 depends on whether  $\ell_j = \ell^X$  or not. Positions  $j \in C$  with  $\ell_j = \ell^X$  need to be handled only if  $\ell^X < |X|$ . In this case,  $X(\ell_j - 2\Delta \dots \ell_j) = Y(j + \ell_j - 2\Delta \dots j + \ell_j)$  holds if and only if  $X(\ell^X - 2\Delta \dots \ell^X)$  has an occurrence in  $T$  at position  $j + \ell^X - 2\Delta + 1$ . Hence, a linear-time pattern matching algorithm is used to find the occurrences of  $X(\ell^X - 2\Delta \dots \ell^X)$  starting in  $T$  between positions  $\min C + \ell^X - 2\Delta + 1$  and  $\max C + \ell^X - 2\Delta + 1$ , inclusive. Due to  $\max C - \min C \leq \Delta$ , this takes  $\mathcal{O}(\Delta)$  time. Moreover, since  $\text{per}(X(\ell^X - 2\Delta \dots \ell^X)) > \Delta$  holds by Lemma 4.8, Fact 2.1 implies that there is at most one such occurrence, i.e., at most one position  $j$  with  $\ell_j = \ell^X$  passes the test in Line 11.

Next, consider positions  $j \in C$  with  $\ell_j \neq \ell^X$ . Since these positions satisfy  $\ell_j = \ell^Y - j + \min C$ , the condition  $\ell_j < |Y| - j$  implies  $\ell^Y + \min C < |Y|$ . Moreover, the condition  $\ell_j < |X|$  implies  $\ell^Y < |X| + j - \min C \leq |X| + \max C - \min C$ . Consequently, such  $j \in C$  need to be handled only if  $\ell^Y < \min(|Y| - \min C, |X| + \max C - \min C) = |\bar{Y}|$ . Our observation is that  $X(\ell_j - 2\Delta \dots \ell_j) = Y(j + \ell_j - 2\Delta \dots j + \ell_j)$  if and only if  $\bar{Y}(\ell^Y - 2\Delta \dots \ell^Y)$  has an occurrence in  $X$  at position  $\ell^Y - j + \min C - 2\Delta + 1$ . Hence, a linear-time pattern matching algorithm is used to find the occurrences of  $\bar{Y}(\ell^Y - 2\Delta \dots \ell^Y)$  starting in  $X$  between positions  $\ell^Y - \max C + \min C - 2\Delta + 1$  and  $\ell^Y - 2\Delta + 1$ , inclusive. Due to  $\max C - \min C \leq \Delta$ , this takes  $\mathcal{O}(\Delta)$  time. Moreover, since  $\text{per}(\bar{Y}(\ell^Y - 2\Delta \dots \ell^Y)) > \Delta$  holds by Lemma 4.8, Fact 2.1 implies that there is at most one such occurrence, i.e., at most one position  $j$  with  $\ell_j \neq \ell^X$  passes the test in Line 11.

We conclude that Line 11 (across all  $j \in C$ ) can be implemented in  $\mathcal{O}(\Delta)$  time and that Line 12 needs to be executed for at most two indices  $j \in C$ . Consequently, the overall cost of executing Lines 10–12 is  $\tilde{\mathcal{O}}(\frac{1}{r}|X| + \Delta)$  with high probability. Due to  $|\bar{Y}| \leq |X| + \Delta$ , the cost of calls to FindBreak2 of Lemma 4.8 is also  $\tilde{\mathcal{O}}(\frac{1}{r}|X| + \Delta)$  with high probability. As explained above, executing Line 2 costs  $\mathcal{O}(|J|)$  time. The cost of Line 5 is  $\tilde{\mathcal{O}}(\frac{1}{r}|X|)$  with high probability. Due to  $\Delta = |J| - 1$ , the total running time is therefore  $\tilde{\mathcal{O}}(\frac{1}{r}|X| + |J|)$  with high probability.  $\square$

We are now ready to prove a counterpart of Proposition 4.4 for  $\overline{\text{LCE}}_r$  queries.

**Lemma 4.11.** *There is a data structure that, initialized with strings  $X$  and  $Y$ , a real parameter  $r > 0$ , and an integer range  $\Delta$ , answers the following queries: given an integer  $x$ , return  $\overline{\text{LCE}}_r^{X,Y}(x, x + \delta)$  for each  $\delta \in \Delta$ . The initialization costs  $\tilde{\mathcal{O}}(\frac{1}{r}|X|)$  time with high probability, and the queries cost  $\tilde{\mathcal{O}}(|\Delta|)$  time with high probability.*

---

**Algorithm 8:** Implementation of the data structure of Lemma 4.11

---

```

1 Construction( $X, Y, r, \Delta$ ) begin
2    $q := \lceil r|J| \rceil$ ;
3    $x := |X|$ ;
4   foreach  $\delta \in \Delta$  do  $\ell_{x,\delta} := 0$ ;
5   while  $x \geq q$  do
6      $x := x - q$ ;
7      $(v_\delta)_{\delta \in \Delta} := \text{Batch}(X[x..x+q], Y, r, \{x + \delta : \delta \in \Delta\})$ ;
8     foreach  $\delta \in \Delta$  do
9       if  $v_\delta < q$  then  $\ell_{x,\delta} := v_\delta$ ;
10      else  $\ell_{x,\delta} := q + \ell_{x+q,\delta}$ ;
11 Query( $x$ ) begin
12   if  $x \notin [0..|X|]$  then return  $(0)_{\delta \in \Delta}$ ;
13    $x' := x + (|X| - x) \bmod q$ ;
14   if  $x' \neq x$  then
15      $(v_\delta)_{\delta \in \Delta} := \text{Batch}(X[x..x'], Y, r, \{x + \delta : \delta \in \Delta\})$ ;
16     foreach  $\delta \in \Delta$  do
17       if  $v_\delta < x' - x$  then  $\ell_{x,\delta} := v_\delta$ ;
18       else  $\ell_{x,\delta} := x' - x + \ell_{x',\delta}$ ;
19   return  $(\ell_{x,\delta})_{\delta \in \Delta}$ ;

```

---

*Proof.* Procedures  $\text{Construction}(X, Y, r, \Delta)$  and  $\text{Query}(x)$  implementing Lemma 4.11 are given as Algorithm 8.

The construction algorithm precomputes the answers for all  $x \in [0..|X|]$  satisfying  $x \equiv |X| \pmod{q}$ , where  $q = \lceil r|\Delta| \rceil$ . More formally, for each such  $x$ , the data structure stores  $\ell_{x,\delta} = \overline{\text{LCE}}_r^{X,Y}(x, x + \delta)$  for all  $\delta \in \Delta$ . First, the values  $\ell_{|X|,\delta}$  are set to 0. In subsequent iterations, the algorithm computes  $\ell_{x,\delta}$  based on  $\ell_{x+q,\delta}$ . For this, the procedure  $\text{Batch}(X[x..x+q], Y, r, \{x + \delta : \delta \in \Delta\})$  of Lemma 4.9 is called. The resulting values  $\overline{\text{LCE}}_r^{X[x..x+q],Y}(0, x + \delta)$  are then combined with  $\ell_{x+q,\delta} = \overline{\text{LCE}}_r^{X[x+q..|X|],Y}(0, x + q + \delta)$  based on Fact 4.6, which yields  $\overline{\text{LCE}}_r^{X[x..|X|],Y}(0, x + \delta) = \overline{\text{LCE}}_r^{X,Y}(x, x + \delta)$ ; the latter values are stored at  $\ell_{x,\delta}$ .

The cost of a single iteration is  $\tilde{\mathcal{O}}(\frac{q}{r} + |\Delta|) = \tilde{\mathcal{O}}(\frac{q}{r})$  with high probability, and the number of iterations is  $\mathcal{O}(\frac{1}{q}|X|)$ , so the total preprocessing time is  $\tilde{\mathcal{O}}(\frac{1}{r}|X|)$  with high probability.

To answer a query for a given integer  $x$ , the algorithm needs to compute  $\overline{\text{LCE}}_r^{X,Y}(x, x + \delta)$  for all  $\delta \in \Delta$ . If  $x \notin [0..|X|]$ , then these values are equal to 0 by definition. Otherwise, the algorithm computes the nearest integer  $x' \geq x$  with  $x' \equiv |X| \pmod{q}$ . If  $x' = x$ , then the sought values have already been precomputed. Otherwise, the algorithm proceeds based the values  $\ell_{x',\delta}$ . For this, the procedure  $\text{Batch}(X[x..x'], Y, r, \{x + \delta : \delta \in \Delta\})$  of Lemma 4.9 is called. The resulting values  $\overline{\text{LCE}}_r^{X[x..x'],Y}(0, x + \delta)$  are then combined with  $\ell_{x',\delta} = \overline{\text{LCE}}_r^{X[x'..|X|],Y}(0, x' + \delta)$  based on Fact 4.6, which yields the sought values  $\overline{\text{LCE}}_r^{X[x..|X|],Y}(0, x + \delta) = \overline{\text{LCE}}_r^{X,Y}(x, x + \delta)$ .

The cost of a query is  $\tilde{\mathcal{O}}(\frac{x'-x}{r} + |\Delta|) = \tilde{\mathcal{O}}(\frac{q}{r} + |\Delta|) = \tilde{\mathcal{O}}(\Delta)$  with high probability.  $\square$

Finally, we recall that  $\overline{\text{LCE}}_r^{X,Y}(x, y)$  satisfies the requirements for  $\text{LCE}_{\leq k}^{X,Y}(x, y)$  with probability at least  $1 - \exp(-\frac{k+1}{r})$ . Consequently, taking sufficiently small  $r = \tilde{\Theta}(k+1)$  guarantees

success with high probability. Thus, Lemma 4.11 yields Proposition 4.4, which we restate below.

**Proposition 4.4.** *There exists a data structure that, initialized with strings  $X$  and  $Y$ , an integer  $k \geq 0$ , and an integer range  $\Delta$ , answers the following queries: given an integer  $x$ , return  $\text{LCE}_{\leq k}^{X,Y}(x, x + \delta)$  for all  $\delta \in \Delta$ . The initialization costs  $\tilde{O}(\frac{1}{k+1}|X|)$  time with high probability, and the queries cost  $\tilde{O}(|\Delta|)$  time with high probability.*

## 5 PTAS for Aperiodic Strings

In this section, we design an algorithm distinguishing between  $\text{ED}(X, Y) \leq k$  and  $\text{ED}(X, Y) > (1 + \varepsilon)k$ , assuming that  $X$  does not have a length- $\ell$  substring with period at most  $2k$ . The high-level approach of our solution is based on the existing algorithms for Ulam distance [3, 32]. The key tool in these algorithms is a method for decomposing  $X = X_0 \cdots X_m$  and  $Y = Y_0 \cdots Y_m$  into short phrases such that  $\text{ED}(X, Y) = \sum_{i=0}^m \text{ED}(X_i, Y_i)$  if  $\text{ED}(X, Y) \leq k$ . While designing such a decomposition in sublinear time for general strings  $X$  and  $Y$  remains a challenging open problem, the lack of long highly periodic substrings makes this task feasible.

**Lemma 5.1.** *There exists an algorithm that, given strings  $X$  and  $Y$ , integers  $k$  and  $\ell$  such that  $\text{per}(X[i..i + \ell]) > 2k$  for each  $i \in [0..|X| - \ell]$ , and a real parameter  $0 < \delta < 1$ , returns factorizations  $X = X_0 \cdots X_m$  and  $Y = Y_0 \cdots Y_m$  with  $m = \mathcal{O}(\frac{\delta}{(k+1)\ell}|X|)$  such that  $|X_i| \leq \lceil \delta^{-1}(k+1)\ell \rceil$  for each  $i \in [0..m]$  and, if  $\text{ED}(X, Y) \leq k$ , then  $\mathbb{P}[\text{ED}(X, Y) = \sum_{i=0}^m \text{ED}(X_i, Y_i)] \geq 1 - \delta$ . The running time of the algorithm is  $\mathcal{O}(\frac{\delta}{k+1}|X|)$ .*

*Proof.* Let  $q = \lceil \delta^{-1}(k+1)\ell \rceil$ . If  $|X| \leq q$ , then the algorithm returns trivial decompositions of  $X$  and  $Y$  with  $m = 0$ . In the following, we assume that  $q < |X|$ . The algorithm chooses  $r \in [0..q]$  uniformly at random and creates a partition  $X = X_0 \cdots X_m$  so that  $|X_0| = r$ ,  $|X_i| = q$  for  $i \in [1..m)$ , and  $|X_m| \leq q$ . This partition clearly satisfies  $m = \mathcal{O}(\frac{1}{q}|X|) = \mathcal{O}(\frac{\delta}{(k+1)\ell}|X|)$  and  $|X_i| \leq q = \lceil \delta^{-1}(k+1)\ell \rceil$  for each  $i \in [0..m]$ .

Let us define  $x_i$  for  $i \in [0..m+1]$  so that  $X_i = X[x_i..x_{i+1})$  for  $i \in [0..m]$ . For each  $i \in [1..m]$ , the algorithm finds the occurrences of  $X[x_i..x_i + \ell)$  in  $Y$  with starting positions between  $x_i - k$  and  $x_i + k$ . If there is no such occurrence (perhaps due to  $|X_i| < \ell$  for  $i = m$ ), then the algorithm declares a failure and returns a partition  $Y = Y_0 \cdots Y_m$  with  $Y_0 = Y$  and  $Y_i = \varepsilon$  for  $i \geq 1$ . Otherwise, due to the assumption that  $\text{per}(X[x_i..x_i + \ell]) > 2k$ , there is exactly one occurrence, say, at position  $y_i$ . (Recall that the distance between two positions in  $\text{Occ}(P, T)$  is either a period of  $P$  or larger than  $|P|$ .) The algorithm defines  $Y_i = Y[y_i..y_{i+1})$  for  $i \in [0..m]$ , where  $y_0 = 0$  and  $y_{m+1} = |Y|$ . This approach can be implemented in  $\mathcal{O}(m\ell) = \mathcal{O}(\frac{\delta}{k+1}|X|)$  time using a classic linear-time pattern matching algorithm [31].

We shall prove that the resulting partition  $Y = Y_0 \cdots Y_m$  satisfies the requirements. Assuming that  $\text{ED}(X, Y) \leq k$ , let us fix an optimal alignment between  $X$  and  $Y$ . We need to prove that, with probability at least  $1 - \delta$ , the fragments  $X[x_i..x_i + \ell)$  are all matched against  $Y[y_i..y_i + \ell)$ . By optimality of the alignment, this will imply  $\mathbb{P}[\text{ED}(X, Y) = \sum_{i=0}^m \text{ED}(X_i, Y_i)] \geq 1 - \delta$ .

We say that a position  $x \in [0..|X|]$  is an *error* if  $x = |X|$  or (in the alignment considered) the position  $X[x]$  is deleted or matched against a position  $Y[y]$  for  $y \in [0..|Y|)$  such that  $Y[y] \neq X[x]$  or  $Y[y+1]$  is inserted. Each edit operation yields at most one error, so the total number of errors is at most  $k+1$ . Moreover, if  $[x_i..x_i + \ell)$  does not contain any error, then  $X[x_i..x_i + \ell)$  is matched exactly against a fragment of  $Y$ , and that fragment must be  $Y[y_i..y_i + \ell)$  (since we considered all starting positions in  $[x_i - k..x_i + k]$ ). Hence, if the algorithm fails, then there is an error  $x \in [x_i..x_i + \ell)$  for some  $i \in [1..m]$ . By definition of the decomposition  $X = X_0 \cdots X_m$ , this implies  $x \bmod q \in [r..r + \ell) \bmod q$ , or, equivalently,  $r \in (x - \ell..x] \bmod q$ . The probability of this event is  $\frac{\ell}{q} \leq \frac{\delta}{k+1}$ . The union bound across all errors yields an upper bound of  $\delta$  on the failure probability.  $\square$

Next, we present a subroutine that will be applied to individual phrases of the decompositions of Lemma 5.1. Given that the phrases are short, we can afford using the classic Landau–Vishkin algorithm [28] whenever we find out that the corresponding phrases do not match exactly.

**Lemma 5.2.** *There exists an algorithm that, given strings  $X$  and  $Y$ , and an integer  $k \geq 0$ , computes  $\text{ED}(X, Y) > 0$  exactly, taking  $\tilde{O}(|X| + \text{ED}(X, Y)^2)$  time, or certifies that  $\text{ED}(X, Y) \leq k$  with high probability, taking  $\tilde{O}(1 + \frac{1}{k+1}|X|)$  time with high probability.*

*Proof.* The algorithm first checks if  $|X| = |Y|$ , and then it samples  $X$  with sufficiently large rate  $\tilde{\Theta}(\frac{1}{k+1})$  checking whether  $X[i] = Y[i]$  for each sampled position  $i$ . If the checks succeed, then the algorithm certifies that  $\text{ED}(X, Y) \leq k$ . This branch takes  $\tilde{O}(1 + \frac{1}{k+1}|X|)$  time with high probability. Otherwise,  $\text{ED}(X, Y) > 0$ , and the algorithm falls back to a procedure of Landau and Vishkin [28], whose running time is  $\tilde{O}(|X| + |Y| + \text{ED}(X, Y)^2) = \tilde{O}(|X| + \text{ED}(X, Y)^2)$ .

As for correctness, it suffices to show that if the checks succeeded, then  $\text{ED}(X, Y) \leq k$  with high probability. We shall prove a stronger claim that  $\text{HD}(X, Y) \leq k$ . For a proof by contradiction, suppose that  $\text{HD}(X, Y) \geq k + 1$  and consider some  $k + 1$  mismatches. Notice that the sampling rate is sufficiently large that at least one of these mismatches is sampled with high probability. This completes the proof.  $\square$

The next step is to design a procedure which distinguishes between  $\sum_{i=0}^m \text{ED}(X_i, Y_i) \leq k$  and  $\sum_{i=0}^m \text{ED}(X_i, Y_i) \geq (1 + \varepsilon)k$ . Our approach relies on the Chernoff bound: we apply Lemma 5.2 to determine  $\text{ED}(X_i, Y_i)$  for a small sample of indices  $i$ , and then we use these values to estimate the sum  $\sum_{i=0}^m \text{ED}(X_i, Y_i)$ .

**Lemma 5.3.** *There exists an algorithm that, given strings  $X_0, \dots, X_m, Y_0, \dots, Y_m$ , and a real parameter  $0 < \varepsilon < 1$ , returns YES if  $\sum_{i=0}^m \text{ED}(X_i, Y_i) \leq k$ , and returns NO if  $\sum_{i=0}^m \text{ED}(X_i, Y_i) \geq (1 + \varepsilon)k$ . The algorithm succeeds with high probability, and its running time is  $\tilde{O}(qk + k^2 + \frac{n}{\varepsilon^2(k+1)})$ , where  $q = \max_{i=0}^m |X_i|$  and  $n = \sum_{i=0}^m (|X_i| + |Y_i|)$ .*

*Proof.* If  $k = 0$ , then the algorithm naively checks if  $X_i = Y_i$  for each  $i$ , which costs  $\mathcal{O}(n)$  time. In the following, we assume that  $k > 0$ .

For  $i \in [0..m]$  and  $j \in [0..|X_i| + |Y_i|]$ , let us define an indicator  $r_{i,j} = [\text{ED}(X_i, Y_i) > j]$ . Observe that  $\sum_{i=0}^m \text{ED}(X_i, Y_i) = \sum_{i=0}^m \sum_{j=0}^{|X_i|+|Y_i|-1} r_{i,j}$ . The algorithm samples independent random variables  $R_1, \dots, R_r$  distributed as a uniformly random among the  $n$  terms  $r_{i,j}$ , where  $r = \tilde{\Theta}(\frac{n}{\varepsilon^2 k})$  is sufficiently large, and returns YES if and only if  $\sum_{t=1}^r R_t \leq (1 + \frac{\varepsilon}{2}) \frac{rk}{n}$ .

Before we provide implementation details, let us prove the correctness of this approach. If  $\sum_{i=0}^m \text{ED}(X_i, Y_i) \leq k$ , then  $\mathbb{E}[R_t] \leq \frac{k}{n}$ . Consequently, the multiplicative Chernoff bound implies  $\mathbb{P}[\sum_{t=1}^r R_t \geq (1 + \frac{\varepsilon}{2}) \frac{rk}{n}] \leq \exp(-\frac{\varepsilon^2 rk}{12n})$ . Since  $r = \tilde{\Theta}(\frac{n}{\varepsilon^2 k})$  is sufficiently large, the complementary event holds with high probability. Similarly, if  $\sum_{i=0}^m \text{ED}(X_i, Y_i) \geq (1 + \varepsilon)k$ , then  $\mathbb{E}[R_t] \geq \frac{(1+\varepsilon)k}{n}$ . Consequently, the multiplicative Chernoff bound implies  $\mathbb{P}[\sum_{t=1}^r R_t \leq (1 + \frac{\varepsilon}{2}) \frac{rk}{n}] \leq \mathbb{P}[\sum_{t=1}^r R_t \leq (1 - \frac{\varepsilon}{4}) \frac{(1+\varepsilon)k}{n}] \leq \exp(-\frac{\varepsilon^2 r(1+\varepsilon)k}{48n})$ . Since  $r = \tilde{\Theta}(\frac{n}{\varepsilon^2 k})$  is sufficiently large, the complementary event holds with high probability. This finishes the correctness proof.

Evaluating each  $R_t$  consists in drawing a term  $r_{i_t, j_t}$  uniformly at random and testing if  $r_{i_t, j_t} = 1$ , that is, whether  $\text{ED}(X_{i_t}, Y_{i_t}) > j_t$ . For this, the algorithm makes a call to Lemma 5.2. If this call certifies that  $\text{ED}(X_{i_t}, Y_{i_t}) \leq j_t$ , then the algorithm sets  $R_t = 0$ . Otherwise, the call returns the exact distance  $\text{ED}(X_{i_t}, Y_{i_t}) > 0$ , and the algorithm sets  $R_t = 1$  if and only if  $\text{ED}(X_{i_t}, Y_{i_t}) > j_t$ . Moreover, the algorithm stores the distance  $\text{ED}(X_{i_t}, Y_{i_t})$  so that if  $i_{t'} = i_t$  for some  $t' > t$ , then the algorithm uses the stored distance to evaluate  $r_{i_{t'}, j_{t'}}$ , instead of calling Lemma 5.2 again. Whenever the sum of the stored distances exceeds  $k$ , the algorithm terminates and returns NO. Similarly, a call to Lemma 5.2 is terminated preemptively (and NO is returned) if the call takes too much time, indicating that  $\text{ED}(X_{i_t}, Y_{i_t}) > k$ . Consequently,

since the function  $x \mapsto x^2$  is convex, the total running time of the calls to Lemma 5.2 that compute  $\text{ED}(X_{i_t}, Y_{i_t})$  is  $\tilde{O}(qk + k^2)$ . The total cost of the remaining calls can be bounded by  $\tilde{O}(\sum_{t=1}^r \frac{1}{j_t+1} |X_{i_t}|)$ . Since  $\mathbb{E}[\frac{1}{j_t+1} |X_{i_t}| \mid i_t = i] \leq \ln(|X_i| + |Y_i|) + 1 \leq \ln n + 1$ , the total expected running time of these calls is  $\tilde{O}(r) = \tilde{O}(\frac{n}{k})$ . When  $\sum_{t=1}^r \frac{1}{j_t+1} |X_{i_t}|$  exceeds twice the expectation, the whole algorithm is restarted; with high probability, the number of restarts is  $\tilde{O}(1)$ .  $\square$

Finally, we obtain the main result of this section by combining Lemma 5.1 with Lemma 5.3.

**Theorem 5.4.** *There exists an algorithm that, given strings  $X$  and  $Y$ , integers  $k$  and  $\ell$  such that  $\text{per}(X[i..i + \ell]) > 2k$  for each  $i \in [0..|X| - \ell]$ , and a real parameter  $0 < \varepsilon < 1$ , returns YES if  $\text{ED}(X, Y) \leq k$ , and NO if  $\text{ED}(X, Y) \geq (1 + \varepsilon)k$ . With high probability, the algorithm is correct and its running time is  $\tilde{O}(\frac{1}{\varepsilon^2(k+1)} |X| + k^2 \ell)$ .*

*Proof.* The algorithm performs logarithmically many iterations. In each iteration, the algorithm calls Lemma 5.1 (with  $\delta = \frac{1}{2}$ ) to obtain decompositions  $X = X_0 \cdots X_m$  and  $Y = Y_0 \cdots Y_m$ . Then, the phrases are processed using Lemma 5.3. If this subroutine returns YES, then the algorithm also returns YES, because  $\text{ED}(X, Y) \leq \sum_{i=0}^m \text{ED}(X_i, Y_i) < (1 + \varepsilon)k$  with high probability. On the other hand, if each call to Lemma 5.3 returns NO, then the algorithm returns NO. If  $\text{ED}(X, Y) \leq k$ , then with high probability,  $\text{ED}(X, Y) = \sum_{i=0}^m \text{ED}(X_i, Y_i)$  holds in at least one iteration; thus,  $\text{ED}(X, Y) > k$  holds with high probability if the algorithm returns NO.

As for the running time, the calls to Lemma 5.1 cost  $\tilde{O}(\frac{1}{k+1} |X|)$  time, and the calls to Lemma 5.3 cost  $\tilde{O}((k+1)^2 \ell + \frac{1}{\varepsilon^2(k+1)} (|X| + |Y|))$  time because  $\max |X_i| = \mathcal{O}((k+1)\ell)$ .  $\square$

## 6 Random Walk over Samples

In this section, we describe the sampled random walk process. This is used in Section 7 to embed edit distance to Hamming distance in sublinear time.

---

### Algorithm 9: SampledRandomWalk( $X, Y, k, p$ )

---

```

1  $x := 0, y := 0, c := 0;$  ▷ Initialization
2 while  $x < |X|$  and  $y < |Y|$  do
3   Let  $s \sim \text{Bin}(1, \frac{2 \ln n}{p});$  ▷ Biased coin
4   if  $s = 1$  and  $X[x] \neq Y[y]$  then
5     Let  $r \sim \text{Bin}(1, \frac{1}{2});$  ▷ Unbiased coin
6      $x := x + r;$ 
7      $y := y + (1 - r);$ 
8      $c := c + 1;$ 
9   else
10     $x := x + 1;$ 
11     $y := y + 1;$ 
12 return  $c + \max(|X| - x, |Y| - y) \leq 1296k^2;$ 

```

---

Given strings  $X, Y \in \Sigma^{\leq n}$  and integer parameters  $k \geq 0, p \geq 2 \ln n$ , Algorithm 9 scans  $X$  and  $Y$  from left to right. The currently processed positions are denoted by  $x$  and  $y$ , respectively. At each iteration, the algorithm tosses a biased coin: with probability  $\frac{2 \ln n}{p}$ , it chooses to compare  $X[x]$  with  $Y[y]$  and, in case of a mismatch ( $X[x] \neq Y[y]$ ), it tosses an unbiased coin to decide whether to increment  $x$  or  $y$ . In the remaining cases, both  $x$  and  $y$  are incremented. Once the algorithm completes scanning  $X$  or  $Y$ , it returns YES or NO depending on whether the number of mismatches encountered is at most  $1296k^2 - \max(|X| - x, |Y| - y)$ .

**Theorem 6.1.** *Given strings  $X, Y \in \Sigma^{\leq n}$  and integers  $k \geq 0$  and  $2 \ln n \leq p \leq n$ , Algorithm 9 returns YES with probability at least  $\frac{2}{3}$  if  $\text{ED}(X, Y) \leq k$ , and NO with probability at least  $1 - \frac{1}{n}$  if  $\text{ED}(X, Y) \geq (1296k^2 + 1)p$ . Moreover, Algorithm 9 can be implemented in  $\tilde{O}(\frac{n}{p})$  time.*

**YES Case** Recall that the *indel distance*  $\text{IDD}(\cdot, \cdot)$  is defined so that  $\text{IDD}(X, Y)$  is the minimum number of character insertions and deletions needed to transform  $X$  to  $Y$  (the cost of a character substitution is 2 in this setting), and observe that  $\text{ED}(X, Y) \leq \text{IDD}(X, Y) \leq 2\text{ED}(X, Y)$ .

We analyze how  $D := \text{IDD}(X[x..|X|], Y[y..|Y|])$  changes throughout the execution of Algorithm 9. Let  $D_0$  be the initial value of  $D$  and let  $D_i$  be the value of  $D$  after the  $i$ th iteration of the algorithm, where  $i \in [1..t]$  and  $t$  is the total number of iterations. We say that iteration  $i$  is a *mismatch iteration* if the condition in Line 4 is satisfied. The following lemma gathers properties of the values  $D_0, D_1, \dots, D_t$ :

**Lemma 6.2.** *We have  $D_0 = \text{IDD}(X, Y) \leq 2\text{ED}(X, Y)$ . Moreover, the following holds for each iteration  $i \in [1..t]$ :*

- (a)  $D_i$  is a non-negative integer,
- (b) If  $i$  is not a mismatch iteration, then  $D_i \leq D_{i-1}$ .
- (c) If  $i$  is a mismatch iteration, then  $D_i = D_{i-1} - 1$  or  $D_i = D_{i-1} + 1$ , and  $D_i = D_{i-1} - 1$  holds for at least one of the two possible outcomes of  $r$  in Line 5.
- (d) If  $D_{i-1} = 0$ , then  $D_i = 0$ .

*Proof.* Property (a) is clear from the definition of  $D$ .

As for Property (b), let us consider an optimal indel-distance alignment resulting in  $D_{i-1} = \text{IDD}(X[x..|X|], Y[y..|Y|])$  at the beginning of iteration  $i$ , and transform it to an alignment between  $X[x+1..|X|]$  and  $Y[y+1..|Y|]$  by discarding  $X[x]$  and  $Y[y]$  and deleting characters matched with  $X[x]$  or  $Y[y]$ , if any. Note that a character  $X[x']$  with  $x' > x$  is deleted only if the original alignment deletes  $Y[y]$ , and a character  $Y[y']$  with  $y' > y$  is deleted only if the original alignment deletes  $X[x]$ . Hence, the alignment cost does not increase and  $D_i \leq D_{i-1}$ .

As for Property (c), observe that incrementing  $x$  or  $y$  changes the value of  $D$  by exactly 1. Since  $X[x] \neq Y[y]$  holds at the beginning of iteration  $i$ , every optimal alignment between  $X[x..|X|]$  and  $Y[y..|Y|]$  deletes  $X[x]$  or  $Y[y]$ . Incrementing  $x$  or  $y$ , respectively, then results in  $D_i = D_{i-1} - 1$ .

As for Property (d), we note that once  $X[x..|X|] = Y[y..|Y|]$ , no subsequent iteration will be a mismatch iteration.  $\square$

Now, consider a 1-dimensional random walk  $(W_0)_{i \geq 0}$  that starts with  $W_0 = 2k$  and moves 1 unit up or down at every step with equal probability  $\frac{1}{2}$ . Let us couple this random walk with the execution of Algorithm 9. Let  $i_1, \dots, i_c$  be the mismatch iterations. For each  $j \in [1..c]$  such that exactly one choice of  $r$  at iteration  $i_j$  results in decrementing  $D$ , we require that  $W_j = W_{j-1} - 1$  if and only if  $D_{i_j} = D_{i_{j-1}} - 1$ . Otherwise, we keep  $W_j - W_{j-1}$  independent of the execution. As each coin toss in Algorithm 9 uses fresh randomness, the steps of the random walk remain unbiased and independent from each other.

Now, Lemma 6.2 implies that  $D_0 \leq W_0$ , that  $D_{i_j} \leq W_j$  holds for  $j \in [1..c]$ , and that  $D_t \leq W_c$  holds upon termination of Algorithm 9. In particular, the hitting time  $T = \min\{j : W_j = 0\}$  satisfies  $T \geq c + W_c \geq c + D_t$ . However, as proved in [30, Theorem 2.17],  $\mathbb{P}[T \leq N] \geq 1 - \frac{12k}{\sqrt{N}}$ . Thus,  $\mathbb{P}[c + D_t \leq 1296k^2] \geq 1 - \frac{12k}{\sqrt{1296k^2}} = \frac{2}{3}$ . Since  $D_t = \max(|X| - x, |Y| - y)$ , this completes the proof of the YES case.

**NO Case** If  $s = 0$  and  $X[x] \neq Y[y]$  holds at the beginning of some iteration of Algorithm 9, we say the algorithm *misses* the mismatch between  $X[x]$  and  $Y[y]$ . Let us bound probability of missing many mismatches in a row.

**Lemma 6.3.** *Consider the values  $x, y$  at the beginning of iteration  $i$  of Algorithm 9. Conditioned on any random choices made prior to iteration  $i$ , the probability that Algorithm 9 misses the leftmost  $p$  mismatches between  $X[x..|X|)$  and  $Y[y..|Y|)$  is at most  $n^{-2}$ .*

*Proof.* Prior to detecting any mismatch between  $X[x..|X|)$  and  $Y[y..|Y|)$ , the algorithm scans these strings from left to right, comparing the characters at positions sampled independently with rate  $\frac{2\ln n}{p}$ . Hence, the probability of missing the first  $p$  mismatches is  $(1 - \frac{2\ln n}{p})^p \leq n^{-2}$ .  $\square$

Now, observe that an execution of Algorithm 9 yields an edit-distance alignment between  $X$  and  $Y$ : consider values of  $x$  and  $y$  at an iteration  $i$  of the algorithm. If  $i$  is a mismatch iteration, then  $X[x]$  or  $Y[y]$  is deleted depending on whether the algorithm increments  $x$  or  $y$ . Otherwise,  $X[x]$  is aligned against  $Y[y]$  (which might be a substitution). Finally, all  $\max(|X| - x, |Y| - y)$  characters remaining in  $X[x..|X|)$  or  $Y[y..|Y|)$  after the last iteration  $t$  are deleted. The total number of deletions is thus  $c + \max(|X| - x, |Y| - y)$ , and every substitution corresponds to a missed mismatch. By Lemma 6.3, for every block of subsequent non-mismatch iterations, with probability at least  $1 - \frac{1}{n^2}$ , there are at most  $p - 1$  missed mismatches. Overall, with probability at least  $1 - \frac{1}{n}$ , there are at most  $(p - 1)(c + 1)$  missed mismatches, and  $\text{ED}(X, Y) < (c + \max(|X| - x, |Y| - y) + 1)p$ . Hence, the algorithm returns NO if  $\text{ED}(X, Y) \geq (1296k^2 + 1)p$ .

**Efficient Implementation** Finally, we observe that iterations with  $s = 0$  do not need to be executed explicitly: it suffices to repeat the following process: draw (from a geometric distribution  $\text{Geo}(0, \frac{2\ln n}{p})$ ) the number  $\delta$  of subsequent iterations with  $s = 0$ , increase both  $x$  and  $y$  by  $\delta$ , and then execute a single iteration with  $s = 1$ . The total number of iterations is at most  $|X| + |Y| \leq 2n$ , and the number of iterations with  $s = 1$  is  $\mathcal{O}(\frac{n\ln n}{p})$  with high probability.

This completes the proof of Theorem 6.1.  $\square$

## 7 Embedding Edit Distance to Hamming Distance

A randomized embedding of edit distance to Hamming distance is given by a function  $f$  such that  $\text{HD}(f(X, R), f(Y, R)) \approx \text{ED}(X, Y)$  holds with good probability over the randomness  $R$  for any two strings  $X$  and  $Y$ . Chakraborty, Goldenberg, and Koucký [17] gave one such randomized embedding:

**Theorem** ([17, Theorem 1]). *For every integer  $n \geq 1$ , there is an integer  $\ell = \mathcal{O}(\log n)$  and a function  $f : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^{3n}$  such that, for every  $X, Y \in \{0, 1\}^n$ ,*

$$\frac{1}{2}\text{ED}(X, Y) \leq \text{HD}(f(X, R), f(Y, R)) \leq \mathcal{O}(\text{ED}^2(X, Y))$$

*holds with probability at least  $\frac{2}{3} - e^{-\Omega(n)}$  over a uniformly random choice of  $R \in \{0, 1\}^\ell$ . Moreover,  $f$  can be evaluated in linear time.*

Their algorithm utilizes  $3n$  hash functions  $h_1, h_2, \dots, h_{3n}$  mapping  $\{0, 1\}$  to  $\{0, 1\}$ . It scans  $X$  sequentially, and if it is at  $X[x]$  in iteration  $i$ , it appends  $X[x]$  to the embedding and increments  $x$  by  $h_i(X[x])$ . The latter can be viewed as tossing an unbiased coin and, depending on its outcome, either staying at  $X[x]$  or moving to  $X[x + 1]$ . The algorithm uses  $\mathcal{O}(n)$  random bits, which can be reduced to  $\mathcal{O}(\log n)$  using Nisan's pseudorandom number generator [33].

By utilizing random walk over samples, we provide the first sublinear-time randomized embedding from edit to Hamming distance. Given a parameter  $p \geq 2\ln n$ , we sample  $S \subseteq [1..3n]$ , with each index  $i \in [1..3n]$  contained in  $S$  independently with probability  $\frac{2\ln n}{p}$ . We then independently draw  $|S|$  uniformly random bijections  $h_1, h_2, \dots, h_{|S|} : \{0, 1\} \rightarrow \{0, 1\}$ . The shared randomness  $R$  consists of  $S$  and  $h_1, h_2, \dots, h_{|S|}$ . We prove the following theorem.



**Theorem 7.1.** For every integer  $n \geq 1$  and  $p \geq 2 \ln n$ , there is an integer  $\ell = \mathcal{O}(\log n)$  and a function  $f : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^{\tilde{\mathcal{O}}(\frac{n}{p})}$  such that, for every  $X, Y \in \{0, 1\}^n$ ,

$$\frac{\text{ED}(X, Y) - p + 1}{p + 1} \leq \text{HD}(f(X, R), f(Y, R)) \leq \mathcal{O}(\text{ED}^2(X, Y))$$

holds with probability at least  $\frac{2}{3} - n^{-\Omega(1)}$  over a uniformly random choice of  $R \in \{0, 1\}^\ell$ . Moreover,  $f$  can be evaluated in  $\tilde{\mathcal{O}}(\frac{n}{p})$  time.

Algorithm 10 provides the pseudocode of the embedding.

---

**Algorithm 10:** SublinearEmbedding( $X, \langle S, h_1, \dots, h_{|S|} \rangle$ )

---

```

1 Output :=  $\varepsilon$ ,  $j := 0$ ,  $x := 0$ ,  $X' := X \cdot 0^{3n}$ ;
2 for  $i := 1$  to  $3n$  do
3   if  $i \in S$  then
4     Output[ $j$ ] :=  $X'[x]$ ;
5      $j := j + 1$ ;
6      $x := x + h_j(X'[x])$ ;
7   else  $x := x + 1$ ;
8 return Output
```

---

**Interpretation through Algorithm 9** Consider running Algorithm 9 for  $X' = X0^{3n}$  and  $Y' = Y0^{3n}$ , modified as follows: if  $s = 1$  and  $X'[x] = Y'[y]$ , then we still toss the unbiased coin  $r$  and, depending on the result, either increment both  $x$  and  $y$ , or none of them. Observe that this has no impact of the outcome of processing  $(x, y)$ : ultimately, both  $x$  and  $y$  are incremented and the cost  $c$  remains unchanged. Now, let us further modify Algorithm 9 so that its execution uses  $R$  as the source of randomness: we use the event  $\{i \in S\}$  for the  $i$ th toss of the biased coin  $s$  and the value  $h_j(X'[x])$  for the  $j$ th toss of the unbiased coin  $r$  (which is now tossed whenever  $s = 1$ ). If  $S$  was drawn  $[1.. \infty)$ , this would perfectly implement the coins. However,  $S \subseteq [1.. 3n]$ , so the transformation is valid only for the first  $3n$  iterations. Nevertheless, for each iteration, the probability of incrementing  $x$  is  $1 - \frac{\ln n}{p} \geq \frac{1}{2}$ , so  $x \geq n$  and, symmetrically,  $y \geq n$  hold with high probability after iteration  $3n$ , and then Algorithm 9 cannot detect any mismatch. Hence, Algorithm 10 with high probability simulates the treatment of  $X'$  and  $Y'$  by Algorithm 9. Moreover,  $c$  in Algorithm 9 corresponds to the number of iterations with  $s = 1$  and  $X'[x] \neq Y'[y]$ , and this number of iterations is precisely  $\text{HD}(f(X, R), f(Y, R))$ .

**YES Case** Algorithm 9 with  $k = \text{ED}(X, Y) = \text{ED}(X', Y')$  returns YES with probability at least  $\frac{2}{3}$ . Thus,  $\text{HD}(f(X, R), f(Y, R)) = c \leq 1296k^2 = \mathcal{O}(\text{ED}(X, Y)^2)$  with probability at least  $\frac{2}{3}$ .

**NO Case** As proved in Section 6,  $\text{ED}(X', Y') \leq c + \max(|X'| - x, |Y'| - y) + (p - 1)(c + 1)$  holds with probability at least  $1 - \frac{1}{n}$ . Due to  $|X'| = |Y'|$  and  $|x - y| \leq c$ , we deduce  $\text{ED}(X, Y) = \text{ED}(X', Y') \leq 2c + (p - 1)(c + 1) = (p - 1) + (p + 1)c$ , i.e.,  $\text{HD}(f(X, R), f(Y, R)) = c \geq \frac{\text{ED}(X, Y) - p + 1}{p + 1}$ .

**Efficient implementation** To complete the proof, we note that Algorithm 10 can be implemented in  $\mathcal{O}(|S|)$  time by batching the iterations  $i$  with  $i \notin S$  (for each such iteration, it suffices to increment  $i$  and  $x$ ). Moreover,  $|S| = \mathcal{O}(\frac{n \ln n}{p})$  holds with high probability.  $\square$

## References

- [1] Alexandr Andoni, Michel Deza, Anupam Gupta, Piotr Indyk, and Sofya Raskhodnikova. Lower bounds for embedding edit distance into normed spaces. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2003*, pages 523–526, 2003. URL: <http://dl.acm.org/citation.cfm?id=644108.644196>.
- [2] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *51st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 377–386. IEEE, 2010. doi:10.1109/FOCS.2010.43.
- [3] Alexandr Andoni and Huy L. Nguyen. Near-optimal sublinear time algorithms for ulam distance. In Moses Charikar, editor, *21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 76–86. SIAM, 2010. doi:10.1137/1.9781611973075.8.
- [4] Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it’s a constant factor. In Sany Irani, editor, *61st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020. doi:10.1109/FOCS46700.2020.00096.
- [5] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM Journal on Computing*, 41(6):1635–1648, 2012. doi:10.1137/090767182.
- [6] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM Journal on Computing*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.
- [7] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *45th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2004*, pages 550–559. IEEE, 2004. doi:10.1109/FOCS.2004.14.
- [8] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In Lawrence L. Larmore and Michel X. Goemans, editors, *35th Annual ACM Symposium on Theory of Computing, STOC 2003*, pages 316–324. ACM, 2003. doi:10.1145/780542.780590.
- [9] Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 792–801. ACM Press, 2006. URL: <http://dl.acm.org/citation.cfm?id=1109557.1109644>.
- [10] Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In Irit Dinur, editor, *57th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2016*, pages 51–60. IEEE, 2016. doi:10.1109/FOCS.2016.15.
- [11] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. In Artur Czumaj, editor, *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1170–1189. SIAM, 2018. doi:10.1137/1.9781611975031.76.
- [12] Joshua Brakensiek, Moses Charikar, and Aviad Rubinfeld. A simple sublinear algorithm for gap edit distance, 2020. arXiv:2007.14368.
- [13] Joshua Brakensiek, Venkatesan Guruswami, and Samuel Zbarsky. Efficient low-redundancy codes for correcting multiple deletions. *IEEE Transactions on Information Theory*, 64(5):3403–3410, 2018. doi:10.1109/TIT.2017.2746566.

- [14] Joshua Brakensiek and Aviad Rubinfeld. Constant-factor approximation of near-linear edit distance in near-linear time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd Annual ACM Symposium on Theory of Computing, STOC 2020*, pages 685–698. ACM, 2020. doi:10.1145/3357713.3384282.
- [15] Dany Breslauer and Zvi Galil. Finding all periods and initial palindromes of a string in parallel. *Algorithmica*, 14(4):355–366, 1995. doi:10.1007/BF01294132.
- [16] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In Mikkel Thorup, editor, *59th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2018*, pages 979–990. IEEE, 2018. doi:10.1109/FOCS.2018.00096.
- [17] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM Symposium on Theory of Computing, STOC 2016*, pages 712–725. ACM, 2016. doi:10.1145/2897518.2897577.
- [18] Moses Charikar, Ofir Geri, Michael P. Kim, and William Kuszmaul. On estimating edit distance: Alignment, dimension reduction, and embeddings. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 34:1–34:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.34.
- [19] Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific, 2003. doi:10.1142/4838.
- [20] Arthur L. Delcher, Simon Kasif, Robert D. Fleischmann, Jeremy Peterson, Owen White, and Steven L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999. doi:10.1093/nar/27.11.2369.
- [21] Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.1090/S0002-9939-1965-0174934-9.
- [22] Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. In David Zuckerman, editor, *60th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2019*, pages 1101–1120. IEEE, 2019. doi:10.1109/FOCS.2019.00070.
- [23] Elazar Goldenberg, Aviad Rubinfeld, and Barna Saha. Does preprocessing help in fast sequence comparisons? In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd Annual ACM Symposium on Theory of Computing, STOC 2020*, pages 657–670. ACM, 2020. doi:10.1145/3357713.3384300.
- [24] Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In David Zuckerman, editor, *60th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2019*, pages 334–347. IEEE, 2019. doi:10.1109/FOCS.2019.00029.
- [25] Russell Impagliazzo and Ramamohan Paturi. On the complexity of  $k$ -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

- [26] Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd Annual ACM Symposium on Theory of Computing, STOC 2020*, pages 699–712. ACM, 2020. doi:10.1145/3357713.3384307.
- [27] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998. doi:10.1137/S0097539794264810.
- [28] Gad M. Landau and Uzi Vishkin. Fast string matching with  $k$  differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.
- [29] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965. URL: <http://mi.mathnet.ru/eng/dan31411>.
- [30] David Levin and Yuval Peres. *Markov Chains and Mixing Times*. American Mathematical Society, 2017. doi:10.1090/mbk/107.
- [31] James H. Morris, Jr. and Vaughan R. Pratt. A linear pattern-matching algorithm. Technical Report 40, Department of Computer Science, University of California, Berkeley, 1970.
- [32] Timothy Naumovitz, Michael E. Saks, and C. Seshadhri. Accurate and nearly optimal sublinear approximations to ulam distance. In Philip N. Klein, editor, *28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 2012–2031. SIAM, 2017. doi:10.1137/1.9781611974782.131.
- [33] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. doi:10.1007/BF01305237.
- [34] Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *Journal of the ACM*, 54(5):23, 2007. doi:10.1145/1284320.1284322.
- [35] Barna Saha. The Dyck language edit distance problem in near-linear time. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014*, pages 611–620. IEEE, 2014. doi:10.1109/FOCS.2014.71.
- [36] Haoyu Zhang and Qin Zhang. Embedjoin: Efficient edit similarity joins via embeddings. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2017*, pages 585–594. ACM, 2017. doi:10.1145/3097983.3098003.