2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)

Sublinear-Time Algorithms for Computing & Embedding Gap Edit Distance

Tomasz Kociumaka
Bar-Ilan University
Ramat Gan, Israel
kociumaka@mimuw.edu.pl

Barna Saha
University of Calfornia, Berkeley
Berkeley, U.S.
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 a greedy algorithm that distinguishes in time $\tilde{\mathcal{O}}(\frac{n}{k}+k^2)$ between length-n input strings with edit distance at most k and those with edit distance more than $4k^2$. This is an improvement and a simplification upon the main result of [Goldenberg, Krauthgamer, Saha, FOCS 2019], where the k vs $\Theta(k^2)$ gap edit distance problem is solved in $\tilde{\mathcal{O}}(\frac{n}{k}+k^3)$ time. We further generalize our result to solve the k vs αk gap edit distance problem in time $\tilde{\mathcal{O}}(\frac{n}{\alpha}+k^2+\frac{k}{\alpha}\sqrt{nk})$, strictly improving upon the previously known bound $\tilde{\mathcal{O}}(\frac{n}{\alpha}+k^3)$. Finally, we show that if the input strings do not have long highly periodic substrings, then the gap edit distance problem can be solved in sublinear time within any factor $\alpha > 1$. Specifically, if the strings contain no substring of length ℓ with the shortest period of length at most $2k$, then the k vs $(1+\varepsilon)k$ gap edit distance problem can be solved in time $\tilde{\mathcal{O}}(\frac{n}{\varepsilon^2 k}+k^2\ell)$.**

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

***Keywords*-edit distance; sublinear algorithms; embedding;**

I. Introduction

The edit distance, also known as the Levenshtein distance [28], is a basic measure of sequence similarity. For two strings X and Y over an alphabet Σ, the edit distance $\mathsf{ED}(X,Y)$ is defined as the minimum number of character insertions, deletions and substitutions required for converting X into Y. A natural dynamic programming computes edit distance of two strings of total length n in $\mathcal{O}(n^2)$ time. While for many applications running a quadratic-time algorithm is prohibitive, the Strong Exponential Time Hypothesis (SETH) [24] implies that there is no truly subquadratic-time algorithm that computes edit distance exactly [6].

A full version of this paper is available at arxiv.org/abs/2007.12762. Proofs of the claims marked with ♠ are presented only in the full version.

The last two decades have seen a surge of interest in designing fast approximation algorithms for edit distance computation [4], [14], [25], [21], [16], [11], [5], [2], [9], [7], [8]. A breakthrough result of Chakraborty, Das, Goldenberg, Koucký, and Saks provided the first constant-factor approximation of edit distance in 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 provide a constant-factor approximation in near-linear time: initially by Brakensiek and Rubinstein [14] as well as Koucký and Saks [25] for the regime of near-linear edit distance, and then by Andoni and Nosatzki [4] for the general case. Designing fast algorithms for edit distance has also been considered in other models, such as in the quantum and massively parallel framework [11], and when independent preprocessing of each string is allowed [22].

In this paper, we focus on sublinear-time algorithms for edit distance, the study of which was initiated by Batu et al. [8], and then continued in [3], [5], [31], [21]. Here, the goal is to distinguish, in time sublinear in n, whether the edit distance is below k or above k' for some $k' > k$. This is known as the *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 [19]. 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, the sequences are highly repetitive, 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 $k = n^\eta$ and $k' = \Omega(n)$ in $\mathcal{O}(n^{\max(2\eta-1,\eta/2)})$ time. However, their algorithm crucially depends on k' being $\Omega(n)$, and cannot distinguish between, say, $n^{0.1}$ and $n^{0.99}$. A more recent algorithm by Andoni and Onak [5] resolves this issue and can distinguish between $k = n^\eta$ and $k' = n^\beta$, with $\beta > \eta$, in $\mathcal{O}(n^{2+\eta-2\beta+o(1)})$ time. However, if we want to distinguish between k vs k^2, then the algorithm of [5] achieves sublinear time only when $k = \omega(n^{1/3})$. (Setting $k' = k^2$ 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 k^2.) In a recent

2575-8454/20/$31.00 ©2020 IEEE
DOI 10.1109/FOCS46700.2020.00112

work, Goldenberg et al. [21] gave an algorithm solving the k vs k^2 gap edit distance in $\tilde{\mathcal{O}}(\frac{n}{k} + k^3)$ time[1], thereby providing a sublinear-time algorithm for *the quadratic gap edit distance* as long as $k = o(n^{1/3})$ and $k = \omega(1)$.

Bar-Yossef et al. [7] introduced the term *gap edit distance* and solved the quadratic gap 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 [27] solves the edit distance problem 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. [21] raised a question whether it is possible to solve the quadratic gap edit problem in sublinear time for all $k = \tilde{\omega}(1)$. In particular, the running times of the algorithms of Goldenberg et al. [21] and Andoni and Onak [5] algorithms meet at $k = n^{1/3}$, when they become nearly-linear. In light of the $\mathcal{O}(n + k^2)$-time exact algorithm [27], the presence of a k^3 term in the time complexity of [21] is undesirable, and it is natural to ask if the dependency can be reduced. In particular, for $k = O(n^{1/3})$ if the polynomial dependency on k can be reduced to k^2, then the contribution of that term can be neglected 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 [21] 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 [21] and simultaneously provides a conceptual simplification.

k vs αk Gap Edit Distance: Combining the greedy approach with the structure of computations in [27], we can solve the k vs αk gap edit distance problem in $\tilde{\mathcal{O}}(\frac{n}{\alpha} + k^2 + \frac{k}{\alpha}\sqrt{kn})$ time. This improves upon the $\tilde{\mathcal{O}}(\frac{n}{\alpha}+k^3)$ time bound of [21] for all values of α and k.

k vs $(1+\varepsilon)k$ Gap Edit Distance: We can distinguish edit distance at most k and at least $(1+\varepsilon)k$ in $\tilde{\mathcal{O}}(\frac{n}{\varepsilon^2 k} + \ell k^2)$ time as long as there is no length-ℓ substring with the shortest period of length 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], [31]. Note that not only we can allow character repetition, we get an $(1+\varepsilon)$-approximation as long as the same periodic structure does not continue for more than ℓ consecutive positions, or the period length is large. This is the case with most text corpora, and for biological sequences with interspersed repeats.

Embedding to Hamming Distance: Along with designing fast approximation algorithms for edit distance, a parallel line of works have investigated how edit distance can be embedded into other metric spaces, especially to the Hamming space [1], [9], [33], [17], [18]. Indeed such embedding results have led to new approximation algorithms for edit distance (e.g., the embedding of [33], [9] applied in [5], [9]), new streaming algorithms and document exchange protocols (e.g., the embedding of [17] applied in [17], [10]). In particular, Chakraborty, Goldenberg, and Koucký [17] provided a probabilistic embedding of edit distance to Hamming distance with quadratic distortion. Their algorithm runs in linear time, and if the original edit distance between two given sequences were k, then the Hamming distance between the resultant sequences is bounded between k and k^2. 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 [34]. 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 [34], [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 $\tilde{\mathcal{O}}(\frac{n}{p})$ time, and guarantees that (with good probability) the Hamming distance of the resultant strings is between $\frac{k}{p}$ and 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 at most p. Just like the algorithm of [17] has been very influential (see its applications in [10], [13], [35], [23]), 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 exact algorithm [27] tests if $\mathsf{ED}(X,Y) \le k$ 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 $j \in [-k\,..\,k]$,[2] aiming at $d_{i,j} = \max\{x : \mathsf{ED}(X[0\,..\,x), Y[0\,..\,x+j)) \le i\}$. In terms of the table of all distances $\mathsf{ED}(X[0\,..\,x), Y[0\,..\,y))$, the value $d_{i,j}$ can be interpreted as the (index of) the farthest cell on the jth *diagonal* with value i or less. After a linear-time preprocessing of X and Y, the cells $d_{i,j}$ can be filled in $\mathcal{O}(1)$ each, which results in $\mathcal{O}(n+k^2)$ overall running time.

Our algorithm for the quadratic gap edit distance problem follows the basic framework of [27]. However, instead of

[1]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.

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

computing each value $d_{i,j}$, it only computes values d_i for $i = [0\,..\,k]$, interpreted as relaxed version of $\max_{j=-k}^{k} d_{i,j}$ allowing for an $\mathcal{O}(k)$-factor underestimation of the number of edits i. Testing whether the value d_k satisfies $d_k < |X|$ is sufficient to distinguish k vs k^2. 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 $\tilde{\mathcal{O}}(k)$ additional characters. This yields $\tilde{\mathcal{O}}(\frac{n}{k}+k^2)$ total running time. Our algorithm not only improves upon the main result of Goldenberg et al. [21], but also provides a conceptual simplification. Indeed, the algorithm of [21] also follows the basic ideas of [27], but it identifies each value $d_{i,j}$ (relaxed in a similar way), paying an extra $\tilde{\Theta}(k)$ time per $\Theta(k^2)$ values. In order to do so, the algorithm follows a more complex row-by-row approach of an online version [26] of the Landau–Vishkin algorithm.

To improve the gap to k vs αk for $\alpha < k$, greedily computing the values d_i for $i \in [0\,..\,k]$ is not sufficient. (It is enough, though, for $\alpha \geq k$, where the time complexity becomes $\tilde{\mathcal{O}}(\frac{n}{\alpha} + k^2)$ if we simply decrease the sampling rate to $\tilde{\mathcal{O}}(\frac{1}{\alpha})$.) 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 k vs αk for $\alpha < k$, we would like to approximate the number of edits within a factor α. In order to do so, we decompose the entire set of $2k+1$ diagonals into $\mathcal{O}(\frac{k}{\alpha})$ groups each consisting of consecutive α 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 with α approximation ratio on the number of edits. Computing each of the $\mathcal{O}(k)$ values for each wide diagonal requires reading $\mathcal{O}(\alpha)$ extra characters. Therefore, for each wide diagonal, the running time is $\tilde{\mathcal{O}}(\frac{n}{\alpha}+\alpha k)$, and across the $\mathcal{O}(\frac{k}{\alpha})$ wide diagonals, the total time complexity becomes $\tilde{\mathcal{O}}(\frac{nk}{\alpha^2} + k^2)$. This bound is incomparable to $\tilde{\mathcal{O}}(\frac{n}{\alpha}+k^3)$ of [21]: While we pay less on the second term, the uniform sampling rate increases. In order to decrease the first term, instead of computing over each wide diagonal independently, we provide a synchronization mechanism so that the global uniform sampling rate remains $\tilde{\mathcal{O}}(\frac{1}{\alpha})$. This leads to an implementation with running time $\tilde{\mathcal{O}}(\frac{n}{\alpha} + \frac{k^3}{\alpha})$, which already improves upon [21]. However, synchronizing only over smaller groups of wide diagonals, we can achieve the running time of $\tilde{\mathcal{O}}(\frac{n}{\alpha} + k^2 + \frac{k}{\alpha}\sqrt{kn})$, which subsumes both $\tilde{\mathcal{O}}(\frac{nk}{\alpha^2} + k^2)$ and $\tilde{\mathcal{O}}(\frac{n}{\alpha} + \frac{k^3}{\alpha})$.

Our algorithm distinguishing k vs $(1+\varepsilon)k$ for strings without length-ℓ substrings with the shortest period of length at most $2k$ follows a very different approach, inspired by the existing solutions for estimating the Ulam distance [3], [31]. This method consists of three ingredients. First, we decompose $X = X_0 \cdots X_m$ and $Y = Y_0 \cdots Y_m$ into phrases of length $\mathcal{O}(\ell k)$ such that if $\mathsf{ED}(X,Y) \leq k$, then $\sum_{i=0}^{m} \mathsf{ED}(X_i,Y_i) \leq k$ holds with good probability (note that $\mathsf{ED}(X,Y) \geq \mathsf{ED}(X_i,Y_i)$ is always true). The second ingredient estimates $\mathsf{ED}(X_i,Y_i)$ for any given i. This subroutine is then applied for a random sample of indices i by the third ingredient, which distinguishes between $\sum_{i=0}^{m} \mathsf{ED}(X_i,Y_i) \leq k$ and $\sum_{i=0}^{m} \mathsf{ED}(X_i,Y_i) > (1+\varepsilon)k$ 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 ℓ 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 $\mathcal{O}(\frac{n}{\varepsilon^2 k}+\ell k^2)$. This helps to keep our implementation of the other two ingredients much simpler than their counterparts in [3], [31].

A simple *random deletion* process, introduced in [34], solves the k vs k^2 gap edit distance problem in linear time. The algorithm simultaneously scans X and Y from left to right. If the two currently processed symbols $X[x]$ and $Y[y]$ match, they are aligned, and the algorithm proceeds to $X[x+1]$ and $Y[y+1]$. Otherwise, one of the symbols 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 k^2p gap edit distance problem. Finally, we observe that this process can be implemented in $\tilde{\mathcal{O}}(\frac{n}{p})$ 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 now the decision whether the algorithm stays at $X[x]$ or moves to $X[x+1]$ can now be made without accessing $Y[y]$. 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 bijection $h_i : \{0,1\} \to \{0,1\}$. 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 Hamming distance between $\frac{k}{p}$ and $\mathcal{O}(k^2)$ with good probability.

Organization: After introducing the main notations in Section II, we provide the algorithm and analysis of the quadratic gap edit distance problem in Section III. In Section IV, we elaborate on our results on k vs αk gap edit distance problem for $\alpha < k$. The $(1+\varepsilon)$ approximation to gap edit distance problem without the presence of long periodic strings is given in Section V. The procedure of random walk over samples is described in Section VI. Finally, the embedding result is provided in Section VII.

Further Remarks: We have recently been aware of an independent work that uses a greedy algorithm similar to ours for the sublinear gap edit distance problem and achieves a running time of $\tilde{O}(\frac{n}{\sqrt{k}})$ to distinguish k vs $\Theta(k^2)$ [12]. This is in contrast to our bound of $\tilde{O}(\frac{n}{k}+k^2)$, which is superior for $k \le n^{\frac{2}{5}}$. At the same time, for $k \ge n^{\frac{2}{5}+o(1)}$, the algorithm of Andoni and Onak [5] has a better running time. We also remark here that there exists an even simpler algorithm (by now folklore) that has query complexity $\tilde{\mathcal{O}}(\frac{n}{\sqrt{k}})$. The algorithm samples both sequences X and Y independently with probability $\tilde{\Theta}(\frac{1}{\sqrt{k}})$ so that $\Pr[X[x]$ and $Y[x+d]$ are sampled$] = \tilde{\Theta}(\frac{1}{k})$ for all $x \in [1\mathinner{..}n]$ and $d \in [-k\mathinner{..}k]$. Then, by running the Landau–Vishkin algorithm [27] suitably over the sampled sequences, one can solve the k vs k^2 gap edit distance problem. Nevertheless, this is significantly worse than the bounds achieved here. It remains open to characterize a tight lower bound for the quadratic gap edit distance problem.

II. Preliminaries

A *string* X is a finite sequence of characters from an *alphabet* Σ. The length of X is denoted by $|X|$ and, for $i \in [0\mathinner{..}|X|)$, the ith 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 $0 \le \ell \le r \le |X|$. We then say that Y *occurs* in X at position ℓ. The set of positions where Y occurs in X is denoted $\mathrm{Occ}(Y, X)$. The *occurrence* of Y at position ℓ in X is denoted by $X[\ell\mathinner{..}r)$ or $X[\ell\mathinner{..}r-1]$. Such an occurrence is a *fragment* of X, and it can be represented by (a pointer to) X and a pair of indices $\ell \le r$. Two fragments (perhaps of different strings) *match* if they are occurrences of the same substring. A fragment $X[\ell\mathinner{..}r)$ 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 $i \in [0\mathinner{..}|X|-p)$. We define $\mathrm{per}(X)$ to be the smallest period of X. The following result relates periods to occurrences:

Fact II.1 (Breslaurer and Galil [15, Lemma 3.2])**.** *If strings P, T satisfy $|T| \le \frac{3}{2}|P|$, then $\mathrm{Occ}(P, T)$ forms an arithmetic progression with difference $\mathrm{per}(P)$.*

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, $\mathsf{HD}(X, Y) = |\{i \in [0\mathinner{..}|X|) : X[i] \ne Y[i]\}|$. The edit distance between two strings X and Y is denoted $\mathsf{ED}(X, Y)$.

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

Our algorithms rely on *approximate* LCE queries.

Definition II.2. Let X, Y be strings and let $k \ge 0$ be an integer. For integers x, y, we set $\mathrm{LCE}_{\le k}^{X,Y}(x, y)$ as any value satisfying $\mathrm{LCE}_0^{X,Y}(x, y) \le \mathrm{LCE}_{\le k}^{X,Y}(x, y) \le \mathrm{LCE}_k^{X,Y}(x, y)$.

III. Improved Gap Edit Distance

The classic Landau–Vishkin exact algorithm [27] for testing if $\mathsf{ED}(X, Y) \le k$ is given below as Algorithm 1. The key property of this algorithm is that $d_{i,j} = \max\{x : \mathsf{ED}(X[0\mathinner{..}x), Y[0\mathinner{..}x+j)) \le i\}$ holds for each $i \in [0\mathinner{..}k]$ and $j \in [-k\mathinner{..}k]$. Since $\mathrm{LCE}_0^{X,Y}$ 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 [27]

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

The main idea behind the algorithm of Goldenberg et al. [21] is that if LCE_0 queries are replaced with $\mathrm{LCE}_{\le k}$ queries, then the algorithm is still guaranteed to return YES if $\mathsf{ED}(X, Y) \le k$ and NO if $\mathsf{ED}(X, Y) > k(k+2)$. The cost of their algorithm is $\tilde{\mathcal{O}}(\frac{1}{k+1}|X|)$ plus $\tilde{\mathcal{O}}(k)$ per $\mathrm{LCE}_{\le k}$ 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 [26] of the Landau–Vishkin algorithm.

An auxiliary result of this paper is that $\mathrm{LCE}_{\le k}^{X,Y}(x, y)$ queries with $|x-y| \le k$ 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 [21]. In fact, we show that $\tilde{\mathcal{O}}(k)$ time is sufficient to answer *all* queries $\mathrm{LCE}_{\le k}^{X,Y}(x, y)$ with fixed x and arbitrary $y \in [x-k\mathinner{..}x+k]$.

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 LCE_0 queries to $\mathrm{LCE}_{\le k}$ queries yields a cost of up to k mismatches for every $\mathrm{LCE}_{\le k}^{X,Y}(x, y)$ 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     $d_i := d'_i + \max_{\delta=-k}^{k} \mathrm{LCE}_{\le k}^{X,Y}(d'_i, d'_i+\delta)$;
4     $d'_{i+1} := \min(|X|, d_i + 1)$;
5 if $||X| - |Y|| \le k$ and $d_k = |X|$ then return YES;
6 else return NO;
```

Lemma III.1. *Algorithm 2 returns YES if* $\mathsf{ED}(X,Y) \le k$ *and NO if* $\mathsf{ED}(X,Y) > (3k+5)k$.

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

Claim III.2. *Each* $i \in [0\,..\,k]$ *has the following properties:*

1) $\mathsf{ED}(X[0\,..\,d'_i), Y[0\,..\,y)) \le (3k+1)i + k$ *for every* $y \in [d'_i - k\,..\,d'_i + k] \cap [0\,..\,|Y|]$*;*
2) $\mathsf{ED}(X[0\,..\,d_i), Y[0\,..\,y)) \le (3k+1)i + 4k$ *for every* $y \in [d_i - k\,..\,d_i + k] \cap [0\,..\,|Y|]$.

Proof: We proceed by induction on i. Our base case is Property 1 for $i = 0$. Since $d'_0 = 0$, we have $\mathsf{ED}(X[0\,..\,d'_0), Y[0\,..\,y)) = y \le k$ for $y \in [d'_0 - k\,..\,d'_0 + k] \cap [0\,..\,|Y|]$.

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

Finally, we shall prove that Property 1 holds for $i > 0$ assuming that Property 2 is true for $i-1$. Since $d'_i \le d_{i-1}+1$, the assumption yields $\mathsf{ED}(X[0\,..\,d'_i - 1), Y[0\,..\,y-1)) \le (3k+1)(i-1) + 4k$, and thus $\mathsf{ED}(X[0\,..\,d'_i), Y[0\,..\,y)) \le 1 + (3k+1)(i-1) + 4k = (3k+1)i + k$. ■

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

Claim III.3. *If* $\mathsf{ED}(X[0\,..\,x), Y[0\,..\,y)) = i \in [0\,..\,k]$ *for* $x \in [0\,..\,|X|]$ *and* $y \in [0\,..\,|Y|]$*, then* $x \le d_i$.

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 $x \le d'_i + \max_{\delta=-k}^{k} \mathrm{LCE}_0^{X,Y}(d'_i, d'_i + \delta)$. Since $d_i \ge d'_i + \max_{\delta=-k}^{k} \mathrm{LCE}_0^{X,Y}(d'_i, d'_i + j)$ holds by definition of $\mathrm{LCE}_{\le k}$ queries, this implies the claim.

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

For $i > 0$, we consider an optimal alignment between $X[0\,..\,x)$ and $Y[0\,..\,y)$, and we distinguish its maximum prefix with $i-1$ errors. This yields positions $x', x'' \in [0\,..\,x]$ and $y', y'' \in [0\,..\,y]$ with $x'' - x' \in \{0,1\}$ and $y'' - y' \in \{0,1\}$ such that $\mathsf{ED}(X[0\,..\,x'), Y[0\,..\,y')) = i - 1$ and $X[x''\,..\,x) = Y[y''\,..\,y)$. The inductive assumption yields $x' \le d_{i-1}$, which implies $x'' \le \min(x, d_{i-1} + 1) \le d'_i$. Due to $X[x''\,..\,x) = Y[y''\,..\,y)$, we have $\mathrm{LCE}_0^{X,Y}(x'', y'') \ge x - x''$. By $x'' \le d'_i$, this implies $\mathrm{LCE}_0^{X,Y}(d'_i, d'_i + y - x) \ge x - d'_i$. Since $|y - x| \le k$, we conclude that $x = d'_i + (x - d'_i) \le d'_i + \mathrm{LCE}_0^{X,Y}(d'_i, d'_i + y - x) \le d'_i + \max_{\delta=-k}^{k} \mathrm{LCE}_0^{X,Y}(d'_i, d'_i + \delta)$. ■

Hence, the algorithm returns YES if $\mathsf{ED}(X,Y) \le k$. ■

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

Proposition III.4. *There exists an algorithm that, given strings X and Y, an integer* $k \ge 0$*, an index i, and a range of indices J, computes* $\ell := \max_{j \in J} \mathrm{LCE}_{\le k}^{X,Y}(i,j)$*. W.h.p. the algorithm is correct and its running time is* $\tilde{\mathcal{O}}(\frac{\ell}{k+1} + |J|)$.

Theorem III.5. *There exists an algorithm that, given strings X and Y, and an integer* $k \ge 0$*, returns YES if* $\mathsf{ED}(X,Y) \le k$*, and NO if* $\mathsf{ED}(X,Y) > (3k+5)k$*. W.h.p. the algorithm is correct and its running time is* $\tilde{\mathcal{O}}(\frac{1}{k+1}|X| + k^2)$.

Proof: The pseudocode is given in Algorithm 2. Queries $\mathrm{LCE}_{\le k}$ are implemented using Proposition III.4. W.h.p. all the queries are answered correctly. Conditioned on this assumption, Lemma III.1 yields that Algorithm 2 is correct w.h.p. It remains to analyze the running time. The cost of instructions other than $\mathrm{LCE}_{\le k}$ queries is $\mathcal{O}(k)$. By Proposition III.4, the cost of computing d_i is $\tilde{\mathcal{O}}(\frac{1}{k+1}(d_i - d'_i) + k)$. Due to $0 \le d'_0 \le d_0 \le d'_1 \le d_1 \le \cdots \le d'_k \le d_k \le |X|$, this sums up to $\tilde{\mathcal{O}}(\frac{1}{k+1}|X| + k^2)$ across all queries. ■

A. Proof of Proposition III.4

Our implementation of $\max_{j \in J} \mathrm{LCE}_{\le k}^{X,Y}(i,j)$ queries heavily borrows from [21]. 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 [21] as black boxes; thus, we opt for a self-contained presentation.

On the highest level, in Lemma III.7, we develop an oracle that, additionally given a threshold ℓ, must return YES if $\max_{j \in J} \mathrm{LCE}_0^{X,Y}(i,j) \ge \ell$, must return NO if $\max_{j \in J} \mathrm{LCE}_k^{X,Y}(i,j) < \ell$, and may return an arbitrary answer otherwise. The final algorithm behind Proposition III.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 $\mathsf{HD}(X[i\,..\,i+\ell), Y[j\,..\,j+\ell)) > k$ for every $j \in J$. Now, if $\ell \le 3|J|$, then we can afford running a classic exact pattern matching algorithm [30] 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 II.1 implies that $X[i\mathinner{.\,.} i+3|J|)$ is periodic with period $p \le |J|$ and that the candidate positions form an arithmetic progression with difference p. We then check whether p remains a period of $X[i\mathinner{.\,.} i+\ell)$, and of $Y[j\mathinner{.\,.} j+\ell)$ for the leftmost candidate j. Even if either check misses $\frac{k}{2}$ mismatches with respect to the period, two positive answers guarantee $\mathsf{HD}(X[i\mathinner{.\,.} i+\ell), Y[j\mathinner{.\,.} j+\ell)) \le k$, which lets us return YES. Thus, the periodicity check (Lemma III.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 $\operatorname{per}(B) > |J|$. For this, we utilize a binary-search-based procedure, which is very similar to finding "*period transitions*" in [21]. Whenever $X[i\mathinner{.\,.} i+\ell) = Y[j\mathinner{.\,.} j+\ell)$, 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 III.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 responsible for certificating (approximate) periodicity or finding a break.

Lemma III.6. *There exists an algorithm that, given a string T and integers $q, k \ge 0$ such that $|T| \ge 2q$, returns either*
- *a length-$2q$* break *B in T such that $\operatorname{per}(B) > q$, or*
- *$\perp$, certifying that $p := \operatorname{per}(T[0\mathinner{.\,.} 2q)) \le q$ and that $|\{i \in [0\mathinner{.\,.} |T|) : T[i] \ne T[i \bmod p]\}| \le k$.*

W.h.p. the algorithm is correct and costs $\tilde{\mathcal{O}}(\frac{|T|}{k+1} + q)$ time.

Proof: A procedure $\mathsf{FindBreak}(T, q, k)$ implementing Lemma III.6 is given as Algorithm 3.

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

Otherwise, the algorithm tries to check if $\perp$ can be returned. If we say that a position $i \in [0\mathinner{.\,.} |T|)$ 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 $S \subseteq [0\mathinner{.\,.} |T|)$ with a sufficiently large rate $\tilde{\mathcal{O}}(\frac{1}{k+1})$. Such sampling rate guarantees that if there are at least $k+1$ incompatible positions, then w.h.p. 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 w.h.p.

Algorithm 3: $\mathsf{FindBreak}(T,\ q,\ k)$
```
1  $p := \operatorname{per}(T[0\mathinner{.\,.} 2q))$;
2  if $p > q$ then return $T[0\mathinner{.\,.} 2q)$;
3  Let $S \subseteq [0\mathinner{.\,.} |T|)$ with each element sampled
     independently with sufficiently large rate $\tilde{\Theta}(\frac{1}{k+1})$;
4  foreach $s \in S$ do
5      if $T[s] \ne T[s \bmod p]$ 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 $T[j] \ne T[j \bmod p]$ then $e := j$;
11             if $e \ge m$ then $b := m$;
12         return $T(b - 2q\mathinner{.\,.} b]$
13 return $\perp$
```

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 $2q \le b \le e < |T|$ such that e is incompatible and positions in $[b-2q\mathinner{.\,.} b)$ are all compatible. The initial choice of $b := 2q$ and $e := s$ satisfies the invariant because positions in $[0\mathinner{.\,.} 2q)$ are all compatible due to $p = \operatorname{per}(T[0\mathinner{.\,.} 2q))$. While $b < e$, the algorithm chooses $m := \lceil \frac{b+e}{2} \rceil$. If $[m-2q\mathinner{.\,.} m)$ contains an incompatible position j, then $j \ge b$ (because $j \ge m - 2q \ge b - 2q$ and positions in $[b-2q\mathinner{.\,.} b)$ are all compatible), so the algorithm maintains the invariant setting $e := j$ for such a position j (Line 10). Otherwise, positions in $[m-2q\mathinner{.\,.} m)$ are all compatible. Due to $m \le e$, 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 $B := T(b-2q\mathinner{.\,.} b]$.

We shall prove that this is a valid break. For a proof by contradiction, suppose that $p' := \operatorname{per}(B) \le q$. Then, p' is also period of $T(b-2q\mathinner{.\,.} b)$. Moreover, the invariant guarantees that positions in $(b-2q\mathinner{.\,.} b)$ are all compatible, so also p is a period of $T(b-2q\mathinner{.\,.} b)$. Since $p+p'-1 \le 2q-1$, the periodicity lemma [20] implies that also $\gcd(p, p')$ is a period of $X(b-2q\mathinner{.\,.} b)$. 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 $\operatorname{per}(B) > q$.

It remains to analyze the running time. Determining $\operatorname{per}(T[0\mathinner{.\,.} 2q))$ in Line 1 costs $\mathcal{O}(q)$ time using a classic algorithm [30]. The number of sampled positions is $|S| = \tilde{\mathcal{O}}(\frac{1}{k+1}|T|)$ w.h.p., 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 $\tilde{\mathcal{O}}(\frac{1}{k+1}|T| + q)$. ∎

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

Lemma III.7. *There exists an algorithm that, given strings X and Y, an integer $k\ge 0$, 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 $\forall_{j\in J}: \mathsf{HD}(X[i\,..\,i+\ell),Y[j\,..\,j+\ell))>k$. W.h.p. the algorithm is correct and its running time is $\tilde{\mathcal{O}}(\frac{\ell}{k+1}+|J|)$.*

Proof: A procedure $\mathsf{Oracle}(P,\ T,\ i,\ J,\ k,\ \ell)$ implementing Lemma III.7 is given as Algorithm 4.

Algorithm 4: $\mathsf{Oracle}(X,\ Y,\ i,\ J,\ k,\ \ell)$

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

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 $\mathsf{FindBreak}$ of Lemma III.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 $\mathsf{FindBreak}$: 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 $M\ne\emptyset$, and it may return NO whenever $M=\emptyset$.

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

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 $\mathsf{FindBreak}$ 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 12 are correct.

Next, note that the calls to $\mathsf{FindBreak}$ satisfy the requirements of Lemma III.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 $\mathsf{HD}(X[i\,..\,i{+}\ell),Y[\min C\,..\,\min C{+}\ell))\le k$ holds w.h.p. in case both calls return $\perp$. Denote $p=\mathrm{per}(X[i\,..\,i+2|J|])$, let $P=X[i\,..\,i+p)$ be the corresponding string period, and let P^∞ be the concatenation of infinitely many copies of P. The outcome $\perp$ of the first call to $\mathsf{FindBreak}$ certifies that $X[i\,..\,i+\ell)$ is w.h.p. at Hamming distance at most $\frac{k}{2}$ from a prefix of P^∞. Due to $X[i\,..\,i+2|J|)=Y[\max C\,..\,\max C{+}2|J|)$, the outcome $\perp$ of the second call to $\mathsf{FindBreak}$ certifies that also $Y[\max C\,..\,\min C+\ell)$ is w.h.p. at Hamming distance at most $\frac{k}{2}$ from a prefix of P^∞. Moreover, by Fact II.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 w.h.p. at Hamming distance at most $\frac{k}{2}$ from a prefix of P^∞. Now, the triangle inequality yields $\mathsf{HD}(X[i\,..\,i+\ell),Y[\min C\,..\,\min C+\ell))\le k$, as claimed.

It remains to justify the answers returned in Lines 15 and 16. Because the breaks B_X and B_Y, if defined, satisfy $\mathrm{per}(B_X)>|J|$ and $\mathrm{per}(B_Y)>|J|$, Lemma III.6 implies that their exact occurrences must be more than $|J|$ positions apart. Consequently, applying Line 9 or Line 11 leaves at most one position in C. Thus, the algorithm correctly returns NO if it detects a mismatch in Line 15 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 w.h.p. at least one of them is detected. Thus, returning YES in Line 16 is also correct.

Running time: Lines 2, 3, 9, and 11 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 [30]. The calls to $\mathsf{FindBreak}$ from Lemma III.6 cost $\tilde{\mathcal{O}}(\frac{\ell}{\lfloor k/2\rfloor+1}+|J|)$ time w.h.p. Finally, the number of sampled positions is $|S|=\tilde{\mathcal{O}}(\frac{\ell}{k+1})$ w.h.p., and this is also the total cost of Line 15. The total running time is $\tilde{\mathcal{O}}(\frac{\ell}{k+1}+|J|)$. ■

Finally, we derive Proposition III.4 via a simple reduction to Lemma III.7.

Proposition III.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 $\ell := \max_{j\in J} \mathrm{LCE}^{X,Y}_{\leq k}(i,j)$. W.h.p. the algorithm is correct and its running time is $\tilde{\mathcal{O}}(\frac{\ell}{k+1}+|J|)$.*

Proof: Observe that Lemma III.7 provides an oracle that returns YES if $\max_{j\in J} \mathrm{LCE}^{X,Y}_{0}(i,j) \geq \ell$ and NO if $\max_{j\in J} \mathrm{LCE}^{X,Y}_{k}(i,j) < \ell$. However, before calling $\mathsf{Oracle}(P,\ T,\ i,\ J,\ k,\ \ell)$, we need to make sure that $\ell > 0$, $i \in [0\,..\,|X|-\ell]$, and $\emptyset \neq J \subseteq [0\,..\,|Y|-\ell]$. Thus, basic corner cases have to be handled separately: The algorithm returns YES if $\ell \leq 0$; otherwise, it sets $J := J \cap [0\,..\,|Y|-\ell]$, returns NO if $i \notin [0\,..\,|X|-\ell]$ or $J = \emptyset$, and makes a call $\mathsf{Oracle}(P,\ T,\ i,\ J,\ k,\ \ell)$ in the remaining case.

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

IV. Improved Approximation Ratio

Goldenberg et al. [21] generalized their algorithm in order to solve the k vs αk gap edit distance problem in $\tilde{\mathcal{O}}(\frac{n}{\alpha}+k^3)$ time for any $\alpha \geq 1$. This transformation is quite simple, because Algorithm 1 (the Landau–Vishkin algorithm) with LCE_0 queries replaced by $\mathrm{LCE}_{\leq \alpha-1}$ queries returns YES if $\mathsf{ED}(X,Y) \leq k$ and NO if $\mathsf{ED}(X,Y) > k+(\alpha-1)(k+1)$.

However, if we replace $\mathrm{LCE}_{\leq k}$ queries with $\mathrm{LCE}_{\leq \alpha-1}$ queries in Algorithm 2, then we are guaranteed to get a NO answer only if $\mathsf{ED}(X,Y) > 2k(k+2)+(\alpha-1)(k+1)$. As a result, with an appropriate adaptation of Proposition III.4, Algorithm 2 yields an $\tilde{\mathcal{O}}(\frac{n}{\alpha}+k^2)$-time solution to the k vs αk 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 $\mathrm{LCE}^{X,Y}(x,y)$. On the other hand, no such shift changes are performed in Algorithm 1, but this results in $\mathrm{LCE}^{X,Y}(x,y)$ queries asked for up to $\Theta(k^2)$ distinct positions x, which is the reason behind the $\tilde{\mathcal{O}}(k^3)$ term in the running time $\tilde{\mathcal{O}}(\frac{n}{\alpha}+k^3)$ of [21].

Nevertheless, since each $\mathrm{LCE}^{X,Y}_{\leq \alpha-1}(x,y)$ 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.

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

If we use Proposition III.4 to implement $\mathrm{LCE}_{\leq \alpha-1}$ queries in Algorithm 5, then the cost of computing $d_{i,j}$ is $\mathcal{O}(\alpha + \frac{1}{\alpha}(d_{i,j}-d'_{i,j}))$, w.h.p. This query is performed only

Algorithm 5: Improved algorithm

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

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 w.h.p., which is not comparable to the running time $\tilde{\mathcal{O}}(\frac{1}{\alpha}|X|+k^3)$ of [21].

However, we can obtain a faster algorithm using the data structure specified below and described in the full version of the paper. In particular, this result dominates Proposition III.4 and, if we set $\Delta = [-k\,..\,k]$, then $\mathrm{LCE}^{X,Y}_{\leq k}(x,y)$ 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 III.

Proposition IV.2 (♠). *There exists a data structure that, initialized with strings X and Y, an integer $k \geq 0$, and an integer range Δ, answers the following queries: given an integer x, return $\mathrm{LCE}^{X,Y}_{\leq k}(x,x+\delta)$ for all $\delta \in \Delta$. The initialization costs $\tilde{\mathcal{O}}(\frac{1}{k+1}|X|)$ time w.h.p., and the queries cost $\tilde{\mathcal{O}}(|\Delta|)$ time w.h.p.*

Since the $\mathrm{LCE}^{X,Y}_{\leq \alpha-1}(x,y)$ 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 IV.2 yields an $\tilde{\mathcal{O}}(\frac{1}{\alpha}|X|+\frac{k^3}{\alpha})$-time implementation of Algorithm 5, which is already better the running time of [21]. However, the running time of a more subtle solution described below subsumes both $\tilde{\mathcal{O}}(\frac{1}{\alpha}|X|+\frac{k^3}{\alpha})$ and $\tilde{\mathcal{O}}(\frac{k}{\alpha^2}|X|+k^2)$ (obtained using Proposition III.4).

Theorem IV.3. *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 $\mathsf{ED}(X,Y) \leq k$, and NO if $\mathsf{ED}(X,Y) > k+3(k+1)(\alpha-1)$. W.h.p. the algorithm is correct and its running time is $\tilde{\mathcal{O}}(\frac{1}{\alpha}|X|+k^2+\frac{k}{\alpha}\sqrt{|X|k})$.*

Proof: We define an integer parameter $b \in [1\,..\,\lceil\frac{k}{\alpha}\rceil]$ (to be fixed later) and initialize $\mathcal{O}(\frac{k}{\alpha b})$ instances of the data structure of Proposition IV.2 for answering $\mathrm{LCE}^{X,Y}_{\le\alpha-1}$ queries. The instances are indexed with $j' \in [\lfloor\frac{-k}{\alpha b}\rfloor \mathinner{.\,.} \lfloor\frac{k}{\alpha b}\rfloor]$, and the j'th instance has interval $\Delta_{j'} = [j'\alpha b \mathinner{.\,.} (j'+1)\alpha b)$. This way, the value $d_{i,j}$ can be retrieved from the values $\mathrm{LCE}_{\le r-1}(d'_{i,j}, d'_{i,j}+\delta)$ for $\delta \in \Delta_{\lfloor \frac{j}{b} \rfloor}$, that is, from a single query to an instance of the data structure of Proposition IV.2.

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

V. PTAS for Aperiodic Strings

In this section, we design an algorithm distinguishing between $\mathsf{ED}(X,Y) \le k$ and $\mathsf{ED}(X,Y) > (1+\varepsilon)k$, assuming that X does not have a length-ℓ substring with period at most $2k$. The high-level approach of our solution is based on the existing algorithms for Ulam distance [3], [31]. 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 $\mathsf{ED}(X,Y) = \sum_{i=0}^{m} \mathsf{ED}(X_i,Y_i)$ if $\mathsf{ED}(X,Y) \le k$. 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 V.1 (♠). *There exists an algorithm that, given strings X and Y, integers k and ℓ such that $\mathrm{per}(X[i \mathinner{.\,.} i+\ell)) > 2k$ for each $i \in [0 \mathinner{.\,.} |X|-\ell]$, 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 $|X_i| \le \lceil \delta^{-1}(k+1)\ell \rceil$ for each $i \in [0 \mathinner{.\,.} m]$ and, if $\mathsf{ED}(X,Y) \le k$, then $\Pr[\mathsf{ED}(X,Y) = \sum_{i=0}^{m}\mathsf{ED}(X_i,Y_i)] \ge 1-\delta$. The running time of the algorithm is $\mathcal{O}(\frac{\delta}{k+1}|X|)$.*

Next, we present a subroutine that will be applied to individual phrases of the decompositions of Lemma V.1. Given that the phrases are short, we can afford using the classic Landau–Vishkin algorithm [27] whenever we find out that the corresponding phrases do not match exactly.

Lemma V.2 (♠). *There exists an algorithm that, given strings X and Y, and an integer $k \ge 0$, computes $\mathsf{ED}(X,Y) > 0$ exactly, taking $\tilde{\mathcal{O}}(|X|+\mathsf{ED}(X,Y)^2)$ time, or certifies that $\mathsf{ED}(X,Y) \le k$ w.h.p., taking $\tilde{\mathcal{O}}(1+\frac{1}{k+1}|X|)$ time w.h.p.*

The next step is to design a procedure which distinguishes between $\sum_{i=0}^{m}\mathsf{ED}(X_i,Y_i) \le k$ and $\sum_{i=0}^{m}\mathsf{ED}(X_i,Y_i) \ge (1+\varepsilon)k$. Our approach relies on the Chernoff bound: we apply Lemma V.2 to determine $\mathsf{ED}(X_i,Y_i)$ for a small sample of indices i, and then we use these values to estimate the sum $\sum_{i=0}^{m}\mathsf{ED}(X_i,Y_i)$.

Lemma V.3 (♠). *There exists an algorithm that, given strings $X_0,\ldots,X_m,Y_0,\ldots,Y_m$, and a real parameter $0 < \varepsilon < 1$, returns YES if $\sum_{i=0}^{m}\mathsf{ED}(X_i,Y_i) \le k$, and NO if $\sum_{i=0}^{m}\mathsf{ED}(X_i,Y_i) \ge (1+\varepsilon)k$. The algorithm succeeds w.h.p., and its running time is $\tilde{\mathcal{O}}(qk+k^2+\frac{n}{\varepsilon^2(k+1)})$, where $q = \max_{i=0}^{m}|X_i|$ and $n = \sum_{i=0}^{m}(|X_i|+|Y_i|)$.*

Finally, we obtain the main result of this section by combining Lemma V.1 with Lemma V.3.

Theorem V.4 (♠). *There exists an algorithm that, given strings X and Y, integers k and ℓ such that $\mathrm{per}(X[i \mathinner{.\,.} i+\ell)) > 2k$ for each $i \in [0 \mathinner{.\,.} |X|-\ell]$, and a real parameter $0 < \varepsilon < 1$, returns YES if $\mathsf{ED}(X,Y) \le k$, and NO if $\mathsf{ED}(X,Y) \ge (1+\varepsilon)k$. W.h.p. the algorithm is correct and its running time is $\tilde{\mathcal{O}}(\frac{1}{\varepsilon^2(k+1)}|X|+k^2\ell)$.*

VI. Random Walk over Samples

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

Algorithm 6: SampledRandomWalk(X, Y, k, p)

```
1  $x := 0$, $y := 0$, $c := 0$;          ▷ Initialization
2  while $x < |X|$ and $y < |Y|$ do
3      Let $s \sim \mathrm{Bin}(1, \frac{2\ln n}{p})$;      ▷ Biased coin
4      if $s = 1$ and $X[x] \neq Y[y]$ then
5          Let $r \sim \mathrm{Bin}(1, \frac{1}{2})$;      ▷ 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 $c + \max(|X| - x, |Y| - y) \le 1296k^2$;
```

Given strings $X, Y \in \Sigma^n$, and integer parameters $k \ge 0$, $p \ge 2\ln n$, Algorithm 6 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 VI.1. *Given strings $X, Y \in \Sigma^{\le n}$ and integers $k \ge 0$ and $2\ln n \le p \le n$, Algorithm 6 returns YES with probability at least $\frac{2}{3}$ if $\mathsf{ED}(X,Y) \le k$, and NO with probability least $1-\frac{1}{n}$ if $\mathsf{ED}(X,Y) \ge (1296k^2+1)p$. Moreover, Algorithm 6 can be implemented in $\tilde{\mathcal{O}}(\frac{n}{p})$ time.*

YES Case: Recall that the *indel distance* $\mathsf{IDD}(\cdot,\cdot)$ is defined so that $\mathsf{IDD}(X,Y)$ 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 $\mathsf{ED}(X,Y) \le \mathsf{IDD}(X,Y) \le 2\mathsf{ED}(X,Y)$.

We analyze how $D := \mathsf{IDD}(X[x\mathinner{.\,.}|X|), Y[y\mathinner{.\,.}|Y|)$ changes throughout the execution of Algorithm 6. Let D_0 be the initial value of D and let D_i be the value of D after the ith iteration of the algorithm, where $i \in [1\mathinner{.\,.}t]$ 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_i:

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

(a) D_i is a non-negative integer,
(b) If i is not a mismatch iteration , then $D_i \le D_{i-1}$.
(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) $D_i = 0$ if $D_{i-1} = 0$.

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

As for (b), let us consider an optimal indel-distance alignment resulting in $D_{i-1} = \mathsf{IDD}(X[x\mathinner{.\,.}|X|), Y[y\mathinner{.\,.}|Y|))$ at the beginning of iteration i, and transform it to an alignment between $X[x+1\mathinner{.\,.}|X|)$ and $Y[y+1\mathinner{.\,.}|Y|)$ by discarding $X[x]$ and $Y[y]$ and deleting characters matched with $X[x]$ or $Y[y]$. Two characters are never deleted, and a character is deleted only if the original alignment deletes $X[x]$ or $Y[y]$. Hence, the alignment cost does not increase and $D_i \le D_{i-1}$.

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

As for (d), we note that once $X[x\mathinner{.\,.}|X|) = Y[y\mathinner{.\,.}|Y|)$, no subsequent iteration will be a mismatch iteration. ∎

Now, consider a 1-dimensional random walk $(W_0)_{i\ge 0}$ that starts with $W_0 = 2k$ and moves 1 unit up or down at every step with equal probability $\frac{1}{2}$. Let us connect this random walk with the execution of Algorithm 6. Let $i_1, \dots, i_c$ be the mismatch iterations. For each $j \in [1\mathinner{.\,.}c]$ 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 6 uses fresh randomness, the steps of the random walk remain unbiased and independent from each other.

Now, Lemma VI.2 implies that $D_0 \le W_0$, that $D_{i_j} \le W_j$ holds for $j \in [1\mathinner{.\,.}c]$, and that $D_t \le W_c$ holds upon termination of Algorithm 6. In particular, the hitting time $T = \min\{j : W_j = 0\}$ satisfies $T \ge c + D_t$. However, as proved in [29, Theorem 2.17], $\Pr[T \le N] \ge 1 - \frac{12k}{\sqrt{N}}$. Thus, $\Pr[c + D_t \le 1296k^2] \ge \frac{2}{3}$. Since $D_t = \max(|X| - x, |Y| - y)$ after iteration t, this completes the proof of the YES case.

NO Case: If $s = 0$ and $X[x] \ne Y[y]$ holds at the beginning of some iteration of Algorithm 6, 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 VI.3. *Consider the values x, y at the beginning of iteration i of Algorithm 6. Conditioned on any random choices made prior to iteration i, the probability that Algorithm 6 misses the leftmost p mismatches between $X[x\mathinner{.\,.}|X|)$ and $Y[y\mathinner{.\,.}|Y|)$ is at most n^{-2}.*

Proof: Prior to detecting any mismatch between $X[x\mathinner{.\,.}|X|)$ and $Y[y\mathinner{.\,.}|Y|)$, 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 $(1 - \frac{2\ln n}{p})^p \le n^{-2}$. ∎

Now, observe that an execution of Algorithm 6 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 $X[x\mathinner{.\,.}|X|)$ or $Y[y\mathinner{.\,.}|Y|)$ after the last iteration i_t are also deleted. The total number of deletions is thus $c + \max(|X| - x, |Y| - y)$, and every substitution corresponds to a missed mismatch. By Lemma VI.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 $\mathsf{ED}(X,Y) < (c + \max(|X| - x, |Y| - y) + 1)p$. In this case, the algorithm returns NO if $\mathsf{ED}(X,Y) \ge (1296k^2 + 1)p$.

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 $\mathrm{Geo}(0, \frac{2\ln n}{p})$) the number δ of subsequent iterations with $s = 0$, increase both x and y by δ, and then execute a single iteration with $s = 1$. The number of total number of iterations is at most $|X| + |Y| \le 2n$, and the number of iterations with $s = 1$ is $\mathcal{O}(\frac{n\ln n}{p})$ w.h.p.

This completes the proof of Theorem VI.1. ∎

VII. Sublinear-Time Embedding of Edit Distance to Hamming Distance

Given any two strings X and Y, a randomized embedding of edit distance to Hamming distance is given by a function f such that $\mathsf{HD}(f(X,R), f(Y,R)) \approx \mathsf{ED}(X,Y)$ holds with good probability over the randomness R. The distortion of an embedding is the product of its expansion and contraction, which are upper bounds on $\frac{\mathsf{HD}(f(X,R),f(Y,R))}{ED(X,Y)}$ and $\frac{\mathsf{ED}(X,Y)}{\mathsf{HD}(f(X,R),f(Y,R))}$, respectively, valid for all $X, Y \in \Sigma^n$.

Chakraborty, Goldenberg, and Koucký [17] gave one such randomized embedding with quadratic distortion:

Theorem ([17, Theorem 1.1]). *For every integer $n \geq 1$, there is an integer $\ell = \mathcal{O}(\log n)$ and a function $f : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^{3n}$ such that for every $X, Y \in \{0,1\}^n$*

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

with probability at least $\frac{2}{3}$ 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 Σ 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 $\mathcal{O}(n|\Sigma|)$ random bits, which can be reduced to $\mathcal{O}(\log n)$ using Nisan's pseudorandom number generator [32].

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 $h_1, h_2, \dots, h_{|S|} : \{0,1\} \to \{0,1\}$. The shared randomness R consists of S and $h_1, h_2, \dots, h_{|S|}$. We prove the following theorem.

Theorem VII.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 $f : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^{\tilde{\mathcal{O}}(\frac{n}{p})}$ such that for every $X, Y \in \{0,1\}^n$*

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

with probability at least $\frac{2}{3}$ 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 7 provides the pseudocode of the embedding.

Algorithm 7: $\mathsf{SublinearEmbedding}(X, \langle S, h_1, \dots, h_{|S|}\rangle)$

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

Interpretation through Algorithm 6: Consider running Algorithm 6 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 6 so that its execution uses R as the source of randomness: we use the event $\{i \in S\}$ for the ith toss of the biased coin s and the value $h_j(X'[x])$ for the jth 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 w.h.p. after iteration $3n$, and then Algorithm 6 cannot detect any mismatch. Hence, Algorithm 7 w.h.p. simulates the treatment of X' and Y' by Algorithm 6. Moreover, c in Algorithm 6 corresponds to the number of iterations with $s = 1$ and $X'[x] \neq Y'[y]$, and this number of iterations is precisely $\mathsf{HD}(f(X,R), f(Y,R))$.

YES Case: Algorithm 6 with $k = \mathsf{ED}(X,Y) = \mathsf{ED}(X',Y')$ returns YES with probability $\geq \frac{2}{3}$. Thus, $\mathsf{HD}(f(X,R), f(Y,R)) = c \leq 1296k^2 = \mathcal{O}(\mathsf{ED}(X,Y)^2)$.

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

Efficient implementation: To complete the proof, we note that Algorithm 7 can be implemented in $\mathcal{O}(|S|)$ time by batching the iterations i with $i \notin S$ (it suffices to increase i and x). Moreover, $|S| = \mathcal{O}(\frac{n\ln n}{p})$ with high probability. ■

Acknowledgments

T. Kociumaka was 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). B. Saha was supported by NSF grants no. 1652303 (NSF CAREER), 1934846 (NSF HDR TRIPODS), and 1909046, and by an Alfred P. Sloan fellowship award.

References

[1] A. Andoni, M. Deza, A. Gupta, P. Indyk, and S. Raskhodnikova, "Lower bounds for embedding edit distance into normed spaces," in *14th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2003, pp. 523–526.

[2] A. Andoni, R. Krauthgamer, and K. Onak, "Polylogarithmic approximation for edit distance and the asymmetric query complexity," in *51st Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2010, pp. 377–386.

[3] A. Andoni and H. L. Nguyen, "Near-optimal sublinear time algorithms for ulam distance," in *21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2010, pp. 76–86.

[4] A. Andoni and N. S. Nosatzki, "Edit distance in near-linear time: it's a constant factor," in *61st Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2020.

[5] A. Andoni and K. Onak, "Approximating edit distance in near-linear time," *SIAM J. Comput.*, vol. 41, no. 6, pp. 1635–1648, 2012.

[6] A. Backurs and P. Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)," *SIAM J. Comput.*, vol. 47, no. 3, pp. 1087–1097, 2018.

[7] Z. Bar-Yossef, T. S. Jayram, R. Krauthgamer, and R. Kumar, "Approximating edit distance efficiently," in *45th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2004, pp. 550–559.

[8] T. Batu, F. Ergün, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami, "A sublinear algorithm for weakly approximating edit distance," in *35th Annual ACM Symposium on Theory of Computing, STOC*, 2003, pp. 316–324.

[9] T. Batu, F. Ergün, and S. C. Sahinalp, "Oblivious string embeddings and edit distance approximations," in *17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2006, pp. 792–801.

[10] D. Belazzougui and Q. Zhang, "Edit distance: Sketching, streaming, and document exchange," in *57th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2016, pp. 51–60.

[11] M. Boroujeni, S. Ehsani, M. Ghodsi, M. T. Hajiaghayi, and S. Seddighin, "Approximating edit distance in truly subquadratic time: Quantum and mapreduce," in *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2018, pp. 1170–1189.

[12] J. Brakensiek, M. Charikar, and A. Rubinstein, "A simple sublinear algorithm for gap edit distance," 2020.

[13] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *IEEE Trans. Inf. Theory*, vol. 64, no. 5, pp. 3403–3410, 2018.

[14] J. Brakensiek and A. Rubinstein, "Constant-factor approximation of near-linear edit distance in near-linear time," in *52nd Annual ACM Symposium on Theory of Computing, STOC*, 2020, pp. 685–698.

[15] D. Breslauer and Z. Galil, "Finding all periods and initial palindromes of a string in parallel," *Algorithmica*, vol. 14, no. 4, pp. 355–366, 1995.

[16] D. Chakraborty, D. Das, E. Goldenberg, M. Koucký, and M. E. Saks, "Approximating edit distance within constant factor in truly sub-quadratic time," in *59th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2018, pp. 979–990.

[17] D. Chakraborty, E. Goldenberg, and M. Koucký, "Streaming algorithms for embedding and computing edit distance in the low distance regime," in *48th Annual ACM Symposium on Theory of Computing, STOC*, 2016, pp. 712–725.

[18] M. Charikar, O. Geri, M. P. Kim, and W. Kuszmaul, "On estimating edit distance: Alignment, dimension reduction, and embeddings," in *45th International Colloquium on Automata, Languages, and Programming, ICALP*, ser. LIPIcs, vol. 107, 2018, pp. 34:1–34:14.

[19] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, "Alignment of whole genomes," *Nucleic Acids Research*, vol. 27, no. 11, pp. 2369–2376, 1999.

[20] N. J. Fine and H. S. Wilf, "Uniqueness theorems for periodic functions," *Proceedings of the American Mathematical Society*, vol. 16, no. 1, pp. 109–114, 1965.

[21] E. Goldenberg, R. Krauthgamer, and B. Saha, "Sublinear algorithms for gap edit distance," in *60th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2019, pp. 1101–1120.

[22] E. Goldenberg, A. Rubinstein, and B. Saha, "Does preprocessing help in fast sequence comparisons?" pp. 657–670, 2020.

[23] B. Haeupler, "Optimal document exchange and new codes for insertions and deletions," in *60th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2019, pp. 334–347.

[24] R. Impagliazzo and R. Paturi, "On the complexity of k-SAT," *Journal of Computer and System Sciences*, vol. 62, no. 2, pp. 367–375, 2001.

[25] M. Koucký and M. E. Saks, "Constant factor approximations to edit distance on far input pairs in nearly linear time," in *52nd Annual ACM Symposium on Theory of Computing, STOC*, 2020, pp. 699–712.

[26] G. M. Landau, E. W. Myers, and J. P. Schmidt, "Incremental string comparison," *SIAM J. Comput.*, vol. 27, no. 2, pp. 557–582, 1998.

[27] G. M. Landau and U. Vishkin, "Fast string matching with k differences," *J. Comput. Syst. Sci.*, vol. 37, no. 1, pp. 63–78, 1988.

[28] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[29] D. Levin and Y. Peres, *Markov Chains and Mixing Times*. American Mathematical Society, 2017.

[30] J. H. Morris, Jr. and V. R. Pratt, "A linear pattern-matching algorithm," Department of Computer Science, University of California, Berkeley, Tech. Rep. 40, 1970.

[31] T. Naumovitz, M. E. Saks, and C. Seshadhri, "Accurate and nearly optimal sublinear approximations to ulam distance," in *28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2017, pp. 2012–2031.

[32] N. Nisan, "Pseudorandom generators for space-bounded computation," *Combinatorica*, vol. 12, no. 4, pp. 449–461, 1992.

[33] R. Ostrovsky and Y. Rabani, "Low distortion embeddings for edit distance," *J. ACM*, vol. 54, no. 5, p. 23, 2007.

[34] B. Saha, "The Dyck language edit distance problem in near-linear time," in *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2014, pp. 611–620.

[35] H. Zhang and Q. Zhang, "Embedjoin: Efficient edit similarity joins via embeddings," in *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, 2017, pp. 585–594.