An Algorithmic Bridge Between Hamming and Levenshtein Distances

Elazar Goldenberg
Academic College of Tel Aviv-Yafo, Israel

Tomasz Kociumaka
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel

Barna Saha
University of California, San Diego, CA, USA

Abstract

The edit distance between strings classically assigns unit cost to every character insertion, deletion, and substitution, whereas the Hamming distance only allows substitutions. In many real-life scenarios, insertions and deletions (abbreviated *indels*) appear frequently but significantly less so than substitutions. To model this, we consider substitutions being cheaper than indels, with cost $\frac{1}{a}$ for a parameter $a \geq 1$. This basic variant, denoted $\texttt{ED}_a$, bridges classical edit distance ($a = 1$) with Hamming distance ($a \to \infty$), leading to interesting algorithmic challenges: Does the time complexity of computing $\texttt{ED}_a$ interpolate between that of Hamming distance (linear time) and edit distance (quadratic time)? What about approximating $\texttt{ED}_a$?

We first present a simple deterministic exact algorithm for $\texttt{ED}_a$ and further prove that it is near-optimal assuming the Orthogonal Vectors Conjecture. Our main result is a randomized algorithm computing a $(1+\epsilon)$-approximation of $\texttt{ED}_a(X, Y)$, given strings X, Y of total length n and a bound $k \geq \texttt{ED}_a(X, Y)$. For simplicity, let us focus on $k \geq 1$ and a constant $\epsilon > 0$; then, our algorithm takes $\tilde{O}(\frac{n}{a} + ak^3)$ time. Unless $a = \tilde{O}(1)$, in which case $\texttt{ED}_a$ resembles the standard edit distance, and for the most interesting regime of small enough k, this running time is sublinear in n.

We also consider a very natural version that asks to find a (k_I, k_S)-*alignment*, i.e., an alignment with at most k_I indels and k_S substitutions. In this setting, we give an exact algorithm and, more importantly, an $\tilde{O}(\frac{nk_I}{k_S} + k_S k_I^3)$-time $(1, 1+\epsilon)$-bicriteria approximation algorithm. The latter solution is based on the techniques we develop for $\texttt{ED}_a$ for $a = \Theta(\frac{k_S}{k_I})$, and its running time is again sublinear in n whenever $k_I \ll k_S$ and the overall distance is small enough.

These bounds are in stark contrast to unit-cost edit distance, where state-of-the-art algorithms are far from achieving $(1+\epsilon)$-approximation in sublinear time, even for a favorable choice of k.

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

Keywords and phrases edit distance, Hamming distance, Longest Common Extension queries

Digital Object Identifier 10.4230/LIPIcs.ITCS.2023.58

Related Version *Full Version*: https://arxiv.org/abs/2211.12496

Funding *Tomasz Kociumaka*: Work mostly carried out while at the University of California, Berkeley, partly supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.
Robert Krauthgamer: Work partially supported by ONR Award N00014-18-1-2364, the Israel Science Foundation grant #1086/18, and a Minerva Foundation grant, and by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center.
Barna Saha: Partly supported by NSF CCF grants 1652303 and 1909046, and an HDR TRIPODS Phase II grant 2217058.

1 Introduction

Edit distance and Hamming distance are the two most fundamental measures of sequence similarity. The (unit-cost) edit distance, also known as the Levenshtein distance, of two strings X and Y is the minimum number of character insertions, deletions, and substitutions required to convert one string to the other, whereas the Hamming distance allows only substitutions (requiring $|X| = |Y|$). From an algorithmic perspective, these two measures exhibit significantly different time complexity in terms of the input size $n = |X| + |Y|$. The Hamming distance can be computed exactly in linear time $O(n)$, and it admits a randomized ϵn-additive approximation in time $O(\epsilon^{-1})$, which implies a $(1+\epsilon)$-approximation in sublinear time when the distance is not too small. In contrast, assuming the Orthogonal Vectors Conjecture [32], no $O(n^{2-\Omega(1)})$-time algorithm can compute the edit distance exactly [5]. Recent developments in designing fast approximation algorithms [2, 4, 6, 9, 13, 19, 25] culminated in an $O(1)$-approximation in near-linear time [3], but the existence of a truly subquadratic-time 3-approximation still remains open. Furthermore, despite many efforts [7, 10, 11, 17, 18, 24], the best approximation ratio achievable in sublinear-time ranges from polylogarithmic to polynomial in n (depending on how large the true edit distance is).

The contrasting complexity landscape between edit and Hamming distance clearly indicates that substitutions are easier to handle than insertions and deletions (abbreviated *indels*). Many real-world applications, for instance in computational biology, compare sequences based on edit distance, but its value is often dominated by the number of substitutions, as indicated by recent studies [15, 22, 29, 30, 33]. Is it possible to design significantly faster algorithms for these scenarios with much more substitutions than indels? Such a bridge between Hamming and edit distances could also provide an explanation for why many heuristics for string comparison are fast on real-life examples. (For another applied perspective, see [28].)

This motivates our study of a basic variant of the edit distance, denoted $\mathtt{ED}_a$, where a substitution is significantly cheaper than an insertion or a deletion, and its cost is $\frac{1}{a}$ for a parameter $a \geq 1$. This simple variant bridges unit-cost edit distance ($a = 1$) and Hamming distance ($a \geq |X| = |Y|$), raising basic algorithmic questions: *Does the time complexity of computing* $\mathtt{ED}_a$ *interpolate between Hamming distance (linear time) and edit distance (quadratic time)? How efficiently can one compute a* $(1+\epsilon)$*-approximation of* $\mathtt{ED}_a$*? What speed-up is feasible as the parameter* a *grows?* We present the first series of results to answer these questions. As an illustrative example, if an intended application expects about k indels and k^2 substitutions between the two strings, then one should set $a = k$ and use the $(1+\epsilon)$-approximation algorithm that we devise, which runs in sublinear time $\tilde{O}(\frac{n}{k} + \mathrm{poly}(k))$ for this parameter setting. In contrast, the state-of-the-art sublinear-time algorithm for unit-cost edit distance [10] only provides an $O(\mathrm{polylog}\, k)$-factor approximation of the total number of edits, so it will likely produce an alignment with $\omega(k^2)$ indels, which blatantly violates the structure of alignments arising in the considered application.

One can achieve a stricter control on the number of indels and substitutions by imposing two independent bounds k_I and k_S on these quantities. Our techniques can be easily adapted to the underlying problem, which we call the (k_I, k_S)-*alignment* problem; in particular, we obtain a sublinear-time $(1, 1+\epsilon)$-bicriteria approximation of the combined cost (k_I, k_S). For $(k_I, k_S) = (k, k^2)$ as in the example above, the running time of our algorithm is still $\tilde{O}(\frac{n}{k} + \mathrm{poly}(k))$, albeit the $\mathrm{poly}(k)$ term is slightly larger.

The weighted edit distance problem and the complementary highest-score alignment problem are widely used in applications and discussed in multiple textbooks; see e.g. [1, 20, 23]. The simplest version involves two costs (for indels and substitutions, respectively) and, up

to scaling, is equivalent to $\texttt{ED}_a$. Theoretical analysis usually focuses on the fundamental unit-cost setting, but many results extend to the setting of constant costs, e.g., they admit the same conditional lower bound [12] (and these problems are clearly equivalent from the perspective of logarithmic approximation). We initiate a deeper investigation of how the costs of edit operations affect the complexity of computing the edit distance (exactly and approximately); going beyond the regime of $a = O(1)$ reveals the gamut between Levenshtein distance ($a = 1$) and Hamming distance ($a = n$, $k < 1$), providing a holistic perspective on these two fundamental metrics.

1.1 Our Contribution

We provide multiple results for both the $\texttt{ED}_a$ and (k_I, k_S)-alignment problems. Even though these are basic problems that seamlessly bridge the gap between Hamming and edit distances, surprisingly little is known about them. Throughout, we assume that the algorithms are given $O(1)$-time random access to characters of X, Y and know the lengths $|X|, |Y|$.

Our main results are approximation algorithms, but let us start with exact algorithms for the two problems. We first observe that a simple adaptation of the approach of Landau and Vishkin [26, 27] computes $\texttt{ED}_a(X, Y)$ in time $O(n + k \min(n, ka))$, where $k \geq \texttt{ED}_a(X, Y)$. In the full version, we then use similar techniques to derive an algorithm for the (k_I, k_S)-alignment problem. For convenience, we state these results for decision-only problems; it is straightforward to accompany every YES answer with a corresponding alignment.

▶ **Proposition 1.1.** *Given two strings $X, Y \in \Sigma^*$ of total length n, a cost parameter $a \in \mathbb{Z}_+$, and a threshold $k \in \mathbb{R}_+$, one can compute $\texttt{ED}_a(X, Y)$ or report that $\texttt{ED}_a(X, Y) > k$ in deterministic time $O(n + k \min(n, ka))$.*

▶ **Proposition 1.2.** *Given two strings $X, Y \in \Sigma^*$ of total length n and two integer thresholds $k_S, k_I > 0$, one can decide whether there is a (k_I, k_S)-alignment or not (report YES or NO) in deterministic time $O(n + k_S k_I^2)$.*

The existing $n^{2-\Omega(1)}$ lower bounds [5, 12] do not shed light as to whether these running times are optimal. Interestingly, we strengthen the result of [12] significantly to show that the bound of Proposition 1.1 is indeed tight for the entire range of parameters $a, k \geq 1$, assuming the Orthogonal Vectors Conjecture [32]. Proving such a lower bound for the bicriteria version remains open.

Approximation Algorithm for $\texttt{ED}_a$. Our main results are $(1 + \epsilon)$-approximation algorithms for the $\texttt{ED}_a$ and (k_I, k_S)-alignment problems. Let us first formally state the gap versions (also known as the promise versions) of these problems.

▶ **Problem 1.3** (Approximate Bounded $\texttt{ED}_a$). *Given two strings $X, Y \in \Sigma^*$ of total length n, a cost parameter $a \in \mathbb{Z}_+$, a threshold $k \in \mathbb{R}_+$, and an accuracy parameter $\epsilon \in (0, 1)$, report YES if $\texttt{ED}_a(X, Y) \leq k$, NO if $\texttt{ED}_a(X, Y) > (1 + \epsilon)k$, and an arbitrary answer otherwise.*

▶ **Problem 1.4** (Bicriteria Approximation). *Given two strings $X, Y \in \Sigma^*$ of total length n, two thresholds $k_I, k_S > 0$, and two approximation factors $\alpha, \beta \geq 1$, return YES if there is a (k_I, k_S)-alignment, NO if there is no $(\alpha k_I, \beta k_S)$-alignment, and an arbitrary answer otherwise.*

Our aim is to solve the above problems using algorithms whose running time is truly sublinear for suitable parameter values. This is in stark contrast to unit-cost edit distance, where state-of-the-art algorithms are far from achieving $(1+\epsilon)$-approximation in sublinear time, even for a favorable choice of parameters.[1]

▶ **Theorem 1.5.** *One can solve Problem 1.3 correctly with high probability within time:*
1. $\tilde{O}(\frac{n}{\epsilon^2 ak})$ *if* $k < 1$;
2. $\tilde{O}(\frac{n}{\epsilon^3 a} + ak^3)$ *if* $1 \leq k < \frac{n}{\epsilon a}$; *and*
3. $O(1)$ *if* $k \geq \frac{n}{\epsilon a}$.

Cases 1 and 3 are boundary cases; in the former, there are only substitutions (Hamming distance), whereas, in the latter, $|X|$ and $|Y|$ must differ significantly when $\mathtt{ED}_a(X,Y) > k$, which results in a trivial $(1+\epsilon)$-factor approximation. Our main technical contribution is Case 2, where we achieve sublinear running time for large enough a and small enough k.[2] In the aforementioned applications [15, 22, 29, 30, 33], the indels are few in number and must be estimated with high accuracy; in this case, a should be set proportionally to the substitution-to-indel ratio, so that our $(1+\epsilon)$-approximation computes, in sublinear time, a highly accurate estimate of both the indels and the substitutions.

It is straightforward to solve Problem 1.4 with $\alpha = \beta = 2+\epsilon$, by simply applying Theorem 1.5 with $a = \frac{k_S}{k_I}$ and threshold $k = 2k_I$. However, as shown in the full version, the techniques we developed for Theorem 1.5 allow for a much stronger guarantee of $\alpha = 1$ and $\beta = 1+\epsilon$.

▶ **Theorem 1.6.** *One can solve Problem 1.4 with* $\alpha = 1$ *and* $\beta = 1+\epsilon$ *(for any given parameter* $\epsilon \in (0,1)$*) correctly with high probability in* $\tilde{O}(\frac{nk_I}{\epsilon^3 k_S} + k_S k_I^3)$ *time.*

Again, the running time is sublinear whenever $k_I \ll k_S$ and the overall distance is small.

1.2 Notation

A *string* $X \in \Sigma^*$ is a finite sequence of characters from an *alphabet* Σ. The length of X is denoted by $|X|$ and, for $i \in [0\,..\,|X|)$,[3] the ith character of X is denoted by $X[i]$. A string Y is a *substring* of X if $Y = X[i]X[i+1]\cdots X[j-1]$ for some $0 \leq i \leq j \leq |X|$; this *occurrence* of Y is denoted by $X[i\,..\,j)$ or $X[i\,..\,j-1]$ and called a *fragment* of X. According to this convention, the empty string has occurrences $X[i\,..\,i)$ for $i \in [0\,..\,|X|]$. We use $\mathtt{HD}$ to denote the Hamming distance between two strings and $\mathtt{ED}$ to denote the standard edit distance.

A key notion in our work is that of Longest Common Extension (LCE) queries, defined as follows for indices $x \in [0\,..\,|X|]$ and $y \in [0\,..\,|Y|]$ in strings $X, Y \in \Sigma^*$:[4]

$$\mathsf{LCE}(x,y) = \max\{\ell : X[x\,..\,x+\ell) = Y[y\,..\,y+\ell)\}.$$

After $O(|X|+|Y|)$-time preprocessing, LCE queries can be answered in $O(1)$ time [16, 27]. This notion is often generalized to find the maximum length for which the corresponding substrings still have a small Hamming distance; formally,

$$\mathsf{LCE}_d(x,y) = \max\{\ell : \mathtt{HD}(X[x\,..\,x+\ell), Y[y\,..\,y+\ell)) \leq d\}.$$

---

[1] The smallest approximation ratio known to improve upon the running time $O(n+k^2)$ of the exact and conditionally optimal algorithm [27] stands at $3+o(1)$ [19]. The approximation ratio currently achievable in sublinear time is polylogarithmic in k (if $k < n^{1/4-\Omega(1)}$) [10] or polynomial in k (otherwise) [17].

[2] For small a, e.g., $a = 1$, using our exact algorithm would improve upon the bound in Case 2 and, in particular, reduce the dependency on k from cubic to quadratic.

[3] For $i, j \in \mathbb{Z}$, denote $[i\,..\,j] = \{k \in \mathbb{Z} : i \leq k \leq j\}$, $[i\,..\,j) = \{k \in \mathbb{Z} : i \leq k < j\}$, $(i\,..\,j] = \{k \in \mathbb{Z} : i < k \leq j\}$.

[4] Here and throughout, we implicitly assume the range $\ell \in [0\,..\,\min(|X|-x, |Y|-y)]$ needed to guarantee that $X[x\,..\,x+\ell)$ and $Y[y\,..\,y+\ell)$ are well-defined.

We further define $\mathsf{LCE}_{d,\epsilon}(x,y)$ as an arbitrary value between $\mathsf{LCE}_d(x,y)$ and $\mathsf{LCE}_{(1+\epsilon)d}(x,y)$; intuitively, this represents $\mathsf{LCE}_d(x,y)$ up to a $(1+\epsilon)$-approximation of the value of d.

1.3 Technical Overview

Exact Algorithm for $\mathtt{ED}_a$. The algorithm behind Proposition 1.1 is rather straightforward, but it serves as a foundation for subsequent results. The naive way of computing $\mathtt{ED}_a(X,Y)$ is to construct a dynamic-programming (DP) table $T[x,y] = \mathtt{ED}_a(X[0\,..\,x), Y[0\,..\,y))$. Considering only entries with $|x-y| \le k$ (others clearly exceed k) easily yields an $O(n+nk)$-time algorithm. If $ak \le n$, we achieve a better running time of $O(n+ak^2)$ by generalizing the Landau–Vishkin algorithm [26, 27] to allow $a > 1$. As explained next, this involves answering $O(ak)$ LCE queries for each of $(2k+1)$ diagonals (which consist of entries $T[x,y]$ with fixed $y-x$). The table T is monotone along the diagonals, and thus one can construct "waves" of entries with a common value v. This structure is conveniently described as another table that, for every possible cost $v \in \{0, \frac{1}{a}, \dots, k-\frac{1}{a}, k\}$ and every possible shift (diagonal) $s \in [-k\,..\,k]$, contains an entry $\mathbf{D}_v[s]$ defined as the furthest row x in T with $T[x,x+s] \le v$, or equivalently, the maximum index x such that $\mathtt{ED}_a(X[0\,..\,x), Y[0\,..\,x+s)) \le v$. To compute $\mathbf{D}_v[s]$, the algorithm uses three previously computed entries (for costs $v' < v$) and then performs a single LCE query. The running time of this algorithm is dominated by the construction and usage of a data structure answering LCE queries; see Section 2 for details.

A Naive Sampling Algorithm. A natural way to speed up the previous algorithm at the expense of accuracy is to use sampling. As a first attempt, let the algorithm sample positions in X at some rate $r \in (0,1)$ and compare each sampled $X[i]$ with $Y[i+s]$ for every $s \in [-k\,..\,k]$. The algorithm then reports the minimum-cost alignment of the samples (i.e., each sampled $X[i]$ is associated with exactly one shift s and is matched to $Y[i+s]$), where shift changes cost 1 per unit (i.e., changing s to s' costs $|s-s'|$) and mismatches cost $\frac{1}{ar}$ each. At best, we may hope to set $r = \Theta(\frac{1}{ka})$, the minimum needed for an optimal alignment with $\Theta(ka)$ substitutions and no indels.

This approach faces two serious obstacles. First, the query complexity is asymmetric: the algorithm queries, in expectation, $O(nr)$ positions in X and $O(knr)$ positions in Y. The query complexity into Y must be improved, ideally to $O(nr)$ as well. Second, the hope for sampling rate $r = \Theta(\frac{1}{ka})$ is not realistic, because, in the case of $a = 1$ (i.e., the standard edit distance), this would distinguish between $\mathtt{ED}(X,Y) \le k$ and $\mathtt{ED}(X,Y) > (1+\epsilon)k$ using $\tilde{O}(\frac{n}{k})$ samples, which is far beyond the reach of current techniques for edit distance estimation.

To overcome the first obstacle, we utilize approximate periodicity in our subroutine for answering approximate LCE queries, described later in this overview. To tackle the second obstacle, we make sure that every observed edit is charged against approximately $\frac{1}{r}$ unobserved mismatches (substitutions), for a judicious choice of $r = \tilde{O}(\frac{1}{\epsilon a})$, as explained next.

Our $(1+\epsilon)$-Approximation Algorithm for $\mathtt{ED}_a$. Let us modify our exact algorithm as follows. Instead of considering all possible values of v, we enumerate only over $v \in \{0, \epsilon, 2\epsilon, \dots, k\}$, where $\epsilon \ge \frac{1}{a}$ (otherwise, the exact algorithm already meets the requirements of Theorem 1.5). More precisely, for each diagonal s, we seek a value $\tilde{\mathbf{D}}_v[s]$ that approximates $\mathbf{D}_v[s]$ up to a $(1+\epsilon)$-factor slack in v. In other words, $\tilde{\mathbf{D}}_v[s]$ is between $\mathbf{D}_v[s]$ and $\mathbf{D}_{v(1+\epsilon)}[s]$;

Suppose that we have already computed $\tilde{\mathbf{D}}_{v'}[s]$ for all $v' < v$. Then, we can compute $\tilde{\mathbf{D}}_v[s]$ using a single $\mathsf{LCE}_{\epsilon a,\epsilon}$ query. Intuitively, this works well because these queries $(1+\epsilon)$-approximate the number of substitutions in the optimal alignment, except for up to ϵa additional substitutions (of total cost ϵ) per indel (of cost 1) in the optimal alignment; see Section 3 for details. Thus, the main engine of our algorithm is the following new tool:[5]

▶ **Problem 1.7** (Approximate LCE queries)**.** *Given strings X, Y of total length n, a threshold $d \in \mathbb{Z}_+$, a parameter $\epsilon \in \mathbb{R}_+$, and a width $w \in \mathbb{Z}_+$, build a data structure that efficiently computes $\mathsf{LCE}_{d,\epsilon}(x,y)$ for any $x \in [0\,..\,|X|]$ and $y \in [0\,..\,|Y|]$ such that $|x-y| \le w$.*

Answering $\mathsf{LCE}_{d,\epsilon}$ Queries. The following theorem captures our key technical innovation.

▶ **Theorem 1.8.** *After $\tilde{O}(\frac{n}{\epsilon^2 d})$-time randomized preprocessing (successful w.h.p.), the queries of Problem 1.7 can be answered deterministically (with no further randomness) in $\tilde{O}(dw)$ time.*

Previous work on edit distance [8, 10, 18, 19, 24] developed data structures for approximate LCE queries in a weaker version allowing any value between $\mathsf{LCE}_0(x,y)$ and $\mathsf{LCE}_d(x,y)$. As discussed in subsequent paragraphs, these results rely on identifying perfectly periodic string fragments, whereas, in our case, periodicity is approximate and holds up to $O(dw)$ mismatches.

In this overview, we focus on a simplified problem that already necessitates the main novel ideas behind Theorem 1.8. Namely, we consider the task of distinguishing, for any given $s \in [0\,..\,w) = [0\,..\,|Y|-|X|]$, whether $\mathtt{HD}(X, Y[s\,..\,s+|X|))$ is $\le d$ or $> 1.1d$. To see that this is a special case of Problem 1.7, observe that an $\mathsf{LCE}_{d,0.1}(0,s)$ query must return $|X|$ if $\mathtt{HD}(X, Y[s\,..\,s+|X|)) \le d$ and a value strictly smaller than $|X|$ if $\mathtt{HD}(X, Y[s\,..\,s+|X|)) > 1.1d$.

In this setting, our goal is to answer queries in $\tilde{O}(dw)$ time after $\tilde{O}(\frac{n}{d})$-time preprocessing. In particular, we want to answer all the w possible queries while probing $\tilde{O}(\frac{n}{d} + dw^2)$ characters in total. For comparison, let us consider as a baseline the guarantees achieved using simple techniques. A very naive approach would be to answer each query independently by sampling characters of X with rate $\tilde{O}(\frac{1}{d})$ and comparing each sampled character $X[x]$ with the aligned character $Y[x+s]$; this solution probes $\tilde{O}(\frac{nw}{d})$ characters in total. Another idea, employed in [8], could be to sample the characters of X and Y with the same rate $\tilde{O}(\frac{1}{\sqrt{d}})$ and, for each $s \in [0\,..\,w)$, compare $X[x]$ with $Y[x+s]$ whenever both characters have been sampled simultaneously, which happens with probability $\tilde{O}(\frac{1}{d})$; this solution probes $\tilde{O}(\frac{n}{\sqrt{d}})$ characters in total. Overall, we conclude that the simple techniques result in an effective sampling rate much larger than $\frac{1}{d}$, while our goal is to achieve the optimal rate $\tilde{O}(\frac{1}{d})$ (necessary already to answer a single query) at the price of a moderate additive cost.

The first step of our solution is to eliminate shifts $s \in [0\,..\,w)$ with very large distance $\mathtt{HD}(X, Y[s\,..\,s+|X|))$. For this, we use the naive approach to distinguish between Hamming distances $\le dw$ and $> 2dw$. This step costs $\tilde{O}(\frac{n}{dw})$ time per query, which is $\tilde{O}(\frac{n}{d})$ in total. If our filtering leaves us with at most one candidate shift $\tilde{s} \in [0\,..\,w)$, the answer for this shift is precomputed naively by sampling.

The interesting case is when at least two shifts $\tilde{s}_1 \neq \tilde{s}_2$ remain. A key observation is that the strings X and Y are then almost periodic with period $p := |\tilde{s}_1 - \tilde{s}_2|$; formally, $X[x] = X[x-p]$ holds for all but $O(dw)$ positions $x \in [p\,..\,|X|)$, and similarly for Y. In order to streamline notation, in this overview, we show how to exploit this structure for $p = 1$.

[5] To be precise, our algorithm makes $O(\epsilon^{-1}k^2)$ queries of Problem 1.7 with $d = \Theta(\epsilon a)$ and $w = k$. A single $\mathsf{LCE}_{\epsilon a,\epsilon}(0,0)$ query may already require accessing $\Omega(\frac{n}{\epsilon^3 a})$ characters, so the best time we could hope for is $\tilde{O}(\frac{n}{\epsilon^3 a} + \epsilon^{-1}k^2)$. If $\epsilon = \Theta(\frac{1}{a})$, this cannot improve upon our exact algorithm of Proposition 1.1.

In this case, most positions x in X have *uniform contexts* of any fixed length c, i.e., $X[x\,..\,x+c) = Y[y\,..\,y+c) = \mathtt{A}^c$ holds for some $\mathtt{A} \in \Sigma$ and all $y = x+s$ across $s \in [0\,..\,w)$. Intuitively, our goal is to quickly process uniform regions and spend our additional budget of $\tilde{O}(dw)$ time per query on the $O(cdw)$ positions with non-uniform contexts (each periodicity violation affects $O(c)$ contexts). Intuitively, uniform regions are much easier to handle due to the following observation: If we sample two strings $U, V \in \Sigma^m$ with an appropriate rate $r = \tilde{O}(\frac{2}{d})$, and this sampling reveals only $\mathtt{A}$s, then $\mathtt{HD}(U,V) \le \mathtt{HD}(U,\mathtt{A}^m) + \mathtt{HD}(V,\mathtt{A}^m) \le \frac{d}{2} + \frac{d}{2} \le d$ holds with high probability. This argument does not require any synchronization between samples, so we can reuse the same sample for all shifts.

The remaining challenge is that a rate-$\tilde{O}(\frac{1}{d})$ sample cannot perfectly identify uniform and non-uniform regions. Thus, we need to design a sampling mechanism that provides an unbiased estimate of $\mathtt{HD}(X, Y[s\,..\,s+|Y|))$ yet allows skipping regions that look uniform from the perspective of a rate-$\tilde{O}(\frac{1}{d})$ sampling. For this, we use two random subsets $S_X \subseteq [0\,..\,|X|)$ and $S_Y \subseteq [0\,..\,|Y|)$, where each index is picked independently with rate $r = \tilde{O}(\frac{1}{d})$. Every mismatch $X[x] \neq Y[y]$ is given two chances for detection: when $x \in S_X$ or $y \in S_Y$. In order to avoid double counting, the mismatch at position $x \in S_X$ will be counted if and only if $X[x]$ has fewer occurrences than $Y[y]$ within the contexts $X[x\,..\,x+c)$ and $Y[y\,..\,y+c)$ of length $c = \tilde{O}(\frac{1}{r})$. In this case, with high probability, the sampling reveals a character other than $X[x]$ within the contexts, and thus the algorithm classifies x as *non-uniform*. For positions $x \in S_X$ classified as non-uniform, our query algorithm explicitly compares $X[x]$ to $Y[y]$ (for $y = s+x$) and, upon detecting a mismatch, reads the entire contexts $X[x\,..\,x+c)$ and $Y[y\,..\,y+c)$ to verify whether indeed $X[x]$ is less frequent than $Y[y]$ within these contexts. We can afford both steps because the expected number of positions $x \in S_X$ with non-uniform contexts is $O(rcdw) = \tilde{O}(dw)$ and the expected number of detected mismatches $X[x] \neq Y[y]$ is $O(rd)$.

For a complete proof of Theorem 1.8, without the simplifying assumptions, see Section 4. The techniques employed to eliminate these assumptions build upon [24], where the simplified problem involves deciding, for every $s \in [0\,..\,w)$, whether $\mathtt{HD}(X, Y[s\,..\,s+|X|))$ is 0 or $> d$. Unlike in our setting, that auxiliary problem allows eliminating a shift s upon discovery of a single mismatch $X[x] \neq Y[x+s]$. This leads to a relatively straightforward $\tilde{O}(\frac{n}{d} + w)$-time solution [10, 24], which we sketch next to facilitate easy comparison with our new method. First, exact pattern matching is used to filter out shifts s such that $X[0\,..\,2w) \neq Y[s\,..\,s+2w)$ If at most one shift $\tilde{s}$ remains, then a naive $\tilde{O}(\frac{n}{d})$-time solution can be used to estimate $\mathtt{HD}(X, Y[\tilde{s}\,..\,\tilde{s}+|X|))$. On the other hand, if there are at least two shifts $\tilde{s}_1 \neq \tilde{s}_2$ left, then $p := |\tilde{s}_1 - \tilde{s}_2|$ is a period of $X[0\,..\,2w)$. A naive rate-$\tilde{O}(\frac{1}{d})$ sampling is used to check whether this period extends to the entire X and the relevant portion of Y. If this procedure does not identify any violation of the period, then all the remaining shifts satisfy $\mathtt{HD}(X, Y[s\,..\,s+|X|)) \le d$ with high probability. Otherwise, another call to exact pattern matching (for substrings near the period violation) eliminates all but at most one of the remaining shifts.

Algorithms for the (k_I, k_S)-Alignment Problem. Our algorithms computing $\mathtt{ED}_a$ (exactly or approximately) can be easily adapted to solve the (k_I, k_S)-alignment problem. The only modification required is to add another dimension to the $\mathbf{D}$ and $\tilde{\mathbf{D}}$ tables so that the algorithm separately keeps track of the number of indels and substitutions (as opposed to their total weight). In particular, $\mathbf{D}_{v_I,v_S}[s]$ is the maximum index x such that $X[0\,..\,x)$ and $Y[0\,..\,x+s)$ admit a (v_I, v_S)-alignment, and $\tilde{\mathbf{D}}_{v_I,v_S}[s]$ approximates $\mathbf{D}_{v_I,v_S}[s]$ up to a small slack in the value v_S. The exact algorithm uses $O(k_S k_I)$ LCE queries for each of the $2k_I + 1$ main diagonals, whereas the approximation algorithm asks $O(\frac{k_S k_I}{d})$ $\mathsf{LCE}_{d,\epsilon}$ queries per diagonal for a carefully chosen parameter $d = \Theta(\frac{\epsilon k_S}{k_I})$; see the full version for details.

Tight Lower Bound for Exact Computation of $\mathtt{ED}_a$. Our lower bound essentially proves that the $O(n+k\cdot\min(n,ak))$ running time in Proposition 1.1 is, up to subpolynomial factors, point-wise optimal as a function of (n,a,k). We formalize this delicate statement as follows:

▶ **Theorem 1.9.** *Consider sequences $(a_n)_{n=1}^{\infty}$ and $(k_n)_{n=1}^{\infty}$ with entries $a_n, k_n \in [1\,..\,n]$ computable in $\mathrm{poly}(n)$ time. Unless the Orthogonal Vectors Conjecture fails, there is no algorithm that, for some fixed $\epsilon>0$, every $n\in\mathbb{Z}_+$, and all strings X,Y with $|X|+|Y|\le n$, in $O((n+k_n\cdot\min(n,a_nk_n))^{1-\epsilon})$ time computes $\mathtt{ED}_{a_n}(X,Y)$ or reports that $\mathtt{ED}_{a_n}(X,Y)>k_n$.*

We prove our conditional lower bound in two steps. First, we show that, for any $a\in[1\,..\,n]$, $\Omega(n^{2-\epsilon})$ time is necessary to compute $\mathtt{ED}_a(X,Y)$ with $|X|+|Y|\le n$. We then build upon this construction to prove a lower bound on computing $\mathtt{ED}_a(X,Y)$ under a guarantee that $\mathtt{ED}_a(X,Y)\le k$ for a given $k=o(n)$. In the first step, we follow the approach of Bringmann and Künnemann [12], who proved the desired lower bound for $a=O(1)$. Unfortunately, when translated to larger a, their arguments only exclude $O(n^{2-\epsilon}a^{-6})$-time algorithms. The tool we adapt from [12] is a generic reduction from the Orthogonal Vectors problem to the problem of computing an abstract string similarity measure. This framework is formulated in terms of an *alignment gadget* that needs to be supplied for the measure in question. Unlike [12], instead of working directly with $\mathtt{ED}_a(X,Y)$, we consider an *asymmetric* measure $\mathtt{D}^+(X,Y):=\mathtt{ED}_a(\$^{|Y|}\cdot X\cdot\$^{|Y|},Y)-|XY|$, where \$ is a character not present in Y. Even for $a=O(1)$, this trick greatly simplifies the proof in [12] at the cost of increasing the alphabet size from 2 to 9. As a result, for each parameter a, every instance of the Orthogonal Vectors problem produces a pair of strings (X,Y) and a threshold k such that $\mathtt{ED}_a(X,Y)\le k$ if and only if the instance is a YES instance. However, since $\mathtt{ED}_a(X,Y)=\Theta(|X|+|Y|)$ for instances obtained through $\mathtt{D}^+(\cdot,\cdot)$, this construction alone provides no information on how the running time depends on k. For this, we express an instance of the Orthogonal Vectors problem as an OR-composition of smaller instances, and, for each of them, we construct a pair of strings (X_i,Y_i) such that the original instance is a YES-instance if and only if $\min_i \mathtt{ED}_a(X_i,Y_i)\le k_{\min}$. An appropriate gadget combines the pairs (X_i,Y_i) into a single pair (X,Y) with $\min_i \mathtt{ED}_a(X_i,Y_i)\le k_{\min}$ if and only if $\mathtt{ED}_a(X,Y)\le k$ for some $k=\Theta(\frac{n}{a})$, yielding an $\Omega((nk)^{1-\epsilon})$-time lower bound for this case. The lower bound of $\Omega((ak^2)^{1-\epsilon})$ for $k=O(\frac{n}{a})$ follows because the problem in question does not get easier as we increase n while preserving a and k; see Section 5 for details.

2 Computing $\mathtt{ED}_a$ Exactly

▶ **Proposition 1.1.** *Given two strings $X,Y\in\Sigma^*$ of total length n, a cost parameter $a\in\mathbb{Z}_+$, and a threshold $k\in\mathbb{R}_+$, one can compute $\mathtt{ED}_a(X,Y)$ or report that $\mathtt{ED}_a(X,Y)>k$ in deterministic time $O(n+k\min(n,ka))$.*

Define a DP table T with $T[x,y]=\mathtt{ED}_a(X[0\,..\,x),Y[0\,..\,y))$ for $x\in[0\,..\,|X|]$ and $y\in[0\,..\,|Y|]$. Then, $T[0,0]=0$ and the other entries can be computed according to the following formula (each argument of min is included only if the corresponding condition holds):

$$T[x,y]:=\min\begin{Bmatrix} T[x-1,y]+1 & \text{if } x>0\\ T[x,y-1]+1 & \text{if } y>0\\ T[x-1,y-1]+\frac{1}{a}\cdot\mathbb{1}_{X[x-1]\neq Y[y-1]} & \text{if } x>0 \text{ and } y>0\end{Bmatrix}.$$

One algorithm runs in time $O(n+nk)$ and follows [31]. It computes all entries $T[x,y]$ that satisfy $T[x,y]\le k$; this is correct because $T[x,y]$ only depends on entries $T[x',y']\le T[x,y]$. As $T[x,y]\ge|x-y|$, this algorithm considers $O(n+nk)$ entries, each computed in $O(1)$ time.

Another algorithm runs in time $O(n + ak^2)$ and follows [27], as explained in Section 1.3. It is implemented as Algorithm 1, where all values $\mathbf{D}_v[s]$ are implicitly initialized to $-\infty$.

Algorithm 1 Exact algorithm for $\mathtt{ED}_a$.

```
foreach v ∈ {0, 1/a, 2/a, ..., 1/a ⌊ak⌋} do
    foreach s ∈ [−⌊v⌋ .. ⌊v⌋] do
        D'_v[s] ← min(|X|, |Y| − s, max(D_{v−1}[s − 1], D_{v−1/a}[s] + 1, D_{v−1}[s + 1] + 1));
        if s = 0 then D'_v[s] ← max(D'_v[s], 0);
        D_v[s] ← D'_v[s] + LCE(D'_v[s], D'_v[s] + s);
    if D_v[|Y| − |X|] = |X| then return v;
return "> k";
```

The correctness of Algorithm 1 follows from Lemma 2.1 and the running time is proportional to the number of LCE queries, which is $O(ak^2)$, plus the $O(n)$ construction time of an LCE data structure [16, 27].

▶ **Lemma 2.1.** *For every $v \in \{0, \frac{1}{a}, \dots, \frac{1}{a}\lfloor ak \rfloor\}$, $x \in [0\,..\,|X|]$, and $y \in [0\,..\,|Y|]$, we have $T[x, y] \le v$ if and only if $\mathbf{D}_v[y - x] \ge x$.*

Proof. Let us first prove, by induction on v, that $\mathbf{D}_v[y-x] \ge x$ if $T[x, y] \le v$. Fix an optimal $\mathtt{ED}_a$-alignment between $X[0\,..\,x)$ and $Y[0\,..\,y)$ and consider the longest suffixes $X[x'\,..\,x) = Y[y'\,..\,y)$ aligned without any edit. We shall prove that $\mathbf{D}'_v[y - x] = \mathbf{D}'_v[y' - x'] \ge x'$ by considering four cases:

- If $x' = y' = 0$, then $\mathbf{D}'_v[y' - x'] = \mathbf{D}'_v[0] \ge 0$.
- If $X[x' - 1]$ is deleted, then $v \ge T[x', y'] = T[x' - 1, y'] + 1$, and the inductive assumption yields $\mathbf{D}'_v[y' - x'] \ge \mathbf{D}_{v-1}[y' - x' + 1] + 1 \ge x'$.
- If $Y[y' - 1]$ is inserted, then $v \ge T[x', y'] = T[x', y' - 1] + 1$, and the inductive assumption yields $\mathbf{D}'_v[y' - x'] \ge \mathbf{D}_{v-1}[y' - x' - 1] \ge x'$.
- Otherwise, $X[x' - 1]$ is substituted for $Y[y' - 1]$; then, $v \ge T[x', y'] = T[x' - 1, y' - 1] + \frac{1}{a}$, and the inductive assumption yields $\mathbf{D}'_v[y' - x'] \ge \mathbf{D}_{v-\frac{1}{a}}[y' - x'] + 1 \ge x'$.

Now, $\mathbf{D}_v[y - x] \ge x$ follows from $\mathbf{D}'_v[y' - x'] \ge x'$ due to $\mathsf{LCE}(x', y') \ge x - x'$.

The converse implication is also proved by induction on v. We consider four cases depending on $x' := \mathbf{D}'_v[y - x]$ and $y' := x' + (y - x)$:

- If $x' \le \mathbf{D}_{v-1}[y'-x'+1]+1$, then, by the inductive assumption, $T[x', y'] \le T[x'-1, y']+1 \le (v - 1) + 1$.
- If $x' \le \mathbf{D}_{v-1}[y' - x' - 1]$, then, by the inductive assumption, $T[x', y'] \le T[x', y' - 1] + 1 \le (v - 1) + 1$.
- If $x' \le \mathbf{D}_{v-\frac{1}{a}}[y'-x']+1$, then, by the inductive assumption, $T[x', y'] \le T[x'-1, y'-1]+\frac{1}{a} \le (v - \frac{1}{a}) + \frac{1}{a}$.
- In the remaining case, we have $x' = y' = 0$, and thus $T[x', y'] = 0 \le v$.

In all cases, $T[x', y'] \le v$ implies $T[x, y] \le v$ because $\mathsf{LCE}(x', y') \ge x - x'$. ◀

3 Approximating $\mathtt{ED}_a$

In this section, we present our solution for Problem 1.3. Recall that the input consists of strings X, Y, a cost parameter $a \in \mathbb{Z}_+$, a threshold $k \in \mathbb{R}_+$, and an accuracy parameter $\epsilon \in (0, 1)$, and the task is to distinguish between $\mathtt{ED}_a(X, Y) \le k$ and $\mathtt{ED}_a(X, Y) > (1 + \epsilon)k$.

Our procedure mimics the behavior of Algorithm 1 with a coarser granularity of the costs. It is implemented as Algorithm 2, where we assume that all values $\tilde{\mathbf{D}}_v[s]$ are implicitly initialized to $-\infty$ and that $\epsilon a \in \mathbb{Z}_+$ (we will fall back to Algorithm 1 whenever $\epsilon a < 1$).

Algorithm 2 Approximation Algorithm.

```
1 foreach v ∈ {0, ε, 2ε, ..., ε⌈ε^{-1}k⌉} do
2     foreach s ∈ [−⌊v⌋ .. ⌊v⌋] do
3         D̃'_v[s] ← min(|X|, |Y| − s, max(D̃_{v−1}[s − 1], D̃_{v−ε}[s], D̃_{v−1}[s + 1] + 1));
4         if s = 0 then D̃'_v[s] ← max(D̃'_v[s], 0);
5         D̃_v[s] ← D̃'_v[s] + LCE_{εa,ε}(D̃'_v[s], D̃'_v[s] + s);
6     if D̃_v[|Y| − |X|] = |X| then return YES;
7 return NO
```

The following lemma, justifying the correctness of Algorithm 2, is proved below through an appropriate adaptation of the arguments behind Lemma 2.1.

▶ **Lemma 3.1.** *For every $v \in \{0, \epsilon, 2\epsilon, \ldots, \epsilon\lceil\epsilon^{-1}k\rceil\}$, $x \in [0\,..\,|X|]$, and $y \in [0\,..\,|Y|]$, if $T[x, y] \le v$, then $\tilde{\mathbf{D}}_v[y - x] \ge x$, and if $\tilde{\mathbf{D}}_v[y - x] \ge x$, then $T[x, y] \le v(1 + \epsilon + \epsilon^2) + \epsilon + \epsilon^2$.*

Proof. We start by proving the first implication inductively on v. Let us fix an optimum $\mathtt{ED}_a$-alignment of $X[0\,..\,x)$ and $Y[0\,..\,y)$ and consider the longest suffixes $X[x'\,..\,x)$ and $Y[y'\,..\,y)$ aligned with at most ϵa substitutions and no indels. We shall prove that $\tilde{\mathbf{D}}'_v[y - x] = \tilde{\mathbf{D}}'_v[y' - x'] \ge x'$ by considering four cases:

- If $x' = y' = 0$, then $\tilde{\mathbf{D}}'_v[y' - x'] = \tilde{\mathbf{D}}'_v[0] \ge 0$.
- If $X[x' - 1]$ is deleted, then $v \ge T[x', y'] = T[x' - 1, y'] + 1$, and the inductive assumption yields $\tilde{\mathbf{D}}'_v[y' - x'] \ge \tilde{\mathbf{D}}_{v-1}[y' - x' + 1] + 1 \ge x'$.
- If $Y[y' - 1]$ is inserted, then $v \ge T[x, y] = T[x', y' - 1] + 1$, and the inductive assumption yields $\tilde{\mathbf{D}}'_v[y' - x'] \ge \tilde{\mathbf{D}}_{v-1}[y' - x' - 1] \ge x'$.
- Otherwise, there are exactly ϵa substitutions between $X[x'\,..\,x)$ and $Y[y'\,..\,y)$ (recall that $\epsilon a \in \mathbb{Z}$). Thus, $v \ge T[x, y] = T[x', y'] + \epsilon$, and the inductive assumption yields $\tilde{\mathbf{D}}'_v[y' - x'] \ge \tilde{\mathbf{D}}_{v-\epsilon}[y' - x'] \ge x'$.

Now, $\tilde{\mathbf{D}}_v[y - x] \ge x$ follows from $\tilde{\mathbf{D}}'_v[y' - x'] \ge x'$ due to $\mathsf{LCE}_{\epsilon a}(x', y') \ge x - x'$.

The second implication is also proved by induction on v. We consider four cases depending on $x' := \tilde{\mathbf{D}}'_v[y - x]$ and $y' := x' + (y - x)$:

- If $x' \le \tilde{\mathbf{D}}_{v-1}[y' - x' + 1] + 1$, then, by the inductive assumption, $T[x', y'] \le T[x' - 1, y'] + 1 \le (v - 1)(1 + \epsilon + \epsilon^2) + \epsilon + \epsilon^2 + 1 = v(1 + \epsilon + \epsilon^2)$.
- If $x' \le \tilde{\mathbf{D}}_{v-1}[y' - x' - 1]$, then, by the inductive assumption, $T[x', y'] \le T[x', y' - 1] + 1 \le (v - 1)(1 + \epsilon + \epsilon^2) + \epsilon + \epsilon^2 + 1 = v(1 + \epsilon + \epsilon^2)$.
- If $x' \le \tilde{\mathbf{D}}_{v-\epsilon}[y' - x']$, then, by the inductive assumption, $T[x', y'] \le (v - \epsilon)(1 + \epsilon + \epsilon^2) + \epsilon + \epsilon^2 < v(1 + \epsilon + \epsilon^2)$.
- In the remaining case, we have $x' = y' = 0$. Trivially, $T[x', y'] = 0 \le v(1 + \epsilon + \epsilon^2)$.

In all cases, $T[x', y'] \le v(1 + \epsilon + \epsilon^2)$ implies $T[x, y] \le v(1 + \epsilon + \epsilon^2) + \epsilon + \epsilon^2$ because $\mathsf{LCE}_{(1+\epsilon)\epsilon a}(x', y') \ge x - x'$. ◀

▶ **Corollary 3.2.** *Algorithm 2 returns*

- *YES if $\mathtt{ED}_a(X, Y) \le k$, and*
- *NO if $\mathtt{ED}_a(X, Y) > k(1 + \epsilon + \epsilon^2) + 2\epsilon + 2\epsilon^2 + \epsilon^3$.*

Moreover, it can be implemented in $\tilde{O}(\frac{n}{\epsilon^3 a} + ak^3)$ time if $k \ge 1$.

Proof. If $\mathtt{ED}_a(X,Y) \le k$, we apply Lemma 3.1 for $v = \epsilon\lceil\epsilon^{-1}k\rceil$, $x = |X|$, and $y = |Y|$. We conclude that $\tilde{\mathbf{D}}_v[|X|-|Y|] \ge |X|$, which means that the algorithm returns YES. On the other hand, if the algorithm returns YES, then $\tilde{\mathbf{D}}_v[|X|-|Y|] \ge |Y|$ holds for some $v \in \{0,\epsilon,2\epsilon,\dots,\epsilon\lceil\epsilon^{-1}k\rceil\}$, which satisfies $v < k+\epsilon$. In this case, Lemma 3.1 implies that

$$\mathtt{ED}(X,Y) \le v(1+\epsilon+\epsilon^2)+\epsilon+\epsilon^2 < (k+\epsilon)(1+\epsilon+\epsilon^2)+\epsilon+\epsilon^2 = k(1+\epsilon+\epsilon^2)+2\epsilon+2\epsilon^2+\epsilon^3.$$

As for the running time, we observe that the number of $\mathsf{LCE}_{\epsilon a,\epsilon}(x,y)$ queries asked is $O((1+k)(1+\epsilon^{-1}k))$ and that each of them satisfies $|x-y| \le k$. Consequently, the claimed running time follows from Theorem 1.8. ◀

▶ **Theorem 1.5.** *One can solve Problem 1.3 correctly with high probability within time:*
1. $\tilde{O}(\frac{n}{\epsilon^2 ak})$ *if* $k < 1$;
2. $\tilde{O}(\frac{n}{\epsilon^3 a} + ak^3)$ *if* $1 \le k < \frac{n}{\epsilon a}$; *and*
3. $O(1)$ *if* $k \ge \frac{n}{\epsilon a}$.

Proof. First, consider $k < 1$. If $|X| \ne |Y|$, then we can safely return NO due to $\mathtt{ED}_a(X,Y) \ge \big||X|-|Y|\big|$. Otherwise, we output YES if $\mathtt{HD}(X,Y) \le ak$ (which implies $\mathtt{ED}_a(X,Y) \le k$) and NO if $\mathtt{HD}(X,Y) > (1+\epsilon)ak$ (which implies $\mathtt{ED}_a(X,Y) \ge \min(1,(1+\epsilon)k)$). Previous work for Hamming distance (e.g., [21]) lets us distinguish these two possibilities in $\tilde{O}(\frac{n}{\epsilon^2 ak})$ time.

Next, consider $1 \le k < \frac{n}{\epsilon a}$. If $\epsilon < \frac{7}{a}$, we use the algorithm of Proposition 1.1, which either computes $\mathtt{ED}_a(X,Y)$ or reports that $\mathtt{ED}_a(X,Y) > k$. This is clearly sufficient to distinguish between $\mathtt{ED}_a(X,Y) \le k$ and $\mathtt{ED}_a(X,Y) > (1+\epsilon)k$ for any $\epsilon \ge 0$. The running time is $O(n+ak^2) = O(\frac{n}{\epsilon a}+ak^2) = \tilde{O}(\frac{n}{\epsilon^3 a}+ak^3)$. The most interesting case is when $\epsilon \ge \frac{7}{a}$. We then run Algorithm 2 with the accuracy parameter decreased to $\bar{\epsilon} := \frac{1}{a}\lfloor\frac{\epsilon a}{7}\rfloor$; in particular, this guarantees $\bar{\epsilon}a \in \mathbb{Z}_+$. By Corollary 3.2, the algorithm returns YES if $\mathtt{ED}_a(X,Y) \le k$, and, due to $(1+\epsilon)k \ge (1+7\bar{\epsilon})k \ge (1+\bar{\epsilon}+\bar{\epsilon}^2)k + 2\bar{\epsilon}+2\bar{\epsilon}^2+\bar{\epsilon}^3$, it returns NO if $\mathtt{ED}_a(X,Y) > (1+\epsilon)k$. The running time is $\tilde{O}(\frac{n}{\bar{\epsilon}^3 a}+ak^3) = \tilde{O}(\frac{n}{\epsilon^3 a}+ak^3)$ due to $\bar{\epsilon} \ge \frac{\epsilon}{14}$.

Finally, consider $k \ge \frac{n}{\epsilon a}$. We return YES if $\big||X|-|Y|\big| \le k$ and NO otherwise. This is correct due to $\big||X|-|Y|\big| \le \mathtt{ED}_a(X,Y) \le \big||X|-|Y|\big| + \frac{1}{a}\min(|X|,|Y|) \le \big||X|-|Y|\big| + \epsilon k$. ◀

4 Answering $\mathsf{LCE}_{d,\epsilon}$ Queries

In this section, we explain how to implement $\mathsf{LCE}_{d,\epsilon}$ queries. We focus on the decision version of these queries, asking to distinguish whether $\mathtt{HD}(X[x\,..\,x+\ell), Y[y\,..\,y+\ell))$ is $\le d$ or $> (1+\epsilon)d$. A naive way of performing such a test would be to count mismatches $X[x+s] \ne Y[y+s]$ at positions $s \in S$, where each $s \in [0\,..\,\ell)$ is sampled into S independently with an appropriate rate r. The resulting estimator, $H := |\{s \in S : X[x+s] \ne Y[y+s]\}|$, follows the binomial distribution $\mathrm{Bin}(h,r)$, where $h = \mathtt{HD}(X[x\,..\,x+\ell), Y[y\,..\,y+\ell))$. Thus, we can compare H against $(1+\frac{\epsilon}{3})rd$ to distinguish between $h \le d$ and $h \ge (1+\epsilon)d$: By the Chernoff bound, if $h \le d$, then

$$\Pr[H \ge (1+\tfrac{\epsilon}{3})rd] \le \exp(-\tfrac{1}{27}\epsilon^2 rd),$$

whereas if $h \ge (1+\epsilon)d$, then

$$\Pr[H < (1+\tfrac{\epsilon}{3})rd] \le \Pr[H < (1-\tfrac{\epsilon}{3})(1+\epsilon)rd] \le \exp(-\tfrac{1}{18}\epsilon^2 rd).$$

In particular, if $r = \Theta(\epsilon^{-2}d^{-1}\log n)$, then w.h.p. the query algorithm is correct and costs $\tilde{O}(1+r\ell) = \tilde{O}(1+\epsilon^{-2}d^{-1}\ell)$ time.

The same approach can be used in the setting resembling that of Theorem 1.8, where queries need to be answered deterministically after randomized preprocessing.

▶ **Fact 4.1.** *There exists a data structure that, after randomized preprocessing of strings $X, Y \in \Sigma^*$ and a rate $r \in (0,1)$, given positions $x \in [0\,..\,|X|]$, $y \in [0\,..\,|Y|]$ and a length $\ell \in [0\,..\,\min(|X|-x, |Y|-y)]$, outputs a value distributed as $\mathrm{Bin}(\mathtt{HD}(X[x\,..\,x+\ell), Y[y\,..\,y+\ell)), r)$. Moreover, the answers are independent for queries with disjoint intervals $[x\,..\,x+\ell)$. With high probability, the preprocessing time is $\tilde{O}(1+r(|X|+|Y|))$ and the query time is $\tilde{O}(1+r\ell)$.*

Proof. At preprocessing time, we draw $S_X \subseteq [0\,..\,|X|)$ with each position sampled independently with rate r. At query time, the algorithm counts $x' \in S_X \cap [x\,..\,x+\ell)$ such that $X[x'] \neq Y[y-x+x']$. It is easy to see that each mismatch $X[x+s] \neq Y[y+s]$, with $s \in [0\,..\,\ell)$, is included with rate r, and these events are independent across the mismatches. Moreover, the sets $S_X \cap [x\,..\,x+\ell)$ across distinct ranges $[x\,..\,x+\ell)$ are independent, so the resulting estimators are independent as well. ◀

Our aim, however, is a query time that does not grow with ℓ. As shown in Sections 4.1 and 4.2, $\tilde{O}(dw)$ query time can be achieved after preprocessing $X[x\,..\,x+\ell)$ in $\tilde{O}(w+r\ell)$-time.[6] While preprocessing each fragment $X[x\,..\,x+\ell)$ is too costly, it suffices to focus on fragments such that $[x\,..\,x+\ell)$ is a *dyadic interval* (where $\log \ell$ and $\frac{x}{\ell}$ are both integers); this is because, for any intermediate value $m \in [0\,..\,\ell)$, the problem of estimating $\mathtt{HD}(X[x\,..\,x+\ell), Y[y\,..\,y+\ell))$ naturally reduces to the problems of estimating both $\mathtt{HD}(X[x\,..\,x+m), Y[y\,..\,y+m))$ and $\mathtt{HD}(X[x+m\,..\,x+\ell), Y[y+m\,..\,y+\ell))$. Estimators following binomial distribution are particularly convenient within such a reduction: if $B \sim \mathrm{Bin}(h,r)$ and $B' \sim \mathrm{Bin}(h',r)$ are independent, then $B+B' \sim \mathrm{Bin}(h+h',r)$. Unfortunately, our techniques do not allow estimating $\mathtt{HD}(X[x\,..\,x+\ell), Y[y\,..\,y+\ell))$ when this value is much larger than d. Thus, we formalize the outcome of our subroutines as *capped* binomial variables define below.

▶ **Definition 4.2.** *An integer-valued random variable B is a d-*capped binomial variable *with parameters h and r, denoted $B \in \mathrm{Bin}_d(h,r)$, if it satisfies the following conditions:*

- *If $h \le d$, then $B \sim \mathrm{Bin}(h,r)$, i.e., $\Pr[B \ge x] = \Pr[\mathrm{Bin}(h,r) \ge x]$ for all $x \in \mathbb{Z}$.*
- *Otherwise, B stochastically dominates $\mathrm{Bin}(h,r)$, i.e., $\Pr[B \ge x] \ge \Pr[\mathrm{Bin}(h,r) \ge x]$ for all $x \in \mathbb{Z}$.*

Moreover, to allow a small deviation from the desired distribution (e.g., in the unlikely event the number of sampled positions is much larger than expected), for $\delta \in [0,1]$, we write $B \in \mathrm{Bin}_{d,\delta}(h,r)$, if B is at total variation distance at most δ from a variable $B' \in \mathrm{Bin}_d(h,r)$. The capped binomial variables can be composed just like their uncapped counterparts.

▶ **Observation 4.3.** *If $B \in \mathrm{Bin}_{d,\delta}(h,r)$ and $B' \in \mathrm{Bin}_{d,\delta'}(h',r)$ are independent, then $B+B' \in \mathrm{Bin}_{d,\delta+\delta'}(h+h',r)$.*

As indicated above, the main building block of our solution to Problem 1.7 is a component for estimating $\mathtt{HD}(X[x\,..\,x+\ell), Y[y\,..\,y+\ell))$ for fixed x, ℓ and varying $y \in [x-w\,..\,x+w]$. This task can be formalized as a problem of estimating *text-to-pattern* Hamming distances [14], that is, the distances between a pattern X and length-$|X|$ fragments of a text Y.

▶ **Problem 4.4.** *Preprocess strings $X, Y \in \Sigma^*$, real numbers $\delta \in (0, \frac{1}{|X|})$ and $r \in (0,1)$, and an integer $d \ge r^{-1}$ into a data structure that, given a shift $s \in [0\,..\,|Y|-|X|]$, outputs a value distributed as $\mathrm{Bin}_{d,\delta}(\mathtt{HD}(X, Y[s\,..\,s+|X|)), r)$.*

In Section 4.1, we study Problem 4.4 in an important special case when X, Y are almost periodic. The general solution to Problem 4.4 is then presented in Section 4.2, and we derive Theorem 1.8 in Section 4.3.

[6] Recall the assumption in Problem 1.7 that $|x-y| \le w$ holds for all queries.

4.1 Text-to-Pattern Hamming Distances for Almost Periodic Strings

In this section, we consider the case when X and Y are almost periodic. Formally, this means that the data structure is additionally given (at construction time) integers p and m such that $\mathtt{HD}(X[0\,..\,|X|-p), X[p\,..\,|X|)) + \mathtt{HD}(Y[0\,..\,|Y|-p), Y[p\,..\,|Y|)) \le m$.

The presentation of our data structure comes in four parts. We start with an overview, where we introduce some notation and provide intuition. This is followed by Algorithm 3, which gives a precise mathematical definition of the values reported by our data structure. Then, in Lemma 4.5, we formalize the intuition and prove that the outputs of Algorithm 3 satisfy the requirements of Problem 4.4, that is, their distributions follow $\mathrm{Bin}_{d,\delta}(\mathtt{HD}(X, Y[s\,..\,s+|X|)), r)$. Finally, in Lemma 4.6, we provide an efficient implementation of Algorithm 3 and the complexity analysis of the resulting procedure.

The randomness in our data structure comes from two sets $S_X \subseteq [0\,..\,|X|)$ and $S_Y \subseteq [0\,..\,|Y|)$ with each element sampled independently with probability r. At a first glance, it may seem that S_X would be sufficient because $|\{x \in S_X : X[x] \neq Y[s+x]\}|$ satisfies the requirements of Problem 4.4 (as proved in Fact 4.1). However, these values cannot be computed efficiently. To see this, suppose that X and Y consist only of the character $\mathtt{A}$, except for a single position y where $Y[y] = \mathtt{B}$. In this case, we would need to return 1 if and only if $y - s \in S_X$; this requires $\Omega(|Y|)$ preprocessing time (to discover y) or $\Omega(|S_X|)$ query time (to check if $y = s + x$ for some $x \in S_X$). Thus, our strategy is more involved: we partition the set of mismatches $\{(x, y) : X[x] \neq Y[y]\}$ into two disjoint classes, M_X and M_Y, and we return $|\{(x, y) \in M_X : x \in S_X \text{ and } y - x = s\}| + |\{(x, y) \in M_Y : y \in S_Y \text{ and } y - x = s\}|$. In other words, depending on its class, a mismatch in $X[x] \neq Y[y]$ is counted either if $x \in S_X$ or if $y \in S_Y$. Our classification is determined by comparing the frequency of characters $X[x]$ and $Y[y]$ in proximity to the two underlying positions. Intuitively, this is helpful because a rare character is unlikely to be discovered unless it happens to be sampled. Formally, for each position $x \in [0\,..\,|X|)$, we define its context $C_x = \{x, x+p, x+2p, \ldots, x+(c-1)p\}$, where $c = \tilde{O}(r^{-1})$ is sufficiently large, and the context C_y of $y \in [0\,..\,|Y|)$ is defined similarly. Then, we count the occurrences of $X[x]$ and $Y[y]$ as $X[x']$ for $x' \in C_x$ and as $Y[y']$ for $y' \in C_y$, denoting the resulting numbers by m_x and m_y, respectively (see Lines 11 and 12 of Algorithm 3). We place the mismatch in M_X if $m_x \le m_y$ and in M_Y otherwise.

Algorithm 3 Answering queries of Problem 4.4 in the almost periodic case.

```
1  H ← 0;
2  foreach x ∈ [0 .. |X|) do
3      y ← s + x;
4      if x ∉ S_X and y ∉ S_Y then continue;
5      c ← ⌈r^{-1} ln δ^{-2}⌉;
6      C_x ← {x, x + p, ..., x + (c − 1)p};
7      C_y ← {y, y + p, ..., y + (c − 1)p};
8      if C_x ⊆ [0 .. |X|) and |{X[x'] : x' ∈ C_x ∩ S_X} ∪ {Y[y'] : y' ∈ C_y ∩ S_Y}| = 1 then
9          continue;
10     if X[x] = Y[y] then continue;
11     m_x ← |{x' ∈ C_x ∩ [0 .. |X|) : X[x'] = X[x]}| + |{y' ∈ C_y ∩ [0 .. |Y|) : Y[y'] = X[x]}|;
12     m_y ← |{x' ∈ C_x ∩ [0 .. |X|) : X[x'] = Y[y]}| + |{y' ∈ C_y ∩ [0 .. |Y|) : Y[y'] = Y[y]}|;
13     if (x ∈ S_X and m_x ≤ m_y) or (y ∈ S_Y and m_y < m_x) then H ← H + 1;
14 return H;
```

The benefit of this approach is that, if $x \in M_X$, then the contexts C_x and C_y (of total size $2c$) contain at least c occurrences of characters other than $X[x]$. In particular, at a small loss of total variation distance, we may assume that the samples S_X and S_Y have revealed such a character distinct from $X[x]$, and thus we need to read $Y[y]$ only conditioned on that event. At the same time, if X and Y are almost periodic and the sought Hamming distance is small, then the contexts C_x and C_y are typically uniform (meaning that all the $2c$ positions are occurrences of the same symbol). Hence, we can afford to investigate each location with non-uniform contexts and, in case a mismatch $X[x] \neq Y[y]$ is detected, to explicitly compute m_x and m_y in order to verify whether this mismatch indeed belongs to M_X or M_Y.

▶ **Lemma 4.5** (Correctness). *Given $s \in [0\,..\,|Y|-|X|]$, Algorithm 3 outputs a random variable H at total variation distance at most δ from $\mathrm{Bin}(h, r)$, where $h = \mathtt{HD}(X, Y[s\,..\,s+|X|))$.*

Proof. We shall first prove that a modification of Algorithm 3 with Lines 8 and 9 removed outputs $H \sim \mathrm{Bin}(h, r)$. Let $P \subseteq [0\,..\,|X|)$ be the set of positions for which H was incremented in Line 13. We claim that P is a subset of $M := \{x \in [0\,..\,|X|) : X[x] \neq Y[s+x]\}$ with each position sampled independently with rate r. Due to Line 10, we have $P \subseteq M$. For fixed positions $x \in M$ and $y = s + x$, let us consider the (deterministic) values m_x and m_y as defined in Lines 11 and 12. Observe that, for every $x \in M$, we have $x \in P$ if and only if ($x \in S_X$ and $m_x \leq m_y$) or ($y \in S_Y$ and $m_x > m_y$). Moreover, these are probability-r events independent across $x \in M$. Thus, in the modified version of the algorithm, we indeed have $H \sim \mathrm{Bin}(h, r)$.

Next, we shall prove that, for every $x \in P$, the probability of losing x due to Line 8 does not exceed δ^2. By the union bound, the total variation distance between H and $\mathrm{Bin}(h, r)$ is then at most $\delta^2|X| \leq \delta$. Consider a multiset $A := \{X[x'] : x' \in C_x \cap [0\,..\,|X|)\} \cup \{Y[y'] : y' \in C_y \cap [0\,..\,|Y|)\}$. The filtering of Line 8 applies only if $|A| = 2c$ and all the sampled characters in A match. If $m_x \leq m_y$ and $x \in S_X$, then $X[x]$ is sampled yet none of the at least c elements of A distinct from $X[x]$ is sampled. Similarly, if $m_y < m_x$ and $y \in S_Y$, then $Y[y]$ is sampled yet none of the at least c elements of A distinct from $Y[y]$ is sampled. In either case, the probability of losing $x \in P$ is bounded by $(1-r)^c \leq \exp(-rc) \leq \delta^2$, as claimed. ◀

▶ **Lemma 4.6** (Efficient implementation). *Algorithm 3 can be implemented so that, after $\tilde{O}(|S_X| + |S_Y|)$-time preprocessing, the query time is $\tilde{O}((m + p + h + r^{-1}) \log^2 \delta^{-1})$ with probability $1 - O(\delta)$, where $h = \mathtt{HD}(X, Y[s\,..\,s+|X|))$.*

Proof. While preprocessing X, we construct a data structure that, given a position $x \in [0\,..\,|X|)$ and a character $a \in \Sigma$, in $\tilde{O}(1)$ time computes the smallest position $x' \in S_X \cap [x\,..\,|X|)$ such that $X[x'] \neq a$ and $x' \equiv x \pmod p$ (if any such x' exists). This preprocessing costs $\tilde{O}(|S_X|)$ time (recall that all our model assumes oracle access to characters of the input strings). The string Y is preprocessed in the same way in $\tilde{O}(|S_Y|)$ time.

In the main loop of Line 2, we process $x \in [0\,..\,|X|)$ grouped by the remainder $x \bmod p$, starting from $x \in [0\,..\,p)$ and $y = s + x$. For each group, instead of increasing x and y by p each time, we utilize the precomputed data structure to perform larger jumps. Specifically, if an iteration terminates due to Line 4, we proceed directly to the subsequent state (x', y') such that $x' \in S_X$ or $y' \in S_Y$ (if any). Moreover, while executing Line 8, we compute $a = X[x]$ (if $x \in S_X$) or $a = Y[y]$ (otherwise) and determine the nearest sampled position $x' \in [x\,..\,|X|) \cap S_X$ with $X[x'] \neq a$ and $x' \equiv x \pmod p$ and the nearest sampled position $y' \in [y\,..\,|Y|) \cap S_Y$ with $Y[y'] \neq a$ and $y' \equiv y \pmod p$. Then, the condition of Line 8 is satisfied if and only if $(c-1)p < \min(|X| - x, x' - x, y' - y)$. In that case, we can increase x and y by $p \cdot \lceil \frac{1}{p} \min(|X| - x, x' - x, y' - y) - (c-1) \rceil$; the number of such increases can be bounded by $O(p + m)$ in total across all the remainders modulo p.

It remains to count positions reaching Lines 10 and 11. The condition $C_x \subseteq [0\,..\,|X|)$ is not satisfied for at most pc positions x. The condition $|\{X[x'] : x' \in C_x \cap [0\,..\,|X|)\} \cup \{Y[y'] : y' \in C_y \cap [0\,..\,|Y|)\}| = 1$ is not satisfied for at most $(m+h)c$ positions x. Hence, at most $(m+h+p)c$ positions may proceed to Line 10. However, due to the filtering of Line 4, the actual number of positions reaching Line 10 can be bounded by $\mathrm{Bin}((m+h+p)c, 2r)$, which is $O((m+h+p)\log\delta^{-1})$ with probability at least $1-\delta$. Hence, Lines 3–10 can be implemented in $\tilde{O}((m+h+p)\log\delta^{-1})$ time with probability at least $1-\delta$.

Furthermore, $X[x] = Y[y]$ is not satisfied for at most h positions, and, for these positions, computing m_x and m_y takes $O(c)$ time. Due to the filtering of Line 4, the actual number of positions reaching Line 11 is at most $\mathrm{Bin}(h, 2r)$, which is $O(\frac{h}{r} + \log\delta^{-1})$ with probability at least $1-\delta$. Hence, Lines 11–13 cost $O(h\log\delta^{-1} + r^{-1}\log^2\delta^{-1})$ time with probability at least $1-\delta$. Overall, the running time of any query is $\tilde{O}((m+p+h+r^{-1})\log^2\delta^{-1})$ with probability at least $1-O(\delta)$. ◀

4.2 Text-to-Pattern Hamming Distances for Arbitrary Strings

In this section, we design an efficient solution to Problem 4.4 (in the general case):

▶ **Proposition 4.7.** *Given an instance of Problem 4.4 with $w := |Y| - |X| + 1$ and $|X|r \geq dw$, after $\tilde{O}(|X|r\log\delta^{-1})$-time randomized preprocessing, one can deterministically (with no further randomness) answer the queries of Problem 4.4 in $\tilde{O}(dw\log^2\delta^{-1})$ time.*

Proof. In the preprocessing, for each shift $s \in [0\,..\,w)$, we distinguish, correctly with probability at least $1-\frac{\delta}{w}$, the case of $\mathtt{HD}(X, Y[s\,..\,s+|X|)) \leq dw$ from the case of $\mathtt{HD}(X, Y[s\,..\,s+|X|)) > 2dw$. This is implemented by sampling positions in X at rate $\Theta(\frac{\log(w\delta^{-1})}{dw})$ and comparing the sampled characters of X against the aligned characters of Y. By the union bound, the tests return correct values (for all $s \in [0\,..\,w)$) with probability at least $1-\delta$. The running time of this preprocessing step is $O(w \cdot \frac{\log(w\delta^{-1})}{dw} \cdot (|X|+|Y|)) = \tilde{O}(|X|r\log\delta^{-1})$ with probability at least $1-\delta$ (if this step takes too much time, we terminate the underlying computation and declare a preprocessing failure, which effectively means that $|X|$ is returned for all queries).

If the test reports a small value $\mathtt{HD}(X, Y[\tilde{s}\,..\,\tilde{s}+|X|))$ for just one shift $\tilde{s} \in [0\,..\,w)$, then we memorize this shift and compute a value distributed as $\mathrm{Bin}(\mathtt{HD}(X, Y[\tilde{s}\,..\,\tilde{s}+|X|)), r)$ by sampling the characters $X[x]$ and $Y[x+\tilde{s}]$ at rate r. This costs $O(|X|r\log\delta^{-1})$ time with probability at least $1-\delta$ (otherwise, we declare a preprocessing failure). At query time, we report the computed value for $\tilde{s}$. For each of the remaining shifts $s \neq \tilde{s}$, we are guaranteed that (with probability at least $1-\delta$) $\mathtt{HD}(X, Y[s\,..\,s+|X|)) > dw \geq d$, and thus we can return $|X|$ as a value distributed according to $\mathrm{Bin}_{d,\delta}(\mathtt{HD}(X, Y[s\,..\,s+|X|)), r)$.

The interesting case is when we detect (at least) two shifts $\tilde{s}_1 \neq \tilde{s}_2$ satisfying $\mathtt{HD}(X, Y[s\,..\,s+|X|)) \leq 2dw$. We then conclude that X and Y are approximately periodic with period $p := |\tilde{s}_1 - \tilde{s}_2|$. Formally, this means that $\mathtt{HD}(X[0\,..\,|X|-p), X[p\,..\,|X|)) + \mathtt{HD}(Y[0\,..\,|Y|-p), Y[p\,..\,|Y|)) \leq 8dw + w = O(dw)$ holds with probability at least $1-\delta$. In this case, at preprocessing time, we draw sets $S_X \subseteq [0\,..\,|X|), S_Y \subseteq [0\,..\,|Y|)$ with each element sampled independently with probability r. This costs $O(|X|r\log\delta^{-1})$ time with probability at least $1-\delta$ (otherwise, we declare a preprocessing failure). We apply the query procedure of Algorithm 3, described in Section 4.1, imposing a hard limit $\tilde{O}(dw\log^2\delta^{-1})$ on the query time. If the query algorithm exceeds the limit, we return a naive upper bound $H = |X|$. If $h \leq d$, the probability of exceeding the limit is $O(\delta)$ (this incorporates both X, Y violating the periodicity assumption and Algorithm 3 taking a long time due to an unlucky choice of S_X, S_Y). Hence, imposing the limit increases the total variation distance of H and $\mathrm{Bin}(h, r)$

by $O(\delta)$. Otherwise (if $h > d$), setting $H := |X|$ in some states of the probability space may only increase H in these states, and thus the resulting random variable stochastically dominates the original one. In particular, H is at total variation distance at most δ from a variable stochastically dominating $\mathrm{Bin}(h, r)$. Combining these two cases, we conclude that, if we shrink δ by an appropriate constant factor, the resulting value H satisfies the requirements of Problem 4.4 ◀

4.3 Proof of Theorem 1.8

In this section, we derive Theorem 1.8 from Proposition 4.7.

▶ **Theorem 1.8.** *After $\tilde{O}(\frac{n}{\epsilon^2 d})$-time randomized preprocessing (successful w.h.p.), the queries of Problem 1.7 can be answered deterministically (with no further randomness) in $\tilde{O}(dw)$ time.*

Proof. Our solution uses an auxiliary parameter $r \in [\frac{1}{d}, 1]$ to be chosen later. At preprocessing, we build the data structure of Fact 4.1 and, for each dyadic range $[x \,..\, x+\ell) \subseteq [0 \,..\, |X|)$ of length $\ell \geq \frac{dw}{r}$, the component of Proposition 4.7 for $X[x \,..\, x+\ell)$ and $Y[x-w \,..\, x+w+\ell)$ (with the latter fragment trimmed to fit within Y if necessary) and $\delta = n^{-c-1}$ for a sufficiently large constant c. While doing so, we make sure that all these components use independent randomness. The construction algorithm takes $\tilde{O}(\ell r)$ time for each such dyadic range, which sums up to $\tilde{O}(nr)$ time across all considered ranges.

As a result, for $x \in [0 \,..\, |X|]$, $y \in [0 \,..\, |Y|]$, and $\ell \in [1 \,..\, \min(|X|-x, |Y|-y)]$, if $[x \,..\, x+\ell)$ is a dyadic range and $|y-x| \leq w$, then we can in $\tilde{O}(dw)$ time compute a value $H \in \mathrm{Bin}_{d,n^{-c-1}}(h, r)$, where $h = \mathtt{HD}(X[x \,..\, x+\ell), Y[y \,..\, y+\ell))$. For $\ell < \frac{dw}{r}$, this follows from Fact 4.1, whereas for $\ell \geq \frac{dw}{r}$, this is a consequence of Proposition 4.7. For an arbitrary range $[x \,..\, x+\ell)$, a value $H \in \mathrm{Bin}_{d,n^{-c}}(h, r)$ can be derived by decomposing $[x \,..\, x+\ell)$ into $O(\log n)$ dyadic ranges and combining the results of the individual queries using Observation 4.3.

With an appropriate r, we can then use H to distinguish $\mathtt{HD}(X[x \,..\, x+\ell), Y[y \,..\, y+\ell]) \leq d$ and $\mathtt{HD}(X[x \,..\, x+\ell), Y[y \,..\, y+\ell]) \geq (1+\epsilon)d$. To see this, first suppose that $H \in \mathrm{Bin}_d(h, r)$. If $h \leq d$, then $\Pr[H \geq (1+\frac{\epsilon}{3})rd] \leq \exp(-\frac{1}{27}\epsilon^2 rd)$ holds by the multiplicative Chernoff bound. Moreover, if $h \geq (1+\epsilon)d$, then $\Pr[H < (1+\frac{\epsilon}{3})rd] \leq \Pr[H < (1-\frac{\epsilon}{3})(1+\epsilon)rd] \leq \exp(-\frac{1}{18}\epsilon^2 rd)$ holds by the multiplicative Chernoff bound. Hence, by testing whether $H < (1+\frac{\epsilon}{3})rd$, we can distinguish between $h \leq d$ and $h \geq (1+\epsilon)d$ with failure probability at most $\exp(-\frac{1}{27}\epsilon^2 rd)$. Setting $r = \Theta(\epsilon^{-2} d^{-1} \log n)$, the failure probability can be bounded by n^{-c}.

Since we are only guaranteed that $H \in \mathrm{Bin}_{d,n^{-c}}(h, r)$, the failure probability increases to $2n^{-c}$. Nevertheless, this yields a data structure that can distinguish in $\tilde{O}(dw)$ time between $\mathtt{HD}(X[x \,..\, x+\ell), Y[y \,..\, y+\ell]) \leq d$ and $\mathtt{HD}(X[x \,..\, x+\ell), Y[y \,..\, y+\ell]) \geq (1+\epsilon)d$ correctly with high probability. Since our query algorithm is deterministic, this means that, with high probability, the randomized construction algorithm yields a data structure that produces correct answers for all $O(n^3)$ possible queries. With binary search over ℓ, we derive a data structure answering $\mathsf{LCE}_{d,(1+\epsilon)d}(x, y)$ queries correctly w.h.p. and in $\tilde{O}(dw)$ time. ◀

5 Lower Bound for Computing $\mathtt{ED}_a$ Exactly

The lower bounds provided in this section are conditioned on the Orthogonal Vectors Conjecture [32], which we state below.

▶ **Conjecture 5.1** (Orthogonal Vectors Conjecture [32]). *In the* OV *problem, the input is a set $V \subseteq \{0,1\}^d$ of n vectors, and the goal is to decide if there exist $u, v \in V$ with $\langle u, v\rangle = 0$.*[7]

The Orthogonal Vectors Conjecture asserts that, for every constant $\epsilon > 0$, there exists a constant $c \geq 1$ such that OV *cannot be solved in $O(n^{2-\epsilon})$-time on instances with $d = c \log n$.*

5.1 Unbounded Distance

Here, we use the setting of [12] to prove that the following auxiliary similarity measures are hard to compute exactly. This immediately yields analogous hardness for $\mathtt{ED}_a$.

▶ **Definition 5.2.** *For strings $X, Y \in \Sigma^*$ and an integer $a \in \mathbb{Z}_+$, let $D_a(X,Y) := \mathtt{ED}_a(X,Y) + |Y| - |X|$. Moreover, let $D_a^+(X,Y) := D_a(\$^{|Y|} \cdot X \cdot \$^{|Y|}, Y)$, where $\$ \notin \Sigma$.*

Intuitively, for an edit-distance alignment between strings X and Y, the measure D_a charges a character $Y[y]$ with cost 0 if it is matched, cost $\frac{1}{a}$ if it is substituted, and cost 2 if it is inserted; deletions in X are free. The measure D_a^+ is justified by the following fact, where $\mathrm{alph}(Z) \subseteq \Sigma$ denotes the set of letters occurring in $Z \in \Sigma^*$.

▶ **Fact 5.3.** *Let $X, Y, L, R \in \Sigma^*$. If $|L|, |R| \geq |Y|$ and $\mathrm{alph}(Y) \cap \mathrm{alph}(LR) = \emptyset$, then $D_a^+(X,Y) = D_a(LXR, Y)$.*

Proof. Due to $\mathrm{alph}(Y) \cap \mathrm{alph}(LR) = \emptyset$, we may assume without loss of generality that $L = \$^{|L|}$ and $R = \$^{|R|}$. Any edit-distance alignment between LXR and Y deletes at least $|L| - |Y|$ characters of L and at least $|R| - |Y|$ characters of R, so $D_a(LXR, Y) \geq D_a(\$^{|Y|} \cdot X \cdot \$^{|Y|}, Y)$. On the other hand, we have $\mathtt{ED}_a(LXR, Y) \leq \mathtt{ED}_a(LXR, \$^{|Y|} \cdot X \cdot \$^{|Y|}) + \mathtt{ED}_a(\$^{|Y|} \cdot X \cdot \$^{|Y|}, Y) = |L| - |Y| + |R| - |Y| + \mathtt{ED}_a(\$^{|Y|} \cdot X \cdot \$^{|Y|}, Y)$ by the triangle inequality, so $D_a(LXR) = \mathtt{ED}_a(LXR, Y) + |Y| - |LXR| \leq \mathtt{ED}_a(\$^{|Y|} \cdot X \cdot \$^{|Y|}, Y) + |LR| - 2|Y| + |Y| - |LXR| = \mathtt{ED}_a(\$^{|Y|} \cdot X \cdot \$^{|Y|}, Y) + |Y| - (|X| + 2|Y|) = D_a^+(X,Y)$. ◀

Bringmann and Künnemann [12] developed a generic scheme of reducing the Orthogonal Vectors problem to the problem of computing a given string similarity measure.[8] This framework requires identifying *types* in the input space (which is Σ^* here) and proving that the similarity measure admits an *alignment gadget* and *coordinate values*.

In our construction, the types are of the form $[0\,..\,\sigma)^n$ for $n, \sigma \in \mathbb{Z}_+$ (we assume $\Sigma \subseteq \mathbb{Z}_{\geq 0}$).

▶ **Lemma 5.4.** *The similarity measure D_a^+ admits coordinate values. That is, there exist strings $\mathbf{0}_X, \mathbf{1}_X$ (of some type t_X) and $\mathbf{0}_Y, \mathbf{1}_Y$ (of some type t_Y) such that $D_a^+(\mathbf{1}_X, \mathbf{1}_Y) > D_a^+(\mathbf{0}_X, \mathbf{0}_Y) = D_a^+(\mathbf{0}_X, \mathbf{1}_Y) = D_a^+(\mathbf{1}_X, \mathbf{0}_Y)$.*

Proof. We set $\mathbf{0}_X = 01, \mathbf{1}_X = 00$ (of type $[0\,..\,2)^2$), and $\mathbf{0}_Y = 0, \mathbf{1}_Y = 1$ (of type $[0\,..\,2)^1$). It is easy to check that $\frac{1}{a} = D_a^+(\mathbf{1}_X, \mathbf{1}_Y) > D_a^+(\mathbf{0}_X, \mathbf{0}_Y) = D_a^+(\mathbf{0}_X, \mathbf{1}_Y) = D_a^+(\mathbf{1}_X, \mathbf{0}_Y) = 0$. ◀

An alignment gadget is a pair of functions GA_X and GA_Y parameterized by integers $n \geq m$ and types t_X, t_Y. The function GA_X, given n strings $X_1, \ldots, X_n$ of type t_X, produces a string X of some fixed type (that only depends on n, m, t_X, t_Y), whereas the function GA_Y, given m strings $Y_1, \ldots, Y_m$ of type t_Y, produces a string Y of some (other) fixed type. Both functions must admit $O((n+m)(\ell_X + \ell_Y))$-time algorithms, where ℓ_X is the common length of strings in t_X and ℓ_Y is the common length of strings in t_Y. The main requirement relates

[7] Here, $\langle u, v\rangle = \sum_{i=1}^{d} u_i \cdot v_i$ denotes the scalar product of u and v.

[8] The authors wish to thank Marvin Künnemann for clarifying a few points about this framework.

the value $D_a^+(X,Y)$ to the values $D_a^+(X_i,Y_j)$ for $i \in [1\,..\,n]$ and $j \in [1\,..\,m]$. Specifically, there must be a fixed value C (depending only on n, m, t_X, t_Y) that satisfies the following conditions:

1. Each $\delta \in [0\,..\,n-m]$ satisfies $D_a^+(X,Y) \le C + \sum_{j=1}^{m} D_a^+(X_{j+\delta}, Y_j)$.
2. There is a set $A = \{(i_1,j_1),\dots,(i_k,j_k)\} \subseteq [1\,..\,n] \times [1\,..\,m]$ such that $i_1 < \cdots < i_k$, $j_1 < \cdots < j_k$, and $D_a^+(X,Y) \ge C + \sum_{(i,j)\in A} D_a^+(X_i,Y_j) + (m - |A|)\max_{i,j} D_a^+(X_i,Y_j)$.

▶ **Construction 5.5.** *Let $n \ge m$ be positive integers and $t_X = [0\,..\,\sigma_X)^{\ell_X}$, $t_Y = [0\,..\,\sigma_Y)^{\ell_Y}$ be input types. Denote $\mathtt{A} = \sigma_Y$, $\mathtt{B} = \sigma_Y + 1$, and $\ell = \ell_Y$. We set GA_X and GA_Y so that*

$$\mathsf{GA}_X(X_1,\dots,X_n) := X := \bigodot_{i=1}^{n} \left(\mathtt{A}^{2\ell} \cdot X_i \cdot \mathtt{A}^{\ell} \cdot \mathtt{B}^{\ell}\right), \text{ and}$$

$$\mathsf{GA}_Y(Y_1,\dots,Y_m) := Y := \mathtt{B}^{n\ell} \cdot \bigodot_{j=1}^{m} \left(\mathtt{A}^{\ell} \cdot Y_j \cdot \mathtt{B}^{\ell}\right) \cdot \mathtt{B}^{n\ell},$$

where $\bigodot$ and $\cdot$ denote concatenation.

It is easy to check that both functions can be implemented in $O((n+m)(\ell_X+\ell_Y))$ time and the output types are $[0\,..\,\max(\sigma_X,\sigma_Y+2))^{n(\ell_X+4\ell_Y)}$ and $Y \in [0\,..\,\sigma_Y+2)^{(2n+3m)\ell_Y}$, respectively. In the next two lemmas, we prove that Construction 5.5 satisfies the two aforementioned conditions with $C = \frac{1}{a}(n+m)\ell$.

▶ **Lemma 5.6.** *Each $\delta \in [0\,..\,n-m]$ satisfies $D_a^+(X,Y) \le \frac{1}{a}(n+m)\ell + \sum_{j=1}^{m} D_a^+(X_{j+\delta},Y_j)$.*

Proof. Let us construct an edit distance alignment between $\$^{|Y|} \cdot X \cdot \$^{|Y|}$ and Y so that the total cost charged to characters of Y does not exceed the claimed upper bound.

- The prefix $\mathtt{B}^{(n-\delta)\ell}$ is matched against the prefix $\$^{(n-\delta)\ell}$ (at cost $\frac{1}{a}(n-\delta)\ell$).
- The subsequent δ blocks $\mathtt{B}^{\ell}$ are matched against the $\mathtt{B}^{\ell}$ blocks within the phrases $\mathtt{A}^{2\ell} \cdot X_i \cdot \mathtt{A}^{\ell} \cdot \mathtt{B}^{\ell}$ for $i \in [0\,..\,\delta)$ (at cost 0).
- Each phrase $\mathtt{A}^{\ell} \cdot Y_j \cdot \mathtt{B}^{\ell}$ with $j \in [1\,..\,m]$ is matched against the phrase $\mathtt{A}^{2\ell} \cdot X_{j+\delta} \cdot \mathtt{A}^{\ell} \cdot \mathtt{B}^{\ell}$ so that:
  - The leading block $\mathtt{A}^{\ell}$ is matched against the leading block $\mathtt{A}^{\ell}$ (at cost 0).
  - The string Y_j is matched against $\mathtt{A}^{\ell}X_{j+\delta}\mathtt{A}^{\ell}$ (at cost $D_a^+(X_{j+\delta},Y_j)$ by Fact 5.3).
  - The trailing block $\mathtt{B}^{\ell}$ is matched against the trailing block $\mathtt{B}^{\ell}$ (at cost 0).
- The subsequent $n-m-\delta$ blocks $\mathtt{B}^{\ell}$ are matched against the $\mathtt{B}^{\ell}$ blocks within the phrases $\mathtt{A}^{2\ell} \cdot X_i \cdot \mathtt{A}^{\ell} \cdot \mathtt{B}^{\ell}$ for $i \in (m+\delta\,..\,n]$ (at cost 0).
- The suffix $\mathtt{B}^{(m+\delta)\ell}$ is matched against the suffix $\$^{(m+\delta)\ell}$ (at cost $\frac{1}{a}(m+\delta)\ell$). ◀

▶ **Lemma 5.7.** *There exists a set $A = \{(i_1,j_1),\dots,(i_k,j_k)\} \subseteq [1\,..\,n] \times [1\,..\,m]$ such that $i_1 < \cdots < i_k$, $j_1 < \cdots < j_k$, and $D_a^+(X,Y) \ge \frac{1}{a}(n+m)\ell + \sum_{(i,j)\in A}^{m} D_a^+(X_i,Y_j) + (m - |A|)\max_{i,j} D_a(X_i,Y_j)$.*

Proof. Let us fix an optimal alignment between $\$^{|Y|} \cdot X \cdot \$^{|Y|}$ and Y. We construct a set $A \subseteq [1\,..\,n] \times [1\,..\,m]$ consisting of pairs (i,j) jointly satisfying the following conditions:

- At least one character of Y_j is aligned against a character of X_i.
- No character of Y_j is aligned against any character of $X_{i'}$ for any $i' \ne i$.
- No character of X_i is aligned against any character of $Y_{j'}$ for any $j' < j$.

It is easy to see that A forms a non-crossing matching, i.e., ordering its elements by the first coordinate is equivalent to ordering them by the second coordinate.

Let us analyze the contribution of each letter of Y to $D_a^+(X,Y)$ with respect to any fixed optimal alignment between $\$^{|Y|} \cdot X \cdot \$^{|Y|}$ and Y. Let $c_{\mathtt{B}}$ denote the number of $\mathtt{B}$s in X that are not matched with any $\mathtt{B}$ in Y. Since the number of $\mathtt{B}$s in X and Y is $n\ell$ and $(2n+m)\ell$, respectively, the contribution of $\mathtt{B}$s in Y to the total cost is at least $\frac{1}{a}((n+m)\ell + c_{\mathtt{B}})$.

If $(i,j) \in A$, then Y_j is aligned against a subsequence of X obtained by deleting $X_{i'}$ for all $i' \neq i$ and (possibly) some further characters (recall that all deletions in X are free). In this case, the contribution of Y_j is at least $D_a^+(X_i, Y_j)$ by Fact 5.3.

If no character of Y_j is aligned against any character of any X_i, then all characters of Y_j are deleted or substituted, meaning that the contribution of Y_j is at least $\frac{1}{a}\ell$. If characters of Y_j are aligned against characters of X_i and $X_{i'}$ for two distinct $i < i'$, then all $\mathtt{B}$s between X_i and $X_{i'}$ contribute to $c_{\mathtt{B}}$; this contribution is at least ℓ, and it cannot be charged to any $Y_{j'}$ with $j' \neq j$. Finally, if characters of both $Y_{j'}$ and Y_j (with $j' < j$) are aligned to characters of the same X_i, then the block $\mathtt{A}^\ell$ preceding Y_j contributes at least $\frac{1}{a}\ell$. Overall, we conclude that the total cost is at least $\frac{1}{a}\ell$ for each $j \in [1\,..\,m]$ with no $(i,j) \in A$ (this contribution is either directly charged to $\mathtt{A}^\ell Y_j$ or via $c_{\mathtt{B}}$). Hence, the total cost $D_a^+(X,Y)$ is at least $\frac{1}{a}(n + 2m - |A|)\ell + \sum_{(i,j)\in A} D_a^+(X_i, Y_j)$. Due to $D_a^+(X_i, Y_j) \leq \frac{1}{a}|Y_j| = \frac{1}{a}\ell$, this yields the claimed lower bound. ◀

Consequently, the framework of [12] yields the following result. (An inspection of [12] reveals that the strings are produced from the coordinate values composed by recursive application of the alignment gadget. The tree of this recursion is of height 3, so the output strings are over an alphabet of size 8.)

▶ **Corollary 5.8.** *There is an $O(nd)$-time algorithm that, given two sets $U, V \subseteq \{0,1\}^d$ of n vectors each and an integer $a \in \mathbb{Z}_+$, constructs strings $X, Y \in [0\,..\,8)^*$ and a threshold k such that $D_a^+(X,Y) \leq k$ if and only if $\langle u, v\rangle = 0$ for some $u \in U$ and $v \in V$. Moreover, $|X|$, $|Y|$, and k depend only on n, d, and a.*

5.2 Bounded Distance

Our next goal is to derive a counterpart of Corollary 5.8 with $k \ll |X| + |Y|$ and $\mathtt{ED}_a$ instead of D_a^+. The following construction is at the heart of our reduction.

▶ **Construction 5.9.** *Let n, ℓ_X, ℓ_Y be positive integers and let $\ell = \ell_X + \ell_Y$. For a sequence of n pairs $(X_i, Y_i) \in \Sigma^{\ell_X} \times \Sigma^{\ell_Y}$, we define strings*

$$X = \mathtt{A}^{\ell_Y} \cdot \bigodot_{j=1}^{n} (\mathtt{B}^\ell \cdot X_j \cdot \mathtt{C}^\ell \cdot \mathtt{A}^{\ell_Y}) \qquad \text{and} \qquad Y = Y_1 \cdot \bigodot_{j=2}^{n} (\mathtt{B}^\ell \cdot \mathtt{D}^{\ell_X} \cdot \mathtt{C}^\ell \cdot Y_j).$$

▶ **Lemma 5.10.** *If $a \geq n$, then the strings X, Y of Construction 5.9 satisfy*

$$D_a(X,Y) = \tfrac{(n-1)\ell}{a} + \min_{i=1}^{n} D_a^+(X_i, Y_i).$$

Proof. First, we prove an upper bound on $D_a(X,Y)$. Let us fix $i \in [1\,..\,n]$. We define an edit distance alignment between

$$X = \bigodot_{j=1}^{i-1} (\mathtt{A}^{\ell_Y} \cdot \mathtt{B}^\ell \cdot X_j \cdot \mathtt{C}^\ell) \cdot \mathtt{A}^{\ell_Y} \cdot \mathtt{B}^\ell \cdot X_i \cdot \mathtt{C}^\ell \cdot \mathtt{A}^{\ell_Y} \cdot \bigodot_{j=i+1}^{n} (\mathtt{B}^\ell \cdot X_j \cdot \mathtt{C}^\ell \cdot \mathtt{A}^{\ell_Y})$$

and

$$Y = \bigodot_{j=1}^{i-1}(Y_j \cdot \mathtt{B}^\ell \cdot \mathtt{D}^{\ell_X} \cdot \mathtt{C}^\ell) \cdot Y_i \cdot \bigodot_{j=i+1}^{n}(\mathtt{B}^\ell \cdot \mathtt{D}^{\ell_X} \cdot \mathtt{C}^\ell \cdot Y_j)$$

so that the total cost charged to characters of Y does not exceed $\frac{(n-1)\ell}{a} + D_a^+(X_i, Y_i)$.

- For $j \in [1\,..\,i)$, the phrase $Y_j \cdot \mathtt{B}^\ell \cdot \mathtt{D}^{\ell_X} \cdot \mathtt{C}^\ell$ is aligned with the phrase $\mathtt{A}^{\ell_Y} \cdot \mathtt{B}^\ell \cdot X_j \cdot \mathtt{C}^\ell$ using substitutions only (at cost $\frac{1}{a}(\ell_X + \ell_Y)$).
- The block Y_i is aligned against $\mathtt{A}^{\ell_Y} \cdot B^\ell \cdot X_i \cdot \mathtt{C}^\ell \cdot \mathtt{A}^{\ell_Y}$ (at cost $D_a^+(X_i, Y_i)$ by Fact 5.3).
- For $j \in (i\,..\,n]$, the phrase $(\mathtt{B}^\ell \cdot \mathtt{D}^{\ell_X} \cdot \mathtt{C}^\ell \cdot Y_j)$ is aligned with the phrase $\mathtt{B}^\ell \cdot X_j \cdot \mathtt{C}^\ell \cdot \mathtt{A}^{\ell_Y}$ using substitutions only (at cost $\frac{1}{a}(\ell_X + \ell_Y)$).

It remains to prove the lower bound on $D_a(X, Y)$. Let us fix an arbitrary alignment of X and Y; we shall prove that its cost is at least as large as the claimed lower bound on $D_a(X, Y)$. First, suppose that, for some $i \neq j$, a character of X_i is aligned against a character of Y_j. Note that $X_i = X[(3i-2)\ell + \ell_Y\,..\,(3i-1)\ell)$ and $Y_j = Y[(3j-3)\ell\,..\,(3j-3)\ell + \ell_Y)$. If $i < j$, then aligning a character of X_i against a character of Y_j requires deleting at least $(3j-3)\ell - (3i-1)\ell = (3j-3i-2)\ell \geq \ell$ characters in the prefix $Y[0\,..\,(3j-3)\ell)$. Similarly, if $i > j$, then aligning a character of X_i against a character of Y_j requires deleting at least $(|Y| - ((3j-3)\ell + \ell_Y)) - ((|X| - (3i-2)\ell + \ell_Y)) = (3(n-j)\ell - (3n-3i+2)\ell) = (3i-3j-2)\ell \geq \ell$ characters in the suffix of $Y[(3j-3)\ell + \ell_Y\,..\,|Y|)$. In either case, the alignment cost is at least $\ell \geq \frac{n\ell}{a} \geq \frac{(n-1)\ell}{a} + \frac{1}{a}\ell_Y \geq \frac{(n-1)\ell}{a} + \min_{i=1}^n D_a^+(X_i, Y_i)$.

Thus, we may assume that no character of Y_j is aligned against a character of X_i for $i \neq j$. In this case, we bound from below the total cost charged to characters of Y. Note that the symbols $\mathtt{D}$ in Y contribute at least $\frac{n-1}{a}\ell_X$ (because there are no $\mathtt{D}$s in X). If, for some $i \in [1\,..\,n]$, no character of Y_i is aligned against a character of X_i, then Y_i is charged at least $\frac{1}{a}\ell_Y$. Otherwise, Y_i is charged at least $D_a^+(X_i, Y_i)$ by Fact 5.3. However, if $i < j$ are subsequent indices such that a character of Y_i is aligned against a character of X_i and a character of Y_j is aligned against a character X_j then, among the $2\ell(j-i)$ characters $\mathtt{B}$ and $\mathtt{C}$ between Y_i and Y_j, at most $\mathtt{LCS}((\mathtt{B}^\ell\mathtt{C}^\ell)^{j-i}, (\mathtt{C}^\ell\mathtt{B}^\ell)^{j-i}) = \ell(2(j-i)-1)$ are matched, incurring a cost of at least $\frac{1}{a}\ell > \frac{1}{a}\ell_Y$ for the mismatched characters. Overall, $\mathtt{D}$s contribute at least $\frac{n-1}{a}\ell_X$, the smallest i such that a character of X_i is aligned against a character of Y_i (if there is any) contributes at least $D_a^+(X_i, Y_i)$, each of the remaining $i \in [1\,..\,n]$ contributes at least $\frac{1}{a}\ell_Y$ (either directly or via $\mathtt{B}$s and $\mathtt{C}$s), for a total of $\frac{(n-1)\ell}{a} + \min_{i=1}^n D_a^+(X_i, Y_i)$. ◀

These properties let us use Construction 5.9 to generalize Corollary 5.8 and finally prove Theorem 1.9.

▶ **Proposition 5.11.** *There is an $O(nmd)$-time algorithm that, given two sets $U, V \subseteq \{0,1\}^d$ of n vectors each and integers $a \in \mathbb{Z}_+$, $m \in [1\,..\,n]$, constructs strings $X, Y \in [0\,..\,12)^*$ and a threshold $K = O(\max(\frac{m}{a}, \frac{1}{m})nd)$ such that $\mathtt{ED}_a(X, Y) \leq K$ if and only if $\langle u, v\rangle = 0$ for some $u \in U$ and $v \in V$.*

Proof. We cover U and V with subsets $U_1, \ldots, U_m \subseteq U$ and $V_1, \ldots, V_m \subseteq V$ of size $\lceil \frac{n}{m} \rceil$. For any $i, j \in [1\,..\,m]$, we construct an instance $(X_{i,j}, Y_{i,j}, k_{i,j})$ using Corollary 5.8 for $U_i, V_j \subseteq \{0,1\}^d$. Note that $|X_{i,j}| = \ell_X$, $|Y_{i,j}| = \ell_Y$, and $k_{i,j}$ depend only on n, m, and d. We then apply Construction 5.9 to the collection of m^2 pairs $(X_{i,j}, Y_{i,j})$, resulting in strings X and Y. The running time of this construction is $O(m^2(\ell_X + \ell_Y)) = O(m^2 \cdot \lceil \frac{n}{m} \rceil d) = O(mnd)$. Moreover, we set $k = \frac{m^2-1}{a}(\ell_X + \ell_Y) + k_{i,j}$ (recall that $k_{i,j}$ is the same for all $i, j \in [1\,..\,m]$); this value satisfies $k \leq \frac{m^2}{a}(\ell_X + \ell_Y) = O(a^{-1}mnd)$. If $\langle u, v\rangle = 0$ for some $u \in U$ and $v \in V$,

then Corollary 5.8 implies that $D_a^+(X_{i,j}, Y_{i,j}) \le k_{i,j}$ for $i, j \in [1\,..\,m]$ such that $u \in U_i$ and $v \in V_j$. Consequently, $D_a(X, Y) \le k$ by Lemma 5.10. Symmetrically, if $D_a(X, Y) \le k$, then, by Lemma 5.10, $D_a^+(X_{i,j}, Y_{i,j}) \le k_{i,j}$ holds for some $i, j \in [1\,..\,m]$ and, by Corollary 5.8, $\langle u, v\rangle = 0$ for some $u \in U_i \subseteq U$ and $v \in V_j \subseteq V$. The threshold k for $D_a(X, Y)$ translates to a threshold $K = k + |X| - |Y| = k + O(\ell) = O(\max(a^{-1}mnd, nd/m))$ for $\mathtt{ED}_a$. ◀

▶ **Theorem 1.9.** *Consider sequences $(a_n)_{n=1}^{\infty}$ and $(k_n)_{n=1}^{\infty}$ with entries $a_n, k_n \in [1\,..\,n]$ computable in $\mathrm{poly}(n)$ time. Unless the Orthogonal Vectors Conjecture fails, there is no algorithm that, for some fixed $\epsilon > 0$, every $n \in \mathbb{Z}_+$, and all strings X, Y with $|X| + |Y| \le n$, in $O((n + k_n \cdot \min(n, a_n k_n))^{1-\epsilon})$ time computes $\mathtt{ED}_{a_n}(X, Y)$ or reports that $\mathtt{ED}_{a_n}(X, Y) > k_n$.*

Proof. For each $n \in \mathbb{Z}_+$, let $t_n = \min(n, a_n k_n)k_n$. Note that $\Omega(n)$ time is already needed to check whether $X = Y$. Thus, the claim holds trivially if there are infinitely many pairs (a_n, k_n) with $t_n^{1-\epsilon} < n$. Consequently, we may assume without loss of generality that $t_n^{1-\epsilon} \ge n$ for each n (any finite prefix of the sequence (a_n, k_n) is irrelevant).

Let us choose a constant c' so that (a_n, k_n) can be constructed in $O(n^{c'})$ time, a constant c of Conjecture 5.1 for $\epsilon/(2c')$, and a constant C so that Proposition 5.11 guarantees $|X| + |Y| \le Cnmd$ and $K \le C(\max(m/a, 1/m)nd)$.

Given an instance $V \subseteq \{0,1\}^d$ of the Orthogonal Vectors problem with $d = c \log |V|$, we set $n := \lceil |V|^{1/c'} \rceil$ and compute the values a_n, k_n, and t_n in $O(n^c) = O(|V|)$ time. Next, we set $N := \lfloor \sqrt{t_n}/(Cd) \rfloor$ and cover V with $v := \lceil \frac{1}{N}|V| \rceil$ subsets $V_1, \ldots, V_v$ of size N. For any $i, j \in [1\,..\,v]$, we construct an instance $(X_{i,j}, Y_{i,j}, K_{i,j})$ using Proposition 5.11 for $V_i, V_j \subseteq \{0,1\}^d$, a_n, and $m_n := \lfloor \sqrt{\min(n/k_n, a_n)} \rfloor$.

Note that $|X_{i,j}| + |Y_{i,j}| \le Cm_nNd \le \sqrt{\min(n/k_n, a_n)t_n} \le \sqrt{n/k_n \cdot nk_n} \le n$ and $K_{i,j} \le C \cdot \max(m_n/a_n, 1/m_n)Nd \le \max(m_n/a_n, 1/m_n)\sqrt{t_n}$. If $a_n k_n \le n$, then $K_{i,j} \le \max(\sqrt{a_n}/a_n, 1/\sqrt{a_n}) \cdot \sqrt{a_n k_n^2} = k_n$. If $a_n k_n \ge n$, on the other hand, then $K_{i,j} \le \max(\sqrt{n/k_n}/a_n, \sqrt{k_n/n}) \cdot \sqrt{nk_n} \le \max(n/a_n, k_n) \le k_n$. Thus, we can test $\mathtt{ED}_a(X_{i,j}, Y_{i,j}) \le K_{i,j}$ using an instance $(X_{i,j}, Y_{i,j}, a_n, k_n)$ of the problem in question. By our hypothesis, this costs $O(n + t_n^{1-\epsilon}) = O(t_n^{1-\epsilon})$ time (including construction of Proposition 5.11). Summing up over $i, j \in [1\,..\,v]$, the running time is $O(v^2 t_n^{1-\epsilon}) = O(|V|^2/N^2 \cdot t_n^{1-\epsilon}) = O(d^2|V|^2 t_n^{-\epsilon}) = O(d^2|V|^2 n^{-\epsilon}) = O(d^2|V|^{2-\epsilon/c'}) = O(|V|^{2-\epsilon/(2c')})$. Overall, including the construction of (a_n, k_n) in $O(|V|)$ time, the total time complexity of solving the OV instance is $O(|V|^{2-\epsilon/(2c')})$, contradicting Conjecture 5.1. ◀

References

1 Srinivas Aluru, editor. *Handbook of Computational Molecular Biology*. Chapman and Hall/CRC, December 2005. doi:10.1201/9781420036275.

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 Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In *61st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020. doi:10.1109/FOCS46700.2020.00096.

4 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.

5 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.

6 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.

7 Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *35th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2003*, pages 316–324. ACM, 2003. doi:10.1145/780542.780590.

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

9 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 685–698. ACM, 2020. doi:10.1145/3357713.3384282.

10 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. Almost-optimal sublinear-time edit distance in the low distance regime. In *54th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2022*, pages 1102–1115. ACM, 2022. doi:10.1145/3519935.3519990.

11 Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. Improved sublinear-time edit distance for preprocessed strings. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 32:1–32:20. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.32.

12 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *56th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2015*, pages 79–97. IEEE, 2015. doi:10.1109/focs.2015.15.

13 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *Journal of the ACM*, 67(6):36:1–36:22, 2020. doi:10.1145/3422823.

14 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern Hamming distances. In *52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 643–656. ACM, 2020. doi:10.1145/3357713.3384266.

15 Jian-Qun Chen, Ying Wu, Haiwang Yang, Joy Bergelson, Martin Kreitman, and Dacheng Tian. Variation in the ratio of nucleotide substitution and indel rates across genomes in mammals and bacteria. *Molecular Biology and Evolution*, 26(7):1523–1531, 2009. doi:10.1093/molbev/msp063.

16 Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *Journal of the ACM*, 47(6):987–1011, 2000. doi:10.1145/355541.355547.

17 Elazar Goldenberg, Tomasz Kociumaka, Robert Krauthgamer, and Barna Saha. Gap edit distance via non-adaptive queries: Simple and optimal. In *63rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2022*, pages 674–685. IEEE, 2022. doi:10.1109/FOCS54457.2022.00070.

18 Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. Sublinear algorithms for gap edit distance. In *60th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2019*, pages 1101–1120. IEEE, 2019. doi:10.1109/FOCS.2019.00070.

19 Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. Does preprocessing help in fast sequence comparisons? In *52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 657–670. ACM, 2020. doi:10.1145/3357713.3384300.

20 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

21 Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012. doi:10.4086/toc.2012.v008a014.

22 John A. Hawkins, Stephen K. Jones, Ilya J. Finkelstein, and William H. Press. Indel-correcting DNA barcodes for high-throughput sequencing. *Proceedings of the National Academy of Sciences*, 115(27):E6217–E6226, 2018. doi:10.1073/pnas.1802640115.

23 Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009. URL: https://www.worldcat.org/oclc/315913020.

24 Tomasz Kociumaka and Barna Saha. Sublinear-time algorithms for computing & embedding gap edit distance. In *61st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2020*, pages 1168–1179. IEEE, 2020. doi:10.1109/focs46700.2020.00112.

25 Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 699–712. ACM, 2020. doi:10.1145/3357713.3384307.

26 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.

27 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.

28 Paul Medvedev. Theoretical analysis of edit distance algorithms: an applied perspective, 2022. doi:10.48550/arXiv.2204.09535.

29 Julienne M. Mullaney, Ryan E. Mills, W. Stephen Pittard, and Scott E. Devine. Small insertions and deletions (INDELs) in human genomes. *Human Molecular Genetics*, 19(R2):R131–R136, 2010. doi:10.1093/hmg/ddq400.

30 Kerstin Neininger, Tobias Marschall, and Volkhard Helms. SNP and indel frequencies at transcription start sites and at canonical and alternative translation initiation sites in the human genome. *PLOS ONE*, 14(4):1–21, 2019. doi:10.1371/journal.pone.0214816.

31 Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985. doi:10.1016/S0019-9958(85)80046-2.

32 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

33 Zhaolei Zhang and Mark Gerstein. Patterns of nucleotide substitution, insertion and deletion in the human genome inferred from pseudogenes. *Nucleic Acids Research*, 31(18):5338–5348, 2003. doi:10.1093/nar/gkg745.