On Estimating Edit Distance: Alignment, Dimension Reduction, and Embeddings

Moses Charikar[1]
Department of Computer Science, Stanford University, Stanford, CA, USA
moses@cs.stanford.edu

Ofir Geri[2]
Department of Computer Science, Stanford University, Stanford, CA, USA
ofirgeri@cs.stanford.edu

Michael P. Kim[3]
Department of Computer Science, Stanford University, Stanford, CA, USA
mpk@cs.stanford.edu

William Kuszmaul
Department of Computer Science, Stanford University, Stanford, CA, USA
kuszmaul@cs.stanford.edu

Abstract

Edit distance is a fundamental measure of distance between strings and has been widely studied in computer science. While the problem of estimating edit distance has been studied extensively, the equally important question of actually producing an alignment (i.e., the sequence of edits) has received far less attention. Somewhat surprisingly, we show that any algorithm to estimate edit distance can be used in a black-box fashion to produce an approximate alignment of strings, with modest loss in approximation factor and small loss in run time. Plugging in the result of Andoni, Krauthgamer, and Onak, we obtain an alignment that is a $(\log n)^{O(1/\varepsilon^2)}$ approximation in time $\tilde{O}(n^{1+\varepsilon})$.

Closely related to the study of approximation algorithms is the study of metric embeddings for edit distance. We show that min-hash techniques can be useful in designing edit distance embeddings through three results: (1) An embedding from Ulam distance (edit distance over permutations) to Hamming space that matches the best known distortion of $O(\log n)$ and also implicitly encodes a sequence of edits between the strings; (2) In the case where the edit distance between the input strings is known to have an upper bound K, we show that embeddings of edit distance into Hamming space with distortion $f(n)$ can be modified in a black-box fashion to give distortion $O(f(\text{poly}(K)))$ for a class of periodic-free strings; (3) A randomized dimension-reduction map with contraction c and asymptotically optimal expected distortion $O(c)$, improving on the previous $\tilde{O}(c^{1+2/\log\log\log n})$ distortion result of Batu, Ergun, and Sahinalp.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases edit distance, alignment, approximation algorithms, embedding, dimension reduction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.34

Related Version A full version is available at `https://arxiv.org/abs/1804.09907`.

[1] Supported by NSF grant CCF-1617577 and a Simons Investigator Award.
[2] Supported by NSF grant CCF-1617577, a Simons Investigator Award, and the Google Graduate Fellowship in Computer Science in the School of Engineering at Stanford University.
[3] Supported by NSF grant CCF-1763299.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 34; pp. 34:1–34:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *edit distance* $\Delta_{\text{ed}}(x, y)$ between two strings x and y is the minimum number of character insertions, deletions, and substitutions needed to transform x into y. This is a fundamental distance measure on strings, extensively studied in computer science [2–5, 9, 11, 17, 21, 24]. Edit distance has applications in areas including computational biology, signal processing, handwriting recognition, and image compression [19]. One of its oldest and most important uses is as a tool for comparing differences between genetic sequences [1, 19, 20].

The textbook dynamic-programming algorithm for edit distance runs in time $O(n^2)$ [20, 23, 24], and can be leveraged to recover a sequence of edits, also known as an *alignment*. The quadratic run time is prohibitively large for massive datasets (e.g., genomic data), and conditional lower bounds suggest that no strongly subquadratic time algorithm exists [5].

The difficulty of computing edit distance has motivated the development of fast heuristics [1, 10, 14, 19]. On the theoretical side, the tradeoff between run time and approximation factor (or distortion) is an important question (see [15, Section 6], and [16, Section 8.3.2]). Andoni and Onak [4] (building on beautiful work of Ostrovsky and Rabani [21]) gave an algorithm that estimates edit distance within a factor of $2^{\tilde{O}(\sqrt{\log n})}$ in time $n^{1+o(1)}$. The current best known tradeoff was obtained by Andoni, Krauthgamer and Onak [3], who gave an algorithm that estimates edit distance to within factor $(\log n)^{O(1/\varepsilon)}$ with run time $O(n^{1+\varepsilon})$.

Alignment Recovery

While these algorithms produce estimates of edit distance, they do not produce an alignment between strings (i.e., a sequence of edits). By decoupling the problem of numerical estimation from the problem of alignment recovery, the authors of [4] and [3] are able to exploit techniques such as metric space embeddings[4] and random sampling in order to obtain better approximations. The algorithm of [3] runs in phases, with the i-th phase distinguishing between whether $\Delta_{\text{ed}}(x, y)$ is greater than or significantly smaller than $\frac{n}{2^i}$. At the beginning of each phase, a nuanced random process is used to select a small fraction of the positions in x, and then the entire phase is performed while examining only those positions. In total, the full algorithm samples an $\tilde{O}\left(\frac{n^{\varepsilon}}{\Delta_{\text{ed}}(x,y)}\right)$-fraction of the letters in x. Note that this is a polynomially small fraction as we are interested in the case where $\Delta_{\text{ed}}(x, y) > n^{1/2}$ (if the edit distance is small, we can run the algorithm of Landau et al. [18] in linear time). Given that the algorithm only views a small portion of the positions in x, it is not clear how to recover a global alignment between the two strings.

We show, somewhat surprisingly, that any edit distance estimator can be turned into an approximate aligner in a black box fashion with modest loss in approximation factor and small loss in run time. For example, plugging the result of [3] into our framework, we get an algorithm with distortion $(\log n)^{O(1/\varepsilon^2)}$ and run time $\tilde{O}(n^{1+\varepsilon})$. To the best of our knowledge, the best previous result that gave an approximate alignment was the work of Batu, Ergun, and Sahinalp [7], which has distortion that is polynomial in n.

[4] The algorithm of [4] has a recursive structure in which at each level of recursion, every substring α of some length l is assigned a vector v_α such that the ℓ_1 distance between vectors closely approximates edit distance between substrings. The vectors at each level of recursion are constructed from the vectors in lower levels through a series of procedures culminating in an application of Bourgain's embedding to a sparse graph metric. As a result, although the distances between vectors in the top level of recursion allow for a numerical estimation of edit distance, it is not immediately clear how one might attempt to extract additional information from the vectors in order to recover an alignment.

Embeddings of Edit Distance Using Min-Hash Techniques

The study of approximation algorithms for edit distance closely relates to the study of embeddings [4, 7, 21]. An *embedding* from a metric space M_1 to a metric space M_2 is a map of points in M_1 to M_2 such that distances are preserved up to some factor D, known as the *distortion*. Loosely speaking, low-distortion embeddings from a complex metric space M_1 to a simpler metric space M_2 allow algorithm designers to focus on the simpler metric space, rather than directly handling the more complex one. Embeddings from edit distance to Hamming space have been widely studied [8, 9, 11, 21] and have played pivotal roles in the development of approximation algorithms [4] and streaming algorithms [8, 9].

The second contribution of this paper is to introduce new algorithms for three problems related to embeddings for edit distance. The algorithms are unified by the use of min-hash techniques to select pivots in strings. We find this technique to be particularly useful for edit distance because hashing the content of strings allows us to split strings in places where their content is aligned, thereby getting around the problem of insertions and deletions misaligning the strings. In several of our results, this allows us to obtain algorithms which are either more intuitive or simpler than their predecessors. The three results are summarized below.

Efficiently Embedding the Ulam Metric into Hamming Space: For the special case of the Ulam metric (edit distance on permutations), we present a randomized embedding ϕ of permutations of size n to poly(n)-dimensional Hamming space with distortion $O(\log n)$. Given strings x and y, the Hamming differences between $\phi(x)$ and $\phi(y)$ not only approximate the edit distance between x and y, but also implicitly encode a sequence of edits from x to y. If the output string of our embedding is stored using a sparse vector representation, then the embedding can be computed in linear time, and its output can be stored in linear space. The logarithmic distortion matches that of Charikar and Krauthgamer's embedding into ℓ_1-space [11], which did not encode the actual edits and needed quadratic time and number of dimensions. Our embedding also supports efficient updates, and can be modified to reflect an edit in expected time $O(\log n)$ (as opposed to the deterministic linear time of [11]).

Embedding Edit Distance in the Low-Distance Regime: Recently, there has been considerable attention devoted to edit distance in the low-distance regime [8, 9]. In this regime, we are interested in finding algorithms that run faster or perform better given the promise that the edit distance between the input strings is small. This regime is of considerable interest from the practical point of view. Landau, Myers and Schmidt [18] gave an exact algorithm for strings with edit distance K that runs in time $O(n+K^2)$. Recently, Chakraborty, Goldenberg and Koucký [9] gave a randomized embedding of edit distance into Hamming space that has distortion linear in the edit distance with probability at least $2/3$.

Given an embedding with distortion $\gamma(n)$ (a function of the input size), could one obtain an embedding whose distortion is a function of K, the edit distance, instead of n? We answer this question in the affirmative for the class of (D, R)-periodic free strings. We say that a string is (D, R)-periodic free if none of its substrings of length D are periodic with period of at most R. For $D \in$ poly(K) and $R = O(K^3)$, we show that the embedding of Ostrovksy and Rabani [21] can be used in a black-box fashion to obtain an embedding with distortion $2^{O\left(\sqrt{\log K \log\log K}\right)}$ for (D, R)-periodic free strings with edit distance of at most K. Our result can be seen as building on the min-hash techniques of [11, Section 3.5] (which in turn extends ideas from [6]). The authors of [11] give an embedding for $(t, 180tK)$-non-repetitive strings with distortion $O(t \log(tK))$ [11]. The key difference is that our notion of (D, R)-periodic free is much less restrictive than the notion of non-repetitive strings studied in [11].

Optimal Dimension Reduction for Edit Distance: The aforementioned work of Batu et al. [7] introduced and studied an interesting notion of dimension reduction for edit distance: An embedding of edit distance on length-n strings to edit distance on length-n/c strings (with larger alphabet size) is called a *dimension-reduction map* with *contraction* c. By first performing dimension reduction, one can then apply inefficient algorithms to the contracted strings at a relatively small overall cost. This idea was used in [7] to design an approximation algorithm with approximation factor $O(n^{1/3+o(1)})$. We provide a dimension-reduction map with contraction c and asymptotically optimal expected distortion $O(c)$, improving on the distortion of $\tilde{O}(c^{1+2/\log\log\log n})$ obtained by the deterministic map of [7].[5]

2 Preliminaries

Throughout the paper, we will use Σ to denote an alphabet,[6] and Σ^n to denote the set of words of length n over that alphabet. Additionally, we use $\mathcal{P}_n$ to denote the set of permutations of length n over Σ, or equivalently, the subset of Σ^n containing words whose letters are distinct. Given a string w of length n, we denote its letters by $w_1, w_2, \ldots, w_n$, and we use $w[i:j]$ to denote the substring $w_i w_{i+1} \cdots w_j$ (which is empty if $j < i$).

An *edit operation* is either an insertion, a deletion, or a substitution of a letter with another letter in a word. Given words x and y, an *alignment* from x to y is a sequence of edits transforming x to y. The *edit distance* $\Delta_{\text{ed}}(x, y)$ is the minimum number of edits needed to transform x to y. Alternatively, it is the length of an optimal alignment.

For convenience, we make the Simple Uniform Hashing Assumption [12], which assumes access to a fully independent family $\mathcal{H}$ of hash functions mapping $\Theta(\log n)$ bits to $\Theta(\log n)$ bits with constant time evaluation. For our applications, this can be simulated using the family of Pagh and Pagh [22], which is independent on any given set of size n with high probability. The family can be constructed in linear time and uses $O(n \log n)$ random bits.

3 Alignment Recovery Using a Black-Box Approximation Algorithm

In this section, we show how to transform a black-box edit distance approximation algorithm $\mathcal{A}$ into an approximate alignment algorithm $\mathcal{B}$. The algorithm $\mathcal{B}$ appears here as Algorithm 1. In the description of the algorithm, we rely on the following definition of a partition.

▶ **Definition 3.1.** A *partition* of a string u into m parts is a tuple $P = (p_0, p_1, p_2, \ldots, p_m)$ such that $p_0 = 0$, $p_m = |u|$, and $p_0 \le p_1 \le \cdots \le p_m$. For $i \in \{1, \ldots, m\}$, the *i-th part of P* is the subword $P_i := u[p_{i-1} + 1 : p_i]$, which is empty if $p_i = p_{i-1}$. A partition of a string u into m parts is an *equipartition* if each of the parts is of size either $\lfloor |u|/m \rfloor$ or $\lceil |u|/m \rceil$.

Formally, we assume that the approximation algorithm $\mathcal{A}$ has the following properties:

1. There is some non-decreasing function γ such that for all $n > 0$, and for any two strings u, v with $|u| + |v| \le n$, $\Delta_{\text{ed}}(u, v) \le \mathcal{A}(u, v) \le \gamma(n) \cdot \Delta_{\text{ed}}(u, v)$.
2. $\mathcal{A}(u, v)$ runs in time at most $T(n)$ for some non-decreasing function T which is super-additive in the sense that $T(j) + T(k) \le T(j + k)$ for $j, k \ge 0$.

We are now ready to state the main theorem of this section.

[5] When comparing these distortions, one should note that $2/\log\log\log n$ goes to zero very slowly; in particular, $c^{1+2/\log\log\log n} \ge c^{1.66}$ for all $n \le 10^{82}$, the number of atoms in the universe.

[6] We assume that characters in Σ can be represented in $\Theta(\log n)$ bits, where n is the size of input strings.

Algorithm 1 Black-Box Approximate Alignment Algorithm.

Input: Strings u, v with $|u| + |v| \leq n$.
Parameters: $m \in \mathbb{N}$ satisfying $m \geq 2$ and an approximation algorithm $\mathcal{A}$ for edit distance.

1. If $|u| \leq 1$, then find an optimal alignment in time $O(|v|)$ naively.
2. Let $P = (p_0, p_1, \ldots, p_m)$ be an equipartition of u.
3. Let S consist of the positions in v which can be reached by adding or subtracting a power of $(1 + \frac{1}{m})$ to some p_i. Formally, define

$$S = \left(\{p_0, \ldots, p_m\} \cup \{|v|\} \cup \left\{ \left\lceil p_i \pm \left(1 + \frac{1}{m}\right)^j \right\rceil \mid i, j \geq 0 \right\} \right) \cap \{0, \ldots, |v|\}.$$

4. Using dynamic programming, find a partition $Q = (q_0, \ldots, q_m)$ of v such that each q_i is in S, and such that the cost $\sum_{i=1}^{m} \mathcal{A}(P_i, Q_i)$ is minimized:
   a. For $l \in S$, let $f(l, j)$ be the subproblem of returning a choice of $q_0, q_1, \ldots, q_j$ with $q_j = l$ which minimizes $\sum_{i=1}^{j} \mathcal{A}(P_i, Q_i)$.
   b. Solve $f(l, j)$ by examining precomputed answers for subproblems of the form $f(l', j-1)$ with $l' \leq l \in S$: if $f(l', j-1)$ gives a choice of $q_0, q_1, \ldots, q_{j-1}$ with $\sum_{i=1}^{j-1} \mathcal{A}(P_i, Q_i) = t$, then we can set $q_j = l$ to get $\sum_{i=1}^{j} \mathcal{A}(P_i, Q_i) = t + \mathcal{A}(P_j, v[l' + 1 : l])$.
   (Here, $\mathcal{A}(P_j, v[l' + 1 : l])$ is computed using $\mathcal{A}$.)
5. Recurse on each pair (P_i, Q_i). Combine the resulting alignments between each P_i and Q_i to obtain an alignment between u and v.

▶ **Theorem 3.2.** *For all u, v with $|u| + |v| \leq n$ and $m \geq 2$, Algorithm 1 outputs an alignment from u to v of size at most $(3\gamma(n))^{O(\log_m n)} \cdot \Delta_{ed}(u, v)$. Moreover, the run time is $\tilde{O}(m^5 \cdot T(n))$.*

Before continuing, we provide a brief discussion of Algorithm 1. The algorithm first breaks u into a partition P of m equal parts. It then uses the black-box algorithm $\mathcal{A}$ to search for a partition Q of v such that $\sum_i \Delta_{\text{ed}}(P_i, Q_i)$ is near minimal; after finding such a Q, the algorithm recurses to find approximate alignments between P_i and Q_i for each i. Rather than considering every option for the partition $Q = (q_0, \ldots, q_m)$, the algorithm limits itself to those for which each q_i comes from a relatively small set S.

The set S is carefully designed so that although it is small, any optimal partition Q^{opt} of v can be in some sense well approximated by some partition Q using only q_i values from S. This limits the multiplicative error introduced at each level of recursion to be bounded by $3\gamma(n)$; across the $O(\log_m n)$ level of recursion, the total multiplicative error becomes $3\gamma(n)^{O(\log_m n)}$. The fact that the recursion depth appears in the exponent of the multiplicative error is why we partition u and v into many parts at each level.

Next we discuss several implications of Theorem 3.2. The parameter m allows us to trade off the approximation factor and the run time of the algorithm. When taken to the extreme, this gives two particularly interesting results.

▶ **Corollary 3.3.** *Let $0 < \varepsilon < 1$ (not necessarily constant). Then m can be chosen so that Algorithm 1 has approximation ratio $(3\gamma(n))^{O(\frac{1}{\varepsilon})}$ and run time $\tilde{O}\left(T(n) \cdot n^{\varepsilon}\right)$.*

▶ **Corollary 3.4.** *Let $0 < \varepsilon < 1$ (not necessarily constant). Then m can be chosen so that Algorithm 1 has approximation ratio $n^{O(\varepsilon)}$ and run time $\tilde{O}(T(n)) \cdot (3\gamma(n))^{O(1/\varepsilon)}$.*

We can apply Corollary 3.3 to the algorithm of Andoni et al. [3] with approximation ratio $(\log n)^{O(1/\varepsilon)}$ and run time $O(n^{1+\varepsilon})$ as follows. (Note that ε may be $o(1)$.)

▶ **Corollary 3.5.** *There exists an approximate-alignment algorithm which runs in time* $\tilde{O}(n^{1+\varepsilon})$*, and has approximation factor* $(\log n)^{O(1/\varepsilon^2)}$ *with probability* $1 - \frac{1}{\text{poly}(n)}$.

3.1 Proof of Theorem 3.2

The proof of the theorem will follow from Proposition 3.6, which bounds the run time of Algorithm 1, and Proposition 3.11, which bounds the approximation ratio.

Throughout this section, let u, v and m be the values given to Algorithm 1. Let $P = (p_0, \ldots, p_m)$ be the equipartition of u, and let S be the set defined by Algorithm 1.

▶ **Proposition 3.6.** *Algorithm 1 runs in time* $\tilde{O}(T(|u| + |v|) \cdot m^5)$.

Proof. If $|u| \leq 1$, then we can find an optimal alignment in time $O(|v|)$ naively. Suppose $|u| > 1$. Notice that $|S| \leq O(m^2 \log n)$. In particular, because $(1 + \frac{1}{m})^{(m+1)\ln n} \geq n$,

$$S \subseteq \{p_0, \ldots, p_m\} \cup \{|v|\} \cup \left\{ \left\lceil p_i \pm \left(1 + \frac{1}{m}\right)^j \right\rceil \mid i \in [0:m], j \in [0:(m+1)\ln n] \right\},$$

which has size at most $O(m^2 \log n)$.

Finding an equipartition of u can be done in linear time, and constructing S takes time $O(|S|) = \tilde{O}(m^2)$. In order to perform the fourth step which selects Q, we must compute $f(l, j)$ for each $l \in S$ and $j \in [0:m]$. To evaluate $f(l, j)$, we must consider each $l' \in S$ satisfying $l' \leq l$, and then compute the cost of $f(l', j-1)$ plus $\mathcal{A}(P_j, v[l'+1:l])$ (which takes time at most $T(|u| + |v|)$ to compute). Therefore, each $f(l, j)$ is computed in time $O(|S| \cdot T(|u|+|v|)) \leq \tilde{O}(T(|u|+|v|) \cdot m^2)$. Because there are $O(m \cdot |S|) = \tilde{O}(m^3)$ subproblems of the form $f(l, j)$, the total run time of the dynamic program is $\tilde{O}(T(|u| + |v|) \cdot m^5)$.

So far we have shown that the first level of recursion takes time $\tilde{O}(T(|u| + |v|)m^5)$. The sum of the lengths of the inputs to Algorithm 1 at a particular level of the recursion is at most $|u| + |v|$. It follows by the super-additivity of $T(n)$ that the time spent in any given level of recursion is at most $\tilde{O}(T(|u| + |v|)m^5)$. Because each level of recursion reduces the sizes of the parts of u by a factor of $\Omega(m)$, the number of levels is at most $O(\log_m n) \leq O(\log n)$. Therefore, the run time is $\tilde{O}(T(|u| + |v|) \cdot m^5)$. ◀

When discussing the approximation ratio of Algorithm 1, it will be useful to have a notion of edit distance between partitions of strings.

▶ **Definition 3.7.** Given two partitions $C = (c_0, \ldots, c_m)$ and $D = (d_0, \ldots, d_m)$ of strings a and b respectively, we define $\Delta_{\text{ed}}(C, D) := \sum_i \Delta_{\text{ed}}(C_i, D_i)$.

In order to bound the approximation ratio of Algorithm 1, we will introduce, for the sake of analysis, a partition $Q^{\text{opt}} = (q_0^{\text{opt}}, \ldots, q_m^{\text{opt}})$ of v satisfying $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = \Delta_{\text{ed}}(u, v)$. Recall that P is fixed, which allows us to use it in the definition of Q^{opt}.

We claim that some partition Q^{opt} satisfying $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = \Delta_{\text{ed}}(u, v)$ must exist. If u and v differed by only a single edit, one could start from P and explicitly define Q^{opt} so that $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = 1$ (by a case analysis of which type of edit was performed). It can then be shown by induction on the number of edits that, in general, we can obtain a partition Q^{opt} satisfying $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = \Delta_{\text{ed}}(u, v)$.

Our strategy for bounding the approximation ratio will be to compare $\Delta_{\text{ed}}(P, Q)$ for the partition Q selected by our algorithm to $\Delta_{\text{ed}}(P, Q^{\text{opt}})$. We do this through three observations.

The first observation upper bounds $\Delta_{\text{ed}}(P, Q)$. Informally, it shows that the cost in edit distance which Algorithm 1 pays for selecting Q instead of Q^{opt} is at most $2\sum_{i=0}^{m} |q_i - q_i^{\text{opt}}|$.

▶ **Lemma 3.8.** *Let $Q = (q_0, \ldots, q_m)$ be a partition of v. Then*

$$\Delta_{ed}(P, Q) \leq \Delta_{ed}(u, v) + 2\sum_{i=1}^{m} |q_i - q_i^{opt}|.$$

Proof. Observe that

$$\Delta_{\text{ed}}(P, Q) \leq \Delta_{\text{ed}}(P, Q^{\text{opt}}) + \Delta_{\text{ed}}(Q^{\text{opt}}, Q) = \Delta_{\text{ed}}(u, v) + \sum_{i=1}^{m} \Delta_{\text{ed}}(Q_i^{\text{opt}}, Q_i).$$

Because Q and Q^{opt} are both partitions of v, $\Delta_{\text{ed}}(Q_i, Q_i^{\text{opt}}) \leq |q_{i-1} - q_{i-1}^{\text{opt}}| + |q_i - q_i^{\text{opt}}|$. In particular, $|q_{i-1} - q_{i-1}^{\text{opt}}|$ insertions to the left side of one of Q_i or Q_i^{opt} (whichever has its start point further to the right) will result in the two substrings having the same start-point; and then $|q_i - q_i^{\text{opt}}|$ insertions to the right side of one of Q_i or Q_i^{opt} (whichever has its end point further to the left) will result in the two substrings having the same end-point. Thus

$$\begin{aligned}\Delta_{\text{ed}}(u, v) + \sum_{i=1}^{m} \Delta_{\text{ed}}(Q_i^{\text{opt}}, Q_i) &\leq \Delta_{\text{ed}}(u, v) + \sum_{i=1}^{m} |q_{i-1} - q_{i-1}^{\text{opt}}| + |q_i - q_i^{\text{opt}}| \\ &\leq \Delta_{\text{ed}}(u, v) + 2\sum_{i=1}^{m} |q_i - q_i^{\text{opt}}|,\end{aligned}$$

where we are able to disregard the case of $i = 0$ because $q_0 = q_0^{\text{opt}} = 0$. ◀

The next observation establishes a lower bound for $\Delta_{\text{ed}}(u, v)$.

▶ **Lemma 3.9.** $\Delta_{ed}(u, v) \geq \frac{1}{m}\sum_{i=1}^{m} |p_i - q_i^{opt}|$.

Proof. Because $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = \Delta_{\text{ed}}(u, v)$, we must have that for each $i \in [m]$,

$$\Delta_{\text{ed}}(u, v) = \Delta_{\text{ed}}(u[1 : p_i], v[1 : q_i^{\text{opt}}]) + \Delta_{\text{ed}}(u[p_i + 1 : |u|], v[q_i^{\text{opt}} + 1 : |v|]).$$

Notice, however, that the strings $u[1 : p_i]$ and $v[1 : q_i^{\text{opt}}]$ differ in length by at least $|q_i^{\text{opt}} - p_i|$. Therefore, their edit distance must be at least $|q_i^{\text{opt}} - p_i|$, implying that $\Delta_{\text{ed}}(u, v) \geq |q_i^{\text{opt}} - p_i|$.

Thus $\frac{1}{m}\Delta_{\text{ed}}(u, v) \geq \frac{1}{m}|q_i^{\text{opt}} - p_i|$. Summing over $i \in [m]$ gives the desired equation. ◀

So far we have shown that the cost in edit distance which Algorithm 1 pays for selecting Q instead of Q^{opt} is at most $2\sum_{i=0}^{m} |q_i - q_i^{\text{opt}}|$ (Lemma 3.8), and that the edit distance from u to v is at least $\frac{1}{m}\sum_{i=1}^{m} |p_i - q_i^{\text{opt}}|$ (Lemma 3.9). Next we compare these two quantities. In particular, we show that if Q is chosen to mimic Q^{opt} as closely as possible, then each of the $|q_i - q_i^{\text{opt}}|$ will become small relative to each of the $|p_i - q_i^{\text{opt}}|$.

▶ **Lemma 3.10.** *There exists a partition $Q = (q_0, \ldots, q_m)$ of v such that each q_i is in S, and such that for each $i \in [0 : m]$, $|q_i - q_i^{opt}| \leq \frac{1}{m}|p_i - q_i^{opt}|$.*

Proof Sketch. Consider the partition Q in which q_i is chosen to be the largest $s \in S$ satisfying $s \leq q_i^{\text{opt}}$. Observe that: (i) because $0 \in S$, each q_i always exists; (ii) because $|v| \in S$, we will have $q_m = |v|$; and (iii) because $q_0^{\text{opt}} \leq q_1^{\text{opt}} \leq \cdots \leq q_m^{\text{opt}}$, we will have that $q_0 \leq q_1 \leq \cdots \leq q_m$. Therefore, Q is a well-defined partition of v.

It remains to prove $|q_i - q_i^{\text{opt}}| \leq \frac{1}{m}|p_i - q_i^{\text{opt}}|$. For brevity, we focus on the case of $p_i < q_i^{\text{opt}}$. The other cases are conceptually similar and appear in the full version of this paper.

Assume $p_i < q_i^{\text{opt}}$. Consider the largest non-negative integer j such that $p_i + (1+\frac{1}{m})^j \leq q_i^{\text{opt}}$. One can verify that by definition of j, we must have

$$\left(1+\frac{1}{m}\right)^j \leq q_i^{\text{opt}} - p_i \leq \left(1+\frac{1}{m}\right)^{j+1}. \tag{3.1}$$

It follows that

$$q_i^{\text{opt}} - \left(p_i + \left(1+\frac{1}{m}\right)^j\right) \leq \left(1+\frac{1}{m}\right)^{j+1} - \left(1+\frac{1}{m}\right)^j = \frac{1}{m}\left(1+\frac{1}{m}\right)^j. \tag{3.2}$$

Since $\lceil p_i + (1+\frac{1}{m})^j \rceil \in S$, the definition of q_i ensures that q_i is between $p_i + (1+\frac{1}{m})^j$ and q_i^{opt} inclusive. Therefore, (3.2) implies $q_i^{\text{opt}} - q_i \leq \frac{1}{m}\left(1+\frac{1}{m}\right)^j$. Combining this with (3.1), it follows that $q_i^{\text{opt}} - q_i \leq \frac{1}{m}(q_i^{\text{opt}} - p_i)$, as desired. ◀

We are now equipped to bound the approximation ratio of Algorithm 1, thereby completing the proof of Theorem 3.2. We will use the preceeding lemmas to bound the approximation ratio at each level of recursion to $O(\gamma(n))$. The approximation ratio will then multiply across the $O(\log_m n)$ levels of recursion, giving total approximation ratio $O(\gamma(n))^{O(\log_m n)}$.

▶ **Proposition 3.11.** *Let $E(u,v)$ be the number of edits returned by Algorithm 1. Then*

$$\Delta_{ed}(u,v) \leq E(u,v) \leq \Delta_{ed}(u,v) \cdot (3\gamma(n))^{O(\log_m n)}.$$

Proof. Because Algorithm 1 finds a sequence of edits from u to v, clearly $\Delta_{\text{ed}}(u,v) \leq E(u,v)$. By Lemma 3.10 there is some partition $Q = (q_0, \ldots, q_m)$ of v such that each q_i is in S, and such that for each $i \in [0:m]$, $|q_i - q_i^{\text{opt}}| \leq \frac{1}{m}|p_i - q_i^{\text{opt}}|$. By Lemma 3.9, it follows that

$$\sum_{i=1}^{m} |q_i - q_i^{\text{opt}}| \leq \frac{1}{m}\sum_{i=1}^{m} |p_i - q_i^{\text{opt}}| \leq \Delta_{\text{ed}}(u,v).$$

Applying Lemma 3.8, we then get that

$$\Delta_{\text{ed}}(P,Q) \leq \Delta_{\text{ed}}(u,v) + 2\sum_{i=1}^{m} |q_i - q_i^{\text{opt}}| \leq 3\Delta_{\text{ed}}(u,v).$$

Thus there is some Q which Algorithm 1 is allowed to select such that $\Delta_{\text{ed}}(P,Q) \leq 3\Delta_{\text{ed}}(u,v)$. Since the approximation ratio of $\mathcal{A}$ is $\gamma(n)$, the true partition Q selected at the first level of recursion must satisfy $\Delta_{\text{ed}}(P,Q) \leq 3\gamma(n)\Delta_{\text{ed}}(u,v)$.

After the i-th level of recursion, u has implicitly been split into a large partition P^i, v has implicitly been split into a large partition Q^i, and the recursive subproblems are searching for edits between pairs of parts of P^i and Q^i. Since there is $3\gamma(n)$ distortion at each level, we can get by induction that $\Delta_{\text{ed}}(P^i, Q^i) \leq (3\gamma(n))^i \Delta_{\text{ed}}(u,v)$. Since there are $O(\log_m n)$ levels of recursion, the number of edits returned by the algorithm is at most $\Delta_{\text{ed}}(u,v) \cdot (3\gamma(n))^{O(\log_m n)}$. ◀

4 Embeddings and Dimension Reduction Using Min-Hash Techniques

4.1 Alignment Embeddings for Permutations

Here we present a randomized embedding from $\mathcal{P}_n$, the set of permutations of length n, into Hamming space with expected distortion $O(\log n)$. The embedding has the surprising

Algorithm 2 Alignment Embedding for Permutations.

Input: A string $w = w_1 \cdots w_n \in \mathcal{P}_n$.
Parameters: ε and $m \geq \log_{1/2+\varepsilon} \frac{1}{n} + 1$.

1. At the first level of recursion only:
   a. Initialize an array A of size $2^m - 1$ (indexed starting at one) with zeros. The array A will contain the output embedding.
   b. Select a hash function h mapping Σ to $r \log n$ bits for a sufficiently large constant r.
2. Let i minimize $h(w_i)$ out of the $i \in [n/2 - \varepsilon n : n/2 + \varepsilon n]$.[8] We call w_i the *pivot* in w.
3. Set $A[2^{m-1}] = w_i$.
4. Recursively embed $w_1 \cdots w_{i-1}$ into $A[1 : 2^{m-1} - 1]$.
5. Recursively embed $w_{i+1} \cdots w_n$ into $A[2^{m-1} + 1 : 2^m - 1]$.

property that it implicitly encodes alignments between strings. If the output is stored using run-length encoding,[7] then the size of the output and the run time are both $O(n)$.

The description of the embedding appears as Algorithm 2. For simplicity, we assume $0 \notin \Sigma$, which allows us to use 0 as a null character. The algorithm takes two parameters: ε and m. The parameter ε controls a trade-off between the distortion and the output dimension. The parameter m dictates the maximum depth of recursion that can be performed within the array A. In particular, m needs to be chosen such that the algorithm does not run out of space for the embedding in the recursive calls.

Since each recursive step takes as input words of size in the range $[(1/2-\varepsilon)n : (1/2+\varepsilon)n]$, the input size at the i-th level of the recursion is at most $(1/2+\varepsilon)^{i-1}n$. We need to choose m such that at the m-th level of recursion, the input size will be at most 1. Therefore, it suffices to pick m satisfying $m \geq \log_{1/2+\varepsilon} \frac{1}{n} + 1$.

We denote the resulting embedding of the input string w into the output array A by $\phi_{\varepsilon,m}(w)$. Moreover, for $m = \lceil \log_{1/2+\varepsilon} \frac{1}{n} + 1 \rceil$, we define $\phi_\varepsilon(w)$ to be $\phi_{\varepsilon,m}(w)$. Note that ϕ_ε embeds w into an array A of size $O\left(2^{\log_{1/2+\varepsilon} 1/n}\right) = O\left(n^{-1/\log(1/2+\varepsilon)}\right)$, which one can verify for $\varepsilon \leq \frac{1}{4}$ is $O(n^{1+6\varepsilon})$.

We call ϕ_ε an *alignment embedding* because ϕ_ε maps a string x to a copy of x spread out across an array of zeros. When we compare $\phi_\varepsilon(x)$ with $\phi_\varepsilon(y)$ by Hamming differences, ϕ_ε encodes an alignment between x and y; it pays for every letter which it fails to match up with another copy of the same letter. In particular, every pairing of a letter with a null corresponds to an insertion or deletion, and every pairing of a letter with a different letter corresponds to a substitution. Thus $\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))$ will always be at least $\Delta_{\mathrm{ed}}(x, y)$.

In the rest of this subsection we will prove the following theorem.

▶ **Theorem 4.1.** *For $\varepsilon \leq \frac{1}{4}$, there exists a randomized embedding ϕ_ε from $\mathcal{P}_n$ to $O(n^{1+6\varepsilon})$-dimensional Hamming space with the following properties.*

- *For $x, y \in \mathcal{P}_n$, $\phi_\varepsilon(x)$ and $\phi_{\varepsilon(y)}$ encode a sequence of $\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))$ edits from x to y. In particular, $\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y)) \geq \Delta_{ed}(x, y)$.*
- *For $x, y \in \mathcal{P}_n$, $\mathbb{E}[\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))] \leq O\left(\frac{1}{\varepsilon} \log n\right) \cdot \Delta_{ed}(x, y)$.*
- *For $x \in \mathcal{P}_n$, $\phi_\varepsilon(x)$ is sparse in the sense that it only contains n non-zero entries. Moreover, if $\phi_\varepsilon(x)$ is stored with run-length encoding, it can be computed in time $O(n)$.*

[7] In run-length encoding, runs of identical characters are stored as a pair whose first entry is the character and the second entry is the length of the run.
[8] With high probability, there are no hash collisions.

The first property in the theorem follows from the discussion above. In order to prove $\mathbb{E}[\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))] \leq \Delta_{\mathrm{ed}}(x,y) O\left(\frac{1}{\varepsilon}\log n\right)$, we will consider a series of at most $2\Delta_{\mathrm{ed}}(x,y)$ insertions or deletions that are used to transform x into y. Each substitution operation can be emulated by an insertion and a deletion. Moreover, note that by ordering deletions before insertions, each of the intermediate strings will still be a permutation. In the following lemma, we bound the expected Hamming distance for just a single insertion (or equivalently, a deletion). By the triangle inequality, we get the bound on $\mathbb{E}[\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))]$.

▶ **Lemma 4.2.** *Let $x \in \mathcal{P}_n$ be a permutation, and let y be a permutation derived from x by a single insertion. Let $0 < \varepsilon \leq \frac{1}{4}$ and let m be large enough so that $\phi_{\varepsilon,m}$ is well-defined on x and y. Then $\mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))] \leq O\left(\frac{1}{\varepsilon}\log n\right)$.*

Proof. Observe that the set of letters in position-range $[(1/2-\varepsilon)|x| : (1/2+\varepsilon)|x|]$ in x differs by at most $O(1)$ elements from the set of letters in position-range $[(1/2-\varepsilon)|y| : (1/2+\varepsilon)|y|]$ in y. Thus with probability $1 - O(1/(\varepsilon n))$, there will be a letter l in the overlap between the two ranges whose hash is smaller than that of any other letter in either of the two ranges. In other words, the pivot in x (i.e., the letter in the position range with minimum hash) will differ from the pivot in y with probability $O(1/(\varepsilon n))$. Therefore,

$$\begin{aligned}
\mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))] &= \Pr[\text{pivots differ}] \cdot \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots differ}] \\
&\quad + \Pr[\text{pivots same}] \cdot \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots same}] \\
&\leq O\left(\frac{1}{\varepsilon n}\right) \cdot \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots differ}] \\
&\quad + \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots same}].
\end{aligned}$$

In general, $\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))$ cannot exceed $O(n)$. Thus

$$\begin{aligned}
\mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))] &\leq O\left(\frac{1}{\varepsilon n}\right) \cdot O(n) + \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots same}] \\
&\leq O\left(\frac{1}{\varepsilon}\right) + \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots same}].
\end{aligned}$$

If the pivot in x is the same as in y, then the insertion must take place to either the left or the right of the pivot. Clearly $\phi_{\varepsilon,m}(x)$ and $\phi_{\varepsilon,m}(y)$ will agree on the side of the pivot in which the edit does not occur. Inductively applying our argument to the side on which the edit occurs, we incur a cost of $O(1/\varepsilon)$ once for each level in the recursion. The maximum depth of the recursion is $O\left(\log_{1/2+\varepsilon}\frac{1}{n}\right) = O(\log n)$. This gives us $\mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))] \leq O\left(\frac{1}{\varepsilon}\right) \cdot \log n$. ◀

It remains only to analyze the run time. Notice that $\phi_\varepsilon(x)$ can be stored in space $\Theta(n)$ using run-length encoding. We can compute $\phi_\varepsilon(x)$ in time $O(n)$, as follows. Using Range Minimum Query [13] we can build a data structure which supports constant-time queries returning minimum hashes in contiguous substrings of x. This allows each recursive step in the embedding to be performed in constant time. Since each recursive step writes one of the n letters to the output, the total run time is bounded by $O(n)$.

4.2 Embedding into Hamming Space in the Low-Distance Regime

Assume we are given an embedding from edit distance in Σ^n into Hamming space with subpolynomial distortion $\gamma(n)$. We wish to use such an embedding as a black box in order to obtain a new embedding for the low edit distance regime: the new embedding, which would

Algorithm 3 Choose Next Block (Informal).

Input: A string x, the index i where the current block begins.
Output: The index where the next block begins.
Parameters: $W'' \ll W' \ll W$ set as needed.

1. Consider the window $x_i \cdots x_{i+W-1}$ of size W. Divide the second half of the window into non-overlapping sub-windows of size W', and pick one such sub-window at random.
2. Each substring of length W'' inside the sub-window is called a sub-sub-window (sub-sub-windows may overlap). Compute a hash of each of the sub-sub-windows.[9]
3. Return the start position of the sub-sub-window with the smallest hash.

be parameterized by a value K, would take any two strings $x, y \in \Sigma^n$ with $\Delta_{\text{ed}}(x, y) \leq K$ and map x and y into Hamming space with distortion $\gamma'(K)$, a function of K rather than n.

We make progress toward such an embedding with the added constraint that our strings x and y are (D, R)-periodic free for $D \in \text{poly}(K)$ of our choice and $R \in O(K^3)$. A string is (D, R)-*periodic free* if it contains no contiguous substrings of length D that are periodic with period at most R. Our embedding takes two such strings with $\Delta_{\text{ed}}(x, y) \leq K$ and maps them into Hamming space with distortion $\gamma(\text{poly}(K))$. If we select the black-box embedding to be embedding of Ostrovsky and Rabani [21], then this gives distortion $2^{O\left(\sqrt{\log K \log\log K}\right)}$.

Our main result in this section is stated as the following theorem.

▶ **Theorem 4.3.** *Suppose we have an embedding from edit distance in Σ^n to Hamming space with subpolynomial distortion $\gamma(n) \geq 2$. Let $K \in \mathbb{N}$ and pick some $D \in \text{poly}(K)$. Then there exists $R \in O(K^3)$ and an embedding α from edit distance into scaled Hamming space with the following property. For strings x and y that are (D, R)-periodic free and of edit distance at most K apart, α distorts the distance between x and y by at most $O(\gamma(K^3 D)) \leq O(\gamma(\text{poly}(K)))$ in expectation.*

The complete proof of Theorem 4.3 appears in the full version of the paper. Below we discuss the key ideas, which once again make use of min-hash techniques.

The main step in our embedding is to partition the strings x and y into parts of length $\text{poly}(K)$ in a way so that the sum of the edit distances between the parts equals $\Delta_{\text{ed}}(x, y)$ (with some probability), and then to apply the black-box embedding to each individual part.

We select the partition of x by going from left to right, and at each step choosing the next index at which the current block ends and the next one begins. The algorithm for selecting each successive block appears as Algorithm 3.

The goal of the algorithm is to find partitions of x and y that are aligned despite the edits. By picking a sub-window uniformly at random, we guarantee that with high probability (as a function of W, W') no edits occur directly within that sub-window. However, if x and y differ by an insertion or deletion prior to that sub-window, the sub-window within x may be misaligned with the sub-window within y. Nonetheless, the set of W''-letter sub-sub-windows of the sub-window of x will be almost the same as the set of W''-letter sub-sub-windows of the sub-window of y. By selecting the sub-sub-window with minimum hash as the start position for the next block, we are then able to guarantee that with high probability (as a function of W') we pick positions in x and y which are aligned with each other.

[9] By selecting W'', W', W appropriately, we can use the (D, R)-periodic free property to guarantee that these sub-sub-windows are distinct.

4.3 Dimension Reduction

A mapping of edit distance on length-n strings to edit distance on length-at-most-n/c strings (with larger alphabet size) is called a *dimension-reduction* map with *contraction* c [7]. The *distortion* of the map measures the multiplicative factor to which edit distance is preserved.

▶ **Theorem 4.4.** *There is a randomized dimension-reduction map ϕ with contraction c and expected distortion $O(c)$. In particular, for $x, y \in \Sigma^n$,*

$$\frac{1}{2c} \cdot \Delta_{ed}(x, y) \leq \Delta_{ed}(\phi(x), \phi(y)),$$

and

$$\mathbb{E}[\Delta_{ed}(\phi(x), \phi(y))] \leq O(1) \cdot \Delta_{ed}(x, y).$$

Moreover, for this definition of distortion, ϕ is within a constant factor of optimal. Additionally, ϕ can be evaluated in time $O(n \log c)$.

Note that the output of a dimension-reduction map may be over a much larger alphabet than the input. One should not be too alarmed by this, however, because alphabet reduction can be performed after the dimension reduction (by simply hashing to $\Theta(\log n)$ bits).

Below we summarize our approach to proving Theorem 4.4. The complete proof appears in the full version of the paper. When computing our dimension-reduction map $\phi(w)$ on a word w, one approach would be to split w into blocks of size c and to then define $\phi(w)$'s letters to correspond to blocks. This would achieve the desired reduction in dimension, and would have the effect that a single edit to $\phi(w)$ would correspond to at most c edits in w. However, a single insertion to w could change the content of linearly many blocks, corresponding to a large number of edits to $\phi(w)$. Thus the challenge is to instead break w into blocks in a way so that an edit to w affects only a small number of blocks.

In order to accomplish this, our actual ϕ breaks w into long periodic substrings and non-periodic substrings. The two types of substrings are then handled as follows:

Handling Long Periodic Substrings: Within the periodic substrings, we break the substring into blocks based on the periodic behavior. The key insight is that the embedding of the periodic substring will consist of the same block repeating many times. If an edit occurs in the middle of a long periodic substring, the embedding will still include the same block repeated many times, but $O(1)$ blocks around the edit will be modified. Since the blocks correspond to letters in $\phi(w)$, the edit to w results in $O(1)$ edits to $\phi(w)$.

Handling Non-Periodic Substrings: Within the non-periodic substrings, we select markers in a randomized fashion to determine block boundaries. The definition of non-periodic substrings guarantees that for every c adjacent letters, the $8c$-letter substrings $w_i w_{i+1} \cdots w_{i+8c-1}$ beginning at each of those c letters w_i are distinct. We utilize this property to select markers based on the minimum hash of $8c$-letter substrings. A letter is selected as a marker if the hash of the $8c$-letter string beginning at that letter is smaller than the hashes of any of the $8c$-letter strings beginning in the $c/2$ letters to its left or right. This localizes the selection so that edits to the string will only affect nearby markers.

When two markers are more than c apart, we additionally break the space between them into sub-blocks of size at most c. By preventing blocks in $\phi(w)$ from exceeding $O(c)$ in size, we can take a sequence of edits in $\phi(w)$ and generate a corresponding sequence of edits in w with distortion $O(c)$. When bounding the distortion in the other direction, we risk edits in

w taking place between two far-apart markers, which in turn could affect all of the blocks between those markers in $\phi(w)$. The main technical challenge is bounding the effect this has on the distortion. We do so by showing probabilistically that wherever there is an edit, there will be markers nearby which mitigate the impact of that edit on the block structure of $\phi(w)$.

References

1 Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

2 Alexandr Andoni and Robert Krauthgamer. The computational hardness of estimating edit distance. *SIAM J. Comput.*, 39(6):2398–2429, 2010.

3 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 377–386. IEEE, 2010.

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

5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the 47th Annual Symposium on Theory of Computing (STOC)*, pages 51–58. ACM, 2015.

6 Ziv Bar-Yossef, T.S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of 45th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 550–559. IEEE, 2004.

7 Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the 17th Annual Symposium on Discrete Algorithms (SODA)*, pages 792–801, 2006.

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

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

10 Kun-Mao Chao, William R. Pearson, and Webb Miller. Aligning two sequences within a specified diagonal band. *Bioinformatics*, 8(5):481–487, 1992.

11 Moses Charikar and Robert Krauthgamer. Embedding the ulam metric into l_1. *Theory of Computing*, 2(11):207–224, 2006.

12 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. The MIT Press, third edition, 2009.

13 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to LCA and LCE. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 36–48, 2006.

14 Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

15 Piotr Indyk. Algorithmic aspects of geometric embeddings (tutorial). In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 10–33. IEEE, 2001.

16 Piotr Indyk and Jiri Matoušek. Low-distortion embeddings of finite metric spaces. *Handbook of Discrete and Computational Geometry*, page 177, 2004.

17 Robert Krauthgamer and Yuval Rabani. Improved lower bounds for embeddings into L_1. In *Proceedings of the 17th Annual Symposium on Discrete Algorithms (SODA)*, pages 1010–1017, 2006.

18 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.

19 Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

20 Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

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

22 Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM Journal on Computing*, 38(1):85–96, 2008.

23 Taras K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

24 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.