Many Flavors of Edit Distance

Sudatta Bhattacharya 🖂 🕩

Charles University, Prague, Czech Republic

Sanjana Dey 🖂 💿 National University of Singapore, Singapore

Elazar Goldenberg 🖂 🕩 The Academic College of Tel-Aviv-Yaffo, Israel

Michal Koucký ⊠© Charles University, Prague, Czech Republic

– Abstract -

Several measures exist for string similarity, including notable ones like the edit distance and the indel distance. The former measures the count of insertions, deletions, and substitutions required to transform one string into another, while the latter specifically quantifies the number of insertions and deletions. Many algorithmic solutions explicitly address one of these measures, and frequently techniques applicable to one can also be adapted to work with the other. In this paper, we investigate whether there exists a standardized approach for applying results from one setting to another. Specifically, we demonstrate the capability to reduce questions regarding string similarity over arbitrary alphabets to equivalent questions over a binary alphabet. Furthermore, we illustrate how to transform questions concerning indel distance into equivalent questions based on edit distance. This complements an earlier result of Tiskin (2007) which addresses the inverse direction.

2012 ACM Subject Classification Theory of computation \rightarrow Random projections and metric embeddings

Keywords and phrases Edit distance, Indel distance, Embedding, LCS, Alphabet Reduction

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.11

Related Version Full Version: https://arxiv.org/abs/2410.09877

Funding Sudatta Bhattacharya: Partially supported by the project of Czech Science Foundation no. 19-27871X, 24-10306S and by the project GAUK125424 of the Charles University Grant Agency. Michal Koucký: Partially supported by the project of Czech Science Foundation no. 19-27871X and 24-10306S.

Acknowledgements The project started at EPAC Workshop: Algorithms and Complexity partially supported by the project of Czech Science Foundation no. 19-27871X.

1 Introduction

String-related metrics, such as edit distance, longest common subsequence distance, are pivotal in numerous applications that deal with text or sequence data. Edit distance metric, also referred to as Levenshtein distance [18], quantifies the minimum number of singlecharacter edits (insertions, deletions, or substitutions) necessary to convert one string into another. This metric finds extensive application in spell checking and correction systems, DNA sequence alignment and bioinformatics for comparing genetic sequences, as well as in natural language processing tasks such as machine translation and text summarization, among other fields. The longest common subsequence metric, also known as *Indel* or *LCS* distance, evaluates the disparity between two strings by determining the minimum number of single-character edits while forbidding substitutions. This metric has diverse applications, including text comparison and plagiarism detection, DNA and protein sequence analysis for recognizing shared regions or motifs, music analysis to uncover similarities between musical sequences, and document clustering and classification based on content similarity.



© Sudatta Bhattacharya, Sanjana Dey, Elazar Goldenberg, and Michal Koucký;

licensed under Creative Commons License CC-BY 4.0 44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).



Editors: Siddharth Barman and Sławomir Lasota; Article No. 11; pp. 11:1–11:16 Leibniz International Proceedings in Informatics

LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

11:2 Many Flavors of Edit Distance

From a computational complexity perspective computing distances under the Edit or Indel metric typically takes quadratic time complexity, as initially demonstrated by Wagner [23]. Subsequent research has marginally improved this complexity by reducing logarithmic factors, as evidenced by Masek and Paterson [19] and Grabowski [15]. Additionally, Backurs and Indyk [5] demonstrated that a truly sub-quadratic algorithm $O(n^{2-\delta})$ for some $\delta > 0$ would lead to a $2^{(1-\gamma)n}$ -time algorithm for CNF-satisfiability, contradicting the Strong Exponential Time Hypothesis. Similarly, Abboud et al. [1] established a similar result for computing the Indel metric between string pairs. Notably, obtaining an efficient isometric embedding for the edit metric into the Indel metric would effortlessly yield the latter result.

Extensive research has been conducted on approximating edit distance, with studies dating back to the work of Landau et al. [17, 6, 8, 4, 7, 9, 11, 14, 16, 10], ultimately culminating in the breakthrough result of Andoni and Nosatzki [3], which offers a (large) constant factor approximation in nearly linear time. However, approximating the Indel distance has not received similar attention, and although one expects the same techniques should provide similar results for Indel distance, one would need to check all the details of the construction to verify the exact properties of such a result. This exhibits a general pattern where results for one of the measures can often be adapted for the other but there is no simple tool that would guarantee such an automatic transformation.

Similar pattern emerges when dealing with the string measures over different size alphabets. For example, the hardness result elucidated by Backurs and Indyk [5] is constrained to some "large" constant-size alphabets. Subsequent research has revealed that the computational task of computing edit distance is also hard for binary alphabets using an ad hoc approach. Once more, the possibility of achieving an efficient isometric embedding between strings residing in large alphabets and those in smaller ones could potentially resolve these questions automatically. Another scenario where the alphabet size becomes relevant is in the simple linear-time approximation algorithm for the length of the LCS. The naive algorithm provides a $|\Sigma|$ -approximation, where Σ represents the alphabet to which the strings reside. Therefore, if one can embed strings from a large alphabet into those from a smaller alphabet while approximately preserving distances, it may lead to an improvement in the approximation factor. Given the current circumstances, we pose the following questions, which we then delve into extensively:

▶ Question 1. Does an isometric embedding exist between the edit metric and the Indel metric? Can it be computed efficiently?

▶ Question 2. Does an isometric embedding exist between edit metric on arbitrary alphabets and the edit distance on binary alphabets? Can it be computed efficiently?

1.1 Our Contribution

This paper introduces multiple mappings that establish connections between various string metrics. Specifically, we transform points residing within a designated input metric space (M, d) into points within an output metric space (M', d'), ensuring that the distance between any pair of points in M under d, is (approximately) preserved on the output pairs under d'. In this paper, we introduce a relaxation of the concept of isometric embedding, permitting scaling factors, as outlined below.

We say an embedding $E : M \to M'$ is **scaled isometric** if there exists a function $f : \mathbb{N} \to \mathbb{N}$ that maps distances from the original space to the embedded one, such that for every pair of points $x, y \in M$ we have: d'(E(x), E(y)) = f(d(x, y)). This implies that such an embedding can be utilized to reduce the computation of distances in the original metric M to computing distances under M', provided that f is invertible and computationally tractable.

A special case of scaling involves preserving the normalized distances. That is, for any metric on a string space, it is intuitive to define the normalized distance between a pair of strings of a specified length, as their distance divided by the maximum distance of pairs of that same length. Subsequently, for an embedding $E: M \to M'$ that preserves lengths (i.e., the length of the output string is a function of the input string), we further assert that it is **normalized scaled isometric** if, for every pair from M, the normalized distance of the embedded strings preserves the normalized distance of the original ones.

Utilizing normalized scaled isometric embedding can facilitate the computational process of approximating distances in the original metric M to compute approximate distances under M', even if the normalized distances are not perfectly preserved by the embedding but distorted by a (multiplicative) factor of c. In such a case, any c' approximation algorithm for the metric M' could thus be transformed into a $c \cdot c'$ approximation algorithm for M.

1.1.1 Our Results

In the sequel, we utilize the notation Δ_{indel} to represent the Indel distance between a pair of strings, Δ_{edit} to denote their edit distance. Additionally, we use $\widetilde{\Delta}_{indel}$ and $\widetilde{\Delta}_{edit}$ to represent the normalized Indel distance between a pair of strings (for precise definitions, refer to Section 2).

Alphabet Reduction – Succinct Embedding. Our first result pertains to alphabet reduction achieving normalized scaled isometric embedding. In this context, as elaborated in Section 3, any embedding contracts the normalized distances of some of the pairs. Consequently, we shift our focus to approximate normalized scaled isometric embedding, where we permit slight distortions in the normalized distances. Our main result is outlined below:

▶ **Theorem 3** (Alphabet Reduction - Succinct Embedding). Let Γ be a finite alphabet, and let $0 < \varepsilon < 1/4$. There exists an alphabet Σ , where $|\Sigma| = O(\frac{1}{\varepsilon^2})$ and there exists $E : \Gamma^* \to \Sigma^*$ satisfying:

$$\forall X,Y \in \Gamma^*: \ \widetilde{\Delta}_{indel}(E(X),E(Y)) \in \left| (1-\varepsilon)\widetilde{\Delta}_{indel}(X,Y), \widetilde{\Delta}_{indel}(X,Y) \right|.$$

Moreover, for every $X \in \Gamma^n$ we have: $|E(X)| = O(n \log(|\Gamma|))$.

Observe that embedded string length is optimal up to logarithmic factors. The size of the alphabet Σ must exceed $1/\varepsilon$, as otherwise, by a claim proved later, there will be a pair of strings whose distance will be contracted by at least $\left(1 - \frac{1}{|\Sigma|}\right)$ -factor.

As a result, we find that when focusing on Δ_{indel} approximation, we can, without loss of generality, limit our scope to a constant alphabet size without significantly impacting the quality of the approximation. This is captured in the following corollary which we provide without a proof.

▶ Corollary 4. Let Γ be a finite alphabet, and let $0 < \varepsilon < 1/4$. Suppose that there exists an algorithm ALG that provides a c-factor approximation for the Δ_{indel} metric for any pairs of strings in Σ of total length N, where $\Sigma = O(1/\varepsilon^2)$, running in time t(N).

Then there exists an algorithm ALG' that, given any pair of string in Γ of total length n, provides a $c + \varepsilon$ -approximation for their Δ_{indel} distance in time $t(n \log |\Gamma|)$.

The proof strategy of Theorem 3 is as follows: Initially, we construct an error-correcting code within the smaller alphabet Σ , where $|\Sigma| = O(1/\varepsilon^2)$. This code ensures that every distinct pair of codewords shares only a small portion of LCS, indicating a significant Δ_{indel}

between them. The code comprises $|\Gamma|$ words, with its dimension being logarithmic in the size of Γ . We interpret this code as a mapping from characters within Γ to short strings in Σ . The existence of such a code can be demonstrated using the probabilistic method. The construction is almost tight in terms of the smaller alphabet size: it is not hard to show that for any code $C \subseteq \Sigma^k$, if $|C| > |\Sigma|$, then there exist $c \neq c' \in C$, satisfying: $\Delta_{indel}(c,c') < (1-\frac{1}{|\Sigma|})2k$.

Employing the code construction, we embed input strings in a straightforward and natural manner: Consider a string residing in Γ^* , then each of its characters is sequentially encoded using the code. This encoding ensures that identical characters are mapped to the same codeword, while the code's distance guarantees that distinct characters may have only a few shared matches, akin to the global nature of Δ_{indel} alignments. If there were no matches between distinct characters' encodings, any alignment for the embedded strings could be converted into an alignment between the input strings without any distortion in the normalized costs. However, these few matches between non-matching codewords introduce a slight distortion and complicate the proof.

Alphabet Reduction – Binary Alphabets. Our previous method relied on a local approach, wherein the characters of the string from the large alphabet were encoded sequentially and independently. It appears that such a local strategy may not result in a normalized scaled isometric embedding into a small, particularly binary alphabet. As codes in such alphabets have a relative distance of at most 1/2, and this distance affects the distortion of the normalized distances. Therefore, achieving alphabet reduction into binary alphabets requires a scaling that is not normalized, as well as a more global approach that does not encode the characters sequentially and independently. However, there's a caveat: the encoding still needs to be performed independently on each string rather than on a pair of strings. Specifically, if we take a particular string X its encoding will remain the same regardless of the second string Y. This aspect forms the focal point of our forthcoming result.

For clarity, we present our results for the Δ_{indel} metric, with a similar approach applicable to the Δ_{edit} metric. Our main result states that one can embed Δ_{indel} over any alphabet Σ into binary alphabet. We employ asymmetric embedding and prove that the distances are preserved up to some scaling function. The dimension of the embedded strings is quasipolynomial, making this result more of a proof of concept at the moment. Nonetheless, we find it conceptually intriguing and pose the question of decreasing the target dimension as an open question. Our main result is as follows:

▶ **Theorem 5** (Informal statement of Theorem 16). For any alphabet Σ and for every $n \in \mathbb{N}$ there exist functions $G, H : \Sigma^n \to \{0, 1\}^N$, $f : [n] \to [N]$ where $N = n^{O(\log n)}$ such that for any $X, Y \in \Sigma^n$,

 $\Delta_{indel}(G(X), H(Y)) = f(\Delta_{indel}(X, Y)).$

Our initial consideration revolves around the fact that deciding whether $\Delta_{indel}(X, Y) \leq k$ for two strings $X, Y \in \Sigma^n$ and a threshold parameter k, can be accomplished by a Turing machine utilizing $\log(n)$ -space. This capability can then be translated into a SAT formula of $\log^2(n)$ -depth.

The foundation of our construction converts this formula into a pair of binary strings X', Y' of quasi-polynomial length, where X' (respectively, Y') depends solely on X (Y) and the Indel distance between X', Y' is contingent on the distance between X, Y. This is achieved by recursively transforming the formula into such a pair of strings gate by gate. Two essential components, referred to as AND- and OR-gadgets, implement this process.

The input for the AND-gadget consists of two pairs of strings $(X_0, Y_0), (X_1, Y_1)$, which can be thought of as outputs from previous levels. Here the X_i 's only depend on X and the Y_i 's only depend on Y. Moreover, we are guaranteed that $\Delta_{indel}(X_i, Y_i)$ can only take two values $\{F, T\}$, where F < T. The goal is to concatenate the X_i into a single string X and the Y_i 's into a different string Y such that: $\Delta_{indel}(X, Y)$ can also take value in $\{F', T'\}$, where F' < T' and: $\Delta_{indel}(X, Y) = T'$ if and only if $\Delta_{indel}(X_0, Y_0) = T \wedge \Delta_{indel}(X_1, Y_1) = T$. Similarly for the OR-gadget. Our construction of the LCS instance from the Formula Evaluation is similar to that of Abboud and Bringmann [2] which considers reduction of the Formula Satisfiability to LCS. The purpose of our reduction is different, though.

Scaled Isometric Embedding of Indel into Edit Metrics. While our previous results aimed on reducing the alphabet size while keeping the underlying metric (either Δ_{indel} or Δ_{edit}), this section focuses on converting one metric into another. Tiskin [21] in section 6.1 proposed a straightforward embedding from the Δ_{edit} metric to the Δ_{indel} metric. This inspired our exploration into embedding in the reverse direction i.e. from the Δ_{indel} metric to the Δ_{edit} metric. Our primary contribution in this realm is a scaled isometric embedding from the Δ_{indel} metric to Δ_{edit} , as outlined below.

▶ **Theorem 6** (Indel Into Edit Metrics Embedding - Approximate embedding). For any alphabet Σ , $n \in \mathbb{N}$ and $\varepsilon \in (0, 1]$, there exist mappings $E : \Sigma^n \to \Sigma^n$ and $E' : \Sigma^n \to (\Sigma \cup \{\$\})^N$, where $N = \Theta(n/\varepsilon)$, such that for any $X, Y \in \Sigma^n$, we have

$$\Delta_{edit}(E(X), E'(Y)) = N - n + k, \text{ where } k \in \left[\frac{\Delta_{indel}(X, Y)}{2}, (1 + \varepsilon)\frac{\Delta_{indel}(X, Y)}{2}\right).$$

Observe that while plugging $\varepsilon \leq \frac{1}{n}$ we obtain a scaled isometry at the expense of a quadratic increase in the length of the second string. Conversely, for constant values of ε , N scales as O(n) albeit with the trade-off of only approximately preserving distances within a constant factor. For intermediate values of ε , we can compromise between the accuracy and the stretch length.

Let us revisit Tiskin's (section 6.1 of [21]) construction of the reverse embedding, namely, from Δ_{edit} into Δ_{indel} . The embedding proceeds as follows: a special character \$ is appended after every symbol of each string. It is easy to check that for each pair of strings, the Δ_{indel} between the embedded pair of strings equals twice the Δ_{edit} between the original pair. In our construction, one string remains unaltered, while for the second string, we append after every symbol a block of length *n* consisting of the special character.

The core of the proof demonstrates the conversion of any Δ_{indel} -alignment for the input strings into an Δ_{edit} -alignment for the embedded strings, preserving the distances up to a scaling factor. This process involves replacing any deletions originally performed on the first string by substituting the characters with the special inserted character. Deletions made on the second string remain unaffected.

1.2 Related Work

The problem of embedding edit distance into other distance measures, such as Hamming distance, ℓ_1 , etc., has attracted significant attention in the literature. Let us briefly survey some of these approaches.

Chakraborty et al. [12] introduced a randomized embedding scheme from the edit distance to the Hamming distance. This embedding transforms strings from a given alphabet into strings that are three times longer. For each pair of strings embedded using the same random

11:6 Many Flavors of Edit Distance

sequence, with high probability the edit distance between the embedded strings is at most quadratic in the Hamming distance of the original strings. Batu et al. [8] introduced a dimensionality reduction technique: Given a parameter r > 1, they reduce the dimension by a factor of r at the expense of distorting the distances by the same factor. They employed the locally consistent parsing technique for their embedding. Ostrovsky and Rabani [20] presented a polynomial time embedding from edit distance to ℓ_1 distance with a distortion of $\mathcal{O}(2^{\sqrt{\log n \log \log n}})$. They proposed a randomized embedding where the length of the output strings is quadratic in the input strings, and the distances are preserved, with high probability, up to the distortion factor.

1.3 Future Directions

Introducing a Robust Concept of Approximation: Transitioning from Approximating Δ_{edit} into Approximating Δ_{indel} . Recall that one of the reasons we aimed to isometrically embed the Δ_{indel} metric into the Δ_{edit} metric stemmed from the abundance of approximation results for Δ_{edit} that might not easily extend to the Δ_{indel} metric. A natural approach, based on our embedding result, is to approximate the Δ_{indel} distance between X, Y by approximating the Δ_{edit} between the embedded strings. However, this is not an immediate consequence due to the substantial disparity in length between the embedded strings and the notion of approximation in this case, as detailed next:

Recall that in Theorem 6 the scaling mechanism is not normalized, i.e, the embedding function did not preserve normalized distances, but instead:

$$\Delta_{edit}(E_1(X), E_3(Y)) = N - n + k, \text{ where } k \in \left[\frac{\Delta_{indel}(X, Y)}{2}, \dots, (1 + \varepsilon) \frac{\Delta_{indel}(X, Y)}{2}\right]$$

where N, n are the length of the embedded strings, and $N = \Theta(\frac{n}{\varepsilon})$. Observe that the Δ_{edit} between the embedded strings lies in the range of [N - n, N].

Considering the substantial difference in length between the embedded strings, an algorithm that consistently outputs the value N - n, regardless of the embedded strings, already yields a $1 + O(\varepsilon)$ -approximation for the distance between the embedded strings. Certainly, such an outcome provides no information about the Δ_{indel} of the original strings. Therefore, we introduce a more robust notion of approximation that generally addresses the discrepancy in string lengths:

▶ **Definition 7** (A Robust Notion of Approximation). Let c > 1, let Σ be a finite set, and let $X, Y \in \Sigma^*$. Define |X| = N, |Y| = n, and assume $N \ge n$. Define $k_{X,Y}$ such that: $\Delta_{edit}(X,Y) = N - n + k_{X,Y}$.

An algorithm is considered to provide a robust c-approximation for Δ_{edit} if for all pairs X, Y it outputs k' such that: $k' \in [k_{X,Y}, c_{K,Y}]$.

We assert that for any value of ε , any algorithm ALG that provides a robust *c*-approximation for Δ_{edit} yields an algorithm ALG' that provides $(1 + \varepsilon)c$ -approximation for Δ_{indel} . Moreover, if the running time ALG on input strings of lengths N, n is t(N, n), then the running time of ALG' is $t\left(\frac{n}{\varepsilon}, n\right)$. The construction of ALG' is straightforward: on input strings X, Y we first apply the embedding, then apply ALG on the resulting strings and finally output: 2k'.

We leave the quest of discovering a robust approximation algorithm for Δ_{edit} as an open question, which falls outside the scope of this paper.

1.4 Organization Of The Paper

We structure the paper as follows. In Section 3 we prove our main result, namely Theorem 3, discussing normalized scaled isometric embedding between large and small alphabets. In Section 4 we establish Theorem 5 focusing on alphabet reduction with binary alphabets. We also demonstrate the existence of an indel to edit scaled isometric embedding, as stated in Theorem 6 the proof of which is in the full version of the paper.

2 Preliminaries and Notations

In this section we introduce the notations that is used throughout the rest of the paper. For any string $X = x_1 x_2 \dots x_n$ and integers $i, j, X[i]^{-1}$ denotes $x_i, X[i, j]$ represents substring $X' = x_i \dots x_j$ of X, and X[i, j] = X[i, j-1]. "-operator denotes concatenation, e.g $X \cdot Y$ is the concatenation of two strings X and Y. Λ denotes the empty string.

Edit Distance with Substitutions (Δ_{edit}). For strings $X, Y \in \Sigma^*$, $\Delta_{edit}(X, Y)$ is defined as the minimal number of edit operations required to transform X into Y. The set of edit operations includes character insertion, deletion, and substitutions.

Indel Distance (Δ_{indel}). For strings $X, Y \in \Sigma^*$, $\Delta_{indel}(X, Y)$ is defined as the LCS (Longest Common Subsequence) metric between X and Y. It counts the minimal number of edit operations needed to convert the strings, where substitutions are excluded.

Normalized Distance. To assess the distance between each pair of strings in a standardized manner, it is advantageous to express it as a normalized value within the range [0, 1]. To achieve this, we introduce the following definition:

$$\widetilde{\Delta}_{edit}(X,Y) = \frac{\Delta_{edit}(X,Y)}{\max(|X|,|Y|)}, \quad \widetilde{\Delta}_{indel}(X,Y) = \frac{\Delta_{indel}(X,Y)}{|X|+|Y|}$$

A string X' is considered a **subsequence** of another string X if $\Delta_{indel}(X, X') = |X| - |X'|$. X' is considered a **substring** of X if X' is a contiguous subsequence of X.

Alignment. For an alphabet Σ and any two strings $X, Y \in \Sigma^*$, an Δ_{edit} alignment of X and Y is a sequence of edit operations (insertions, deletions, and substitutions) that transform the string X into Y. The cost of the alignment is determined by the number of edit operations. An alignment is optimal if it achieves the lowest possible cost. Observe that for each character of X that wasn't deleted or substituted, can be matched with a unique character from Y. The collection of matched characters is referred to as the matching characters of the alignment.

Similarly we define an Δ_{indel} alignment of X and Y as a sequence of edit operations, with the exception that substitutions are not permitted. The cost, optimality, and matching characters of an alignment are defined analogously. See Figure 1 for an example.

¹ We use x_i or X_i or X[i] to denote the i^{th} character of the string X interchangeably.



Figure 1 Example for (a) Δ_{indel} alignment and (b) Δ_{edit} alignment (the matched characters are highlighted in blue, the deleted characters in red and the substituted characters in orange).

3 Alphabet Reduction

Within this section, we tackle the task of embedding strings from a sizable alphabet into a smaller one while preserving the global nature of the original metric space. Our main result demonstrates that it's possible to embed strings from any large alphabet Γ into strings of a smaller alphabet Σ , where the length of the strings remains approximately unchanged, and the normalized distances are distorted by at most a factor of $(1 + \varepsilon)$. The size of the alphabet Σ increases quadratically with $1/\varepsilon$. To provide clarity, we present our results for the Δ_{indel} metric; a similar approach can be applied to the Δ_{edit} metric. This section is structured as follows: In subsection 3.1 we outline our findings regarding normalized scaled alphabet reductions, covering both lower and upper bounds. Section 3.2 discusses our upper bounds, while Section 3.3 addresses lower bounds.

3.1 Normalized Scaled Isometric Embedding – Our Finding

An embedding E is said to preserve lengths, if there exists: $\ell : \mathbb{N} \to \mathbb{N}$ such that: $\forall X \in \Gamma^n : |E(X)| = \ell(n)$. It is non-shrinking if $\ell(n) \ge n$. It is natural to focus on non-shrinking embeddings otherwise if the embedding maps from a large alphabet to smaller one some distinct strings will get mapped to the same string.

The following claim shows that any length-preserving embedding, mapping strings from large alphabet into strings of smaller one, necessarily contracts the normalized distances between certain pairs of strings. The proof of the claim is deferred to Section 3.3.

 \triangleright Claim 8. Let Γ, Σ be finite alphabets, such that: $|\Gamma| > |\Sigma|$, and let $E : \Gamma^* \to \Sigma^*$, which is a length-preserving embedding, then for any $n \in \mathbb{N}$ we have:

$$\exists X, Y \in \Gamma^n : \Delta_{indel}(E(X), E(Y)) < \Delta_{indel}(X, Y).$$

Therefore, we redirect our attention to approximate embedding, where we allow for a slight distortion in distances. Our main result is as follows:

▶ **Theorem 3** (Alphabet Reduction - Succinct Embedding). Let Γ be a finite alphabet, and let $0 < \varepsilon < 1/4$. There exists an alphabet Σ , where $|\Sigma| = O(\frac{1}{\varepsilon^2})$ and there exists $E : \Gamma^* \to \Sigma^*$ satisfying:

$$\forall X, Y \in \Gamma^* : \quad \widetilde{\Delta}_{indel}(E(X), E(Y)) \in \left[(1 - \varepsilon) \widetilde{\Delta}_{indel}(X, Y), \widetilde{\Delta}_{indel}(X, Y) \right].$$

Moreover, for every $X \in \Gamma^n$ we have: $|E(X)| = O(n \log(|\Gamma|))$.

Note that the distortion is "one-sided" in the sense that the normalized distances of the embedded strings cannot surpass the normalized distance of the original strings. However, for the lower bound, a $(1 - \varepsilon)$ -factor may be incurred. Furthermore, we demonstrate that for

any embedding, the normalized distances cannot be uniformly scaled by a fixed factor. In particular, we demonstrate that there exist pairs of strings whose normalized distances are reduced, while for other pairs, their normalized distances converge to each other arbitrarily closely. This, in turn, illustrates that we cannot deduce the value of $\widetilde{\Delta}_{indel}(X,Y)$ directly from the value of $\widetilde{\Delta}_{indel}(E(X), E(Y))$ by a simple scaling.

 \triangleright Claim 9. Let Γ, Σ be finite alphabets, such that: $|\Gamma| > |\Sigma|$, and let $E : \Gamma^* \to \Sigma^*$, which is a length-preserving non-shrinking embedding. We have:

1. For any $n \in \mathbb{N}$, there exist $X, Y \in \Gamma^n$ such that

$$\widetilde{\Delta}_{indel}(E(X), E(Y)) \le (1 - \frac{1}{|\Sigma|})\widetilde{\Delta}_{indel}(X, Y).$$

2. For any sequence Z_1, Z_2, \ldots where $|Z_n| = n$,

 $\lim_{n\to\infty}\widetilde{\Delta}_{indel}(E(Z_n),E(\Lambda)) - \widetilde{\Delta}_{indel}(Z_n,\Lambda) = 0, \text{ where } \Lambda \text{ denotes the empty string.}$

3.2 Upper Bounds

The crux of Theorem 3 lies in the existence of an error correcting code with respect to the Δ_{indel} metric, even when the alphabet size is small. More specifically, given a proximity parameter $\varepsilon > 0$, and $|\Gamma|$, we pick a set C of strings residing in Σ^k of cardinality $|\Gamma|$, with large pairwise distance. The construction of C follows a greedy approach reminiscent of the Gilbert-Varshamov bound [13, 22]. Properties of the code are summarized in the next statement.

▶ Lemma 10. For any $\varepsilon < 1/2$, let Σ be a finite alphabet satisfying $|\Sigma| > 32/\varepsilon^2$. For every $n \in \mathbb{N}$, there exists $k \in \mathbb{N}$ with $k = O(\log n)$ for which the following conditions are satisfied: 1. There exists a code $C_{n,\varepsilon} \subseteq \Sigma^k$ with $|C_{n,\varepsilon}| = n$.

2. $\forall c \neq c' \in C_{n,\varepsilon}$: $\Delta_{indel}(c,c') \ge (2-\varepsilon)k$.

The proof of Lemma 10 is given in the full version of the paper.

Endowed with the existence of such an error correcting code we assign a distinct codeword to each of the characters in the larger alphabet Γ . The embedding procedure is as follows: Given a string $X \in \Gamma^n$, we embed it into a string in $(\Sigma^k)^n$, where the encoding of X is formed by concatenating the codewords assigned to each of its characters. The presentation of the embedding, along with its proof of correctness, is provided in Section 3.2.1.

3.2.1 The Embedding

Let Γ be an alphabet, and let ε be a proximity parameter, and let $C := C_{|\Gamma|,\varepsilon}$ be the code whose existence is guaranteed by Lemma 10. We interpret the code C as a function mapping characters from Γ into (short) strings in Σ^k . The embedding proceeds as follows: each string in Γ^* is encoded sequentially character by character, where the encoding of each character is performed using the code C. Our main technical lemma is as follows:

▶ Lemma 11. For any finite alphabet Γ and $\varepsilon > 0$, consider the encoding $C_{|\Gamma|,\varepsilon} : \Gamma \to \Sigma^k$ as implied by Lemma 10. For any string $X \in \Gamma^*$ define: $E(X) = C(X_1) \dots C(X_n)$ (where n = |X|).

Then for any pair of strings $X, Y \in \Gamma^*$ the following inequality holds:

$$(1-48\varepsilon)\Delta_{indel}(X,Y) \le \Delta_{indel}(E(X),E(Y)) \le \Delta_{indel}(X,Y)$$

Moreover, for every $X \in \Gamma^n$ we have: $|E(X)| = O(n \log |\Gamma|)$.

11:10 Many Flavors of Edit Distance

In the sequel, an Δ_{indel} -alignment converting E(X) into E(Y) is simply referred as an alignment. In the course of the proof, we introduce the concepts of blocks and blockstructured alignments. For $X \in \Gamma^n$ we define the *i*-th block of E(X) to be the substring of E(X) corresponding to X_i , namely it equals $C(X_i)$. Furthermore, given an alignment \mathcal{A} that transforms E(X) into E(Y), we label it as a *block-structured* alignment if, for each *i*-th block in E(X), the alignment either fully matches all the characters of the block to some block jin E(Y) or entirely deletes the *i*-th block. It is clear that block-structured alignments for the embedded strings correspond one-to-one with alignments for the original strings, and their normalized distance remains unchanged. To prove our main technical lemma we transform any alignment converting E(X) to E(Y) into a block-structured one without significantly increasing its cost.

We will introduce certain notations to facilitate the presentation of the proof. For any $X \in \Gamma^n$ we employ lowercase letters such as: i, j, k etc. to represent indices of X_i . We utilize tuples from $[n] \times [k]$ to represent indices of E(X), where the first index signifies the block index denoted by lowercase letters, and the second one describes the index within the block represented by a lowercase Greek letter.

The following claim, stated without a formal proof, will be useful in the subsequent proof.

 \triangleright Claim 12. For an alignment \mathcal{A} transforming E(X) into E(Y), we have that the set of matching characters has to be monotone, indicating that for $(i, \alpha) < (i', \alpha')$ in lexicographical order if (i, α) is matched to (j, β) and (i', α') to (j', β') by \mathcal{A} , we must have: $(j, \beta) < (j', \beta')$.

Given an alignment \mathcal{A} , we partition E(Y) into n segments based on its matching blocks in E(X). Define the *i*-th segment as follows: if no character of the *i*-th block of E(X) is matched under \mathcal{A} , then the *i*-th segment is empty. Otherwise, let j denote the first block of E(Y) that includes a matching character for one of the characters in the *i*-th block. If *i* is the smallest block containing a matching coordinate within j, then the *i*-th segment starts at (j, 1), otherwise it starts at the first coordinate within the *j*-th block matching with the *i*-th block.

The ending point of the segment is defined similarly: Let j' be the initial block of E(Y) containing a match for the (i + 1)-th block of E(X). If the *i*-th block does not match any of the j'-th coordinates, then the *i*-th segment ends at (j - 1, k). Otherwise, it ends at the coordinate preceding the first match of (i + 1) and j. We define the starting point of the first non-empty segment as (1, 1) and the end point of the last non-empty segment as (n, k). See Figure 2 for an illustration.

Additionally, we define $cost_{\mathcal{A}}(i)$, the cost of the *i*-th block, as the sum of unmatched coordinates in E(X) within its *i*-th block and the unmatched coordinates in E(Y) within its *i*-th segment. For example, in the illustration provided in Figure 2, we have $cost_{\mathcal{A}}(1) = 0 + 3$ since all the characters of the first block of E(X) are matched, and there are 3 unmatched coordinates in the first segment of E(Y). As the decomposition of E(Y) results in disjoint parts, the sum of the costs across the different blocks equals the cost of \mathcal{A} , which we denote by: $cost(\mathcal{A})$.

3.2.1.1 Converting Into Block-Structured Alignments

In this section, we introduce an algorithm that takes an arbitrary alignment \mathcal{A} transforming E(X) into E(Y) as input and produces an alignment \mathcal{A}^* . The resulting alignment \mathcal{A}^* is block-structured, and its cost does not substantially exceed that of \mathcal{A} .

To design the new matching we need few definitions. We say that blocks i and j are partially matched by an alignment \mathcal{A} if there exists a pair (i, α) and (j, β) matched by \mathcal{A} . Furthermore, i and j are significantly matched by \mathcal{A} if more than εk characters in the *i*-th



Figure 2 An illustration of the matching between the strings E(X) and E(Y). Arrows indicate matching coordinates, and dashed lines represent the beginning/end points of the segments. The first segment starts at (1, 1) and ends at (2, 3), as the first matched coordinate in the second block of E(X) is mapped to the third block, and no character from the first block of E(X) is mapped to that block. The second segment starts at (3, 1) and ends at (4, 1) (as the first matched coordinate in the second block of the second block of E(X) is mapped to the third block, and there exists a character from the second block of E(X) is mapped to that block, and there exists a character from the second block of E(X) that is mapped to that block).

block of X are matched into the *j*-th block; we say that *i* and *j* are perfectly matched if \mathcal{A} matches every character in E(X) into a character in E(Y). The algorithm operates in two stages. In the first stage, the algorithm iteratively takes two significantly matched blocks that are not perfectly matched, removes all matches that the characters of the two blocks participate in, and introduces a perfect match between the two blocks. In the second stage, we remove all the matches from blocks that are not matched perfectly.

A key observation is that if *i* and *j* are significantly matched then we have the following inequality: $\Delta_{indel}(C(X_i), C(Y_j)) < (2 - \varepsilon)k$. Therefore, by the distance guarantee regarding *C*, we must have $X_i = Y_j$. The algorithm is given next. For the sake of the analysis its second stage is divided into two cases.

3.2.1.2 The Correctness of the Algorithm

We break down the proof of correctness into three claims. Claim 13 states that the output produced by Algorithm 1 is a block-structured alignment. The subsequent two claims provide bounds on the cost difference between the input and output alignments.

 \triangleright Claim 13. The alignment \mathcal{A}' produced by Algorithm 1 from any alignment \mathcal{A} converting E(X) into E(Y) is a block-structured alignment converting E(X) into E(Y).

The proof of Lemma 11 is derived from the following claims, let us first state the claims.

 \triangleright Claim 14. Let $\varepsilon < 1/4$ and let \mathcal{A} be any alignment converting E(X) into E(Y). Let \mathcal{A}_{I} be the resulting alignment obtained by applying the algorithm described in stage I on \mathcal{A} with a proximity parameter of ε . Then,

 $cost(\mathcal{A}_{\mathrm{I}}) \leq (1+4\varepsilon)cost(\mathcal{A}).$

 \triangleright Claim 15. Let \mathcal{A}_{I} be any resulting alignment obtained by applying the algorithm described in stage I on some alignment \mathcal{A} with a proximity parameter of ε . Let \mathcal{A}_{II} be the resulting alignment obtained by applying the algorithm described in stage II on \mathcal{A}_{I} with a proximity parameter of ε . Then,

 $cost(\mathcal{A}_{\mathrm{II}}) \leq (1+4\varepsilon)cost(\mathcal{A}_{\mathrm{I}}).$

The proofs of claims 13, 14 and 15 are given in the full version of the paper.

Algorithm 1 Converting Into Block-Structured Alignments. **Data:** An alignment \mathcal{A} converting E(X) into E(Y)**Result:** An alignment \mathcal{A}' that is block-structured, converting E(X) into E(Y) $\mathcal{A}' \leftarrow A;$ /* Stage I: */ for $i = 1 \cdots |X|$ do if *i* has some significant match then $j \leftarrow$ smallest block in E(Y) that significantly matches i; Remove all matches incident with blocks i and j from \mathcal{A}' , and add a *perfect* match between the two blocks; end \mathbf{end} /* Stage II: */ $i \leftarrow 1$; while $i \leq |X|$ do if *i* has a partial and not perfect match then if *i* is partially matched to more than a single block then Delete from \mathcal{A}' all matches of characters from the *i*-th block of E(X); i + +;end else $j \leftarrow \text{smallest } E(Y)\text{-block that partially matches } i;$ $i' \leftarrow$ smallest E(X) that does not match with the *j*-th block; Delete from \mathcal{A}' all matches of characters from the *j*-th block of E(Y); $i \leftarrow i';$ end end end

Proof of Lemma 11 (using Claim 14 and Claim 15). Let OPT represent the normalized cost of the optimal alignment between X and Y, and OPT denote the normalized cost of the optimal alignment between E(X) and E(Y). Notice that any alignment between X and Y can be paired with an alignment between E(X) and E(Y) having the same cost. Consequently, we have: $OPT \leq OPT$. To complete the argument, it remains to establish that: $(1 - 48\varepsilon)OPT \leq OPT$, which can be achieved by demonstrating: $OPT \leq (1 + 24\varepsilon)OPT$.

Consider the optimal alignment \mathcal{A} that transforms X into Y, and let \mathcal{A}' be the alignment generated by Algorithm 1 when applied to \mathcal{A} . According to Claims 14 and 15, we obtain:

$$\frac{1}{2|E(X)|} \cdot cost(\mathcal{A}') \leq \frac{1}{2|E(X)|} \cdot (1+4\varepsilon)^2 cost(\mathcal{A}) \leq \frac{1}{2|E(X)|} \cdot (1+24\varepsilon) cost(\mathcal{A}) = (1+24\varepsilon)OPT.$$

We conclude the proof by noting that: $\widetilde{OPT} \leq \frac{1}{2|E(X)|} \cdot cost(\mathcal{A}').$

◀

3.3 Lower Bounds

Proof of Claim 8. For any value of $n \in \mathbb{N}$, define A_n as the set of length n strings composed of a single character from Γ . Clearly, $|A_n| = |\Gamma|$ and moreover, for every distinct pair of strings in A_n , their Indel distance is 2n. Now consider any embedding $E : \Gamma^* \to \Sigma^*$. Since $|A_n| = |\Gamma| > |\Sigma|$, by the pigeonhole principle there exist $X \neq Y \in A_n$ satisfying: $E(X)_1 = E(Y)_1^2$. Hence, $\Delta_{indel}(E(X), E(Y)) < 2\ell(n)$ while $\Delta_{edit}(X, Y) = 2n$.

Proof of Claim 9.

1. Fix $n \in \mathbb{N}$ and consider any embedding $E : \Gamma^* \to \Sigma^*$. For any $X \in \Gamma^n$, define the value $p(E(X)) \in \Sigma$ as the plurality value among $\{E(X)_i\}_{i \in \mathbb{N}}$ (ties are broken arbitrarily). Observe that the character p(E(X)) appears at least $\frac{\ell(n)}{|\Sigma|}$ times in the string E(X). Furthermore, for any $X, Y \in \Sigma^n$ if: p(E(X)) = p(E(Y)), then we get: $LCS(E(X), E(Y)) \geq \frac{\ell(n)}{|\Sigma|}$ and hence: $\Delta_{indel}(E(X), E(Y)) \leq \left(1 - \frac{1}{|\Sigma|}\right) 2\ell(n)$.

As in the proof of Claim 8, define A_n as the set of strings composed of a single character from Γ . Recall that $|A_n| = |\Gamma|$ and moreover, for every distinct pair of points in A_n , their *LCS* distance is 2n.

Since $|A_n| = |\Gamma| > |\Sigma|$, by the pigeonhole principle there exist $X \neq Y \in A_n$ satisfying: p(E(X)) = p(E(Y)), yielding: $\Delta_{indel}(E(X), E(Y)) \leq \left(1 - \frac{1}{|\Sigma|}\right) 2\ell(n)$, whereas $\Delta_{indel}(X, Y) = 2n$, as claimed.

2. Let $k = |E(\Lambda)|$. For $Z \in \Gamma^n$, we have: $\Delta_{indel}(E(Z), E(\Lambda)) \geq \ell(n) - k$ so $\widetilde{\Delta}_{indel}(E(Z), E(\Lambda)) \geq 1 - \frac{k}{\ell(n)} \geq 1 - \frac{k}{n}$. On the other hand, $\Delta_{indel}(Z, \Lambda) = n$ so $\widetilde{\Delta}_{indel}(Z, \Lambda) = 1$.

4 Alphabet Reduction – Binary Alphabets

In this section we show a reduction of Δ_{edit} and Δ_{indel} over an arbitrary alphabet to the binary alphabet. The reduction expands the strings super-polynomially, but one can think of it as a proof of concept that more efficient reduction might exist. The main theorem of this section is the following statement which is a formal statement of Theorem 5. For ease of presentation it is beneficial to think about Longest Common Subsequence instead of Δ_{indel} . That is how we state the theorem here.

▶ **Theorem 16.** For any integer $n \ge 1$, any alphabet Σ of size at most n^3 , there exist integers S, R, N where $N = n^{O(\log n)}$ and functions $G, H, G', H' : \Sigma^n \to \{0, 1\}^N$ such that for any $X, Y \in \Sigma^n$,

$$LCS(X,Y) = \frac{LCS(G(X), H(Y)) - R}{S}$$
$$\Delta_{edit}(X,Y) = \frac{LCS(G'(X), H'(Y)) - R}{S}$$

Hence, for any pair of strings X, Y one can recover $\Delta_{edit}(X, Y)$ from $\Delta_{indel}(G'(X), H'(Y))$ over a binary alphabet. Both mappings G, H and G', H' can be computed efficiently in the length of their output. Indeed, they will be defined explicitly below. We remark that the bound n^3 on the size of Σ is essentially arbitrary and could be replaced for example by a bound 2^n without change in the other parameters (except for multiplicative constants). However, the n^3 bound allows for hashing any large alphabet by a random pair-wise independent hash function to an alphabet of size n^3 without affecting the distance of any given pair of strings except with probability < 1/n.

² $E(X)_i$ is the *i*th character of the string E(X).

11:14 Many Flavors of Edit Distance

In order to prove the theorem we will need several auxiliary functions. We say that a 0-1 string is *balanced* if it contains the same number of 0's and 1's. We say a formula ϕ is *normalized* if it consists of alternating layers of binary AND and OR and all of its literals are at the same depth; each literal is either a constant, a variable or its negation.

We define two functions $g, h: \{0, 1\}^* \times \{\text{normalized formulas}\} \to \{0, 1\}^*$ and two threshold functions $f, t: \{\text{normalized formulas}\} \to \mathbb{N}$ as follows: Let us consider sets of variables $U = \{u_1, \ldots, u_p\}$ and $V = \{v_1, \ldots, v_q\}$, and let $A = \{a_1, \ldots, a_p\}$ where a_i is the assignment to the variable u_i for all $1 \le i \le p$, and $B = \{b_1, \ldots, b_q\}$ where b_i is the assignment to the variable v_i for all $1 \le i \le q$.

Let $\phi(U, V)$ be a normalized formula which is defined over two disjoint sets of variables $U = \{u_1, \ldots, u_p\}$ and $V = \{v_1, \ldots, v_q\}$. Let $A \in \{0, 1\}^p, B \in \{0, 1\}^q$ where A and B are interpreted as assignments for U and V respectively. We define two functions g, h, such that g gets as an input a pair (ϕ, A) and outputs a string in $\{0, 1\}^*$, similarly h takes a pair (ϕ, B) as its input and outputs a string in $\{0, 1\}^*$. We also define threshold functions f, t which take such a formula as input and output a natural number. The crux of the construction is that for any assignment A for U and B for V we have that if ϕ is satisfied by the assignment pair A, B then $LCS(g(\phi, A), h(\phi, B)) = t(\phi)$, otherwise $LCS(g(\phi, A), h(\phi, B)) = f(\phi)$

We establish the recursive definitions of g, h, f and t based on the depth of the formula. The base case is when ϕ is either a constant 0, 1 or single literals $u_i, \neg u_i, v_j, \neg v_j$, where, $u_i \in U, v_j \in V$. Here by $\neg 0$ we understand symbol 1, and similarly by $\neg 1$ we understand symbol 0.

	$\phi = u_i$	$\phi = \neg u_i$	$\phi = v_j$	$\phi = \neg v_j$	$\phi = 1$	$\phi = 0$
$g(A,\phi)$	$\neg a_i a_i$	$a_i \neg a_i$	01	0 1	01	10
$h(B,\phi)$	01	01	$ eg b_j b_j$	$b_j \neg b_j$	01	01
$t(\phi)$	2	2	2	2	2	2
$f(\phi)$	1	1	1	1	1	1

and further inductively:

	$\phi = \phi_0 \; OR \; \phi_1$	$\phi = \phi_0 \ AND \ \phi_1$
$g(A,\phi)$	$1^{k/2} 1^{4k} g(A, \phi_0) 1^{4k} 0^{4k} g(A, \phi_1) 0^{4k} 0^{k/2}$	$0^{T+F} 1^{11k+T+F} 0^{5k} g(A,\phi_0) 0^k 1^k 0^k g(A,\phi_1) 0^{5k}$
$h(B,\phi)$	$0^{k/2} 0^{4k} h(B,\phi_0) 0^{4k} 1^{4k} h(B,\phi_1) 1^{4k} 1^{k/2}$	$0^{T+F} 0^{5k} h(B,\phi_0) 0^k 1^k 0^k h(B,\phi_1) 0^{5k} 1^{11k+T+F}$
$t(\phi)$	9k + T	13k + 3T + F
$f(\phi)$	9k + F	13k + 2T + 2F.

where $k = |g(x, \phi_0)|$, $T = t(\phi_0)$, and $F = f(\phi_0)$.

Key properties of our functions are summarized in the next lemma.

▶ Lemma 17. Let $\phi(U, V)$ be a balanced formula of depth d with set of variables $U = \{u_1, \ldots, u_p\}$ and $V = \{v_1, \ldots, v_q\}$. For every two assignments $A, A' \in \{0, 1\}^p$ to variables U, we have $|g(A, \phi)| = |g(A', \phi)|$. Similarly, for every two assignments $B, B' \in \{0, 1\}^q$ to variables $V, |h(B, \phi)| = |h(B', \phi)|$. Additionally, $|g(A, \phi)| = |h(B, \phi)| \leq 30^d$.

Furthermore, the following holds:

In order to prove the above lemma we also need two gadgets which we call the AND-gadget and the OR-gadget. We need the lemmas on these gadgets (statement and proofs included in the full version of the paper) which analyze the composition of AND and OR.

The proofs of theorem 16 and lemma 17 can also be found in the full version of the paper.

— References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations* of Computer Science, FOCS 2015, pages 59–78, 2015. doi:10.1109/F0CS.2015.14.
- 2 Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, volume 107 of LIPIcs, pages 8:1-8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.8.
- 3 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In Sandy Irani, editor, 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020, pages 990–1001. IEEE, 2020. doi:10.1109/F0CS46700.2020.00096.
- 4 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing, STOC '09, pages 199–204, New York, NY, USA, 2009. ACM. doi:10.1145/1536414.1536444.
- 5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC '15, pages 51–58, New York, NY, USA, 2015. ACM. doi: 10.1145/2746539.2746612.
- 6 Ziv Bar-Yossef, TS Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In 45th Annual IEEE Symposium on Foundations of Computer Science, pages 550–559. IEEE, 2004.
- 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 *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 316–324, New York, NY, USA, 2003. ACM. doi:10.1145/780542.780590.
- 8 Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 792–801, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.
- 9 Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. Journal of the ACM (JACM), 68(3):1–41, 2021. doi:10.1145/3456807.
- 10 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 685–698. ACM, 2020. doi:10.1145/3357713.3384282.
- 11 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Kouckỳ, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *Journal of the ACM (JACM)*, 67(6):1–22, 2020. doi:10.1145/3422823.

11:16 Many Flavors of Edit Distance

- 12 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 ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 712–725, 2016. doi:10.1145/2897518.2897577.
- E. N. Gilbert. A comparison of signalling alphabets. The Bell System Technical Journal, 31(3):504-522, 1952. doi:10.1002/j.1538-7305.1952.tb01393.x.
- 14 Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. Does preprocessing help in fast sequence comparisons? In Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, pages 657–670, 2020. doi:10.1145/3357713.3384300.
- 15 Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016. doi: 10.1016/J.DAM.2015.10.040.
- 16 Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, pages 699–712. ACM, 2020. doi:10.1145/3357713.3384307.
- 17 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. SIAM J. Comput., 27(2):557–582, April 1998. doi:10.1137/S0097539794264810.
- 18 Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10(8), pages 707–710. Soviet Union, 1966.
- William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. Journal of Computer and System sciences, 20(1):18–31, 1980. doi:10.1016/0022-0000(80) 90002-1.
- 20 Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. J. ACM, 54(5):23, 2007. doi:10.1145/1284320.1284322.
- 21 Alexander Tiskin. Semi-local string comparison: Algorithmic techniques and applications. Mathematics in Computer Science, 1:571–603, 2008. doi:10.1007/S11786-007-0033-3.
- 22 Rom Rubenovich Varshamov. Estimate of the number of signals in error correcting codes. Docklady Akad. Nauk, SSSR, 117:739–741, 1957.
- 23 Robert A Wagner and Michael J Fischer. The string-to-string correction problem. Journal of the ACM (JACM), 21(1):168–173, 1974. doi:10.1145/321796.321811.