The Smoothed Complexity of Edit Distance

Alexandr Andoni[1,*] and Robert Krauthgamer[2,**]

[1] MIT
Email: andoni@mit.edu
[2] Weizmann Institute and IBM Almaden
Email: robert.krauthgamer@weizmann.ac.il

Abstract. We initiate the study of the smoothed complexity of sequence alignment, by proposing a semi-random model of edit distance between two input strings, generated as follows. First, an adversary chooses two binary strings of length d and a longest common subsequence A of them. Then, every character is perturbed independently with probability p, except that A is perturbed in exactly the same way inside the two strings.

We design two efficient algorithms that compute the edit distance on smoothed instances up to a constant factor approximation. The first algorithm runs in near-linear time, namely $d^{1+\varepsilon}$ for any fixed $\varepsilon > 0$. The second one runs in time sublinear in d, assuming the edit distance is not too small. These approximation and runtime guarantees are significantly better then the bounds known for worst-case inputs, e.g. near-linear time algorithm achieving approximation roughly $d^{1/3}$, due to Batu, Ergün, and Sahinalp [SODA 2006].

Our technical contribution is twofold. First, we rely on finding matches between substrings in the two strings, where two substrings are considered a match if their edit distance is relatively small, a prevailing technique in commonly used heuristics, such as PatternHunter of Ma, Tromp and Li [Bioinformatics, 2002]. Second, we effectively reduce the smoothed edit distance to a simpler variant of (worst-case) edit distance, namely, edit distance on permutations (a.k.a. Ulam's metric). We are thus able to build on algorithms developed for the Ulam metric, whose much better algorithmic guarantees usually do not carry over to general edit distance.

1 Introduction

The *edit distance* (aka *Levenshtein distance*) between two strings is the number of insertions, deletions, and substitutions needed to transform one string into the other. This distance is of key importance in several fields, such as computational biology and text processing, and consequently computational problems involving the edit distance were studied extensively, both theoretically and experimentally, see e.g. the detailed survey on edit distance by Navarro [Nav01]. Despite extensive research, the worst-case guarantees currently known for algorithms dealing with edit distance are quite poor, especially in comparison to the Hamming distance (which is just the number of substitutions to transform one string into the other).

* Work done in part while visiting IBM Almaden Research Center.
** Work supported in part by a grant from the Fusfeld Research Fund.

The most basic problem is to compute the edit distance between two strings of length d over alphabet Σ. The worst-case running time known for this problem has not improved in three decades — the problem can be solved using dynamic programming in time $O(d^2)$ [WF74], and in time $O(d^2/\log^2 d)$ when the alphabet has constant size [MP80]. Unfortunately, such near-quadratic time is prohibitive when working on large datasets, which is common in areas such as computational biology. The gold standard is to achieve a linear-time algorithm, or even sublinear in several cases, which has triggered the study of very efficient *distance estimation* algorithms – algorithms that compute an approximation to the edit distance. In particular, the best quasi-linear time algorithm, due to Batu, Ergün, and Sahinalp [BES06], achieves $d^{1/3+o(1)}$ approximation (improving over [BJKK04]), and the only known sublinear time algorithm, due to Batu, Ergün, Kilian, Magen, Raskhodnikova, Rubinfeld and Sami [BEK$^+$03], decides whether the edit distance is $O(d^\alpha)$ or $\Omega(d)$ in time $O(d^{\max\{\alpha/2,1-2\alpha\}})$. In fact, distance estimation with sublogarithmic approximation factor was recently proved impossible in a certain model of low communication complexity [AK07]. In practice, this situation is mitigated by heuristic algorithms. In computational biology settings for instance, tools such as BLAST [AGM$^+$90] are commonly used to solve the problem quickly, essentially by relying on heuristic considerations that sacrifice some sensitivity.

We initiate the study of the smoothed complexity of sequence alignment, by proposing a semi-random model of edit distance (the input is a worst-case instance modified by a random perturbation), and design for it very efficient approximation algorithms. Specifically, an adversary chooses two strings and a longest common subsequence of them, and every character is perturbed independently with probability $0 \le p \le 1$, except that every character in the common subsequence is perturbed in the same way in the two strings. Semi-random models appeared in the literature in other contexts, but to the best of our knowledge, not for sequence alignment problems; see Sect. 1.2 for more details. Our algorithms for the smoothed model approximate the edit distance within a constant factor in linear, and even sublinear time.

Why study semi-random models of sequence alignment? First, they elude the extreme difficulty posed by worst-case inputs, while avoiding the naivete of average-case (random) inputs. Using these models as a theoretical testbed for practical algorithms may lead to designing new algorithmic techniques, and/or to providing rigorous explanation for the empirical success of well-known heuristics. Second, studying algorithms for semi-random models may be viewed as an attack on the worst-case complexity. It is difficult to quantify the progress we manage to make in this direction, but we certainly achieve much better performance guarantees on a very large collection of inputs (including random inputs as an extreme case), by delineating rather general assumptions on the input, under which we have efficient algorithms.

1.1 Our Contribution

A smoothed model. Let $0 < p \le 1$ be a *perturbation probability*. In our smoothed model for edit distance, an input consisting of two strings, x and y, is generated as follows. (A more formal description is given in Sect. 1.3.)

1. An adversary chooses two strings $x^*, y^* \in \{0,1\}^d$, and a longest common subsequence $\mathcal{A}$ of x^*, y^*.
2. Every character in x^* and y^* is replaced independently with probability p by a random bit, except that the perturbation of $\mathcal{A}$ inside x and that of $\mathcal{A}$ inside y are identical.

Results. We start by investigating the typical properties of a smoothed instance (x, y), including proving that the expected edit distance $\mathrm{ed}(x, y)$ is comparable to that of the generating strings, $\mathrm{ed}(x^*, y^*)$.

Our first result is a deterministic algorithm that approximates the edit distance within a constant factor, and its smoothed runtime complexity is near-linear. Specifically, for any desired $0 < \varepsilon < 1$, the algorithm always obtains $O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$ approximation, and with high probability over the randomness in the smoothing, runs in time $O(d^{1+\varepsilon})$. For comparison, the algorithm of Batu, Ergün, and Sahinalp [BES06] for worst-case inputs requires a similar running time of $O(d^{1+\varepsilon})$ and achieves approximation $d^{(1-\varepsilon)/3+o(1)}$.

Our second result is a sublinear time algorithm for smoothed instances. Specifically, for every desired $0 < \varepsilon < 1$, the algorithm computes a $O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$ approximation to $\mathrm{ed}(x, y)$ in time $O(d^{1+\varepsilon}/\sqrt{\mathrm{ed}(x, y)})$. For comparison, recall that the algorithm of Batu et al. [BEK+03] for worst-case inputs can only distinguish a polynomially large gap in the edit distance, and only at the highest regime $\Omega(d)$.

Techniques. Our algorithms are based on two new technical ideas. The first one is to find *matches* of blocks (substrings) of length $L = O(\frac{1}{p} \log d)$ between the two strings, where two blocks are considered a match if they are at a small edit distance (say εL). This same idea, but in a more heuristic form, is used by practical tools. In particular, PatternHunter [MTL02] uses such a notion of matches (to identify "seeds"), significantly improving over BLAST [AGM+90], which considers only identical blocks to be a match. Thus, our smoothed analysis may be viewed as giving some rigorous explanation for the empirical success of such techniques.

The second idea is to reduce the problem to edit distance on permutations (in worst-case), called in the literature *Ulam's distance*, or the *Ulam metric*. Here and throughout, a *permutation* is a string in which every symbol appears at most once.[3] The Ulam metric is a submetric of edit distance, but the algorithmic bounds known for it are significantly better than those for the general edit distance. In particular, Ulam's distance between permutations of length d can be computed in linear time $O(d \log d)$, e.g. using Patience Sorting. The main challenge we overcome is to design a reduction that distorts distances by at most a constant factor. Indeed, there is an easy reduction with distortion $L = O(\frac{1}{p} \log d)$, that follows simply because with high probability, in each string, the blocks of length L are all distinct, see [CK06, Section 3.1].

[3] It is sometimes convenient, though not crucial, to use an alphabet Σ with size larger than d. We then define a permutation as a string whose characters are all distinct.

1.2 Related Work

Average-case analysis of edit distance. Random models for edit distance were studied in two contexts, for pattern matching and for nearest neighbor searching. In the former, the text is typically assumed to be random, i.e., each character is chosen uniformly and independently from the alphabet, and the pattern is usually not assumed to be random. We refer the reader to the survey [Nav01, Section 5.3] for details and references. For nearest neighbor search, the average-case model is quite similar, see [NBYST01,GP06].

Our model is considerably more general than the random strings model. In particular, the average-case analysis often relies on the fact that no short substring of the text is identical to any substring of the pattern, to quickly "reject" most candidate matches. In fact, for distance estimation, it is easy to distinguish the case of two random strings from the case of two (worst-case) strings at a smaller edit distance — just choose one random block of logarithmic length in the first string and check whether it is close in edit distance to at least one block in the second string. We achieve a near-linear time algorithm for a more adversarial model, albeit by allowing constant factor approximation.

Smoothed complexity and semi-random models. Smoothed analysis was pioneered by Spielman and Teng [ST04] as a framework aimed to explain the practical success of heuristics that do not admit traditional worst-case analysis. They analyzed the simplex algorithm for linear programming, and since then researchers investigated the smoothed complexity of several other problems, mostly numerical ones, but also some discrete problems. An emerging principle in smoothed analysis is to perform *property-preserving* perturbation [ST03], example of which is our model. Specifically, our model may be seen as performing a perturbation of x^* and y^* that preserves the common subsequence $\mathcal{A}$.

In combinatorial optimization problems, smoothed analysis is closely related to an earlier notion of semi-random models, which were initiated by Blum and Spencer [BS95]. This research program encompasses several interesting questions, such as *what algorithmic techniques are most effective (spectral methods?)*, and *when is the optimum solution likely to be unique, hard to find, or easy to certify*, see e.g. [FM97,FK01] and the references therein.

To the best of our knowledge, smoothed analysis and/or semi-random models were not studied before for sequence alignment problems.

Distance estimation. Algorithms for distance estimation are studied also in other scenarios, using different notions of efficiency. One such model is the communication complexity model, where two parties are each given a string, and they wish to estimate the distance between their strings using low communication. The sketching model falls into this category, with further restriction to simultaneous communication protocols. A communication lower bound was recently proved in [AK07] for the edit distance metric, even on permutations, and it holds for approximations as large as $\Omega(\log d/ \log\log d)$.

1.3 Preliminaries

Strings. Let x be a string of length d over alphabet Σ. A *position* in the string is an index $i \in [d]$. We write $x[i]$ or x_i to denote the symbol appearing in position i

in x. Let $[i : j]$ denote the sequence of positions $(i, i+1, \ldots, j)$. We write $x[i : j]$ or $x_{[i:j]}$ for the corresponding substring of x. A *block* is a substring, often of a predetermined length.

A variant of edit distance. Let x, y be two strings. Define $\underline{\mathrm{ed}}(x, y)$ to be the minimum number of character insertions and deletions needed to transform x into y. Character substitution are not allowed, in contrast to $\mathrm{ed}(x, y)$, but a substitution can be simulated by a deletion followed by an insertion, and thus $\mathrm{ed}(x, y) \leq \underline{\mathrm{ed}}(x, y) \leq 2\,\mathrm{ed}(x, y)$. Observe that

$$\underline{\mathrm{ed}}(x, y) = |x| + |y| - 2\,\mathrm{LCS}(x, y),$$

where $\mathrm{LCS}(x, y)$ is the length of the longest common subsequence of x and y.

Alignments. For two strings x, y of length d, an *alignment* is a function $A : [d] \to [d] \cup \{\perp\}$ that is monotonically increasing on $A^{-1}([d])$ and satisfies $x[i] = y[A(i)]$ for all $i \in A^{-1}([d])$. Define the *length* (or size) of the alignment as $\mathrm{len}(A) = |A^{-1}([d])|$, i.e., the number of positions in x that are matched by A. Let the *cost* of A be $\mathrm{cost}(A) = 2(d - \mathrm{len}(A)) = 2|A^{-1}(\perp)|$, i.e. the number of positions in x and in y that are not matched by A. Observe that an alignment between x and y corresponds exactly to a common subsequence to x and y. Thus, if A is an alignment between x and y, then

$$\mathrm{cost}(A) = 2(d - \mathrm{len}(A)) \geq 2d - 2\,\mathrm{LCS}(x, y) = \underline{\mathrm{ed}}(x, y),$$

with equality if and only if A is an alignment of maximum length.

Block matches. Consider two strings x, y and a block length $L \in [d]$. For blocks $x_{[i:i+L-1]}$ and $y_{[j:j+L-1]}$ of length L, we let $\mathrm{ed}_A(x_{[i:i+L-1]}, y_{[j:j+L-1]})$ be the number of positions $k \in [i : i+L-1]$ such that $A(k) \notin [j : j+L-1]$. We let $\mathrm{match}(x_{[i:i+L-1]})$ denote the block $y_{[j:j+L-1]}$, where $j \in [d-L+1]$ minimizes $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]})$, breaking ties arbitrarily. For an alignment A between x and y, let $\mathrm{match}_A(i, L)$ be the block $y_{[j:j+L-1]}$, where $j \in [d-L+1]$ minimizes $\mathrm{ed}_A(x_{[i:i+L-1]}, y_{[j:j+L-1]})$. Slightly abusing notation, we sometimes let match and match_A represent the corresponding position j (instead of the substring $y_{[j:j+L-1]}$), but the distinction will be clear from the context.

Smoothed model. Let $0 \leq p \leq 1$, let $x^*, y^* \in \{0,1\}^d$ be two strings, and fix a maximum-length alignment A^* between x^* and y^*. Let $x, y \in \{0,1\}^d$ be the strings obtained from x^*, y^* respectively, by replacing, independently with probability p, each character with a random one, except that the positions aligned by A^* are kept correlated. Formally, let $\pi_x \in \{0,1\}^d$ be a string where each $\pi_x[j]$ is drawn independently to be 1 with probability $p/2$ and 0 otherwise, and let π_y be defined similarly (and independently), except for position $j \in A^*([d])$, for which we set $\pi_y[j] = \pi_x[(A^*)^{-1}(j)]$. Now let $x[i] = x^*[i] + \pi_x[i]$ and $y[i] = y^*[i] + \pi_y[i]$, where addition is done modulo 2. We call the pair (x, y) a *smoothed instance* of edit distance, and denote its distribution by $\mathrm{SMOOTH}_p(x^*, y^*, A^*)$.

2 Typical properties of smoothed instances

We first show that the edit distance of a smoothed instance is likely to be similar to that of the strings used to generate it. We then turn our attention to the distance between different substrings of the smoothed strings x and y. Specifically, we show that blocks of length $L = O(p^{-1} \log d)$ are likely to be far from each other in terms of edit distance, with the few obvious exceptions of overlapping blocks and blocks that are aligned via the original alignment A^*.

Besides the inherent interest, these bounds are useful in the smoothed analysis of our algorithms carried out in subsequent sections.

2.1 Edit Distance of a Smoothed Instance

Theorem 2.1. *Let A^* be an optimal alignment between $x^*, y^* \in \{0,1\}^d$, and fix $0 < p \leq 1$. Then a smoothed instance $(x, y) \in \mathrm{SMOOTH}_p(x^*, y^*, A^*)$ satisfies*

$$\Pr_{(x,y)} \left[\Omega(\tfrac{p}{\log(1/p)}) \operatorname{ed}(x^*, y^*) \leq \operatorname{ed}(x, y) \leq \operatorname{ed}(x^*, y^*) \right] \geq 1 - 2^{-\Omega(p) \operatorname{ed}(x^*, y^*)}.$$

Proof. Observe that $\operatorname{ed}(x, y) \leq \operatorname{ed}(x^*, y^*)$ always holds (i.e. with probability 1). We proceed to show that with high probability, $\underline{\operatorname{ed}}(x, y) \geq \Omega(\frac{p}{\log(1/p)}) \cdot \underline{\operatorname{ed}}(x^*, y^*)$, which by Sect. 1.3 would complete the proof. We let U denote the unaligned positions in x under A^*, i.e. $U = (A^*)^{-1}(\perp)$ and $|U| = \frac{1}{2} \underline{\operatorname{ed}}(x^*, y^*)$.

Consider a *potential alignment* A between x and y, i.e. a map $A : [d] \mapsto [d] \cup \{\perp\}$ that is monotonically increasing on $A^{-1}([d])$, and suppose that $\operatorname{cost}(A) = 2|A^{-1}(\perp)|$ is at most $\alpha \cdot \underline{\operatorname{ed}}(x^*, y^*)$ for a small $0 < \alpha \leq 1/4$ to be chosen later. For A to be an actual alignment, we must additionally have that $x[i] = y[A(i)]$ for every position $i \notin A^{-1}(\perp)$, and in particular for every position $i \in U \setminus A^{-1}(\perp)$. The number of such positions is at least $|U| - |A^{-1}(\perp)| \geq \frac{1}{2} \underline{\operatorname{ed}}(x^*, y^*) - \frac{1}{2}\alpha \, \underline{\operatorname{ed}}(x^*, y^*) \geq \frac{1}{4} \underline{\operatorname{ed}}(x^*, y^*)$. For each of them, $x^*[i]$ is perturbed independently of $y^*[A(i)]$, and thus $x[i] \neq y[A(i)]$ occurs with probability at least $p/2$. These events might not be mutually independent due to correlations via A^*, but it is easy to see that for at least half of such i, the probability is at least $p/2$ even when conditioned on earlier events (namely, $x[i]$ is independent of $x[i']$ and $y[A(i')]$ for all $i' < i$). Thus, the probability that A is an actual alignment is at most

$$\Pr \left[x[i] = y[A(i)] \text{ for all } i \in U \setminus A^{-1}(\perp) \right] \leq \left(1 - \tfrac{p}{2}\right)^{\underline{\operatorname{ed}}(x^*, y^*)/8} \leq e^{-p \cdot \underline{\operatorname{ed}}(x^*, y^*)/16}.$$

We will apply a union bound on all potential alignments, and thus it suffices to have an upper bound on the number of different values taken by $A_{|U}$, the restriction of A to the positions in U. We note that $A_{|U}$ is determined by the number of insertions and deletions occurring between every two successive positions in U (including the insertions and deletions before the first position in U and after the last position in U). Thus we can count the number of $A_{|U}$ as:

$$\#\{A_{|U}\} \leq \binom{|U| + \frac{1}{2}\alpha \, \underline{\operatorname{ed}}(x^*, y^*)}{\frac{1}{2}\alpha \, \underline{\operatorname{ed}}(x^*, y^*)}^3 \leq \left(\tfrac{e(1+\alpha)}{\alpha}\right)^{1.5\alpha \, \underline{\operatorname{ed}}(x^*, y^*)} \leq (\tfrac{1}{\alpha^2})^{1.5\alpha \, \underline{\operatorname{ed}}(x^*, y^*)}.$$

Choosing $\alpha = \frac{cp}{\log(1/p)}$ for a sufficiently small constant $c > 0$, we get by a union bound $\Pr \left[\underline{\operatorname{ed}}(x, y) \leq \alpha \, \underline{\operatorname{ed}}(x^*, y^*) \right] \leq e^{[3\alpha \ln(1/\alpha) - p/16] \cdot \underline{\operatorname{ed}}(x^*, y^*)} \leq e^{-(p/32) \, \underline{\operatorname{ed}}(x^*, y^*)}$.
□

2.2 Edit Distance Between Different Blocks

Our next lemma concerns typical distances between two blocks from x and y. The main technical difficulty in this lemma, beyond technique used to prove Theorem 2.1, is that here we consider blocks whose perturbations are correlated, e.g. overlapping blocks in the same string, thus impeding direct concentration bounds. The proof of this lemma is deferred to the full version of the article.

Lemma 2.1. *Let A^* be an optimal alignment between $x^*, y^* \in \{0,1\}^d$ and fix $0 < p \leq 1$. Let $L \geq \frac{C}{p} \log d$ for a sufficiently large constant $C > 0$, and let $c_a, c_b, c_c > 0$ be sufficiently small constants. Then with probability at least $1 - d^{-\Omega(C)}$, a smoothed instance $(x, y) \in \mathrm{SMOOTH}_p(x^*, y^*, A^*)$ satisfies the following for all $i, j \in [d]$:*

(a). $\mathrm{ed}(x_{[i:i+L-1]}, x_{[j:j+L-1]}) \geq c_a \cdot \min\{pL, |j-i|\}$*, and similarly in y.*
(b). *If* $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j+L-1]}) \geq L/4$*, then* $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \geq c_b \cdot pL$.
(c). *Let $k^* = \mathrm{match}_{A^*}(i, L)$, then*

$$\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \geq \min\{c_c \cdot pL, c_c \cdot |j-k^*| - \mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[k^*:k^*+L-1]})\}.$$

Furthermore, if $|j - k^*| \geq L$*, then* $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \geq c_c \cdot pL$.

3 Near-Linear Time Distance Estimation

Our first algorithm is guaranteed to give a correct answer for any input strings, but has an improved runtime for smoothed inputs, coming from a distribution $\mathrm{SMOOTH}_p(x^*, y^*, A^*)$.

Theorem 3.1. *For every $\varepsilon > 0$ and $p > 0$ there is a deterministic algorithm that, given as input two strings $x, y \in \{0,1\}^d$, approximates $\mathrm{ed}(x, y)$ within factor $O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$, and on a p-smoothed instance, with high probability its running time is $O(d^{1+\varepsilon})$.*

We will need three lemmas, the first two of which do not deal directly with smoothed instances and may be useful in other scenarios as well.

Lemma 3.1. *Consider a bipartite graph $G = ([d], [d], E)$, and call two edges $(i, j) \in E$ and $(k, l) \in E$* intersecting *if $(i-k)(j-l) \leq 0$. Then a maximum-cardinality subset of non-intersecting edges can be found in time $O(d + |E| \log d)$ by reducing the problem to Patience Sorting.*

Proof (sketch). Construct a string z of length $|E|$ as follows. Start with an empty string. For each node $i = 1, \ldots, d$, append to the end of z the list of symbols $(j_1, -i), \ldots, (j_k, -i)$ where $j_1 > j_2 > \ldots > j_k$ are the neighbors of i, i.e. $\{j_1, \ldots j_k\} = \{j \mid (i, j) \in E\}$. Then the longest increasing subsequence of z (when the order on $(j, -i)$ is lexicographic) gives a maximum-size subset of non-intersecting edges. We can find it using Patience Sorting. $\square$

Lemma 3.2. *Fix an optimal alignment A between two strings $x, y \in \{0,1\}^d$. Let $L \in [d]$ divide d. Partition x into successive blocks of length L, denoted $(X_i)_{i=1}^{d/L}$, and let $Y_i = \mathrm{match}_A(X_i)$. Then $\sum_{i=1}^{d/L} \mathrm{ed}_A(X_i, Y_i) \le 2\,\mathrm{ed}(x,y)$. Hence, for every $\varepsilon > 0$, the number of indices $i \in [d/L]$ such that $\mathrm{ed}(X_i, Y_i) > \varepsilon L$ is at most $\frac{4}{\varepsilon L} \cdot \mathrm{ed}(x,y)$.*

Proof (sketch). For $i \in [d/L]$, let MIN_i and MAX_i, respectively, be the positions of the first and last aligned symbol in X_i, i.e., $MIN_i = \min\{j \in [iL-i+1 : iL] \mid A(j) \in [d]\}$ and similarly for MAX_i. Let u_i^x be the number of unaligned positions in $X_i = x_{[iL-L+1:iL]}$, i.e. $u_i^x = |\{j \in [iL-L+1 : iL] \mid A(j) = \perp\}|$. Also, let u_i^y be the number of unaligned position in $y[A(MIN_i) : A(MAX_i)]$.

If $A(MAX_i) - A(MIN_i) < L$, then $\mathrm{ed}_A(X_i, Y_i) = u_i^x$. If $A(MAX_i) - A(MIN_i) \ge L$, then $\mathrm{ed}_A(X_i, Y_i) \le u_i^x + u_i^y$. Observing that each of $\sum_i u_i^x$ and $\sum_i u_i^y$ is bounded by $\mathrm{ed}(x,y)$ proves the first part. The second part follows immediately because for such blocks $\mathrm{ed}_A(X_i, Y_i) \ge \frac{1}{2}\,\mathrm{ed}(X_i, Y_i) > \frac{1}{2}\varepsilon L$. □

Lemma 3.3. *Let $C > 1$ and $0 < c' < 1$ be sufficiently large and sufficiently small constants, respectively, and let $L = \frac{C}{p}\log d$. Let A^* be a maximum-length alignment between $x^*, y^* \in \{0,1\}^d$. Then for every $i \in [d]$ there is $j_i^* \in [d]$ such that, for $(x,y) \in \mathrm{SMOOTH}_p(x^*, y^*, A^*)$, with probability at least $1 - d^{-2}$, for all j with $|j - j_i^*| > L$ we have $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) > c'pL$.*

Proof. Take $j_i^* = \mathrm{match}_{A^*}(x^*_{[i:i+L-1]})$. If $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j_i^*:j_i^*+L-1]}) < L/4$, then for all j with $|j - j_i^*| > L$ we have $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j+L-1]}) \ge L - L/4$. Otherwise, for all j we have $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j+L-1]}) \ge L/4$. In both cases the conclusion results by applying Lemma 2.1(b). □

Proof (of Theorem 3.1). We use as a building block a near neighbor (NN) data structure under edit distance, defined as follows. Preprocess a database of m strings each of length L, so that given a query string, the algorithm returns *all* database strings at distance $\le \varepsilon L$ from the query. We will construct such data structure at the end, and for now assume it can be implemented with preprocessing $P(m, L)$ and query time $Q(m, L) + O(|output|)$, where *output* is the list of points reported by the query.

Let $C > 1$ and L be as in Lemma 3.3 and assume $\varepsilon < c'p$. Our algorithm proceeds in two stages. The first one uses the NN data structure to find, for each position in x, a few "candidate matches" in y, presumably including the correct match (under optimal alignment) for a large fraction of positions in x. The second stage views the candidate matches between positions in x and in y as the edge-set E of a bipartite graph and applies the algorithm from Lemma 3.1, thereby reconstructing an alignment.

Let us describe the algorithm in more detail. The first stage builds an NN data structure on all the substrings of length L in y. Then, it partitions x into successive blocks $x_{[iL-L+1:iL]}$, and for each such block, queries the NN data structure to identify all blocks in y that are within distance εL. For each such block in y, collect all the character matches between the two blocks, i.e., every zero in the block in x with every zero in the block in y, and same for ones. Let

E be the resulting list of all candidate matches. The second stage simply applies Lemma 3.1 to this list E to retrieve an alignment between x and y. The reported approximation to $\mathrm{ed}(x, y)$ is then twice the cost of this alignment.

Next we argue the correctness of the algorithm. Consider an optimal alignment A between x and y. Lemma 3.2 guarantees that for all but $4\,\mathrm{ed}(x, y)/\varepsilon L$ blocks from x, there exists a corresponding block $y_{[s_i:s_i+L-1]}$ at distance $\leq \varepsilon L$. Since the algorithm detects all pairs of blocks at distance $\leq \varepsilon L$, the lemma implies that all but $O(\frac{1}{\varepsilon})\,\mathrm{ed}(x, y)$ of aligned pairs from the alignment A will appear in the list of candidate matches. The algorithm will then compute an alignment A' that has at least $d - O(\frac{1}{\varepsilon})\,\mathrm{ed}(x, y)$ aligned pairs. Concluding, the algorithm will output a distance D such that $\mathrm{ed}(x, y) \leq D \leq O(\frac{1}{\varepsilon})\,\mathrm{ed}(x, y)$.

Next we show that, with high probability, the running time of the algorithm is $O(dL \log d + P(d, L) + \frac{d}{L} \cdot Q(d, L))$. Indeed, by Lemma 3.3, for each query block $x_{[iL-L+1:iL]}$, only blocks $y_{[j:j+L-1]}$ for $|j - j^*_{iL-L+1}| \leq L$ can be at distance εL. Thus, for each position in $x_{[iL-L+1:iL]}$, we have at most $3L$ candidate matches, hence $|E| \leq O(dL)$. We can now conclude that the first stage runs in $O(P(d, L)+ d/L \cdot (Q(d, L) + L^2))$, and the second stage runs in $O(|E| \log d) = O(dL \log d)$.

Finally, it remains to describe the NN data structure. We achieve $P(m, L) = m \log m \cdot 2^{L \cdot O(\varepsilon \log 1/\varepsilon)}$ preprocessing and $Q(m, L) = O(L)$ query time. The data structure simply prepares all answers in advance: for each string σ in the database and every string τ at edit distance $\leq \varepsilon L$ from σ, store the pair (σ, τ) in a trie keyed by τ. To query a string q, the algorithm accesses the trie using q as the key, and for every pair (η, q) returned by the trie, it reports the string η. Recall that a trie with t strings of length L, has query time $O(L)$, and preprocessing time $O(tL \log t)$. Thus, $Q(m, L) \leq O(L)$ and since there are at most $\binom{2L}{\varepsilon L}^3$ strings at edit distance $\leq \varepsilon L$ from a given string,

$$P(m, L) \leq O(m \log m \cdot \binom{2L}{\varepsilon L}^4 \cdot L) \leq m \log m \cdot 2^{L \cdot O(\varepsilon \log(1/\varepsilon))}.$$

The overall running time becomes $d^{1+O(p^{-1}\varepsilon \log(1/\varepsilon))}$ for $O(1/\varepsilon)$ approximation. To complete the proof, apply the above to $\varepsilon' = \Theta(\varepsilon p/ \log \frac{1}{p\varepsilon})$. The resulting running time is $d^{1+\varepsilon}$ and the approximation is $O(1/\varepsilon') = O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$. □

4 Sublinear Time Distance Estimation

We now present a sublinear time algorithm that estimates the edit distance of a smoothed instance (x, y) within a constant factor. Full proof of the following theorem is deferred to the full version of this paper.

Theorem 4.1. *For every $\varepsilon > 0$ there is a randomized algorithm that, given as input $(x, y) \in \mathrm{SMOOTH}_p(x^*, y^*, A^*)$, approximates $\mathrm{ed}(x, y)$ within factor $O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$ in time $O(d^{1+\varepsilon}/\sqrt{\mathrm{ed}(x, y)})$, with success probability at least $1 - d^{-2}$ (over the randomness in the smoothing operation and the algorithm's coins).*

The high-level approach is to map the smoothed instance (x, y) to a pair of permutations (P, Q), such that the edit distance between x and y is approximately equal to the Ulam distance between P and Q. We can then estimate the Ulam distance between P and Q using an off-the-shelf sublinear algorithm for estimating Ulam distance. Specifically, we use the following algorithm of [AIK08].

Theorem 4.2 ([AIK08]). *There exists a randomized algorithm that, given access to two permutations* P, Q *of length* d*, approximates* $\mathrm{ed}(P, Q)$ *up to a constant factor in time* $\tilde{O}(d/\sqrt{\mathrm{ed}(P,Q)})$*, with success probability at least* $2/3$*.*

The first key observation is that every algorithm for Ulam distance estimation can work independently of the actual names of symbols it reads from P, Q. Specifically, when the algorithm queries one character, say position i in P, it suffices to know whether it is identical to a previously queried character $Q[j]$, and vice versa. This observation can be leveraged in the following way: if at the time that Ulam algorithm asks to query $P[i]$, the matching character $Q[j]$ (i.e. position j such that $P[i] = Q[j]$) was not queried yet, then we may "delay" revealing the actual symbol $P[i]$ until $Q[j]$ is queried (if at all, as the running time is sublinear). Hence, for the sake of analysis we may decide (by relabeling symbols) that Q is a fixed permutation, say the identity ($Q[j] = j$ for all $j \in [d]$). In the sequel, P, Q will be permutation of length d over the alphabet $\Sigma = [2d]$.

Our construction of P, Q is based on the following principle. Let A be an alignment between x and y. Then we can construct P (while Q is the identity) so that A is the *optimal* alignment between P and Q, as follows: set $P[i] = Q[A(i)]$ whenever $A(i) \in [d]$, and set $P[i] = d+i$ whenever $A(i) = \perp$. To be useful for our sublinear algorithm (when x, y is a smoothed instance), the alignment A must have cost $O(\mathrm{ed}(x, y))$, and furthermore it has to be computable "on the fly". More precisely we require that, for queried positions i, j in P, Q respectively, if $A(i) = j$, then we can detect this by only querying $x[i]$, possibly together with a small local neighborhood around $x[i]$ and around $y[j]$ (in particular, the question whether $A(i) \stackrel{?}{=} j$ is independent of the rest of the strings x, y). We term this property "locality"; ensuring locality of the alignment A we construct is the main technical part of the proof of the theorem. We note that, for worst-case strings (x, y), constructing a near-optimal alignment A that satisfies the locality property seems hard; for a smoothed instance, on the other hand, we show this is possible, due to, in part, Lemma 2.1 and a stronger version of Lemma 3.2. For the sake of presentation, we show how to construct P directly.

4.1 Reducing Smoothed Instances to Ulam's Metric

We proceed to show how to efficiently translate a smoothed instance of edit distance into an instance of Ulam's distance, while distorting the distance by only a constant factor.

As mentioned above we set Q to be the identity permutation, and construct P as a function of x and y. (We now define the entire permutation P, even though only a sublinear portion of it will be queried by the algorithm.) The basic idea appears simple. First, we partition P into blocks of length $L = O(\frac{1}{p} \log d)$. Then, each such block $x_{[i:i+L-1]}$ we match to its closest block in y, say $y_{[j:j+L-1]}$, and define $P_{[i:i+L-1]}$ based on $Q_{[j:j+L-1]}$ and $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]})$ only. Namely, it is such that $\mathrm{ed}(P_{[i:i+L-1]}, Q_{[j:j+L-1]}) = \mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]})$. One difference from our earlier high-level description using A is that we work at the level of blocks, not single characters. But the main problem we now face is that some characters may repeat in P, because the blocks we match against in y may have overlaps. Once we fix this issue, we can apply a form of Lemma 3.2 to argue that $\mathrm{ed}(P, Q)$ is approximately $\mathrm{ed}(x, y)$. Unfortunately, a straightforward fix to the above issue would introduce dependencies between different blocks in P, violating the locality requirement. We thus need additional transformations of P, under which each block can locally certify it does not interfere with other blocks.

Lemma 4.1 (Reduction Lemma). *Fix $\varepsilon > 0$, $0 < p < 1$, and and an optimal alignment A^* between strings x^*, y^*. Let $(x, y) \in \mathrm{SMOOTH}_p(x^*, y^*, A^*)$ and let $L = \frac{C}{p} \log d$ for a large constant $C > 0$. Then there exists two permutations P and $Q = (1, 2, \ldots d)$ such that, with high probability, the following hold:*
Distance. $\Omega(1) \cdot \mathrm{ed}(x, y) \le \mathrm{ed}(P, Q) \le O(\frac{\log 1/p}{p} + \frac{1}{\varepsilon p}) \cdot \mathrm{ed}(x, y)$.
Locality. *For $k \in [d/L]$, $j \in [kL - L + 1 : kL]$, and $s_k = \mathrm{match}(x_{[kL-L+1:kL]})$,*
- *$P[j]$ can be computed from only s_k, $x_{[kL-2L+1:kL+L-1]}$, and $y_{[s_k-4L+1:s_k+5L-1]}$, in time $O(L^3)$.*
- *Unless $P[j] = d + j$, we have: $P[j] \in [d]$, $\mathrm{ed}(x_{[kL-L+1:kL]}, y_{[s_k:s_k+L-1]}) \le \varepsilon pL$ and $\mathrm{ed}(x_{[kL-L+1:kL]}, y_{[z:z+L-1]}) > \Omega(pL)$ for all z s.t. $|z - s_k| \ge 2L$.*

Next we describe the construction of the permutation P, deferring the proof of the distance and locality properties to the full version of the paper.

Proof (Sketch). Some positions in P will be "invalidated", which means that we set $P[j] = d + j$ for such a position j. However for the other positions we will have $P[j] \in [d]$. We construct P in three stages: first we define a permutation P^1, then we invalidate some of the positions in P^1 to obtain P^2, and again invalidate more positions to obtain the final P.

Let $L = \frac{C}{p} \log d$ denote the block length. Partition x into d/L blocks of length L, called X_k, and for each $k \in [d/L]$, let $Y_k = \mathrm{match}(X_k)$. Let M be the set of k's such that $\mathrm{ed}(X_k, Y_k) \le \varepsilon pL$. Let s_k be the starting position of Y_k and let $c_k = \mathrm{ed}(X_k, Y_k)$.

We construct P^1 by setting, for every $k \in M$, $P^1_{[kL-L+1:kL]}$ to be equal to the block $Q[s_k : s_k + L - 1]$, except that the first c_k symbols are invalidated (and thus $\mathrm{ed}(P^1_{[kL-L+1:kL]}, Q_{[s_k:s_k+L-1]}) = c_k$). For $k \in [d/L] \setminus M$, we simply invalidate the entire block $P^1_{[kL-L+1:kL]}$.

In the second stage, we construct P^2 from P^1. We start by defining a set $F \subseteq M$. For $k \in M$, $k \ge 1$, consider a block X_k and the matching Y_k. We put k into F iff either of the following holds: (**i**) $k - 1 \notin M$, or (**ii**) $k - 1 \in M$ and $s_k - s_{k-1} > 2L$. We obtain P^2 by invalidating all blocks $P^1_{[s_k:s_k+L-1]}$ for $k \in F$.

In the third stage, to obtain from P^2 a permutation P, we invalidate all positions $j \in [d]$ such that $P^2[j]$ occurs also somewhere else in P^2 (all such symbols are invalidated concurrently). □

5 Conclusions

It seems challenging to obtain a distance estimation algorithm whose smoothed running time is quasi-linear, i.e. $d \cdot \log^{O(1)} d$, or whose approximation is independent of the smoothing parameter p at the expense of only $O(1/p)$ increase in the runtime. Perhaps more important is to extend the smoothed analysis framework to other problems, such as nearest neighbor search (or pattern matching). One may hope to match the $O(\log \log d)$ approximation that was recently obtained for the Ulam metric [AIK08].

Acknowledgments. We thank Dick Karp for useful discussions at an early stage of this research.

References

AGM+90. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *J. of Molecular Biology*, 215(3):403–410, 1990.

AIK08. A. Andoni, P. Indyk, and R. Krauthgamer. Overcoming the ℓ_1 non-embeddability barrier: Algorithms for product metrics. *Manuscript*, 2008.

AK07. A. Andoni and R. Krauthgamer. The computational hardness of estimating edit distance. In *48th Annual IEEE Symposium on Foundations of Computer Science*, pages 724–734. IEEE, October 2007.

BEK+03. T. Batu, F. Ergün, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the Symposium on Theory of Computing*, pages 316–324, 2003.

BES06. T. Batu, F. Ergün, and C. Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 792–801, 2006.

BJKK04. Z. Bar-Yossef, T. S. Jayram, R. Krauthgamer, and R. Kumar. Approximating edit distance efficiently. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 550–559, 2004.

BS95. A. Blum and J. Spencer. Coloring random and semi-random k-colorable graphs. *J. Algorithms*, 19(2):204–234, September 1995.

CK06. M. Charikar and R. Krauthgamer. Embedding the ulam metric into ℓ_1. *Theory of Computing*, 2(11):207–224, 2006.

FK01. U. Feige and J. Kilian. Heuristics for semirandom graph problems. *J. Comput. Syst. Sci.*, 63(4):639–673, 2001.

FM97. A. Frieze and C. McDiarmid. Algorithmic theory of random graphs. *Random Structures and Algorithms*, 10(1-2):5–42, 1997.

GP06. S. Gollapudi and R. Panigrahy. A dictionary for approximate string search and longest prefix search. In *15th ACM international conference on Information and knowledge management*, pages 768–775. ACM, 2006.

MP80. W. J. Masek and M. Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.

MTL02. B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

Nav01. G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

NBYST01. G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19–27, 2001. Special issue on Text and Databases. Invited paper.

ST03. D. A. Spielman and S.-H. Teng. Smoothed analysis: Motivation and discrete models. In *WADS, Lecture Notes in Computer Science*. Springer-Verlag, 2003.

ST04. D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004.

WF74. R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.