# The Smoothed Complexity of Edit Distance

ALEXANDR ANDONI, Microsoft Research SVC
ROBERT KRAUTHGAMER, The Weizmann Institute of Science



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 than the bounds that were known for worst-case inputs.

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 et al. [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.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.3 [**Probability and Statistics**]

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Edit distance, smooth complexity, linear time, sublinear time

**ACM Reference Format:**
Andoni, A. and Krauthgamer, R. 2012. The smoothed complexity of edit distance. ACM Trans. Algor. 8, 4, Article 44 (September 2012), 25 pages.
DOI = 10.1145/2344422.2344434 http://doi.acm.org/10.1145/2344422.2344434

## 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

An extended abstract of this article appeared in *Proceedings of ICALP 2008* [Andoni and Krauthgamer 2008]. Part of this work of was done while A. Andoni was a student at MIT, supported in part by NSF CAREER award CCR-0133849, David and Lucille Packard Fellowship and Alfred P. Sloan Fellowship, and while an intern at IBM Almaden Research Center.
Part of the work of R. Krauthgamer was done while he was at the IBM Almaden Research Center. This research was supported in part by a grant from the Fusfeld Research Fund, and by the Israel Science Foundation grant #452/08.
Authors' addresses: A. Andoni, Microsoft Research Silicon Valley, 6/1171, 1065 La Avenida Street, Mountain View, CA 94043; email: andoni@mit.edu; R. Krauthgamer, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, 234 Herzel Street, P.O.B. 26, Rehovot 76100, Israel; email: robert.krauthgamer@weizmann.ac.il.

DOI 10.1145/2344422.2344434 http://doi.acm.org/10.1145/2344422.2344434

and text processing, and consequently computational problems involving the edit distance were studied extensively, both theoretically and experimentally, see, for example, the detailed survey on edit distance by Navarro [2001]. 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). In this article, we discuss the problems of computing and/or estimating the distance between two input strings, which are the focus of our article, but the situation is similar for other problems like pattern matching and near-neighbor searching.

The most basic problem is to compute the edit distance between two strings of length $d$ over alphabet $\Sigma$. 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 $O(d^2)$ [Wagner and Fischer 1974], and in time $O(d^2/\log^2 d)$ when the alphabet has constant size [Masek and Paterson 1980] (see also Bille and Farach-Colton [2008]).[1] 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, prior to our work the best quasi-linear time algorithm, due to Batu et al. [2006], achieves $d^{1/3+o(1)}$ approximation (improving over Bar-Yossef et al. [2004]).[2] The only known sublinear time algorithm, due to Batu et al. [2003], decides whether the edit distance is $O(d^\alpha)$ or $\Omega(d)$ in time $O(d^{\max\{\alpha/2,1-2\alpha\}})$.[3] In fact, distance estimation with sublogarithmic approximation factor was recently proved impossible in a certain model of low communication complexity [Andoni and Krauthgamer 2010].[4] In practice, this situation is mitigated by heuristic algorithms. In computational biology settings, for instance, tools such as BLAST [Altschul et al. 1990] 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 semirandom 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 $0 \leq p \leq 1$, except that every character in the common subsequence is perturbed in the same way in the two strings. Semirandom models appeared in the literature in other contexts, but to the best of our knowledge, not for sequence alignment problems; see Section 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 semirandom models may be viewed as an attack on the worst-case complexity. It is

[1] In contrast, the Hamming distance can clearly be computed in $O(d)$ time.

[2] After the conference version of our paper appeared [Andoni and Krauthgamer 2008], the approximation factor was improved to $2^{\tilde{O}(\sqrt{\log d})}$ [Andoni and Onak 2009], and then to $(\log d)^{O(1/\varepsilon)}$ in $d^{1+\varepsilon}$ time, for any $0 < \varepsilon < 1$ [Andoni et al. 2010b].

[3] In contrast, the analogous decision problem under Hamming distance can clearly be solved in $O(1)$ time.

[4] In contrast, the analogous problem under Hamming distance can be solved within $1+\varepsilon$ approximation [Kushilevitz et al. 2000].

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 $0 < p \leq 1$ 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 Section 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 $(x, y)$, proving along the way 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 an $O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$ approximation to $\mathrm{ed}(x, y)$, and with high probability over the randomness in the smoothing, it runs in time $O(d^{1+\varepsilon})$. For comparison, the algorithm of Batu et al. [2006] 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 achieves an $O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$ approximation in time $O(d^{0.5+\varepsilon} + d^{1+\varepsilon}/\mathrm{ed}(x, y))$. For comparison, the algorithm of Batu et al. [2003] for worst-case inputs can only distinguish a polynomially large gap in the edit distance, and only at the highest regime $\Omega(d)$. This second result obviously subsumes the first one; we nevertheless present the first result because its algorithm is simpler (and deterministic), and because it gradually introduces the ideas necessary for the second algorithm.

While not the focus of this article, we note that it is likely that the results may be extended to larger alphabets (adapting a natural extension of the smoothed model) without degradation in the parameters of the algorithms. We concentrate on the binary alphabet case since this seems to be the hardest regime, as suggested by the recent work of Andoni et al. [2010b] (see Theorem 4.12 and Lemma 4.13).

*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 $\varepsilon L$). This same idea, but in a more heuristic form, is used by practical tools. In particular, PatternHunter [Ma et al. 2002] uses such a notion of matches (to identify "seeds"), significantly improving over BLAST [Altschul et al. 1990], 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.[5] 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 l ength $d$ can be computed in linear time $O(d \log d)$, for example, using Patience Sorting [Aldous and Diaconis 1999]. 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 Charikar and Krauthgamer [2006, Section 3.1].

### 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, that is, 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 [Navarro 2001, Section 5.3] for details and references. For nearest neighbor search, the average-case model is quite similar, see Navarro et al. [2001] and Gollapudi and Panigrahy [2006].

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 Semirandom Models.* Smoothed analysis was pioneered by Spielman and Teng [2004] 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* perturbations [Spielman and Teng 2003], 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 [1995]. 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, for example, Frieze and McDiarmid [1997] and Feige and Kilian [2001] and the references therein.

To the best of our knowledge, smoothed analysis and/or semirandom 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

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

the distance between their strings using low communication [Kushilevitz and Nisan 1997]. A communication lower bound was recently proved in Andoni and Krauthgamer [2010] and Andoni et al. [2010a] 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 $\Sigma$. A *position* in the string is an index $i \in [d]$, where throughout we let $[k] = \{1, 2, \ldots, k\}$. 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 $\mathrm{ed}(x, y) \le \underline{\mathrm{ed}}(x, y) \le 2\,\mathrm{ed}(x, y)$. 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 $y$.

*Example.* For $x = 010111$ and $y = 101000$, $\mathrm{LCS}(x, y) = 3$ (corresponding, e.g., to substring 101), and $\underline{\mathrm{ed}}(x, y) = 6$, whereas $\mathrm{ed}(x, y) = 4$ (corresponding to a deletion, insertion and two substitutions).

*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 $\mathrm{len}(A) = |A^{-1}([d])|$, that is, the number of positions in $x$ that are matched by $A$. Let the *cost* of $A$ be $\mathrm{cost}(A) = 2(d - \mathrm{len}(A)) = 2|A^{-1}(\perp)|$, that is, 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

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

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

*Example.* For the same $x = 010111$ and $y = 101000$, the alignment corresponding to the substring 101 would be defined as $A = (\perp, 1, 2, 3, \perp, \perp)$, hence $\mathrm{cost}(A) = 6$.

*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 the ties arbitrarily. For an alignment $A$ between $x$ and $y$, let $\mathrm{match}_A(x_{[i:i+L-1]})$ be the block $y_{[j:j+L-1]}$, where $j \in [d-L+1]$ minimizes $\mathrm{ed}_A(x_{[i:i+L-1]}, y_{[j:j+L-1]})$, breaking the ties arbitrarily. Slightly abusing notation, we sometimes let match and $\mathrm{match}_A$ represent the corresponding position $j$ (instead of the substring $y_{[j:j+L-1]}$), but the distinction will be clear from the context.

*Example.* Consider the previous example and $L = 3$. Then, $\mathrm{ed}_A(x_{[1:3]}, y_{[2:4]}) = 2$, even if $\mathrm{ed}(x_{[1:3]}, y_{[2:4]}) = 0$. Also, $\mathrm{match}(x_{[1:3]}) = y_{[2:4]}$, but $\mathrm{match}_A(x_{[1:3]}) = y_{[1:3]}$.

*Smoothed Model.* Let $0 \le p \le 1$, 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 $\pi_y$ 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 $\mathtt{Smooth}_p(x^*, y^*, A^*)$.

## 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 optimal alignment $A^*$. Besides the inherent interest, these bounds are useful in the smoothed analysis of our algorithms carried out in subsequent sections.

### 2.1. Typical Edit Distance of a Smoothed Instance

We start by proving that for any two strings $x^*, y^*$, their smoothed instance preserves the original edit distance, up to a constant factor.

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

$$\Pr_{(x,y)}\left[\Omega\left(\frac{p}{\log(2/p)}\right)\mathrm{ed}(x^*, y^*) \le \mathrm{ed}(x, y) \le \mathrm{ed}(x^*, y^*)\right] \ge 1 - 2^{-\Omega(p)\cdot \mathrm{ed}(x^*, y^*)}.$$

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

Consider a *potential alignment* $A$ between $x$ and $y$, that is, a map $A : [d] \mapsto [d] \cup \{\perp\}$ that is monotonically increasing on $A^{-1}([d])$, and suppose that $\mathrm{cost}(A) = 2|A^{-1}(\perp)|$ is at most $\alpha \cdot \underline{\mathrm{ed}}(x^*, y^*)$ for a small $0 < \alpha \le 1/4$ 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 $|U| - |A^{-1}(\perp)| \ge \frac{1}{2}\,\underline{\mathrm{ed}}(x^*, y^*) - \frac{1}{2}\alpha\,\underline{\mathrm{ed}}(x^*, y^*) \ge \frac{1}{4}\,\underline{\mathrm{ed}}(x^*, y^*)$. For each of them, $x^*[i]$ is perturbed independently of $y^*[A(i)]$, and thus $x[i] \ne y[A(i)]$ occurs with probability at least $p/2$. Thus, the probability that $A$ is an actual alignment is at most

$$\Pr[x[i] = y[A(i)] \text{ for all } i \in U \setminus A^{-1}(\perp)] \le \left(1 - \frac{p}{2}\right)^{\underline{\mathrm{ed}}(x^*, y^*)/4} \le e^{-p/8 \cdot \underline{\mathrm{ed}}(x^*, y^*)}.$$

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 $U$. Observe 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 $U$), and thus we can count the number of $A|_U$ as:

$$\#\{A|_U\} \le \binom{|U| + \frac{1}{2}\alpha\,\underline{\mathrm{ed}}(x^*, y^*)}{\frac{1}{2}\alpha\,\underline{\mathrm{ed}}(x^*, y^*)}^2 \le \left(\frac{e(1+\alpha)}{\alpha}\right)^{\alpha\,\underline{\mathrm{ed}}(x^*, y^*)} \le \left(\frac{1}{\alpha^2}\right)^{\alpha\,\underline{\mathrm{ed}}(x^*, y^*)}.$$

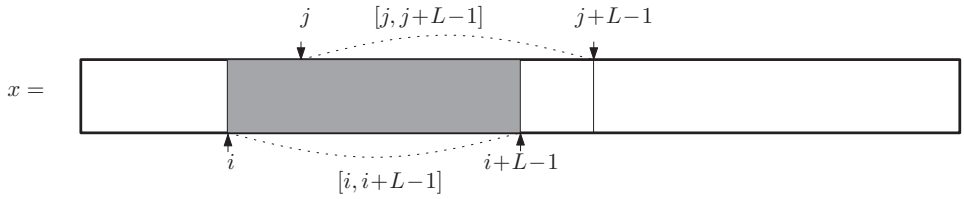


Fig. 1. Illustration of Lemma 2.2(a) addressing two blocks from the same string $x$.

Applying a union bound and choosing $\alpha = \frac{cp}{\log(2/p)}$ for a sufficiently small constant $c > 0$, we get

$$\Pr[\underline{\mathrm{ed}}(x, y) \le \alpha\, \underline{\mathrm{ed}}(x^*, y^*)] \le e^{[2\alpha \ln(1/\alpha) - p/8] \cdot \underline{\mathrm{ed}}(x^*, y^*)} \le e^{-(p/16) \cdot \underline{\mathrm{ed}}(x^*, y^*)},$$

which completes the proof of Theorem 2.1. □

### 2.2. Typical Edit Distance Between Substrings (Blocks)

We now turn to showing some finer properties of a smoothed instance. The next lemma analyzes the distances between two arbitrary blocks of logarithmic length from $x$ and $y$. We show that, two such blocks are almost always far away in terms of edit distance, modulo two obvious exceptions: (1) blocks at nearby positions in the same string; and (2) blocks from different strings that are (mostly) matched under the original optimal alignment $A^*$. This lemma is similar in spirit to Theorem 2.1, but the main difference is that here we also consider blocks whose perturbations are correlated, for example, overlapping blocks in the same string. This technical difficulty impedes direct concentration bounds used in the previous theorem, and thus will require more ideas to complete the proof.

The lemma comprises three parts, taking care of three types of pairs of blocks: (a) both blocks coming from the same string (either $x$ or $y$); (b) a block from $x$ and a block from $y$ that are (mostly) not matched under $A^*$; and finally (c) a block from $x$ and a block from $y$ that are largely matched under $A^*$. See Figures 1 and 2 for illustrations of the first two parts. We shall use of the notation introduced in Section 1.3.

LEMMA 2.2. *Let $A^*$ be an optimal alignment between $x^*, y^* \in \{0,1\}^d$ and fix $0 < p \le 1$. Let $L \ge \frac{C}{p} \log d$ 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 $(x, y) \in \mathtt{Smooth}_p(x^*, y^*, A^*)$ satisfies the following for all $i, j \in [d - L + 1]$:*

(1) $\mathrm{ed}(x_{[i:i+L-1]}, x_{[j:j+L-1]}) \ge c_a \cdot \min\{pL, |j - i|\}$, *and similarly for $y$.*
(2) *If* $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j+L-1]}) = \beta L$ *for some* $\frac{c_c}{32} < \beta \le 1$, *then* $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \ge c_b \frac{\beta}{\log(2/\beta)} \cdot pL$.
(3) *Let* $k^* = \mathrm{match}_{A^*}(i, L)$, *then*

$$\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \ge \min\{c_c \cdot pL,\ c_c \cdot |j - k^*| - 2\,\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[k^*:k^*+L-1]})\}.$$

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

PROOF. It suffices to prove these bounds for fixed $i, j \in [d - L + 1]$ and $\beta L \in [L]$, because the lemma follows by a union bound, when $C > 0$ is sufficiently large.

We start by proving part (a). Consider the case when $|j - i| \ge L$. Since the corresponding blocks $x_{[i:i+L-1]}$ and $x_{[j:j+L-1]}$ do not overlap, they are perturbed independently of each other. We will use the following observation: for every collection of events

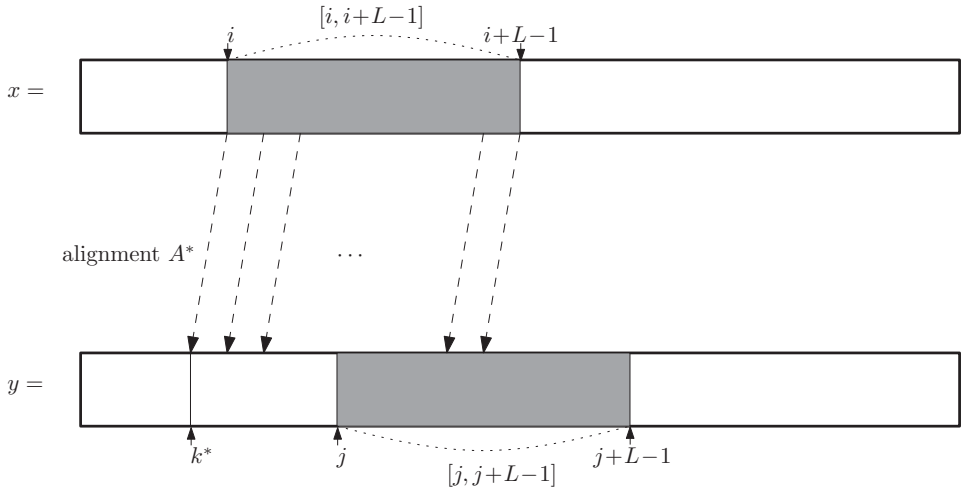


Fig. 2. Illustration of Lemma 2.2(b) addressing blocks from $x$ and from $y$ that are mostly not matched under $A^*$.

$\mathcal{E}, F_1, \ldots, F_t$,

$$\Pr[\cup_i F_i] \le \Pr[\bar{\mathcal{E}}] + \sum_i \min\{\Pr[F_i], \Pr[F_i|\mathcal{E}]\}. \tag{2.1}$$

In particular, let $\mathcal{E}$ be the event that at least $pL/4$ bits in $x_{[i:i+L-1]}$ are flipped by the perturbation. Note that $\Pr[\bar{\mathcal{E}}] \le e^{-\Omega(pL)}$ by Chernoff bound. We shall use the notation $\hat{p} = \min\{p, \frac{1}{2}\}$ (the reason to consider $\hat{p}$ is that the cases $p \le \frac{1}{2}$ and $p > \frac{1}{2}$ require slightly different bounds).

Consider a potential alignment $A$ between the two blocks $x_{[i:i+L-1]}$ and $x_{[j:j+L-1]}$, that is, a map $A : [L] \to [L] \cup \{\perp\}$ that is monotonically increasing on $A^{-1}([L])$, and assume that $\text{cost}(A) = 2|A^{-1}(\perp)|$ equals $\alpha pL$ for a constant $0 < \alpha < 1/6$ to be determined later. Consider the mismatches under $A$ between the unperturbed blocks namely,

$$M_A = \{k \in A^{-1}([L]) : \ x^*_{[i-1+k]} \ne x^*_{[j-1+A(k)]}\}. \tag{2.2}$$

First, suppose that $|M_A| \ge pL/16$. Then for $A$ to be an actual alignment between the two blocks in $x$, for every mismatch $k \in M_A$, the perturbation must flip exactly one of the two relevant bits, which happens with probability $2 \cdot p/2 \cdot (1 - p/2) \le \hat{p}$. Since these events are independent, in this case

$$\Pr[A \text{ is an alignment}] \le \hat{p}^{pL/16}.$$

Now we consider the case when $|M_A| < pL/16$, and assume that the event $\mathcal{E}$ occurs. Then the number of mismatches after perturbing only $x^*_{[i:i+L-1]}$, that is, number of $k \in A^{-1}([L])$ such that $x_{[i-1+k]} \ne x^*_{[j-1+A(k)]}$, is at least $pL/4 - |M_A| - |A^{-1}(\perp)| \ge pL/16$. For $A$ to be an actual alignment between the two perturbed blocks, all the corresponding positions $j - 1 + A(k)$ must also be flipped by the perturbation. Since each of these happens with probability $p/2$ and they are independent,

$$\Pr[A \text{ is an alignment} \mid \mathcal{E}] \le (p/2)^{pL/16} \le \hat{p}^{pL/16}.$$

The number of potential alignments of cost $\alpha pL$ is exactly (as one needs to determine the unaligned positions in each block)

$$\binom{L}{\frac{1}{2}\alpha pL}^2 \le \left(\frac{2e}{\alpha p}\right)^{\alpha pL} \le \left(\frac{1}{\alpha^2 p}\right)^{\alpha pL}. \tag{2.3}$$

Finally, we apply (2.1) with events $F_i$ corresponding to all potential alignments $A$. Choosing $\alpha > 0$ to be a sufficiently small constant independent of $p$, we obtain that

$$\Pr[\underline{\mathrm{ed}}(x_{[i:i+L-1]}, x_{[j:j+L-1]}) \le \alpha pL] \le e^{-\Omega(pL)} + \left(\left(\tfrac{1}{\alpha^2 p}\right)^{16\alpha} \cdot \hat{p}\right)^{pL/16} \le e^{-\Omega(pL)},$$

which proves part (a) in the case $|j-i| \ge L$.

This proof immediately extends to the case $|j-i| \ge L/4$ (the constant $1/4$ is arbitrary here). Indeed, consider in each block the initial segment of length $t = |j-i|$, which do not overlap. By this argument, with high probability $\underline{\mathrm{ed}}(x_{[i:i+t-1]}, x_{[j:j+t-1]}) \ge \Omega(pt) = \Omega(pL)$, implying a similar lower bound for the two blocks of length $L$.

Next we prove part (a) in the remaining case where $t = |j-i| < L/4$. Note that in this slightly harder case, the blocks $x_{[i:i+L-1]}$ and $x_{[j:j+L-1]}$ have a large overlap and thus we do not have the easy independence from before.

Assume without loss of generality that $i < j$. As before, consider a potential alignment $A : [L] \to [L] \cup \{\perp\}$ of cost $\alpha \cdot \min\{pL, t\}$ for a constant $0 < \alpha < 1/16$ to be determined later. Observe that for every $k \in A^{-1}([L])$, we have $|k - A(k)| \le \frac{1}{2}\,\mathrm{cost}(A) \le \frac{1}{2}\alpha t$, thus $i - 1 + k \ne j - 1 + A(k)$, and in particular these two positions are perturbed independently of each other.

Define $M_A$ as in Eq. (2.2), and consider the case where $|M_A| \ge pL/64$. Then, for $A$ to be an actual alignment, for every $k \in M_A$ the event $x[i-1+k] = x[j-1+A(k)]$ must hold, that is, exactly one of the two relevant bits must be flipped by the perturbation. These events might not be independent, but we can easily find at least $1/3$ of them that are independent (here is a simple nonoptimized argument: every bit $x[l]$ appears in at most two such events, so if we take a subset of the events greedily, for every event taken, at most two need to be discarded). Thus, in this case,

$$\Pr[A \text{ is an alignment}] \le \hat{p}^{pL/192}.$$

Next suppose that $|M_A| < pL/64$. Partition the interval $[i : i+L-1]$ into subintervals of length $t/2$, and take every fourth subinterval starting from the first one, namely $I = [i : i + \frac{t}{2} - 1] \cup [i + 2t : i + \frac{5t}{2} - 1] \cup \ldots$. We define $\mathcal{E}$ to be the event that at least $pL/16$ positions in $I$ are flipped by the perturbation. Notice that this event does not depend on the choice of $A$, and that by a Chernoff bound, $\Pr[\bar{\mathcal{E}}] \le e^{-\Omega(pL)}$. As before, we shall assume that the event $\mathcal{E}$ occurs. Observe that, if $k \in A^{-1}([L])$ and $i-1+k \in I$, then $j-1+A(k) \notin I$ (because the difference between these two positions is $j-i+A(k)-k$, which falls in the range $[t - \frac{1}{2}\alpha t, t + \frac{1}{2}\alpha t]$). After conditioning on the outcomes of the perturbations inside $I$ (only), the number of such $k$ for which $x_{[i-1+k]} \ne x^*_{[j-1+A(k)]}$ is at least $pL/16 - |M_A| - |A^{-1}(\perp)| \ge pL/64$. For $A$ to be an actual alignment between the two perturbed blocks, all the corresponding positions $j-1+A(k)$ must also be flipped by the perturbation. Since each of these happens with probability $p/2$ and they are independent,

$$\Pr[A \text{ is an alignment} \mid \mathcal{E}] \le (p/2)^{pL/64} \le \hat{p}^{pL/64}.$$

The number of potential alignments of cost $\alpha \cdot \min\{pL, t\} \le \alpha pL$ is at most $(\frac{1}{\alpha^2 p})^{\alpha pL}$ by Eq. (2.3). Hence, applying (2.1) with events $F_i$ corresponding to all potential alignments

$A$, and choosing $\alpha > 0$ to be a sufficiently small constant independent of $p$, we have $\Pr[\underline{\text{ed}}(x_{[i:i+L-1]}, x_{[j:j+L-1]}) \le \alpha \cdot \min\{pL, t\}] \le e^{-\Omega(pL)}$. This completes the proof of part (a).

Before continuing to parts (b) and (c), we prove the claim below, which will be used in both parts. It is a variant of the argument from above for alignments between blocks of $x$ and $y$.

CLAIM 2.3. *Fix $i, j \in [d - L + 1]$, and let $0 < \beta \le 1$ and $\alpha \le \frac{\beta}{64 \log(1/\beta)}$. Suppose that for every potential alignment $A : [L] \to [L] \cup \{\perp\}$ between $x_{[i:i+L-1]}$ and $y_{[j:j+L-1]}$ of cost at most $\alpha pL$, there is $S \subset [L]$ of size $|S| = \beta L$, such that for all $k \in S \setminus A^{-1}(\perp)$ we have $j + A(k) - 1 \ne A^*(i + k - 1)$ (i.e., $A$ and $A^*$ map positions in $S$ differently). Then*

$$\Pr[\underline{\text{ed}}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \le \alpha pL] \le e^{-\Omega(\beta pL)}.$$

PROOF. Fix a potential alignment $A$ of cost $\alpha pL$. First, we can pick a subset $\hat{S} \subset S$, such that all events $x[i - 1 + k] \stackrel{?}{=} y[j - 1 + A(k)]$ for $k \in \hat{S} \setminus A^{-1}(\perp)$ are independent. Formally, $\hat{S}$ is such that $A^*(i - 1 + \hat{S} \setminus A^{-1}(\perp)) \cap (j + A(\hat{S}) \setminus A^{-1}(\perp)) = \emptyset$. The largest such set has size $|\hat{S}| \ge |S|/2$.

Define $M_A \subseteq \hat{S}$ to be those positions in $\hat{S}$ which are nonmatching positions under $A$ in $x^*, y^*$:

$$M_A = \{k \in A^{-1}([L]) \cap \hat{S} : \; x^*_{[i-1+k]} \ne y^*_{[j-1+A(k)]}\}.$$

First, suppose $M_A \ge \frac{p}{32}\beta L$. Then, for each $k \in M_A$, the event $x_{[i-1+k]} = y_{[j-1+A(k)]}$ happens only with probability at most $\hat{p}$. Since all these events are independent (due to that fact that $M_A \subseteq \hat{S}$), we conclude that $A$ is a valid alignment with probability at most $\hat{p}^{|M_A|} \le \hat{p}^{p\beta L/32}$.

Now suppose $M_A < \frac{p}{32}\beta L$. Define $\mathcal{E}$ to be the event that there are at least $\frac{p}{8}\beta L$ positions $k \in \hat{S}$ that are flipped: $x[i + k - 1] \ne x^*[i + k - 1]$. Note that $\Pr[\bar{\mathcal{E}}] \le e^{-\Omega(p\beta L)}$ by Chernoff bound. Now we condition on the event $\mathcal{E}$. Consider the positions $k \in \hat{S} \setminus A^{-1}(\perp)$ such that $x[i + k - 1] \ne y^*[j + A(k) - 1]$; the number of such positions is at least $\frac{p}{8}\beta L - |M_A| - |A^{-1}(\perp)| \ge \frac{p}{16}\beta L$. For each such position $k$, the event $x[i + k - 1] = y[j + A(k) - 1]$ happens with probability $p/2 \le \hat{p}$. Furthermore, all such events are independent, even after we condition on $\mathcal{E}$, and thus $A$ is a valid alignment with probability at most $\hat{p}^{p\beta L/16}$.

The number of alignments of cost $\alpha pL$ is at most $(\frac{1}{\alpha^2 p})^{\alpha pL}$ by Eq. (2.3). Finally, applying (2.1), we obtain that

$$\Pr[\underline{\text{ed}}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \le \alpha pL] \le e^{-\Omega(p\beta L)} + \hat{p}^{p\beta L/16} \cdot (\tfrac{1}{\alpha^2 p})^{\alpha pL}.$$

The conclusion follows as long as $\alpha \le \frac{\beta}{64 \log(1/\beta)}$. This completes the proof of Claim 2.3. □

Part (b) now follows easily from this claim. In particular, suppose $\text{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j+L-1]}) = \beta L$ for some $\beta > 0$. Let

$$U = \left\{k \in [L] \; : \; A^*(i + k - 1) \notin [j : j + L - 1]\right\}$$

be the set of positions in $x^*_{[i:i+L-1]}$ not matched into $y^*_{[j:j+L-1]}$ under $A^*$. Note that $|U| = \beta L$. Then just apply Claim 2.3 with $S = U$.

We proceed to proving part (c). Suppose $\beta L = \text{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[k^*:k^*+L-1]}) \ge L/4$. Then $\text{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j+L-1]})$ cannot be smaller, and the proof follows by applying part (b). So

assume henceforth that $\beta L \le L/4$. Now if $|j-k^*| \ge L/2$, then $\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[j:j+L-1]}) \ge L/2 - L/4$, and the again the proof follows by applying part (b).

It remains to deal with the case that $|j-k^*| < L/2$ (and $\beta L \le L/4$). We use the triangle inequality to deduce that

$$\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \ge \mathrm{ed}(y_{[k^*:k^*+L-1]}, y_{[j:j+L-1]}) - \mathrm{ed}(x_{[i:i+L-1]}, y_{[k^*:k^*+L-1]}).$$

Thus, by part (a), and using the fact that $\mathrm{ed}(x_{[i:i+L-1]}, y_{[k^*:k^*+L-1]}) \le 2\,\mathrm{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[k^*:k^*+L-1]}) = 2\beta L$, we have

$$\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \ge c_a \cdot \min\{pL, |j-k^*|\} - 2\beta L.$$

If $\beta L < \frac{c_a}{4} pL$, then we are done:

$$\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \ge \min\{\tfrac{c_a}{2} pL, c_a \cdot |j-k^*| - 2\beta L\}.$$

We are thus left with the case $\frac{c_a}{4} pL \le \beta L \le L/4$. We may further assume that $|j-k^*| \ge \frac{2}{c_c} \cdot \beta L$, as otherwise, the inequality we need to prove is trivial (asserting some edit distance is at least some negative number). Assuming this last condition, we shall prove that $\mathrm{ed}(x_{[i:i+L-1]}, y_{[j:j+L-1]}) \ge \Omega(pL)$. Recall that $c_c > 0$ is a sufficiently small absolute constant.

We now want to show that we can apply Claim 2.3. Without loss of generality, suppose $k^* < j$. Consider a potential alignment $A : [L] \to [L] \cup \{\perp\}$ between $x_{[i:i+L-1]}$ and $y_{[j:j+L-1]}$ of cost $\alpha pL$ for $\alpha \le O(\frac{\beta}{\log(2/\beta)})$ (in particular $\alpha \le \beta/4$). Let $S$ be the set of positions in $x[i : i+L-1]$ that $A$ matches to $y[k^* : k^*+L-1]$, formally

$$S = \left\{ z \in [L] : \ A(z) \in [1 : k^* - j + L] \right\}$$

For each $z \in S$, the alignments $A$ and $A^*$ cannot map $x[i+z-1]$ to the same symbol in $y$, formally, $j + A(z) - 1 \ne A^*(i+z-1)$, because $j + A(z) - 1 \ge j + (z - \alpha pL/2) - 1$ and at the same time $A^*(i+z-1) \le k^* + z - 1 + \beta L$ or $A^*(z) = \perp$ (recall $j - k^* \ge \frac{2}{c_c} \cdot \beta L$ and $\alpha p \le \alpha \le \beta$). Moreover, by definition of $S$ we have $|S| \ge (k^* - j + L) - \frac{1}{2}\,\mathrm{cost}(A) \ge L/4 \ge \beta L$ (recall $|j-k^*| < L/2$). We are thus in position to apply Claim 2.3, and this completes the proof of part (c) and of the entire Lemma 2.2.

## 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 $\mathtt{Smooth}_p(x^*, y^*, A^*)$.

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 $O(d^{1+\varepsilon})$.*

Before proving the theorem, we present two lemmas that establish useful properties of the edit distance between two strings and lead us to the algorithm. These lemmas are driven by the basic approach of the algorithm—to break the two input strings into blocks (short substrings later chosen to be of logarithmic length), and rely *only* on distances between blocks, by essentially finding for every block in $x$ its best match in $y$, with no attempt to "coordinate" the decisions for successive blocks in $x$. We show that this crude information is enough for estimating the edit distance between the two strings, up to a constant factor. We believe these lemmas may be useful in other scenarios as well.

### 3.1. Structural Lemmas

The first lemma gives properties of an optimal alignment between two (worst-case) strings and, as such, does not deal with smoothed instances. In this section, we will only need the first two parts of the lemma, which are easier to state, but in Section 4 we will use all four parts. We shall use the notation $\text{match}_A(\cdot)$ and related definitions of block matches from Section 1.3.

Let us briefly outline the intuition behind this lemma. Consider an optimal alignment $A$ between two strings $x, y$ and fix a block-length $L$. Suppose we partition $x$ into $d/L$ such blocks, each starting at a position $(i-1)L+1$ for $i \in [d/L]$. We would like to find, for each such block $X_i = x_{[iL-L+1:iL]}$, a corresponding block in $y$, denoted $Y_i = y_{[s_i:s_i+L-1]}$, such that the alignment $A$ between $x$ and $y$ is largely contained inside the pairs $(X_i, Y_i)$ (i.e., only a small part of $A$ maps a position in $X_i$ to a position in $Y_j$ for $j \neq i$). Indeed, part (a) shows that, for an appropriate choice of $Y_i$'s, this can be accomplished without increasing too much the cost of the alignment (equivalently, the edit distance between the two strings). Furthermore, part (b) shows that if we remove from $A$ matches within pairs $(X_i, Y_i)$ that are at a large edit distance, then the cost of the resulting alignment will not increase too much. Finally, parts (c) and (d) describe the relative positions of the blocks $Y_i$, namely that the blocks $Y_i$ do not have much overlap. This property will be needed in Section 4, where an alignment between $x$ and $y$ is not constructed explicitly (in full), and we need to ensure that any single position in $y$ contributes to the alignment of at most one pair $(X_i, Y_i)$.

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 $Y_i = \text{match}_A(X_i)$. Let $V$ be the set of $i$ for which $\text{ed}_A(X_i, Y_i) < L$. Then, the following holds.*

(1) $\sum_{i \in [d/L]} \text{ed}_A(X_i, Y_i) \leq 2\,\text{ed}(x, y)$.
(2) *For $\varepsilon > 0$, let $B_\varepsilon = \{i \in [d/L] : \text{ed}(X_i, Y_i) > \varepsilon L\}$. Then $|B_\varepsilon| \leq \frac{4}{\varepsilon L} \cdot \text{ed}(x, y)$.*
(3) *For $i \in [d/L]$, let $s_i$ be the starting position of $Y_i$. Then, for all $i, i' \in V$ and $i < i'$, we have*

$$s_{i'} - s_i \geq L - \text{ed}_A(X_i, Y_i) - \text{ed}_A(X_{i'}, Y_{i'}).$$

(4) *For $i \in [d/L]$, let $S_i$ be the positions in $Y_i = y_{[s_i:s_i+L-1]}$ that appear also in some block $Y_{i'}$ for $i' \neq i$. Then $\sum_{i \in [d/L]} |S_i| \leq 2\,\text{ed}(x, y)$.*

PROOF. For $i \in [d/L]$, let $\text{MIN}_i$ and $\text{MAX}_i$, respectively, be the positions of the first and last aligned symbol in $X_i$, that is, $\text{MIN}_i = \min\{j \in [iL-L+1 : iL] \mid A(j) \in [d]\}$ and similarly for $\text{MAX}_i$. It could be that $\text{MIN}_i$ and $\text{MAX}_i$ are undefined, when $A(j) = \perp$ for all $j \in [iL-L+1 : iL]$, in which case, abusing the notion, we define $A(\text{MIN}_i) = 0$ and $A(\text{MAX}_i) = -1$. Let $u_i^x$ be the number of unaligned positions in $X_i = x_{[iL-L+1:iL]}$, that is, $u_i^x = |\{j \in [iL-L+1 : iL] \mid A(j) = \perp\}|$. Also, let $u_i^y$ be the number of unaligned positions in $y_{[A(\text{MIN}_i):A(\text{MAX}_i)]}$. If $\text{MIN}_i, \text{MAX}_i$ are undefined, then set $u_i^x = L$ and $u_i^y = 0$.

If $A(\text{MAX}_i) - A(\text{MIN}_i) < L$, then $\text{ed}_A(X_i, Y_i) = u_i^x$. If $A(\text{MAX}_i) - A(\text{MIN}_i) \geq L$, then $\text{ed}_A(X_i, Y_i) \leq u_i^x + u_i^y$. Observing that each of $\sum_i u_i^x$ and $\sum_i u_i^y$ is bounded by $\text{ed}(x, y)$ proves part (a).

To prove part (b), notice that if $\text{ed}(X_i, Y_i) > \varepsilon L$, then $\text{ed}_A(X_i, Y_i) \geq \frac{1}{2}\,\text{ed}(X_i, Y_i) > \frac{1}{2}\varepsilon L$. By previous part, there could be at most $\frac{4}{\varepsilon}$ such blocks, and thus the claimed bound on the size of $B_\varepsilon$.

For part (c), note that, since $A(\mathrm{MAX}_i) < A(\mathrm{MIN}_{i'})$, we have

$$\begin{aligned} s_i &\le A(\mathrm{MAX}_i) + 1 - (L - \mathrm{ed}_A(X_i, Y_i)) \\ &\le A(\mathrm{MIN}_{i'}) - L + \mathrm{ed}_A(X_i, Y_i). \end{aligned}$$

We also observe that $s_{i'} \ge A(\mathrm{MIN}_{i'}) - \mathrm{ed}_A(X_{i'}, Y_{i'})$. Taking $s_{i'} - s_i$ implies the inequality.

Finally, we prove part (d). For $i \in [d/L]$, define $r'_i$ to be the number of $j \in [s_i : s_i + L - 1]$ such that $j \notin [A(\mathrm{MIN}_i) : A(\mathrm{MAX}_i)]$. Then, we argue that $2\sum_{i \in V} r'_i \ge \sum_{i \in V} |S_i|$. We charge all contribution in $\sum_{i \in V} |S_i|$ to the positions $j \in [s_i : s_i + L - 1]$ with $j \notin [A(\mathrm{MIN}_i) : A(\mathrm{MAX}_i)]$. Any such position $j$ contributes at most twice to the sum $\sum_{i \in V} |S_i|$: at most once in $|S_i|$ and once in $|S_k|$ for $k \in V$ such that $j \in [A(\mathrm{MIN}_k) : A(\mathrm{MAX}_k)]$.

We conclude that $\sum_{i \in [d/L]} |S_i| \le 2\,\mathrm{ed}(x, y)$. We upper bound $r'_i$ by $u^x_i$. Indeed, if $A(\mathrm{MAX}_i) - A(\mathrm{MIN}_i) \ge L$, then $r'_i = 0$. If $A(\mathrm{MAX}_i) - A(\mathrm{MIN}_i) \le L - 1$, then $r'_i = L - 1 - (A(\mathrm{MAX}_i) - A(\mathrm{MIN}_i)) \le u^x_i$. Also, for $i \notin V$, we have that $|S_i| \le L = u^x_i$. Thus, $\sum_{i \in [d/L]} |S_i| \le 2\sum_{i \in V} r'_i + L \cdot (d/L - |V|) \le 2\sum u^x_i \le 2\,\mathrm{ed}(x, y)$. □

The next lemma proves a converse to the previous lemma, and applies to smoothed instances only. The previous lemma essentially said that, for a typical block $X_i$ in $x$, there exists a block $Y_i$ in $y$ that contains most of the alignment of $X_i$ and hence is "close" in edit distance to $X_i$. The next lemma says that, after the smoothing operation, the block $X_i$ is also far from all "other" blocks of $y$ (those that do not overlap with $Y_i$).

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 $(x, y) \in \mathtt{Smooth}_p(x^*, y^*, A^*)$, 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 2.2(b). □

### 3.2. A Near-Linear Time Algorithm for Smoothed Instances

Having established the two structural lemmas, we proceed to present our near-linear time algorithm. We will need the following algorithmic result, which can be seen as a generalization of the Patience Sorting algorithm for computing the edit distance between two nonrepetitive strings (such as permutations), to handle (worst-case) strings with only mild repetitions. For a cleaner statement, we give a graph-theoretic interpretation, where edges of a bipartite graph should be viewed as potential matches between positions in the two strings. In this language, nonrepetitive strings imply that the number of edges in the graph is $|E| \le d$.

LEMMA 3.4. *Consider a bipartite graph $G = ([d], [d], E)$, and call two edges $(i, j) \in E$ and $(k, l) \in E$* intersecting *if $(i - k)(j - l) \le 0$. 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. Construct a string $z$ of length $|E|$ via the following procedure. Start with an empty string. For each node $i = 1, \ldots, d$, we append to the end of $z$ the list of characters $(j_1, -i), \ldots, (j_k, -i)$, where $j_1 > j_2 > \cdots > j_k$ are the neighbors of $i$, that is, $\{j_1, \ldots, j_k\} = \{j \in [d] : (i, j) \in E\}$. Notice that $j_1, \ldots, j_k$ are appended in *decreasing* order.

**ALGORITHM 1:** Matching(edge set $E \subset \{1, \ldots, d\} \times \{1, \ldots, d\}$)

```
// Finds a maximum subset of non-intersecting edges
```
Set $z$ to be an empty string
**for** $i = 1 \ldots d$ **do**
  Let $j_1 > \cdots > j_k$ be the neighbors of $i$ in the edge set $E$
  Append to the end of $z$ the characters $(j_1, -i), \ldots (j_k, -i)$
Find a longest increasing subsequence of $z$ using Patience Sorting (where tuples are ordered lexicographically), and denote it by $(j_1, -i_1), \ldots (j_l, -i_l)$
**return** $\{(j_1, i_1), \ldots, (j_l, i_l)\}$

It should now be clear that the longest increasing subsequence of $x$ gives a maximum subset of nonintersecting edges (the order of symbols $(j, -i)$ is the lexicographic one). More precisely an increasing sequence $(j_1, -i_1) < \cdots < (j_l, -i_l)$ forms a nonintersecting set $(i_1, j_1), (i_2, j_2), \ldots (i_l, j_l)$ and vice-versa.

The string $z$ has length $|E|$, and thus, using Patience Sorting (or just straightforward dynamic programming), we can find the longest increasing subsequence of $z$ in $O(|z| \log |z|) = O(|E| \log d)$ time. The complete procedure is given below as Algorithm 1. □

We now prove Theorem 3.1. The main algorithm follows the intuition built up by the structural lemmas. Consider a smoothed instance $(x, y) = \texttt{Smooth}_p(x^*, y^*, A^*)$. We partition the string $x$ into blocks $X_i = x_{[iL-L+1:iL]}$, and for each $X_i$ we find all the blocks (substrings) of $y$ that are close to $X_i$ in edit distance, and treat them as potential candidate matches for positions in $X_i$. Using Lemma 3.2, we know that we will discover in this fashion most of the original alignment $A^*$. Furthermore, Lemma 3.3 predicts that the number of such potential candidates is small, and hence we can apply the algorithm from Lemma 3.4. An important step of the algorithm is to find, for each $X_i$, the substrings of $y$ that are at a small edit distance. While a naive implementation of this step would take a quadratic time, we can obtain a near-linear time by using a Near Neighbor data structure, in the case where the block length $L$ is logaritmic. This step is the only one using the fact that the strings are a smoothed instance. Full details follow below.

PROOF OF THEOREM 3.1. Our algorithm uses 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 $\leq \varepsilon L$ 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 3.4, 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 $x_{[iL-L+1:iL]}$ for $i \in [d/L]$, and for each such block, queries the NN data structure to identify all blocks in $y$ that are within edit distance $\varepsilon L$. For each such block in $y$, collect all the character matches between the two blocks, that is, every zero in the block in $x$ with every zero in the block in $y$, and similarly for ones. Let $E$ be the resulting list of all candidate matches. The second stage simply applies Lemma 3.4 to this list $E$ to

**ALGORITHM 2:** EDIT(strings $x, y \in \{0,1\}^d$, reals $p, \varepsilon \in (0,1)$)

```
// Approximates edit distance on a smoothed instance (x, y) ∈ Smooth_p(x*, y*, A*).
```
$L \leftarrow \frac{C}{p} \log d$, where $C$ is the constant from Lemma 3.3
NNS-Preprocess (all length $L$ substrings of $y$, $\varepsilon$)
$E \leftarrow \emptyset$
**for** $i \leftarrow 1$ **to** $n/L$ **do**
  $Q \leftarrow$ NNS-Query($x_{[iL-L+1:iL]}$)
  **foreach** $y_{[j:j+L-1]} \in Q$ **do** add to $E$ all the pairs of the form $(a, b)$, such that $x[a] = y[b]$ and $a \in [iL - L + 1 : iL]$, $b \in [j : j + L - 1]$
$M \leftarrow$ Matching($E$)
**return** $2(d - |M|)$

retrieve an alignment between $x$ and $y$. The reported approximation to $\mathrm{ed}(x, y)$ is then twice the cost of this alignment. We present the complete algorithm as Algorithm 2.

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 $\varepsilon L$. 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 $O(P(d, L) + d/L \cdot (Q(d, L) + L^2))$, where $O(L^2)$ is the time to compute all the character matches between a block in $x$ and the corresponding $3L$ positions in $y$. The second stage runs in time $O(|E| \log d) = O(dL \log d)$.

Finally, it remains to describe the NN data structure. We achieve preprocessing time $P(m, L) = m \cdot 2^{L \cdot O(\varepsilon \log 1/\varepsilon)}$ and query time $Q(m, L) = O(L)$. The data structure simply prepares all answers in advance: for each string $\sigma$ in the database and every string $\tau$ at edit distance $\leq \varepsilon L$ from $\sigma$, store the pair $(\tau, \sigma)$ in a trie (we can also use a hash table if we allow a randomized algorithm). To query a string $q$, the algorithm uses the trie to find all pairs $(q, \eta)$, where $\eta \in \{0,1\}^L$, and, for each such pair, reports the string $\eta$. The complete description of the data structure is presented as Algorithm 3.

Recall that a trie with $t$ strings of length $L$, has query time $O(L)$, and preprocessing time $O(tL)$. 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 (the exponent 3 provides a crude bound on the number of deletions, insertions of zeros, and insertions of ones), we have $t \leq m\binom{2L}{\varepsilon L}^3$ and

$$P(m, L) \leq O(m \cdot \binom{2L}{\varepsilon L}^3 \cdot L) \leq m \cdot 2^{L \cdot O(\varepsilon \log(1/\varepsilon))}.$$

The overall running time for $O(1/\varepsilon)$ approximation is (with high probability) $d^{1+O(p^{-1}\varepsilon \log(1/\varepsilon))}$. To complete the proof with respect to a given $\varepsilon > 0$, apply the entire analysis shown so far to a smaller $\varepsilon' = \Theta(\varepsilon p / \log \frac{1}{p\varepsilon})$. The resulting running time is now $d^{1+\varepsilon}$ and the approximation factor is $O(1/\varepsilon') = O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$. □

**ALGORITHM 3:** Data structure for Near Neighbors within edit distance $\varepsilon L$

`NNS-Preprocessing`(set $S$ of $m$ strings in $\{0,1\}^L$, real $0 < \varepsilon < 1$):
1 Compute all tuples $(\tau, \sigma) \in \{0,1\}^L \times \{0,1\}^L$ where $\sigma \in S$ and $\mathrm{ed}(\tau, \sigma) \le \varepsilon L$
2 Construct a trie on this set

`NNS-Query`(string $q \in \{0,1\}^L$):
1 Retrieve from the trie all tuples of the form $(q, \eta)$ where $\eta \in \{0,1\}^L$
2 For each such tuple $(q, \eta)$, report $\eta$

## 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. The precise guarantees are stated in the following theorem. As before, we assume that $(x, y) \in \mathtt{Smooth}_p(x^*, y^*, A^*)$, where $A^*$ is the optimal alignment between two strings $x^*, y^* \in \{0,1\}^d$, and $0 < p \le 1$.

THEOREM 4.1. *For every $\varepsilon > 0$ and $p > 0$, there is a randomized algorithm that, given as input $(x, y) \in \mathtt{Smooth}_p(x^*, y^*, A^*)$, approximates $\mathrm{ed}(x, y)$ within factor $O(\frac{1}{\varepsilon p} \log \frac{1}{\varepsilon p})$ in time $(\sqrt{d} + d/\mathrm{ed}(x, y)) \cdot (\frac{d^\varepsilon \log d}{p})^{O(1)}$, 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 Andoni and Nguyen [2010b].[6] We use $\tilde{O}(f(d))$ as a shorthand for $O(f(d) \cdot (\log d)^{O(1)})$.

THEOREM 4.2. [ANDONI AND NGUYEN 2010]. *There exists a randomized algorithm that, given access to two permutations $P, Q$ of length $d$, approximates $\mathrm{ed}(P, Q)$ within a constant factor in time $\tilde{O}(\sqrt{d} + d/\mathrm{ed}(P, Q))$, with success probability at least $2/3$.*

We remark that this algorithm is based on adaptive sampling, that is, query positions depend on the outcome of earlier queries. As mentioned earlier, a direct application of this theorem implies a much weaker version of Theorem 4.1, with approximation factor $O(\log d)$, by employing the mapping of Charikar and Krauthgamer [2006, Theorem 3.1], which views each block (with overlaps) in $x$ or $y$ as a symbol in a large alphabet $\{0,1\}^L$. Thus, the main challenge we face is to obtain $O(1)$-approximation.

A key observation is that the algorithm in Theorem 4.2 (for Ulam distance estimation) works exactly the same way regardless of any relabeling of the symbols used in $P, Q$. More precisely, when the algorithm queries some position $i$ in $P$, the value of $P[i]$ is used only to check whether $P[i]$ is equal to any previously queried character $Q[j]$, and vice versa. Other than the value of $j$, and the information whether such $j$ exists, the name of the read symbol is not important.[7] This observation can be leveraged in the following way: if the algorithm is about to query $P[i]$, and the matching character $Q[j]$ (i.e. position $j$ such that $P[i] = Q[j]$) was not queried yet, then we may relabel this

[6] We note that the original conference version used the earlier result of Andoni et al. [2009], whose runtime is $\tilde{O}(d/\sqrt{\mathrm{ed}(P, Q)})$, and hence led to a $O(d^{1+\varepsilon}/\sqrt{\mathrm{ed}(P, Q)})$ time algorithm for smoothed instances.

[7] In fact, it is plausible that every algorithm for Ulam distance estimation can be assumed, in effect, to satisfy this property, by a simple transformation that incurs no loss in approximation factor and query complexity.

unread symbol, changing both $P[i]$ and $Q[j]$ to an arbitrary other symbol that does not appear anywhere at all. (Of course, such $Q[j]$ might not exist or might not be queried at all by a sublinear algorithm.) For the sake of analysis (but not in the algorithm) we may further assume, again by relabeling symbols, that $Q$ is a fixed permutation, say the identity, that is, $Q[j] = j$ for all $j \in [d]$. In what follows, the permutations $P, Q$ will always be of length $d$ and over the alphabet $\Sigma = [2d]$.

Our algorithm constructs $P, Q$ (from $x, y$) based on the following principle. Let $A$ be an alignment between $x$ and $y$, say of near-optimal cost $O(\mathrm{ed}(x, y))$. Then we can construct $P$ (while $Q$ is the identity) so that $A$ is an *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 $A(i) = \perp$. For our purpose, $A$ has to be computable "on the fly". More precisely we require that, for every two queried positions $i, j$ in $P, Q$ respectively, we can determine whether $A(i) = j$ by querying $x$ and $y$ only at (or near) positions $i$ and $j$, respectively; in particular, it is independent of the rest of the strings $x, y$. We term this property *locality*, and ensuring it is the main technical part of our proof. 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, largely due to Lemmas 2.2 and 3.2. In our presentation below, we will not describe the alignment $A$ explicitly, but instead construct $P$ directly. The actual construction of $P, Q$ will differ from this description in that it will actually work with whole blocks rather than single characters.

### 4.1. Block Structure Lemma

We now prove a lemma that provides further structural properties of the edit distance between two smoothed strings. These properties have a local nature, based on the substrings of $x$ and $y$, and will be useful later when we design our reduction. Namely, the lemma guarantees that our local operations in reducing to permutations result in a correct (global) edit distance.

This lemma should be seen as an extension of Lemma 3.2 for the more restricted case of smoothed instances. As before, we assume that $(x, y) \in \mathtt{Smooth}_p(x^*, y^*, A^*)$, where $A^*$ is the optimal alignment between two strings $x^*, y^* \in \{0, 1\}^d$, and $0 < p \leq 1$. Intuitively, Lemma 3.2 shows that after partitioning the string $x$ into blocks $X_i$, for each such block $X_i$ there is a "good" matching block $Y_i$ in $y$. In contrast, the lemma below shows how to efficiently find such a block $Y_i$ without knowledge of the original optimal alignment $A^*$. Indeed, in the case of a smoothed instance, it is essentially enough to choose $Y_i$ to be the substring of $y$ that minimizes the edit distance to $X_i$ (more precisely $Y_i = \mathrm{match}(X_i)$). For this choice of $Y_i$'s, we prove essentially the same properties as in Lemma 3.2.

LEMMA 4.3. *Consider a smoothed instance* $(x, y) \in \mathtt{Smooth}_p(x^*, y^*, A^*)$. *Let* $L = \frac{C}{p} \log d$ *for a sufficiently large constant* $C > 0$. *Partition* $x$ *into successive blocks of length* $L$, *denoted* $X_1, X_2, \ldots, X_{d/L}$, *and let* $Y_k = \mathrm{match}(X_k)$ *for* $k \in [d/L]$. *For sufficiently small* $\varepsilon > 0$, *let* $M = \{k \in [d/L] : \mathrm{ed}(X_k, Y_k) \leq \varepsilon pL\}$. *Then, with probability at least* $1 - d^{-\Omega(C)}$, *we have:*

(1) $\sum_{k \in M} \mathrm{ed}(X_k, Y_k) \leq 4 \cdot \mathrm{ed}(x^*, y^*)$.
(2) $d/L - |M| \leq \frac{4}{\varepsilon pL} \cdot \mathrm{ed}(x, y)$.
(3) *For* $k \in [d/L]$, *let* $S_k$ *be the set of positions in* $Y_k$ *that appear also in some block* $Y_{k'}$ *for* $k' \in M \setminus \{k\}$; *then* $\sum_{k \in M} |S_k| \leq O(1) \cdot \mathrm{ed}(x^*, y^*)$.
(4) *The starting positions of blocks* $Y_k$, *for* $k \in M$, *are in a strictly increasing order, and moreover the distance between two consecutive such positions is greater than* $L/2$.

PROOF. We start by proving part (a). Let $X_k^*, Y_k^*$ denote the blocks from $x^*, y^*$ that correspond to (i.e., have the same positions as) blocks $X_k, Y_k$, respectively. Let $s_k^* =$

$\text{match}_{A^*}(X_k^*)$ for all $k \in [d/L]$. The following inequality is immediate.

$$\text{ed}(X_k, Y_k) \le \text{ed}(X_k, y_{[s_k^*:s_k^*+L-1]}) \le \text{ed}(X_k^*, y^*_{[s_k^*:s_k^*+L-1]}). \tag{4.1}$$

Combining Eq. (4.1) with Lemma 3.2(a) and using an immediate relation between ed and $\text{ed}_A$, we obtain

$$\sum_{k\in[d/L]} \text{ed}(X_k, Y_k) \le 2 \sum_{k\in[d/L]} \text{ed}_{A^*}(X_k^*, y^*_{[s_k^*:s_k^*+L-1]}) \le 4 \cdot \text{ed}(x^*, y^*).$$

We now prove part (b). The upper bound on $d/L - |M|$ follows from Lemma 3.2(b) applied to the strings $x, y$, since $B_{\varepsilon p}$ in the language of that lemma is precisely $[d/L] \setminus M$.

Next, we turn to part (c), but before bounding $\sum_{k\in M} |S_k|$ itself, we prove the following two claims. Let $j_k$ denote the starting position of $Y_k$, and let $0 < c_c < 1$ be the constant from Lemma 2.2. The first claim bounds the distance between the starting position of the empirical match of $X_k$, namely $Y_k$, and the starting position of the "ideal" match of $X_k$ under the original alignment $A^*$, namely $\text{match}_{A^*}(X_k^*)$.

CLAIM 4.4. *With probability at least* $1 - d^{-\Omega(C)}$, *for all* $k \in M$:

$$|j_k - s_k^*| \le \tfrac{4}{c_c} \cdot \text{ed}_{A^*}(X_k^*, y^*_{[s_k^*:s_k^*+L-1]}). \tag{4.2}$$

PROOF. Fix $k \in M$ and notice that, by Lemma 2.2(c), with high probability,

$$\text{ed}(X_k, Y_k) \ge \min\{c_c \cdot pL,\ c_c \cdot |j_k - s_k^*| - 2\,\text{ed}_{A^*}(X_k^*, y^*_{[s_k^*:s_k^*+L-1]})\}.$$

For $k \in M$ and assuming $\varepsilon < \min\{c_b, c_c\}$, the minimum must be attained by the second term, hence $|j_k - s_k^*| \le \frac{1}{c_c}(\text{ed}(X_k, Y_k) + 2\,\text{ed}_{A^*}(X_k^*, y^*_{[s_k^*:s_k^*+L-1]}))$. Finally, by Eq. (4.1),

$$\text{ed}(X_k, Y_k) \le \text{ed}(X_k^*, y^*_{[s_k^*:s_k^*+L-1]}) \le 2\,\text{ed}_{A^*}(X_k^*, y^*_{[s_k^*:s_k^*+L-1]}),$$

and altogether this proves Claim 4.4. □

The second claim proves that the sum to be estimated, $\sum_{k\in M} |S_k|$ (corresponding to empirical matches of $X_k$), is composed of two parts: the analogous sum corresponding to "ideal" matches under $A^*$, plus (twice) the deviation in the starting positions of empirical matches $Y_k$ versus the "ideal" matches $Y_k^*$. Specifically, define $S_k^*$ to be the positions in $y^*_{[s_k^*:s_k^*+L-1]}$ that appear also in some block $y^*_{[s_{k'}^*:s_{k'}^*+L-1]}$ for $k' \in M \setminus \{k\}$. We have the following claim.

CLAIM 4.5. *With probability at least* $1 - d^{-\Omega(C)}$, *we have* $\sum_{k\in M} |S_k| \le \sum_{k\in M} \left(|S_k^*| + 2|s_k^* - j_k|\right)$.

PROOF. Consider $k, k' \in M$ with $k < k'$. Observe that $\text{ed}_{A^*}(X_k^*, y^*_{[s_k^*:s_k^*+L-1]})$ and $\text{ed}_{A^*}(X_{k'}^*, y^*_{[s_{k'}^*:s_{k'}^*+L-1]})$ must be at most $\frac{c_c}{32} \cdot L$, or otherwise, by Lemma 2.2(b), with high probability, $\text{ed}(X_k, Y_k)$ or $\text{ed}(X_{k'}, Y_{k'})$, respectively, is at least $c_b \cdot \frac{c_c/32}{\log 64/c_c} \cdot pL$, and thus $k$ or $k'$, respectively, is not in $M$.

We can now apply Lemma 3.2(c) to obtain that

$$s_{k'}^* - s_k^* \ge L - \text{ed}_{A^*}\left(X_k^*, y^*_{[s_k^*:s_k^*+L-1]}\right) - \text{ed}_{A^*}\left(X_{k'}^*, y^*_{[s_{k'}^*:s_{k'}^*+L-1]}\right)$$

and this, together with Claim 4.4, yields

$$\begin{aligned} j_{k'} - j_k &\ge s_{k'}^* - s_k^* - |j_k - s_k^*| - |j_{k'} - s_{k'}^*| \\ &\ge L - \tfrac{5}{c_c} \cdot \left[\text{ed}_{A^*}\left(X_k^*, y^*_{[s_k^*:s_k^*+L-1]}\right) + \text{ed}_{A^*}\left(X_{k'}^*, y^*_{[s_{k'}^*:s_{k'}^*+L-1]}\right)\right]. \end{aligned}$$

Concluding, since $\mathrm{ed}_{A^*}(X^*_k, y^*_{[s^*_k:s^*_k+L-1]})$ and $\mathrm{ed}_{A^*}(X^*_{k'}, y^*_{[s^*_{k'}:s^*_{k'}+L-1]})$ are at most $\frac{c_e}{32} \cdot L$, we have

$$j_{k'} - j_k > L/2. \tag{4.3}$$

Thus, every position in $Y_k$ for $k \in M$ can appear in at most one other block $Y_{k'}$ for $k' \in M \setminus \{k\}$. It not difficult to see that $\sum_{k\in M} |S_k| \le \sum_{k\in M}(|S^*_k| + 2|s^*_k - j_k|)$, since every position in $S_k$ either contributes also to $S^*_k$, or to $|s^*_k - j_k|$, or to some $|s^*_{k'} - j_{k'}|$, for $k' \in M$, and furthermore the contributions of the same type are all distinct. Claim 4.5 follows. □

We can now prove part (c) by combining all the above. Specifically, applying Claim 4.5, then Lemma 3.2(d) and Claim 4.4, and finally Lemma 3.2(a), we have

$$\sum_{k\in M}|S_k| \le \sum_{k\in M}(|S^*_k|+2|s^*_k-j_k|) \le 2\,\mathrm{ed}(x^*, y^*)+O(1)\sum_{k\in M}\mathrm{ed}_{A^*}\left(X^*_k, y^*_{[s^*_k:s^*_k+L-1]}\right) \le O(\mathrm{ed}(x^*, y^*)).$$

Part (d) of the lemma follows directly from Eq. (4.3).

### 4.2. Reducing a Smoothed Instance to an Ulam Instance

Next we 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 earlier, for the sake of analysis we may set $Q$ to be the identity permutation, and construct $P$ as a function of $x$ and $y$. Lemma 4.6 defines $P$ in its entirety, while ensuring the locality property: that every character in $P$ can be computed from local information. (As we shall see later, the algorithm uses this locality property to compute "on the fly" the same $P$, $Q$, up to relabeling of the symbols.)

The basic idea appears simple. First, we partition $x$ into blocks of length $L = O(\frac{1}{p}\log d)$. Then, each such block $x_{[kL-L+1:kL]}$ is matched to its closest block in $y$, say $y_{[l:l+L-1]}$, and then $P_{[kL-L+1:kL]}$ is defined in a simple way that depends only on $Q_{[l:l+L-1]}$ and on $\mathrm{ed}(x_{[kL-L+1:kL]}, y_{[l:l+L-1]})$, and satisfies $\mathrm{ed}(P_{[kL-L+1:kL]}, Q_{[l:l+L-1]}) = \mathrm{ed}(x_{[kL-L+1:kL]}, y_{[l:l+L-1]})$. This reduction preserves the edit distance locally (at the block level), although it is not clear it is true also globally (for the entire strings). We indeed prove the latter, that is, that $\mathrm{ed}(P, Q)$ approximates $\mathrm{ed}(x, y)$, using the technical machinery developed in Lemma 4.3.

The main challenge we face in implementing this basic idea is that characters may repeat in $P$, because the blocks we match against in $y$ may overlap with each other. A straightforward fix to this issue could be to change these repetitions to completely new symbols (distinct symbols that do not appear in $Q$). This fix increases $\mathrm{ed}(P, Q)$, although, as we show, only by a small factor. Unfortunately, this fix also introduces dependencies between different blocks in $P$, violating the locality requirement. We thus refine this fix by going through two smaller transformations of $P$, which reduces the dependencies of a position to only the nearby blocks (in $x$ and in $y$).

LEMMA 4.6 (REDUCTION TO ULAM). *Fix $\varepsilon > 0$, let $(x, y) \in \mathtt{Smooth}_p(x^*, y^*, A^*)$ and $L = \frac{C}{p}\log d$ for a sufficiently large constant $C > 0$. Then, there exist two permutations $P$ and $Q = (1, 2, \ldots, d)$ such that, with probability at least $1 - d^{-\Omega(C)}$, the following holds.*

—**Distance:** $\Omega(1) \cdot \mathrm{ed}(x, y) \le \mathrm{ed}(P, Q) \le O(\frac{\log 1/p}{p} + \frac{1}{\varepsilon p}) \cdot \mathrm{ed}(x, y)$*; and*

—**Locality:** *For all $k \in [d/L]$, $j \in [kL - L + 1 : kL]$, and $s_k = \mathrm{match}(x_{[kL-L+1:kL]})$:*

(1) *$P[j]$ can be computed in time $O(L^3)$, using only $s_k$, $x_{[kL-2L+1:kL]}$, and $y_{[s_k-6L:s_k+L-1]}$. Furthermore, either $P[j] \in [d]$ or $P[j] = d + j$.*

(2) *For all* $l \in [d - L + 1]$ *such that* $|l - s_k| \geq 2L$, *we have* $\mathrm{ed}(x_{[kL-L+1:kL]}, y_{[l:l+L-1]}) \geq \Omega(pL)$. *Furthermore, if* $P[j] \in [d]$, *then necessarily* $\mathrm{ed}(x_{[kL-L+1:kL]}, y_{[s_k:s_k+L-1]}) \leq \varepsilon pL$.

PROOF. We shall say that position $j \in [d]$ (in $P$) is *invalidated* if it is set to the symbol $d + j$. All other positions will be set to symbols in the range $[d]$. Recall that the alphabet is $\Sigma = [2d]$ and that $Q$ is the identity, hence invalidated positions in $P$ match no character in $Q$. We first give a complete description of the construction of $P$, and then prove its properties (distance and locality).

The permutation $P$ is constructed by first defining a string $P^1$, then invalidating some positions to obtain $P^2$, and then invalidating more positions to obtain the final $P$. The intermediate strings $P^1$ and $P^2$ might not be permutations. We now describe these three stages and a preceding setup stage.

*Setup*. Partition $x$ into $d/L$ blocks of length $L$, denoted $X_k$, and for each $k \in [d/L]$ let $Y_k = \mathrm{match}(X_k)$. Let $s_k$ be the starting position of $Y_k$ and let $c_k = \mathrm{ed}(X_k, Y_k)$. Let $M = \{k \in [d/L] : \mathrm{ed}(X_k, Y_k) \leq \varepsilon pL\}$.

$P^1$. For every $k \in M$, set $P^1_{[kL-L+1:kL]}$ to be equal to the block $Q_{[s_k:s_k+L-1]}$, except that the first $c_k$ symbols are invalidated (thus ensuring $\mathrm{ed}(P^1_{[kL-L+1:kL]}, Q_{[s_k:s_k+L-1]}) = c_k$). For each $k \in [d/L] \setminus M$, invalidate the entire block $P^1_{[kL-L+1:kL]}$.

$P^2$. Let $F \subseteq M$ be the following set. For each $k \in M, k > 1$, put $k$ into $F$ if (**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 $P^1_{[kL-L+1:kL]}$ with $k \in F$.

$P$. Invalidate all positions $j \in [d]$ for which the symbol $P^2[j]$ occurs previously in $P^2$ to the left of the position $j$. That is, for each symbol, we invalidate all its occurrences except the very first one.

It should be evident why we invalidate the entire blocks $X_k$ for $k \notin M$. The reason we further invalidate blocks $X_k$ for $k \in F$ during the construction of $P^2$ is to ensure that the computation of the last step ($P$) is local. In particular, for a particular symbol $P^1[j]$, we need to be able to check whether the symbol has occured to the left of $j$ in $P^1$. In particular, it is possible that there is some $j'$ satisfying $P^1[j'] = P^1[j]$ with $j - j' \gg \Omega(L)$, and hence hard to find locally. However, such a situation—where $j - j' \gg \Omega(L)$—may be possible only when $k - 1 \notin M$. Hence, we invalidate all blocks $k$ with $k - 1 \notin M$, which is condition (i) in the definition of $F$. Checking condition (i) by itself may also not be a local operation, and this concern is rectified by condition (ii), because checking the combination of (i) or (ii) is now a local operation.

We proceed to prove the distance property, using two claims that provide a lower bound and an upper bound on $\mathrm{ed}(P, Q)$, respectively.

CLAIM 4.7. $\mathrm{ed}(x, y) \leq 6 \cdot \mathrm{ed}(P, Q)$.

PROOF. First, observe that $\mathrm{ed}(P^1, Q) \leq \mathrm{ed}(P, Q)$ because invalidating some positions can only increase the edit distance. We proceed to show that $\mathrm{ed}(x, y) \leq 6 \cdot \mathrm{ed}(P^1, Q)$. Fix an optimal alignment $\tilde{A}$ between $P^1$ and $Q$, and construct an alignment $A$ between $x$ and $y$ as follows. For each $k \in [d/L]$, consider the block $X_k$. If $k \notin M$ (i.e., $c_k > \varepsilon pL$), then the corresponding block in $P^1$ has only invalidated positions, which cannot be aligned to $Q$, hence the same alignment is valid for $x$ on these blocks. Otherwise, by construction, $\mathrm{ed}(x_{[kL-L+1:kL]}, y_{[s_k:s_k+L-1]}) = c_k = \mathrm{ed}(P^1_{[kL-L+1:kL]}, Q_{[s_k:s_k+L-1]})$. Let $t_k$ be the number of nonaligned positions in the $k$th block of $P^1$ under $\tilde{A}$. Clearly, $t_k \geq c_k$ since the $k^{\text{th}}$ block of $P^1$ has $c_k$ invalidated positions that cannot match any position in $Q$. We construct $A$ by aligning the corresponding $k$th block in $x$ against only the middle $L - 2t_k$ symbols in $y_{[s_k:s_k+L-1]}$, in the best possible way. The number of unaligned positions in $x_{[kL-L+1:kL]}$

is at most $c_k + 2t_k \le 3t_k$, and thus $\mathrm{ed}(x,y) \le 2\,\mathrm{ed}_A(x,y) \le 2\sum_k 3t_k = 6\,\mathrm{ed}_{\tilde{A}}(P^1, Q) \le 6\,\mathrm{ed}(P^1, Q)$.

It remains to show that $A$ is a valid alignment. It suffices to consider $k, k' \in M$ with $k < k'$, and prove that matches under $A$ from the $k$th block are to the left of those from the $k'$th block. If it were true that $s_k + L - 1 < s_{k'}$, we would have been done; but this is not generally true. Instead, by definition of $t_k$ we must have $(s_k + L - 1) - t_k < s_{k'} + t_{k'}$: as $\tilde{A}$ (a valid alignment between $P$ and $Q$) must align a position from the $k$th block to character of value at least $s_k + L - 1 - t_k$ and, at the same tie, $\tilde{A}$ must align a position from the $k'$th block to a character of value at most $s_{k'} + t_{k'}$. Since our construction of $A$ is limited to the middle $L - 2t_k$ symbols in each $y_{[s_k:s_k+L-1]}$, we get that $A$ is indeed monotonically increasing on $A^{-1}([d])$. □

Claim 4.8. *With high probability,* $\mathrm{ed}(P, Q) \le O(\frac{\log(1/p)}{p} + \frac{1}{\varepsilon p})\,\mathrm{ed}(x,y)$.

Proof. First, we argue that the noninvalidated positions of $P$(in any of the three stages) form an increasing sequence, and thus $\frac{1}{2}\,\mathrm{ed}(P, Q)$ is upper bounded by the number of the invalidated positions. Note that we are precisely in the conditions of Lemma 4.3. We use the notation from that lemma for the rest of this proof, and assume that the high-probability event holds (i.e., all conclusions hold). The lemma says that the starting positions of $Y_k$, for $k \in M$, are strictly increasing and moreover increase by $> L/2$ each time. Thus, after the invalidations, a block $P_{[kL-L+1:kL]}$ is either completely invalidated (if $k \notin M \setminus F$), or its invalidated positions form a contiguous sequence at the beginning of the block. Furthermore, in the latter case, the symbol in the position of $kL$ is smaller than any non-invalidated symbol in $P_{[kL+1:d]}$. Thus, all the non-invalidated position of $P$ form an increasing sequence.

We now upper bound the number of invalidated positions in the construction of $P^1$, in the transformation to $P^2$, and in the transformation to $P$. The number of positions invalidated in the construction of $P^1$ is $L \cdot (d/L - |M|) + \sum_{k \in M} c_k$. The number of positions invalidated in the transformation to $P^2$ is at most $L \cdot |F|$. The number of positions invalidated in the transformation to $P$ is at most $\sum_k |S_k|$, because the number of positions in $P_{[kL-L+1:kL]}$ invalidated in this step is at most $|S_k|$. Lemma 4.3 bounds all these quantities, except for $|F|$.

We now bound $L \cdot |F|$. Notice that each $k \in F$ corresponds either to a block $k - 1 \in [d/L] \setminus M$ (case (i) in the definition of $F$) or to some block of length $L$ strictly between the blocks $Y_{k-1}$ and $Y_k$ (case (ii)). Positions appearing in blocks of the latter type do not belong to any $Y_k$ for $k \in M$, and thus their number is at most $L \cdot (d/L - |M|)$ plus the number of positions that appear in two blocks $Y_k, Y'_k$ for distinct $k, k' \in M$. Since the number of such latter positions is at most $\sum_k |S_k|$ by the definition of $S_k$ (in Lemma 4.3), the total comes out to $L \cdot |F| \le L \cdot (d/L - |M|) + (L \cdot (d/L - |M|) + \sum_k |S_k|)$.

Concluding, using Lemma 4.3, we get that, with high probability,

$$\begin{aligned}
\tfrac{1}{2}\,\mathrm{ed}(P, Q) &\le \sum_{k \in M} c_k + L \cdot (d/L - |M|) + L \cdot |F| + \sum_{k \in M} |S_k| \\
&\le \sum_{k \in M} \mathrm{ed}(X_k, Y_k) + 3L \cdot (d/L - |M|) + 2 \sum_{k \in M} |S_k| \\
&\le O(1) \cdot \mathrm{ed}(x^*, y^*) + O\left(\frac{1}{\varepsilon p}\right) \mathrm{ed}(x,y).
\end{aligned}$$

Futhermore, using Theorem 2.1, we conclude that $\mathrm{ed}(P, Q) \le O(\frac{\log(1/p)}{p} + \frac{1}{\varepsilon p})\,\mathrm{ed}(x,y)$. □

We proceed to proving the locality property. In our language, $j$ is inside the block $X_k$ and we need to prove that $P[j]$ depends only on the blocks $X_k, Y_k$ together with

regions of $6L$ positions preceding them. The algorithm for computing $P[j]$ follows the description of the construction of $P$.

Specifically, we compute $P[j]$ in a local manner as follows. For $Y_k = \text{match}(X_k)$ starting at $s_k$ in $y$, we set $P[j] = s_k + j - (kL - L + 1)$, unless it is invalidated according to the following procedure, in which case $P[j] = d + j$. If $\text{ed}(X_k, Y_k) > \varepsilon pL$, then $j$ in invalidated (stage 1 invalidation). Also, if $kL-L+1+\text{ed}(X_k, Y_k) > j$, then $j$ is invalidated (stage 1 invalidation). In case $k > 1$, we consider the block $X_{k-1}$ and define $s'_{k-1}$ to be the position $s \in [s_k - 4L : s_k]$ that minimizes $\text{ed}(X_{k-1}, y_{[s:s+L-1]})$. If $\text{ed}(X_{k-1}, y_{[s'_{k-1}:s'_{k-1}+L-1]}) > \varepsilon pL$ or if $s_k - s'_{k-1} > 2L$, then $j$ is invalidated (stage 2 invalidation). Next, we take

$$s'_{k-2} = \operatorname*{argmin}_{s \in [s'_{k-1}-4L:s'_{k-1}]} \text{ed}(X_{k-2}, y_{[s:s+L-1]}).$$

If $k = 2$ or $\text{ed}(X_{k-2}, y_{[s'_{k-2}:s'_{k-2}+L-1]}) \le \varepsilon pL$ and $s'_{k-1} - s'_{k-2} \le 2L$, and also if $j - (kL - L + 1) < (s'_{k-1} + L) - s_k$, then $j$ is invalidated (stage 3 invalidation).

The correctness of this algorithm follows from the claim that $P[j]$ is invalidated in this procedure if and only if it is invalidated in the original three-stages construction. We argue this claim next. It is easy to see that stage 1 invalidations are correct. Next, suppose $k > 1$ and $j$ is not invalidated in stages 1 or 2 of the original construction, which happens if $k - 1 \in M$ and $s_k - s_{k-1} \le 2L$. In this case, our algorithm will obtain $s'_{k-1} = s_{k-1}$, and thus $P[j]$ will not be invalidated in stage 2 of the above algorithm. Conversely, if $k-1 \notin M$, then we have $\text{ed}(X_{k-1}, y_{[s'_{k-1}:s'_{k-1}+L-1]}) > \varepsilon pL$, or else $s_k - s_{k-1} > 2L$, and then $s'_{k-1} \le \max\{s_{k-1}, s_k - 4L + (2L - 1)\} < s_k - 2L$ (namely, if $s_{k-1} < s_k - 4L$, then $\text{ed}(X_{k-1}, y_{[s:s+L-1]}) > \varepsilon pL$ for all $s \ge s_k - 2L$ by Lemma 2.2(c)). In both cases, the algorithm invalidates $P[j]$. Finally, if $P[j]$ was not invalidated in stages 1 or 2, $P[j]$ can be invalidated in the third stage if $Y_k$ intersects with $Y_{k-1}$ (when $k - 1 \in M \setminus F$). To check the intersection with $Y_{k-1}$, the algorithm checks additionally that $k - 2 \in M$ (unless $k = 2$) and $s_{k-2} - s_{k-1} \le 2L$ (implying $k - 1 \in M \setminus F$). If this condition passes, then we invalidate $P[j]$ iff symbol $P[j] = s_k + j - (kL - L + 1)$ is in the intersection of $Y_k$ and $Y_{k-1}$ (note that the matching symbol $P[j - L - s_k + s_{k-1}]$ in $P_{[kL-2L+1:kL-L]}$ cannot have been invalidated in stages 1 and 2).

We now prove the second part of the locality property. By construction, $P[j] = d + j$ if $\text{ed}(x_{[kL-L+1:kL]}, y_{[s_k:s_k+L-1]}) > \varepsilon pL$. Now, let $k^* = \text{match}_{A^*}(i, L)$, and let $c_c$ be the constant from Lemma 2.2. If $\text{ed}_{A^*}(x^*_{[i:i+L-1]}, y^*_{[k^*:k^*+L-1]}) \ge \frac{c_c}{4} L$, then, for all $l \in [d - L + 1]$, we have $\text{ed}(x_{[kL-L+1:kL]}, y_{[l:l+L-1]}) = \Omega(pL)$ by Lemma 2.2(b), and we reach the desired conclusion. Next, suppose that $\text{ed}_{A^*}(x^*_{[kL-L+1:kL]}, y^*_{[k^*:k^*+L-1]}) < \frac{c_c}{4} L$. Then, for every $l \in [d - L + 1]$ such that $|l - k^*| \ge L$, we have that $\text{ed}(x_{[kL-L+1:kL]}, y_{[l:l+L-1]}) = \Omega(pL)$ by Lemma 2.2(c). If $\text{ed}(x_{[kL-L+1:kL]}, y_{[s_k:s_k+L-1]}) \le \varepsilon pL$, then $|s_k - k^*| \le L$, and thus, for all $l \in [d - L + 1]$ s.t. $|l - s_k| \ge 2L$, we have $\text{ed}(x_{[kL-L+1:kL]}, y_{[l:l+L-1]}) > \Omega(pL)$. But if $\text{ed}(x_{[kL-L+1:kL]}, y_{[s_k:s_k+L-1]}) > \varepsilon pL$ then there is nothing to prove since, by definition, $s_k$ is the $l$ that minimizes $\text{ed}(x_{[kL-L+1:kL]}, y_{[l:l+L-1]})$. This completes the proof of Lemma 4.6.

### 4.3. The Sublinear-Time Algorithm for Smoothed Instances

We now describe the sublinear algorithm for the smoothed instance $\texttt{Smooth}_p(x^*, y^*, A^*)$ and thus prove Theorem 4.1. The algorithm basically performs a reduction to a similar problem (distance estimation) under the Ulam metric, and solves the latter using the algorithm of Andoni and Nguyen [2010] in a black-box fashion.

Proof of Theorem 4.1. We will use the sublinear algorithm from [Andoni and Nguyen 2010] as a black box. We call their algorithm $\mathcal{A}$ and note that $\mathcal{A}$ makes $\tilde{O}(\sqrt{d} + d/\text{ed}(P, Q))$ queries to $P, Q$ and has the same running time, while succeeding with constant probability. For completeness, we note that the algorithm $\mathcal{A}$ reduces

the problem to several *decision version* problems of distance estimation, and then solves the decision version problem. More precisely, the decision version is, for a given threshold $R \in [d]$, to decide whether the distance is $\mathrm{ed}(P, Q) \leq R$ or $\mathrm{ed}(P, Q) > \alpha R$ for approximation factor $\alpha = O(1)$. The algorithm of Andoni et al. [2009] for the decision version runs in time $\tilde{O}(\sqrt{d} + d/R)$.

We would like to run $\mathcal{A}$ on the permutations $P, Q$ obtained from applying Lemma 4.6 to our input $(x, y)$. Since we cannot afford to compute the entire $P$, our reduction will generate on the fly (and feed them into $\mathcal{A}$) two permutations that are equivalent to $P$ and $Q$, up to a relabeling of the symbols. As explained at the beginning of section 4, the algorithm $\mathcal{A}$ is independent of the actual names of the symbols, hence its output is invariant under this relabeling.

We describe here our reduction, viewing it as a data structure that has random access to $x$ and $y$, and provides a random access interface to the permutations $P$ and $Q$ (modulo relabeling). This data structure will be used by the algorithm $\mathcal{A}$. Let $L$ be defined as in Lemma 4.6, and assume that the high-probability event described in the lemma holds.

Our reduction keeps for each of $P$ and $Q$ two data structures, one to keep track of the relabeling and one to keep track of the blocks. Let $\hat{P}$ store the relabeling of $P$, namely, $\hat{P}(i)$ for a position $i \in [d]$ represents the new symbol given to $P[i]$ by the relabeling, or the value $\perp$ if position $i$ in $P$ has not been queried yet. We assume $\hat{P}$ is implemented so as to support fast inverse search, that is, given a symbol $a \in [2d]$ it can report $\hat{P}^{-1}(a)$. Let $T_P$ be a trie (or another data structure implementing a dictionary) that stores the substrings $x_{[kL-L+1:kL]}$ for all $k \in [d/L]$ such that at least one of the positions among $[kL - L + 1 : kL]$ was queried in $P$. We define $\hat{Q}$ for $Q$ analogously to $\hat{P}$; note that if $P[i]$ and $Q[j]$ have already been queried, then $\hat{P}(i) = \hat{Q}(j)$ if and only if $P[i] = Q[j] = j$. Finally, the trie $T_Q$ for $Q$ stores:

(1) all substrings $y_{[l:l+L-1]}$ where at least one queried position $j \in [d]$ in $Q$ satisfies $|l - j| \leq L$; and
(2) all length $L$ strings within edit distance $\varepsilon pL$ from such $y_{[l:l+L-1]}$.

The query to a position $P[i]$ works as follows. If $\hat{P}(i) \neq \perp$, return $\hat{P}(i)$. Otherwise, add the substring $X_k = x_{[kL-L+1:kL]}$, where $i \in [kL - L + 1 : kL]$, into the trie $T_P$ (unless it is already present there). Then, check whether $X_k$ is present in the trie $T_Q$; if it is not, then assign a new symbol to $P[i]$, update $\hat{P}(i)$ accordingly, and return this symbol $\hat{P}(i)$. Suppose now that $X_k$ is present in $T_Q$, that is, it matches a string at edit distance at most $\varepsilon pL$ from a block $y_{[l:l+L-1]}$. Then, apply the locality algorithm from Lemma 4.6, and compute $P[i]$. More precisely, compute $s_k$, with $|l - s_k| \leq 2L$, that minimizes $\mathrm{ed}(x_{[kL-L+1:kL]}, y_{[s_k:s_k+L-1]})$. Note that this $s_k$ indeed satisfies $s_k = \mathrm{match}(x_{[kL-L+1:kL]})$ by Lemma 4.6, Locality-2 property. Applying the algorithm from Lemma 4.6, we compute $P[i]$ and store the value in $\hat{P}$. If $P[i] \notin [d]$, assign a new symbol to $P[i]$ and add it to $\hat{P}$. Otherwise (i.e., $P[i] \in [d]$ and thus $P[i] = Q[j] = j$ for some $j$), check whether the position $P[i]$ has been queried in $Q$ by checking whether $\hat{Q}(P[i]) \neq \perp$. If indeed $\hat{Q}(P[i]) \neq \perp$, then update accordingly $\hat{P}(i) = \hat{Q}(P[i])$. And if $\hat{Q}(P[i]) = \perp$, then assign a new symbol to $P[i]$ and update $\hat{P}(i)$ accordingly. In the end, return the symbol $\hat{P}(i)$.

The query to a position $Q[j]$ is almost analogous, with the obvious modifications to account for the asymmetry in the two tries $T_P$ and $T_Q$. Namely, if $\hat{Q}$ already holds a symbol for $Q[j]$, return it. Otherwise, add every substring $y_{[l:l+L-1]}$, where $|l - j| \leq L$, together with all the length $L$ strings at edit distance $\leq \varepsilon pL$, into the trie $T_Q$. For each added string, check whether the string is present in $T_P$; if it is, that is, the added string

matches some $X_k$ stored in $T_P$, then compute all of $P_{[kL-L+1:kL]}$ using Lemma 4.6, and if for any of the computed position $i$ we have $P[i] = j$ and $\hat{P}(i) \neq \perp$, then the new symbol for $Q[j]$ is that symbol, that is, set $\hat{Q}(j) = \hat{P}(i)$. If, at the end, $\hat{Q}(j)$ is still $\perp$, then we assign $Q[j]$ with a new symbol, and update $\hat{Q}(j)$ accordingly. Either way, return the symbol $\hat{Q}(j)$.

The correctness of the algorithm then follows immediately from the Locality part of Lemma 4.6. In particular, essentially by construction, $\hat{P}$, $\hat{Q}$ form a consistent relabeling of $P$, $Q$, respectively, although a partial one in the sense that some of the values are replaced by $\perp$.

To obtain the stated bounds on query complexity and runtime, we note that the overhead on each query to $P$ or $Q$ is $O(L)$ queries to $x, y$ and $O(L^{O(1)}d^{O(\varepsilon \log 1/\varepsilon)})$ time, where $d^{O(\varepsilon \log 1/\varepsilon)}$ is an upper bound on the number of strings at distance $\leq \varepsilon pL$ from any single string (of length $L$). To obtain the claimed dependence on $\varepsilon$, we replace the $\varepsilon$ used in this algorithm with $\varepsilon' = O(\varepsilon / \log \frac{1}{\varepsilon})$. □

## 5. CONCLUSIONS

It seems challenging to obtain a distance estimation algorithm whose smoothed running time is quasi-linear, that is, $d \cdot (\log d)^{O(1)}$, or whose approximation is independent of the smoothing parameter $p$ at the expense of increasing the runtime only by an $O(1/p)$ factor. Perhaps it is more important 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 obtained for the Ulam metric [Andoni et al. 2009].

## ACKNOWLEDGMENTS

We thank Dick Karp for useful discussions at an early stage of this research. We also thank the anonymous reviewers for their careful proofreading and several comments that improved the presentation.

## REFERENCES

ALDOUS, D. AND DIACONIS, P. 1999. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bull. Amer. Math. Soc. (N.S.) 36,* 4, 413–432.

ALTSCHUL, S. F., GISH, W., MILLER, W., MYERS, E. W., AND LIPMAN, D. J. 1990. A basic local alignment search tool. *J. Molec. Biol. 215,* 3, 403–410.

ANDONI, A., INDYK, P., AND KRAUTHGAMER, R. 2009. Overcoming the $\ell_1$ non-embeddability barrier: Algorithms for product metrics. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 865–874.

ANDONI, A., JAYRAM, T., AND PĂTRAŞCU, M. 2010a. Lower bounds for edit distance and product metrics via Poincaré-type inequalities. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*.

ANDONI, A. AND KRAUTHGAMER, R. 2008. The smoothed complexity of edit distance. In *Proceedings of the 35st International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science. Springer, 357–369.

ANDONI, A. AND KRAUTHGAMER, R. 2010. The computational hardness of estimating edit distance. *SIAM J. Comput. 39,* 6, 2398–2429. (Previously appeared in FOCS'07.)

ANDONI, A., KRAUTHGAMER, R., AND ONAK, K. 2010b. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the Symposium on Foundations of Computer Science*. (A full version is available at `http://arxiv.org/abs/1005.4033`.)

ANDONI, A. AND NGUYEN, H. L. 2010. Near-tight bounds for testing Ulam distance. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*.

ANDONI, A. AND ONAK, K. 2009. Approximating edit distance in near-linear time. In *Proceedings of the Symposium on Theory of Computing*. 199–204.

BAR-YOSSEF, Z., JAYRAM, T. S., KRAUTHGAMER, R., AND KUMAR, R. 2004. Approximating edit distance efficiently. In *Proceedings of the Symposium on Foundations of Computer Science*. 550–559.

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

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

Bille, P. and Farach-Colton, M. 2008. Fast and compact regular expression matching. *Theoret. Comput. Science 409,* 28, 486–496.

Blum, A. and Spencer, J. 1995. Coloring random and semi-random $k$-colorable graphs. *J. Algor. 19,* 2, 204–234.

Charikar, M. and Krauthgamer, R. 2006. Embedding the ulam metric into $\ell_1$. *Theory Comput. 2,* 11, 207–224.

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

Frieze, A. and McDiarmid, C. 1997. Algorithmic theory of random graphs. *Rand. Struct. Algor. 10,* 1-2, 5–42.

Gollapudi, S. and Panigrahy, R. 2006. A dictionary for approximate string search and longest prefix search. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*. ACM, 768–775.

Kushilevitz, E. and Nisan, N. 1997. *Communication Complexity*. Cambridge University Press.

Kushilevitz, E., Ostrovsky, R., and Rabani, Y. 2000. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput. 30,* 2, 457–474.

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

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

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

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

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

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

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

Received March 2009; revised April 2012; accepted May 2012