Polylogarithmic Approximation for Edit Distance and the Asymmetric Query Complexity*

Alexandr Andoni†
Princeton University / CCI
Princeton, NJ, USA
andoni@mit.edu

Robert Krauthgamer‡
Weizmann Institute of Sciences
Rehovot, Israel
robi@weizmann.ac.il

Krzysztof Onak§
MIT
Cambridge, MA, USA
konak@mit.edu

***Abstract*—We present a near-linear time algorithm that approximates the edit distance between two strings within a polylogarithmic factor. For strings of length n and every fixed $\varepsilon > 0$, the algorithm computes a $(\log n)^{O(1/\varepsilon)}$ approximation in $n^{1+\varepsilon}$ time. This is an *exponential* improvement over the previously known approximation factor, $2^{\tilde{O}(\sqrt{\log n})}$, with a comparable running time [Ostrovsky and Rabani, J. ACM 2007; Andoni and Onak, STOC 2009].**

This result arises naturally in the study of a new *asymmetric query* model. In this model, the input consists of two strings x and y, and an algorithm can access y in an unrestricted manner, while being charged for querying every symbol of x. Indeed, we obtain our main result by designing an algorithm that makes a small number of queries in this model. We then provide a nearly-matching lower bound on the number of queries.

Our lower bound is the first to expose hardness of edit distance stemming from the input strings being "repetitive", which means that many of their substrings are approximately identical. Consequently, our lower bound provides the first rigorous separation between edit distance and Ulam distance.

***Keywords*-edit distance, sampling, query complexity, linear-time algorithms, sublinear algorithms**

I. Introduction

Manipulation of strings has long been central to computer science, arising from the high demand to process texts and other sequences efficiently. For example, for the simple task of *comparing* two strings (sequences), one of the first methods emerged to be the *edit distance* (aka the Levenshtein distance) [25], defined as the minimum number of character insertions, deletions, and substitutions needed to transform one string into the other. This basic distance measure, together with its more elaborate versions, is widely used in a variety of areas such as computational biology, speech recognition, and information retrieval. Consequently, improvements in edit distance algorithms have the potential of major impact. As a result, computational problems involving edit distance have been studied extensively (see [29], [17] and references therein).

The most basic problem is that of computing the edit distance between two strings of length n over some alphabet. It can be solved in $O(n^2)$ time by a classical algorithm [33]; in fact this algorithm is a prototypical dynamic programming algorithm, see, e.g., the textbook [12] and references therein. Hence it is natural to ask whether we can speed up this canonical algorithm. Despite significant research over more than three decades, the running time has so far been improved only slightly, to $O(n^2/\log^2 n)$ [26], which remains the fastest algorithm known to date.[1]

Furthermore, a near-quadratic runtime is often unacceptable in modern applications that must deal with massive datasets, such as the genomic data. Hence practitioners tend to rely on faster heuristics [17], [29]. This has motivated the quest for faster algorithms at the expense of approximation, see, e.g., [18, Section 6] and [19, Section 8.3.2]. Indeed, the past decade has seen a serious effort in this direction.[2] One general approach is to design linear time algorithms that approximate edit distance. A linear-time $\sqrt{n}$-approximation algorithm immediately follows from the exact algorithm of [24], which runs in time $O(n + d^2)$, where d is the edit distance between the input strings. Subsequent research improved the approximation factor, first to $n^{3/7}$ [8], then

*This is an extended abstract. For a full version, refer to http://arxiv.org/abs/1005.4033

†Supported in part by NSF CCF 0832797.
‡Supported in part by The Israel Science Foundation (grant #452/08), and by a Minerva grant.
§Supported in part by NSF grants 0732334 and 0728645.

[1]The result of [26] applies to constant-size alphabets. It was extended to arbitrarily large alphabets in [11], albeit with an $O(\log \log n)^2$ factor loss in runtime.

[2]We shall not attempt to present a complete list of results for restricted settings (e.g., average-case/smoothed analysis, weakly-repetitive strings, and bounded distance-regime), for variants of the distance function (e.g., allowing more edit operations), or for related computational problems (such as pattern matching, nearest neighbor search, and sketching). See also the surveys of [29] and [31].

to $n^{1/3+o(1)}$ [10], and finally to $2^{\tilde{O}(\sqrt{\log n})}$ [7] (building on [30]). Predating some of this work was the *sublinear-time* algorithm of [9] achieving n^{ε} approximation, but only when the edit distance d is rather large.

Better progress has been obtained on *variants* of edit distance, where one either restricts the input strings, or allows additional edit operations. An example from the first category is the edit distance on non-repetitive strings (e.g., permutations of $[n]$), termed *the Ulam distance* in the literature. The classical Patience Sorting algorithm computes the exact Ulam distance between two strings in $O(n \log n)$ time. An example in the second category is the case of two variants of the edit distance where certain block operations are allowed. Both of these variants admit an $\tilde{O}(\log n)$ approximation in near-linear time [15], [28], [14], [13].

Despite the efforts, achieving a polylogarithmic approximation factor for the classical edit distance has eluded researchers for a long time. In fact, this has been the case not only in the context of linear-time algorithms, but also in the related tasks, such as nearest neighbor search, ℓ_1-embedding, or sketching. From a lower bounds perspective, only a *sublogarithmic* approximation has been ruled out for the latter two tasks [21], [22], [4], thus giving evidence that a sublogarithmic approximation for the distance computation might be much harder or even impossible to attain.

A. Results

Our first and main result is an algorithm that runs in near-linear time and approximates edit distance within a *polylogarithmic factor*. Note that this is *exponentially better* than the previously known factor $2^{\tilde{O}(\sqrt{\log n})}$ (in comparable running time), due to [30], [7].

Theorem I.1 (Main)**.** *For every fixed $\varepsilon > 0$, there is an algorithm that approximates the edit distance between two input strings $x, y \in \Sigma^n$ within a factor of $(\log n)^{O(1/\varepsilon)}$, and runs in $n^{1+\varepsilon}$ time.*

This development stems from a principled study of edit distance in a computational model that we call the *asymmetric query* model, and which we shall define shortly. Specifically, we design a query-efficient procedure in the said model, and then show how this procedure yields a near-linear time algorithm. We also provide a query complexity lower bound for this model, which matches or nearly-matches the performance of our procedure.

A conceptual contribution of our query complexity lower bound is that it is the first one to expose hardness stemming from "repetitive substrings", which means that many small substrings of a string may be approximately equal. Empirically, it is well-recognized that such repetitiveness is a major obstacle for designing efficient algorithms. All previous lower bounds (in any computational model) failed to exploit it, while in our proof the strings' repetitive structure is readily apparent. More formally, our lower bound provides the first rigorous separation of edit distance from Ulam distance (edit distance on non-repetitive strings). Such a separation was not previously known in any studied model of computation, and in fact all the lower bounds known for the edit distance hold to (almost) the same degree for Ulam distance. These models include: non-embeddability into normed spaces [21], [22], [4], lower bounds on sketching complexity [4], [2], and (symmetric) query complexity [9], [6].

Asymmetric Query Complexity: Before stating the results formally, we define the problem and the model precisely. Consider two strings $x, y \in \Sigma^n$ for some alphabet Σ, and let $\mathrm{ed}(x, y)$ denote the edit distance between the two strings. The computational problem is the promise problem known as the Distance Threshold Estimation Problem (DTEP) [32]: distinguish whether $\mathrm{ed}(x, y) > R$ or $\mathrm{ed}(x, y) \leq R/\alpha$, where $R > 0$ is a parameter (known to the algorithm) and $\alpha \geq 1$ is the *approximation factor*. We use DTEP_β to denote the case of $R = n/\beta$, where $\beta \geq 1$ may be a function of n.

In the *asymmetric query model*, the algorithm knows in advance (has unrestricted access to) one of the strings, say y, and has only *query access* to the other string, x. The *asymmetric query complexity* of an algorithm is the number of coordinates in x that the algorithm has to probe in order to solve DTEP with success probability at least $2/3$.

We now state our upper and lower bound results. Both exhibit a smooth *tradeoff* between the approximation factor and the query complexity. For simplicity, we state the bounds in two extreme regimes of approximation, when $\alpha = \mathrm{polylog}(n)$ and $\alpha = \mathrm{poly}(n)$. (The complete statements appear in the full version.)

Theorem I.2 (Query complexity upper bound)**.** *Let $\beta = \beta(n) \geq 2$. For every $0 < \varepsilon < 1$ there is an algorithm for DTEP_β achieving approximation $\alpha = (\log n)^{O(1/\varepsilon)}$ with βn^{ε} queries into x, and running in $n^{1+\varepsilon}$ time. For every integer $t \geq 2$ there is an algorithm for DTEP_β achieving approximation $\alpha = O(n^{1/t})$ with $O(\log^{t-1} n)$ queries into x.*

It is an easy observation that our general edit distance algorithm in Theorem I.1 follows immediately from the above query complexity upper bound theorem, by

running the latter for all β that are a power of 2.

Theorem I.3 (Query complexity lower bound)**.** *For a sufficiently large constant $\beta > 1$, every algorithm that solves DTEP_β with approximation $\alpha = \alpha(n) \geq 2$ has asymmetric query complexity $2^{\Omega\left(\frac{\log n}{\log \alpha + \log\log n}\right)}$. Moreover, for every fixed non-integer $t > 1$, every algorithm that solves DTEP_β with approximation $\alpha = n^{1/t}$ has asymmetric query complexity $\Omega(\log^{\lfloor t \rfloor} n)$.*

We summarize in Table I our results and previous bounds for DTEP_β under edit distance and Ulam distance. For completeness, we also present known results for a common query model where the algorithm has query access to both strings (henceforth referred to as the *symmetric query* model). We point out two implications of our bounds on the asymmetric query complexity:

- There is a strong separation between edit distance and Ulam distance. In the Ulam metric, a *constant* approximation is achievable with only $O(\log n)$ asymmetric queries (see [1], which builds on [16]). In contrast, for edit distance, we show an exponentially higher complexity lower bound, of $2^{\Omega(\log n/\log\log n)}$, even for a larger (polylogarithmic) approximation.
- Our query complexity upper and lower bounds are nearly-matching, at least for a range of parameters. At one extreme, approximation $O(n^{1/2})$ can be achieved with $O(\log n)$ queries, whereas approximation $n^{1/2-\varepsilon}$ already requires $\Omega(\log^2 n)$ queries (similar "phase transitions" happen at approximations around $n^{1/t}$ for $t \in \mathbb{N}$). At the other extreme, approximation $\alpha = (\log n)^{1/\varepsilon}$ can be achieved using $n^{O(\varepsilon)}$ queries, and requires $n^{\Omega(\varepsilon/\log\log n)}$ queries.

B. Connections of Asymmetric Query Model to Other Models

The asymmetric query model is related to two well-studied models, namely the communication complexity model and the symmetric query model (where the algorithm has query access to both strings). Specifically, the former is less restrictive than our model (i.e., easier for algorithms) while the latter is more restrictive (i.e., harder for algorithms).

Communication Complexity: In this setting, Alice and Bob each have a string, and they need to solve the DTEP_β problem by way of exchanging messages. The measure of complexity is the number of bits exchanged in order to solve DTEP_β with probability at least $2/3$.

The best upper bound known is $2^{\tilde{O}(\sqrt{\log n})}$ approximation with constant communication via [30], [23]. The only known lower bound says that approximation α requires $\Omega(\frac{\log n \ / \ \log\log n}{\alpha})$ communication [4], [2].

Our results immediately imply a one-way communication complexity for DTEP_β achieving $(\log n)^{O(1/\varepsilon)}$ approximation with $O(\beta n^\varepsilon)$ communication.

Symmetric Query Complexity: In another related model, the measure of complexity is the number of characters the algorithm has to query in *both* strings (rather than only in one of the strings). Naturally, the query complexity in this model is at least as high as the query complexity in the asymmetric model. This model has been introduced (for edit distance) in [9], and its main advantage is that it leads to *sublinear-time* algorithms for DTEP_β. The algorithm of [9] makes $\tilde{O}(n^{1-2\varepsilon} + n^{(1-\varepsilon)/2})$ queries (and runs in the same time), and achieves n^ε approximation. However, it only works for $\beta = O(1)$. The best query lower bound is of $\Omega(\sqrt{n/\alpha})$ for any approximation factor $\alpha > 1$ for both edit and Ulam distance [9], [6].

C. Future Directions

We study a new query model that seems to better elucidate the hardness stemming from "repetitiveness" of strings, obtaining eventually the first algorithm that computes a polylogarithmic approximation for edit distance in near-linear time. We believe that our techniques may foster progress on other tasks involving edit distance, such as the nearest neighbor search. We mention below a few natural, tangible goals for future investigation.

Symmetric Model: Extend our results to the symmetric query model. A query upper bound would likely lead to improved *sub-linear time* algorithms, the problem studied in [9]. From the perspective of a lower bound, it seems plausible that a variation of our hard distribution gives a bound of $n^{1/2+\Omega(1/\log\log n)}$ for polylogarithmic approximation, improving over the state-of-the-art $\tilde{\Omega}(\sqrt{n})$ lower bound in this model.

Embedding Lower Bounds: Is there an $\omega(\log n)$ lower bound for the distortion required to embed edit distance into ℓ_1? Such a lower bound would answer a well-known open question [27]. Note that the core component of our hard distribution, the shift metric (i.e., hamming cube augmented with cyclic shift operations), is known to require distortion $\Omega(\log n)$ [22].

Communication Complexity: Prove a communication complexity upper bound of n^ε for all distance regimes, i.e., independent of β (instead of the current $\beta \cdot n^\varepsilon$), for DTEP_β with polylogarithmic approximation.

Improved Time Complexity: Tighten the asymmetric query complexity upper bound to $2^{o(\varepsilon \log n)}$ for approximation $(\log n)^{O(1/\varepsilon)}$. This may ultimately lead

Model	Metric	Approx.	Complexity	Remarks
Near-linear time	Edit	$(\log n)^{O(1/\varepsilon)}$	$n^{1+\varepsilon}$	Theorem I.1
	Edit	$2^{\tilde{O}(\sqrt{\log n})}$	$n^{1+o(1)}$	[7]
Asymmetric query complexity	Edit	$n^{1/t}$	$O(\log^{t-1} n)$	Theorem I.2 (fixed $t \in \mathbb{N}, \beta > 1$)
	Edit	$n^{1/t}$	$\Omega(\log^{\lfloor t \rfloor} n)$	Theorem I.3 (fixed $t \notin \mathbb{N}, \beta > 1$)
	Edit	$(\log n)^{1/\varepsilon}$	$\beta n^{O(\varepsilon)}$	Theorem I.2
	Edit	$(\log n)^{1/\varepsilon}$	$n^{\Omega(\varepsilon/\log\log n)}$	Theorem I.3 (fixed $\beta > 1$)
	Ulam	$2+\varepsilon$	$O_\varepsilon(\beta \log\log\beta \cdot \log n)$	[1]
Symmetric query complexity	Edit	n^{ε}	$\tilde{O}(n^{\max\{1-2\varepsilon,(1-\varepsilon)/2\}})$	[9] (fixed $\beta > 1$)
	Ulam	$O(1)$	$\tilde{O}(\beta + \sqrt{n})$	[6]
	Ulam+edit	$O(1)$	$\tilde{\Omega}(\beta + \sqrt{n})$	[6]

Table I
KNOWN RESULTS FOR DTEP_β AND ARBITRARY $0 < \varepsilon < 1$.

to an algorithm that runs in time $n^{1+o(1)}$ and approximates edit distance within a factor of, say, $O(\log^2 n)$. Such a goal seems plausible by perhaps a more careful sampling.

More ambitiously, can one directly use our edit distance characterization to compute an $O(\log n)$ approximation in subquadratic time?

II. FAST ALGORITHMS VIA ASYMMETRIC QUERY COMPLEXITY

In this section we give an overview of our near-linear time algorithm for estimating the edit distance between two strings, stated in Theorem I.1. The algorithm is obtained via an efficient query algorithm for the DTEP_β problem, i.e., Theorem I.2. Here we include only an overview of the algorithm steps, and defer the details and proofs to the full version.

A high-level intuition for the near-linear time algorithm is as follows. The classical dynamic programming for edit distance runs in time that is the product of the lengths of the two strings. It seems plausible that, if we manage to "compress" one string to size n^ε, we may be able to compute the edit distance in time only $n^\varepsilon \cdot n$. Indeed, this is exactly what we accomplish. Specifically, our "compression" is achieved via a sampling procedure, which samples $\approx n^\varepsilon$ positions of x, and then approximates $\mathrm{ed}(x, y)$ in time $n^{1+\varepsilon}$. Of course, the main challenge is, by far, sampling x so that the above is even possible.

Our query upper bound has two major components. The first component is a *characterization* of the edit distance by a different "distance", denoted $\mathcal{E}$, which approximates $\mathrm{ed}(x, y)$ well. The characterization is parametrized by an integer parameter $b \geq 2$ governing the following tradeoff: a small b leads to a better approximation, whereas a large b leads to a faster algorithm. The second component is a *sampling algorithm* that approximates $\mathcal{E}$ for some settings of the parameter b, up to a constant factor, by querying a small number of positions in x.

A. Edit Distance Characterization: the $\mathcal{E}$-distance

Our characterization is based on a hierarchical decomposition of the edit distance computation, which is obtained by recursively partitioning the string x, each time into b blocks. We shall view this decomposition as a b-ary tree. Then, intuitively, the $\mathcal{E}$-distance at a node is the sum, over all b children, of the minima of the $\mathcal{E}$-distances at these children over a certain range of displacements (possible "shifts" with respect to the other strings). At the leaves (corresponding to single characters of x), the $\mathcal{E}$-distance is simply the Hamming distance to corresponding positions in y. We note that the characterization is asymmetric in the two strings.

We give a formal definition below, after we establish some notation. We fix the arity $b \geq 2$ of the tree, and let $h \stackrel{\text{def}}{=} \log_b n \in \mathbb{N}$ be the height of the tree. We denote by $x[s : s + l]$ the substring of x starting at s and ending at $s + l - 1$ (i.e., $[s : s + l]$ stands for the interval $\{s, s + 1, \ldots, s + l - 1\}$). For some tree level i, where $0 \leq i \leq h$, let $l_i \stackrel{\text{def}}{=} n/b^i$ be the length of blocks at that level. Then we define $B_i \stackrel{\text{def}}{=} \{1, l_i + 1, 2l_i + 1, \ldots\}$ to be the set of starting positions of blocks at level i.

The characterization is asymmetric in the two strings and is defined from a node of the tree to a position $u \in [n]$ of the string y. Specifically, if $i = h$, then the $\mathcal{E}$-*distance* of $x[s]$ to a position u is 0 only if $x[s] = y[u]$ and $u \in [n]$, and 1 otherwise. For $i \in \{0, 1, \ldots h - 1\}$ and $s \in B_i$, we recursively define the $\mathcal{E}$-distance $\mathcal{E}(i, s, u)$ of $x[s : s + l_i]$ to a position u as follows. Partition $x[s : s + l_i]$ into b blocks of length $l_{i+1} = l_i/b$, starting at positions $s + t_j$, where $t_j \stackrel{\text{def}}{=} j \cdot l_{i+1}$, $j \in \{0, 1, \ldots b - 1\}$. Intuitively, we would expect to define the $\mathcal{E}$-distance $\mathcal{E}(i, s, u)$ as the summation of the $\mathcal{E}$-distances of each block $x[s + t_j : s + t_j + l_{i+1}]$ to

the corresponding position in y, i.e., $u+t_j$. Instead, we additionally allow each block to be displaced by some shift r_j, incurring an additional charge of $|r_j|$ in the $\mathcal{E}$-distance. The shifts r_j are chosen such as to minimize the final distance. Formally,

$$\mathcal{E}(i,s,u) \stackrel{\text{def}}{=} \sum_{j=0}^{b-1} \min_{r_j \in \mathbb{Z}} \mathcal{E}(i+1, s+t_j, u+t_j+r_j) + |r_j| \,. \tag{1}$$

The $\mathcal{E}$-distance from x to y is just the $\mathcal{E}$-distance from $x[1:n+1]$ to position 1, i.e., $\mathcal{E}(0,1,1)$.

We illustrate the $\mathcal{E}$-distance for $b = 4$ in Fig. 1. Notice that without the shifts (i.e., when all $r_j = 0$), the $\mathcal{E}$-distance is exactly equal to the Hamming distance between the corresponding strings. Hence allowing the shifts r_j is what differentiates the $\mathcal{E}$-distance from the Hamming distance.

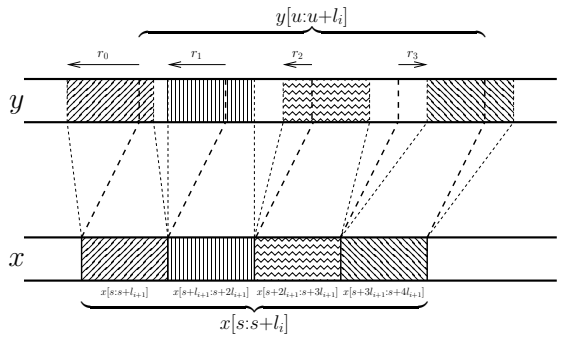

Figure 1. Illustration of the $\mathcal{E}$-distance $\mathcal{E}(i,s,u)$ for $b = 4$. The pairs of blocks of the same shading are the blocks whose $\mathcal{E}$-distance is used for computing $\mathcal{E}(i,s,u)$.

We prove that the $\mathcal{E}$-distance between x and y is an $O(bh) = O(\frac{b}{\log b}\log n)$ approximation to $\mathrm{ed}(x,y)$. Intuitively, the characterization manages to break-up the edit distance computation into *independent* distance computations on smaller substrings. The independence is crucial here as it removes the need to find a *global* alignment between the two strings, which is one of the main reasons why computing edit distance is hard. We note that while the high-level approach of recursively partitioning the strings is somewhat similar to the previous approaches from [9], [30], [7], the technical development here is quite different. The previous hierarchical approaches all relied on the following recurrence relation for the approximation factor α:

$$\alpha(n) = c \cdot \alpha(n/b) + O(b),$$

for some $c \geq 2$. It is easy to see that one obtains $\alpha(n) \geq 2^{\Omega(\sqrt{\log n})}$ for any choice of $b \geq 2$. In contrast, our characterization achieves *no multiplicative factor loss*, i.e., $c = 1$ and hence $\alpha(n) = O(bh)$. Some of the ideas behind the design and analysis of the characterization were inspired from [3].

We note that, for $b = 2$, the $\mathcal{E}$-distance is only an $O(\log n)$ approximation to $\mathrm{ed}(x,y)$. However, we do not know how to compute or approximate it well in better than quadratic time (one can easily compute a $1+\varepsilon$ approximation to $\mathcal{E}$-distance in $\tilde{O}_\varepsilon(n^2)$ time via a dynamic programming). Instead, we show that, using the sampling algorithm (described next), we can compute a $1+\varepsilon$ approximation to $\mathcal{E}$-distance for $b = (\log n)^{O(1/\varepsilon)}$ in $n^{1+\varepsilon}$ time.

B. Sampling Algorithm

We now describe the ideas behind our sampling algorithm. The sampling algorithm approximates the $\mathcal{E}$-distance between x and y up to a constant factor. The query complexity is $Q \leq \beta \cdot (\log n)^{O(h)} = \beta \cdot (\log n)^{\log_b n}$ for distinguishing $\mathcal{E}(0,1,1) > n/\beta$ from $\mathcal{E}(0,1,1) \leq n/(2\beta)$. For the rest of this overview, it is instructive to think about the setting where $\beta = n^{0.1}$.

The idea of the algorithm is to prune the $\mathcal{E}$-characterization tree, and in particular prune the children of each node. If we retain only $\mathrm{polylog}\, n$ children for each node, we would obtain $Q \leq (\log n)^{O(h)}$ leaves at the bottom, which correspond to the claimed sampled positions in x. The main challenge is how to perform this pruning.

A natural approach is to uniformly sample $\mathrm{polylog}\, n$ out of b children at each node, and use high concentration bounds to argue that the summation in the Equation (1) may be approximated only from the $\mathcal{E}$-distance estimates of the sampled children. Note that, since we use the minimum operator at each node, we have to aim, at each node, for an estimate that holds with high probability.

How much do we have to sample at each node? The "rule of thumb" for a Chernoff-type bound to work well is as follows. Suppose we have quantities $a_1, \ldots a_m \in [0,\rho]$ respecting an upper bound $\rho > 0$, and let $\sigma = \sum_{j\in[m]} a_j$. Suppose we sample several $j \in [m]$ to form a set J. Then, in order to estimate σ well (up to a small multiplicative factor) from a_j for $j \in J$, we need to sample essentially a total of $|J| \approx \frac{\rho}{\sigma} \cdot m \log m$ positions $j \in [m]$. We call this Uniform Sampling Lemma (see Lemma II.2 for a complete statement).

With the above "sampling rule" in mind, we can readily see that, at the top of the tree, until a level i, where $l_i = n/\beta$, there is no pruning that may be done (with the notation from above, we have $\rho \geq l_i = n/\beta$ and $\sigma = n/\beta$). Then, we hope to prune the tree at the subsequent levels.

However, it turns out that such pruning is not possible as described. Specifically, consider a node v at level i and its children v_j, for $j = 0, \dots b-1$. Suppose each child contributes a distance a_j to the sum $\mathcal{E}$ at node v (in Equation (1), for fixed u). Then, because of the bound on length of the strings, we have that $a_j \le l_{i+1} = (n/\beta)/b$. At the same time, for an average node v, we have $\sum_{j=0}^{b-1} a_j \approx l_i/\beta = n/\beta^2$. By the Uniform Sampling Lemma from above, we need to take a sample of size $|J| \approx \frac{n/(\beta b)}{n/\beta^2} \cdot b \log b = \beta \log b$. If β were constant, we would obtain $|J| \ll b$ and hence prune the tree (and, indeed, this approach works for $\beta \ll b$). However, once $\beta \gg b$, such pruning does not seem possible. In fact, one can give counter-examples where such pruning approach fails to approximate the $\mathcal{E}$-distance.

To address the above challenge, we develop a way to prune the tree *non-uniformly*. For different nodes we will sample its children at different, well-controlled rates. In fact, for each node we will assign a "precision" w with the requirement that a node v, at level i, with precision w, must estimate its $\mathcal{E}$-distances to positions u up to an *additive error* l_i/w. The pruning and assignment of precision will proceed top-bottom, starting with assigning a precision 4β to the root node. Intuitively, the higher the precision of a node v, the denser is the sampling in the subtree rooted at v.

Technically, our main tool is the *Precision Sampling Lemma*, which we use to assign the necessary precisions to nodes. It may be stated as follows (see Lemma II.3 for a more complete statement). The lemma says that there exists some distribution $\mathcal{W}$ and a reconstruction algorithm R such that the following two conditions hold:

- Fix some $a_j \in [0,1]$ for $j \in [m]$, with $\sigma = \sum_j a_j$. Also, pick w_j i.i.d. from the distribution $\mathcal{W}$ for each $j \in [m]$. Let $\hat{a}_j$ be estimators of a_j, up to an additive error of $1/w_j$, i.e., $|a_j - \hat{a}_j| \le 1/w_j$. Then the algorithm R, given $\hat{a}_j$ and w_j for $j \in [m]$, outputs a value that is inside $[\sigma - 1, \sigma + 1]$, with high probability.
- $\mathbb{E}_{w \in \mathcal{W}}[w] = \operatorname{polylog} m$.

To internalize this statement, fix $\sigma = 10$, and consider two extreme cases. At one extreme, consider some set of 10 j's such that $a_j = 1$, and all the others are 0. In this case, the previous uniform sampling rule does not yield any savings (to continue the parallel, uniform sampling can be seen as having $w_j = m$ for the sampled j's and $w_j = 1$ for the non-sampled j's). Instead, it would suffice to take all j's, but approximate them up to "weak" (cheap) precision (i.e., set $w_j \approx 100$ for all j's). At the other extreme is the case when $a_j = 10/m$ for all j. In this case, sampling would work but then one requires a much "stronger" (expensive) precision, of the order of $w_j \approx m$. These examples show that one cannot choose all w_j to be equal. If w_j's are too small, it is impossible to estimate σ. If w_j's are too big, the expectation of w cannot be bounded by $\operatorname{polylog} m$, and the sampling is too expensive.

The above lemma is somewhat inspired by the sketching/streaming technique introduced by Indyk and Woodruff [20], used for the F_k moment estimation. In [20], they partition elements a_j by weight level, and then perform corresponding sampling in each level. Although related, our approach to the above lemma differs: for example, we avoid any definition of the weight level (which was usually the source of some additional complexity of the use of the [20] technique).

In fact, in a follow-up work [5], we show that the Precision Sampling technique naturally gives a different, arguably simpler, generic algorithm for a number of streaming problems, including F_k moment estimation.

In our $\mathcal{E}$-distance estimation algorithm, we use both uniform and Precision Sampling lemmas at each node to both prune the tree and assign the precisions to the sampled children. We note that the lemmas may be used to obtain a multiplicative $(1+\varepsilon')$-approximation for arbitrary small $\varepsilon' > 0$ for each node. To obtain this, it is necessary to use $\varepsilon \approx \varepsilon'/\log n$, since over $h \approx \log n$ levels, we collect a multiplicative approximation factor of $(1+\varepsilon)^h$, which remains constant only as long as $\varepsilon = O(1/h)$.

We give precise description of the algorithms in Figures 1 and 2. We also give the formal statements of the two sampling lemmas that are applied, recursively, at each node of the tree.

The first lemma, on uniform sampling, is a restatement of the Chernoff/Hoeffding bounds in a suitable regime. We use the following approximation notion that captures both an additive and a multiplicative error. For convenience, we work with e^{ε} instead of the usual $1+\varepsilon$.

Definition II.1. *Fix $\rho > 0$ and some $f \in [1,2]$. For a quantity $\tau \ge 0$, we call its (ρ, f)-*approximator *any quantity $\hat{\tau}$ such that $\tau/f - \rho \le \hat{\tau} \le f\tau + \rho$.*

Lemma II.2 (Uniform Sampling)**.** *Fix $b \in \mathbb{N}$, $\varepsilon > 0$, and error probability $\delta > 0$. Consider some a_j, $j \in [b]$, such that $a_j \in [0, 1/b]$. For arbitrary $w \in [1, \infty)$, construct the set $J \subseteq [b]$ by sampling each $j \in [b]$ with probability $p_w = \min\{1, \frac{w}{b} \cdot \zeta \frac{\log 1/\delta}{\varepsilon^2}\}$ for some constant $\zeta > 0$. Then, with probability at least $1 - \delta$, the value $\frac{1}{p_w} \sum_{j \in J} a_j$ is a $(1/w, e^{\varepsilon})$-approximator to $\sum_{j \in [b]} a_j$,*

Algorithm 1: Sampling Algorithm

Take C_0 to be the root vertex (indexed $(i,s) = (0,1)$), with precision $w_{(0,1)} = 4\beta$.
for *each level* $i = 1, \ldots, h$, *we construct* C_i *as follows* **do**
  Start with C_i being empty.
  for *each node* $v = (i-1, s) \in C_{i-1}$ **do**
    Let w_v be its precision, and set $p_v = \frac{w_v}{b} \cdot O(\log^3 n)$.
    If $p_v \geq 1$, then set $J_v = \{(i, s + jl_i) \mid 0 \leq j < b\}$ to be the set of all the b children of v, and add them to C_i, each with precision p_v.
    Otherwise, when $p_v < 1$, sample each of the b children of v with probability p_v, to form a set $J_v \subseteq \{i\} \times ([s : s + l_{i-1}] \cap B_i)$. For each $v' \in J_v$, draw $w_{v'}$ i.i.d. from $\mathcal{W}$, and add node v' to C_i with precision $w_{v'}$.
Query the characters $x[s]$ for all $(h,s) \in C_h$ — this is the output of the algorithm.

Algorithm 2: Estimation Algorithm

For each sampled leaf $v = (h,s) \in C_h$ and $z \in [n]$ we set $\tau(v,z) = \mathrm{H}(x[s], y[z])$.
for *each level* $i = h-1, j-2, \ldots, 0$, *position* $z \in [n]$, *and node* $v \in C_i$ *with precision* w_v **do**
  We apply the following procedure $P(v,z)$ to obtain $\tau(v,z)$.
  For each $v' \in J_v$, where $v' = (i+1, s + jl_{i+1})$ for some $0 \leq j < b$, let
$$\delta_{v'} \stackrel{\text{def}}{=} \min_{k : |k| \leq n} \tau(v', z + jl_{i+1} + k) + |k|.$$
  If $p_v \geq 1$, then let $\tau(v,z) = \sum_{v' \in J_v} \delta_{v'}$.
  If $p_v < 1$, set $\tau(v,z)$ to be the output of the algorithm R on the vector $(\frac{\delta_{v'}}{l_{i+1}})_{v' \in J_v}$ with precisions $(w_{v'})_{v' \in J_v}$, multiplied by l_{i+1}/p_v.
The output of the algorithm is $\tau(r,1)$ where $r = (0,1)$ is the root of the tree.

and $|J| \leq O(w \cdot \frac{\log 1/\delta}{\varepsilon^2})$.

The second lemma is the *Precision Sampling Lemma*, which the heart of our non-uniform sampling.

Lemma II.3 (Precision Sampling)**.** *Fix integers* $n \leq N$, *approximation* $\varepsilon > 0$, *factor* $1 < f < 1.1$, *error probability* $\delta > 0$, *and an "additive error bound"* $\rho > 6n/\varepsilon/N^3$. *There exists a distribution* $\mathcal{W}$ *on the real interval* $[1, N^3]$ *with* $\mathbb{E}_{w \in \mathcal{W}}[w] \leq O(\frac{1}{\rho} \cdot \frac{\log 1/\delta}{\varepsilon^3} \cdot \log N)$, *as well as a "reconstruction algorithm" R, with the following property.*

Take arbitrary $a_i \in [0,1]$, *for* $i \in [n]$, *and let* $\sigma = \sum_{i \in [n]} a_i$. *Suppose one draws* w_i *i.i.d. from* $\mathcal{W}$, *for each* $i \in [n]$, *and let* $\hat{a}_i$ *be a* $(1/w_i, f)$*-approximator of* a_i. *Then, given* $\hat{a}_i$ *and* w_i *for all* $i \in [n]$, *the algorithm R generates a* $(\rho, f \cdot e^{\varepsilon})$*-approximator to* σ, *with probability at least* $1 - \delta$.

The distribution $\mathcal{W}$ *is generated as follows. For* $k = O(\frac{1}{\rho} \cdot \frac{\log 1/\delta}{\varepsilon^3})$, *let the sample be* $x = \max_{i=1\ldots k} x_i$, *where each* $x_i \in [1, N^3]$ *is distributed i.i.d. with pdf* $f(z) = \nu/z^2$, *for a normalization constant* ν. *The algorithm R, on input* $\hat{a}_1, \ldots \hat{a}_n$, *and* $w_1, \ldots w_n$, *outputs* $\frac{t}{\nu} \sum_{i=1}^{n} \chi[\hat{x} w_i \geq t] \cdot \left(\frac{1}{k} + \frac{k-1}{k} \cdot \frac{\hat{x} w_i / t - 1}{w_i - 1}\right)$, *where* $t = 3/\varepsilon$ *and* $\chi[E]$ *is the indicator variable of event* E.

We remark that an improved version of this lemma has subsequently appeared in [5]. Although the version from [5] achieves better parameters and uses only pairwise independence, it gives no further asymptotic improvements to the results in this paper.

III. Query Complexity Lower Bound

In this section we give an outline of the proof of Theorem I.3. The proof itself appears in the full version.

As usual, the lower bound is based on constructing "hard distributions", i.e., distributions (over inputs) that cannot be distinguished using few queries, but are very different in terms of edit distance. We sketch the construction of these distributions in Section III-A. In Section III-B, we sketch the machinery that we developed to prove that distinguishing these distributions requires many queries. We then sketch in Section III-C the tools needed to prove that the distributions are indeed very different in terms of edit distance.

A. The Hard Distributions

We shall construct two distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ over strings of a given length n. The distributions satisfy the following properties. First, every two strings in the support of the same distribution $\mathcal{D}_i$, denoted $\mathrm{supp}(\mathcal{D}_i)$, are close in edit distance. Second, every string in $\mathrm{supp}(\mathcal{D}_0)$ is far in edit distance from every string in $\mathrm{supp}(\mathcal{D}_1)$. Third, if an algorithm correctly distinguishes (with probability at least $2/3$) whether its input string is drawn from $\mathcal{D}_0$ or from $\mathcal{D}_1$, it must make many queries to the input.

Given two such distributions, we let x be any string from $\mathrm{supp}(\mathcal{D}_0)$. This string is fully known to the algorithm. The other string y, to which the algorithm only has query access, is drawn from either $\mathcal{D}_0$ or

$\mathcal{D}_1$. Since distinguishing the distributions apart requires many queries to the string, so does approximating edit distance between x and y.

Randomly Shifted Random Strings: The starting point for constructing these distributions is the following idea. Choose at random two base strings $z_0, z_1 \in \{0,1\}^n$. These strings are likely to satisfy some "typical properties", e.g. be far apart in edit distance (at least $n/10$). Now let each $\mathcal{D}_i$ be the distribution generated by selecting a cyclic shift of z_i by r positions to the right, where r is a uniformly random integer between 1 and $n/1000$. Every two strings in the same $\operatorname{supp}(\mathcal{D}_i)$ are at distance at most $n/500$, because a cyclic shift by r positions can be produced by r insertions and r deletions. At the same time, by the triangle inequality, every string in $\operatorname{supp}(D_0)$ and every string in $\operatorname{supp}(\mathcal{D}_1)$ must be at distance at least $n/10 - 2 \cdot n/500 \geq n/20$.

How many queries are necessary to learn whether an input string is drawn from $\mathcal{D}_0$ or from $\mathcal{D}_1$? If the number q of queries is small, then the algorithm's view is close to a uniform distribution on $\{0,1\}^q$ under both $\mathcal{D}_0$ and $\mathcal{D}_1$. Thus, the algorithm is unlikely to distinguish the two distributions with probability significantly higher than $1/2$. This is the case because each base string z_i is chosen at random and because we consider many cyclic shifts of it. Intuitively, even if the algorithm knows z_0 and z_1, the random shift makes the algorithm's view a nearly-random pattern, because of the random design of z_0 and z_1. Below we introduce rigorous tools for such an analysis. They prove, for instance, that even an adaptive algorithm for this case, and in particular every algorithm that distinguishes edit distance $\leq n/500$ and $\geq n/20$, must make $\Omega(\log n)$ queries.

It is natural to ask whether the $\Omega(\log n)$ lower bound for the number of queries in this construction can be improved. The answer is negative: for a sufficiently large constant C, querying any consecutive $C \log n$ symbols of z_1, one obtains a pattern that does not occur in z_0 w.h.p., and therefore, can be used to distinguish between the distributions.

To obtain a superlogarithmic lower bound for the DTEP problem, we show how we can amplify this basic hard distribution construction.

Substitution Product: We now introduce the *substitution product*, which plays an important role in amplifying our lower bound construction. Let $\mathcal{D}$ be a distribution on strings in Σ^m. For each $a \in \Sigma$, let $\mathcal{E}_a$ be a distribution on $(\Sigma')^{m'}$, and denote their entire collection by $\mathcal{E} \stackrel{\text{def}}{=} (\mathcal{E}_a)_{a \in \Sigma}$. Then the substitution product $\mathcal{D} \circledast \mathcal{E}$ is the distribution generated by drawing a string z from $\mathcal{D}$, and independently replacing every symbol z_i in z by a string B_i drawn from $\mathcal{E}_{z_i}$.

Strings generated by the substitution product consist of m blocks. Each block is independently drawn from one of the $\mathcal{E}_a$'s, and a string drawn from $\mathcal{D}$ decides which $\mathcal{E}_a$ each block is drawn from.

Recursive Construction: We build on the previous construction with two random strings shifted at random, and extend it by introducing recursion. For simplicity, we show how this idea works for two levels of recursion. We select two random strings z_0 and z_1 in $\{0,1\}^{\sqrt{n}}$. We use a sufficiently small positive constant c to construct two distributions $\mathcal{E}_0$ and $\mathcal{E}_1$. $\mathcal{E}_0$ and $\mathcal{E}_1$ are generated by taking a cyclic shift of z_0 and z_1, respectively, by r symbols to the right, where r is a random integer between 1 and $c\sqrt{n}$. Let $\mathcal{E} \stackrel{\text{def}}{=} (\mathcal{E}_i)_{i \in \{0,1\}}$.

Our two hard distributions on $\{0,1\}^n$ are $\mathcal{D}_0 \stackrel{\text{def}}{=} \mathcal{E}_0 \circledast \mathcal{E}$, and $\mathcal{D}_1 \stackrel{\text{def}}{=} \mathcal{E}_1 \circledast \mathcal{E}$. As before, one can show that distinguishing a string drawn from $\mathcal{E}_0$ and a string drawn from $\mathcal{E}_1$ is likely to require $\Omega(\log n)$ queries. In other words, the algorithm has to *know* $\Omega(\log n)$ symbols from a string selected from one of $\mathcal{E}_0$ and $\mathcal{E}_1$. Given the recursive structure of $\mathcal{D}_0$ and $\mathcal{D}_1$, the hope is that distinguishing them requires at least $\Omega(\log^2 n)$ queries, because, at least intuitively, the algorithm must "know" for at least $\Omega(\log n)$ blocks which $\mathcal{E}_i$ they come from, each of the blocks requiring $\Omega(\log n)$ queries. Indeed, below we describe techniques that we use to formally prove such a lower bound. It is straightforward to show that every two strings drawn from the same $\mathcal{D}_i$ are at most $4cn$ apart. It is slightly harder to prove that strings drawn from $\mathcal{D}_0$ and $\mathcal{D}_1$ are far apart. The important ramification is that for some constants c_1 and c_2, distinguishing edit distance $< c_1 n$ and $> c_2 n$ requires $\Omega(\log^2 n)$ queries, where one can make c_1 much smaller than c_2. For comparison, under the Ulam metric, $O(\log n)$ queries suffice for such a task (deciding whether distance between a known string and an input string is $< c_1 n$ or $> c_2 n$, when $2c_1 < c_2$ [1]).

To prove even stronger lower bounds, we apply the substitution product several times, not just once. Pushing our approach to the limit, we prove that distinguishing edit distance $O(n/\operatorname{polylog} n)$ from $\Omega(n)$ requires $n^{\Omega(1/\log\log n)}$ queries. In this case, $\Theta(\log n/\log\log n)$ levels of recursion are used. One slight technical complication arises in this case. Namely, we need to work with a larger alphabet (rather than binary). Our result holds true for the binary alphabet nonetheless, since we show that one can effectively reduce the larger alphabet to the binary alphabet at the end of the day.

B. Bounding the Number of Queries

To describe the technical tools, we introduce some further definitions. Let $\mathcal{D}_0, \ldots, \mathcal{D}_k$ be k distributions

on the same finite set Ω with $p_1, \ldots, p_k : \Omega \to [0,1]$ as the corresponding probability mass functions. We say that the distributions are α-*similar*, where $\alpha \geq 0$, if for every $\omega \in \Omega$,

$$(1-\alpha) \cdot \max_{i=1,\ldots,k} p_i(\omega) \leq \min_{i=1,\ldots,k} p_i(\omega).$$

For a distribution $\mathcal{D}$ on Σ^n and $Q \subseteq [n]$, we write $\mathcal{D}|_Q$ to denote the distribution created by projecting every element of Σ^n to its coordinates in Q. Let this time $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be probability distributions on Σ^n. We say that they are *uniformly* α-*similar* if for every subset Q of $[n]$, the distributions $\mathcal{D}_1|_Q, \ldots, \mathcal{D}_k|_Q$ are $\alpha|Q|$-similar. Intuitively, think of Q as a sequence of queries that the algorithm makes. If the distributions are uniformly α-similar for a very small α, and $|Q| \ll 1/\alpha$, then from the limited point of view of the algorithm (even an adaptive one), the difference between the distributions is very small.

In order to use the notion of uniform similarity for our construction, we prove the following three properties.

Uniform Similarity Implies a Lower Bound on the Number of Queries: We formalize the ramifications of uniform α-similarity for a pair of distributions. We show that if an algorithm (even an adaptive one) distinguishes the two distributions with probability at least $2/3$, then it has to make at least $1/(6\alpha)$ queries. This implies that it suffices to bound the uniform similarity in order to prove a lower bound on the number of queries.

The proof is based on the fact that for every setting of the algorithm's random bits, the algorithm can be described as a decision tree of depth q, if it always makes at most q queries. Then, for every leaf, the probability of reaching it does not differ by more than a factor in $[1-\alpha q, 1]$ between the two distributions. This is enough to bound the probability the algorithm outputs the correct answer for both the distributions.

Random Cyclic Shifts of Random Strings Imply Uniform Similarity: We construct distributions that are uniformly similar using cyclic shifts of random base strings. We show that if one takes n random base strings in Σ^n and creates n distributions by shifting each of the strings by a number of random indices in $[1, s]$, then with probability at least $2/3$ (over the choice of the base strings) the created distributions are uniformly $O(1/\log_{|\Sigma|} \frac{s}{\log n})$-similar.

It is easy to prove this property for any set Q of size 1. In this case, every shift gives an independent random bit, and the bound directly follows from the Chernoff bound. A slight obstacle is posed by the fact that for $|Q| \geq 2$, sequences of $|Q|$ symbols produced by different shifts are not necessarily independent, since they can share some of the symbols. To address this issue, we show that there is a partition of shifts into at most $|Q|^2$ large sets such that no two shifts of Q in the same set overlap. Then we can apply the Chernoff bound independently to each of the sets to prove the bound.

In particular, using this and the previous property, one can show the basic statement claimed earlier: namely, that shifts of two random strings in $\{0,1\}^n$ by an offset in $[1, cn]$ produce distributions that require $\Omega(\log n)$ queries to be distinguished. It follows from this property that the distributions are likely to be uniformly $O(1/\log n)$-similar.

Substitution Product Amplifies Uniform Similarity: Perhaps the most surprising property of uniform similarity is that it nicely composes with the substitution product. Let $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be uniformly α-similar distributions on Σ^n. Let $\mathcal{E} = (\mathcal{E}_a)_{a \in \Sigma}$, where $\mathcal{E}_a$, $a \in \Sigma$, are uniformly β-similar distributions on $(\Sigma')^{n'}$. We show that $\mathcal{D}_1 \circledast \mathcal{E}, \ldots, \mathcal{D}_k \circledast \mathcal{E}$ are uniformly $\alpha\beta$-similar.

The intuition behind the proof of this property is the following. Querying q locations in a string that comes from $\mathcal{D}_i \circledast \mathcal{E}$, we can detect a difference between distributions in at most βq blocks in expectation. Seeing the difference is necessary to discover which $\mathcal{E}_j$ each of the blocks comes from. Then only these blocks can reveal the identity of $\mathcal{D}_i \circledast \mathcal{E}$, and the difference in the distribution, conditioned on q' blocks being revealed, is bounded by $\alpha q'$.

This property can be used to prove the earlier claim that the two-level construction produces distributions that require $\Omega(\log^2 n)$ queries to be distinguished.

C. Preserving Edit Distance

We also need tools for analyzing the edit distance between strings generated by our distributions. Our analysis mostly uses the distance $\underline{\text{ed}}$, which is a variation of the edit distance that allows only for insertions and deletions. It is clearly a 2-approximation to the edit distance, which is enough for the analysis. The main advantage of $\underline{\text{ed}}$ is that it has a direct connection to the longest increasing subsequence (LCS), which is easier to analyze.

We start by giving a tight bound for the length of LCS of randomly chosen strings, over big alphabet.

The main challenge here is to show that strings constructed via the substitution product behave as expected. Indeed, we show how to control the edit distance between two strings in Σ^n when we substitute every symbol with a longer string using a function $B : \Sigma \to (\Sigma')^{n'}$. The relative edit distance (that is, edit distance divided by the length of the strings) shrinks

by an additive term that polynomially depends on the maximum relative length of the longest common string between $B(a)$ and $B(b)$ for different a and b.

We note that the relative distance shrinks relatively fast as a result of substitutions. This implies that we have to use an alphabet of size polynomial in the number of recursion levels. The alphabet never has to be larger than polylogarithmic, because the number of recursion levels is always $o(\log n)$. (This issue was the only reason to use large alphabets.)

Finally, we show how to reduce the alphabet size to a binary one. We show that a lower bound for the binary alphabet follows immediately from the one for a large alphabet, with only a constant factor loss in the edit distance. To achieve this it suffices to map every element of the large alphabet Σ to a random binary string of length $\Theta(\log |\Sigma|)$.

The main idea behind proofs of the above is that strings constructed using a substitution product are composed of rather rigid blocks, in the sense that every alignment between two such strings, say $x \circledast \mathcal{E}$ and $y \circledast \mathcal{E}$, must respect (to a large extent) the block structure, in which case one can extract from it an alignment between the two initial strings x and y.

REFERENCES

[1] N. Ailon, B. Chazelle, S. Comandur, and D. Liu, "Estimating the distance to a monotone function," *Random Structures and Algorithms*, vol. 31, pp. 371–383, 2007, previously appeared in RANDOM'04.

[2] A. Andoni, T. Jayram, and M. Pătraşcu, "Lower bounds for edit distance and product metrics via Poincaré-type inequalities," in *Proc. of SODA*, 2010.

[3] A. Andoni and R. Krauthgamer, "The smoothed complexity of edit distance," in *Proc. of ICALP*, 2008, pp. 357–369.

[4] ——, "The computational hardness of estimating edit distance," *SIAM Journal on Computing*, vol. 39, no. 6, pp. 2398–2429, 2010, previously appeared in FOCS'07.

[5] A. Andoni, R. Krauthgamer, and K. Onak, "Streaming applications from precision sampling," 2010, manuscript.

[6] A. Andoni and H. L. Nguyen, "Near-tight bounds for testing Ulam distance," in *Proc. of SODA*, 2010.

[7] A. Andoni and K. Onak, "Approximating edit distance in near-linear time," in *Proc. of STOC*, 2009, pp. 199–204.

[8] Z. Bar-Yossef, T. S. Jayram, R. Krauthgamer, and R. Kumar, "Approximating edit distance efficiently," in *Proc. of FOCS*, 2004, pp. 550–559.

[9] T. Batu, F. Ergün, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami, "A sublinear algorithm for weakly approximating edit distance," in *Proc. of STOC*, 2003, pp. 316–324.

[10] T. Batu, F. Ergün, and C. Sahinalp, "Oblivious string embeddings and edit distance approximations," in *Proc. of SODA*, 2006, pp. 792–801.

[11] P. Bille and M. Farach-Colton, "Fast and compact regular expression matching," *Theoretical Computer Science*, vol. 409, no. 28, pp. 486–496, 2008.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[13] G. Cormode, *Sequence Distance Embeddings*, ser. Ph.D. Thesis, University of Warwick, 2003.

[14] G. Cormode and S. Muthukrishnan, "The string edit distance matching problem with moves," *ACM Trans. Algorithms*, vol. 3, no. 1, 2007, special issue on SODA'02.

[15] G. Cormode, M. Paterson, S. C. Sahinalp, and U. Vishkin, "Communication complexity of document exchange," in *Proc. of SODA*, 2000, pp. 197–206.

[16] F. Ergün, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan, "Spot-checkers," *J. Comput. Syst. Sci.*, vol. 60(3), pp. 717–751, 2000.

[17] D. Gusfield, *Algorithms on strings, trees, and sequences*. Cambridge: Cambridge University Press, 1997.

[18] P. Indyk, "Algorithmic aspects of geometric embeddings (tutorial)," in *Proc. of FOCS*, 2001, pp. 10–33.

[19] P. Indyk and J. Matoušek, "Low distortion embeddings of finite metric spaces," *CRC Handbook of Discrete and Computational Geometry*, 2003.

[20] P. Indyk and D. Woodruff, "Optimal approximations of the frequency moments of data streams," *Proc. of STOC*, 2005.

[21] S. Khot and A. Naor, "Nonembeddability theorems via Fourier analysis," *Math. Ann.*, vol. 334, no. 4, pp. 821–852, 2006, preliminary version appeared in FOCS'05.

[22] R. Krauthgamer and Y. Rabani, "Improved lower bounds for embeddings into L_1," in *Proc. of SODA*, 2006, pp. 1010–1017.

[23] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, "Efficient search for approximate nearest neighbor in high dimensional spaces," *SIAM J. Comput.*, vol. 30, no. 2, pp. 457–474, 2000, preliminary version appeared in STOC'98.

[24] G. M. Landau, E. W. Myers, and J. P. Schmidt, "Incremental string comparison," *SIAM J. Comput.*, vol. 27, no. 2, pp. 557–582, 1998.

[25] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals (in russian)," *Doklady Akademii Nauk SSSR*, vol. 4, no. 163, pp. 845–848, 1965, appeared in English as: V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10(8), 707–710, 1966.

[26] W. J. Masek and M. Paterson, "A faster algorithm computing string edit distances," *J. Comput. Syst. Sci.*, vol. 20, no. 1, pp. 18–31, 1980.

[27] J. Matoušek, "Collection of open problems on low-distortion embeddings of finite metric spaces," March 2007, available online. Last access in August, 2007.

[28] S. Muthukrishnan and C. Sahinalp, "Approximate nearest neighbors and sequence comparison with block operations," *Proc. of STOC*, pp. 416–424, 2000.

[29] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surv.*, vol. 33, no. 1, pp. 31–88, 2001.

[30] R. Ostrovsky and Y. Rabani, "Low distortion embedding for edit distance," *J. ACM*, vol. 54, no. 5, 2007, preliminary version appeared in STOC'05.

[31] S. C. Sahinalp, "Edit distance under block operations," in *Encyclopedia of Algorithms*, M.-Y. Kao, Ed. Springer, 2008.

[32] M. Saks and X. Sun, "Space lower bounds for distance approximation in the data stream model," in *Proc. of STOC*, 2002.

[33] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *J. ACM*, vol. 21, no. 1, pp. 168 – 173, 1974.