Polylogarithmic Approximation for Edit Distance and the Asymmetric Query Complexity

Alexandr Andoni*
Princeton University/C.C.I.

Robert Krauthgamer†
Weizmann Institute

Krzysztof Onak‡
MIT

May 21, 2010

Abstract

We present a near-linear time algorithm that approximates the edit distance between two strings within a polylogarithmic factor; specifically, for strings of length n and every fixed $\varepsilon > 0$, it can compute a $(\log n)^{O(1/\varepsilon)}$ approximation in $n^{1+\varepsilon}$ time. This is an *exponential* improvement over the previously known factor, $2^{\tilde{O}(\sqrt{\log n})}$, with a comparable running time [OR07, AO09]. Previously, no efficient polylogarithmic approximation algorithm was known for any computational task involving edit distance (e.g., nearest neighbor search or sketching).

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, which is edit distance on non-repetitive strings, such as permutations.

*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 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) [Lev65], 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 [Nav01, Gus97] 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 [WF74]; in fact this is a prototypical dynamic programming algorithm, see, e.g., the textbook [CLRS01] and references therein. Despite significant research over more than three decades, this running time has so far been improved only slightly to $O(n^2/\log^2 n)$ [MP80], which remains the fastest algorithm known to date.[1]

Still, 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 [Gus97, Nav01]. This has motivated the quest for faster algorithms at the expense of approximation, see, e.g., [Ind01, Section 6] and [IM03, 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 the edit distance. A linear-time $\sqrt{n}$-approximation algorithm immediately follows from the exact algorithm of [LMS98], 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 $n^{3/7}$ [BJKK04], then to $n^{1/3+o(1)}$ [BES06], and finally to $2^{\tilde{O}(\sqrt{\log n})}$ [AO09] (building on [OR07]). Predating some of this work was the *sublinear-time* algorithm of [BEK+03] 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 [CPSV00, MS00, CM07, Cor03].

Despite the efforts, achieving a polylogarithmic approximation factor for the classical edit distance has eluded researchers for a long time. In fact, this is 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 [KN06, KR06, AK10], thus giving evidence that a sublog-

[1] The result of [MP80] applies to constant-size alphabets. It was recently extended to arbitrarily large alphabets, albeit with an $O(\log \log n)^2$ factor loss in runtime [BFC08].

[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 [Nav01] and [Sah08].

arithmic approximation for the distance computation might be much harder or even impossible to attain.

1.1 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 [OR07, AO09].

Theorem 1.1 (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 the Ulam distance. These models include: non-embeddability into normed spaces [KN06, KR06, AK10], lower bounds on sketching complexity [AK10, AJP10], and (symmetric) query complexity [BEK$^+$03, AN10].

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 these two strings. The computational problem is the promise problem known as the Distance Threshold Estimation Problem (DTEP) [SS02]: 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 give complete statements of our upper and lower bound results. Both exhibit a smooth *tradeoff* between approximation factor and query complexity. For simplicity, we state the bounds in two extreme regimes of approximation ($\alpha = \mathrm{polylog}(n)$ and $\alpha = \mathrm{poly}(n)$). See Theorem 3.1 for the full statement of the upper bound, and Theorems 4.15 and 4.16 for the full statement of the lower bound.

Theorem 1.2 (Query complexity upper bound)**.** *For every $\beta = \beta(n) \geq 2$ and fixed $0 < \varepsilon < 1$ there is an algorithm that solves* DTEP_β *with approximation $\alpha = (\log n)^{O(1/\varepsilon)}$, and makes βn^ε asymmetric queries. This algorithm runs in time $O(n^{1+\varepsilon})$.*

For every $\beta = O(1)$ and fixed 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 1.1 follows immediately from the above query complexity upper bound theorem, by running the latter for all β that are a power of 2.

Theorem 1.3 (Query complexity lower bound)**.** *For a sufficiently large constant $\beta > 1$, every algorithm that solves* DTEP_β *with approximation $\alpha = \alpha(n) > 2$ 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 1 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 distances. In the Ulam metric, a *constant* approximation is achievable with only $O(\log n)$ asymmetric queries (see [ACCL07], which builds on [EKK+00]). 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. 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.

Model	Metric	Approx.	Complexity	Remarks
Near-linear time	Edit	$(\log n)^{O(1/\varepsilon)}$	$n^{1+\varepsilon}$	Theorem 1.1
	Edit	$2^{\tilde{O}(\sqrt{\log n})}$	$n^{1+o(1)}$	[AO09]
Symmetric query complexity	Edit	n^ε	$\tilde{O}(n^{\max\{1-2\varepsilon,(1-\varepsilon)/2\}})$	[BEK+03] (fixed $\beta > 1$)
	Ulam	$O(1)$	$\tilde{O}(\beta + \sqrt{n})$	[AN10]
	Ulam+edit	$O(1)$	$\tilde{\Omega}(\beta + \sqrt{n})$	[AN10]
Asymmetric query complexity	Edit	$n^{1/t}$	$O(\log^{t-1} n)$	Theorem 1.2 (fixed $t \in \mathbb{N}, \beta > 1$)
	Edit	$n^{1/t}$	$\Omega(\log^{\lfloor t \rfloor} n)$	Theorem 1.3 (fixed $t \notin \mathbb{N}, \beta > 1$)
	Edit	$(\log n)^{1/\varepsilon}$	$\beta n^{O(\varepsilon)}$	Theorem 1.2
	Edit	$(\log n)^{1/\varepsilon}$	$n^{\Omega(\varepsilon/\log\log n)}$	Theorem 1.3 (fixed $\beta > 1$)
	Ulam	$2 + \varepsilon$	$O_\varepsilon(\beta \log\log\beta \cdot \log n)$	[ACCL07]

Table 1: Known results for DTEP_β and arbitrarily $0 < \varepsilon < 1$.

1.2 Connections of Asymmetric Query Model to Other Models

The asymmetric query model is connected and has implications for two previously 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). Our upper bound gives an $O(\beta n^{\varepsilon})$ one-way communication complexity protocol for DTEP_{β} for polylogarithmic approximation.

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 non-trivial upper bound known is $2^{\tilde{O}(\sqrt{\log n})}$ approximation with constant communication via [OR07, KOR00]. The only known lower bound says that approximation α requires $\Omega(\frac{\log n \ / \ \log\log n}{\alpha})$ communication [AK10, AJP10].

The asymmetric model is "harder", in the sense that the query complexity is at least the communication complexity, up to a factor of $\log|\Sigma|$ in the complexity, since Alice and Bob can simulate the asymmetric query algorithm. In fact, our upper bound implies a communication protocol for the same DTEP_{β} problem with the same complexity, and it is a one-way communication protocol. Specifically, Alice can just send the $O(\beta n^{\varepsilon})$ characters queried by the query algorithm in the asymmetric query model. This is the first communication protocol achieving polylogarithmic approximation for DTEP_{β} under edit distance with $o(n)$ 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 the edit distance) in [BEK$^+$03], and its main advantage is that it leads to *sublinear-time* algorithms for DTEP_{β}. The algorithm of [BEK$^+$03] 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 $\beta = O(1)$.

In the symmetric query model, the best query lower bound is of $\Omega(\sqrt{n/\alpha})$ for any approximation factor $\alpha > 1$ for both edit and Ulam distance [BEK$^+$03, AN10]. The lower bound essentially arises from the birthday paradox. Hence, in terms of separating edit distance from the Ulam metric, this symmetric model can give at most a quadratic separation in the query complexity (since there exists a trivial algorithm with $2n$ queries). In contrast, in our asymmetric model, there is no lower bound based on the birthday paradox, and, in fact, the Ulam metric admits a constant approximation with $O(\log n)$ queries [EKK$^+$00, ACCL07]. Our lower bound for edit distance is exponentially bigger.

1.3 Techniques

This section briefly highlights the main techniques and tools used in the course of proving our results. A more informative proof overview for the algorithmic results, including the near-linear time algorithm and the query upper bounds, appears in Section 2.1. The proof overview for the query lower bounds appears in Section 2.2. The complete proofs are on Sections 3 and 4, respectively.

Algorithm and Query Complexity Upper Bound. 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 subsamples $\approx n^\varepsilon$ positions of x, and then computes $\mathrm{ed}(x, y)$ in time $n^{1+\varepsilon}$. Of course, the main challenge is, by far, subsampling x so that the above is possible.

Our asymmetric 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.

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 show that our characterization is an $O(\frac{b}{\log b} \log n)$ 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 [BEK+03, OR07, AO09], 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 is much more refined and has *no multiplicative factor loss*, i.e., $c = 1$ and hence $\alpha(n) = O(b \log_b n)$. We note that our characterization achieves a *logarithmic* approximation for $b = O(1)$ (although, we do not know efficient algorithms for this setting of b).

The second component of our query algorithm is a careful sampling procedure that approximates $\mathcal{E}$-distance up to a constant factor. The basic idea is to prune the above tree by subsampling at each node a subset of its children. In particular, for a tree with arity $b = (\log n)^{1/\varepsilon}$, the hope is to subsample $(\log n)^{O(1)}$ children and use Chernoff-type bounds to argue that the subsample approximates well the $\mathcal{E}$-distance at that node. We note that $\Omega(\log n)$ samples of children seem necessary due to the minimum operation taken at each node. The estimate at each node has to hold with high probability so that we can apply the union bound. After such a pruning of the tree, we would be left with only $(\log n)^{O(\log_b n)} = n^{O(\varepsilon)}$ leaves, i.e., $n^{O(\varepsilon)}$ positions of x to query.

However, this natural approach of subsampling $(\log n)^{O(1)}$ children at each node does not work when $\beta \gg 1$. Instead, we develop a *non-uniform subsampling technique*: for different nodes we subsample children at different, carefully-chosen rates. From a high-level, our deployed technique is somewhat reminiscent of the hierarchical decomposition and subsampling technique introduced by Indyk and Woodruff [IW05] in the context of sketching and streaming algorithms.

Query Complexity Lower Bound. The gist of our lower bound is designing two "hard distributions" $\mathcal{D}_0$ and $\mathcal{D}_1$, on strings in Σ^n, for which it is hard to distinguish with only a few queries to x whether $x \in \mathcal{D}_0$ or $x \in D_1$. At the same time, every two strings x, y in the support of the same $\mathcal{D}_i$ are at a small edit distance: $\mathrm{ed}(x, y) \leq n/(\alpha\beta)$; but for a mixed pair $x \in \mathcal{D}_0$ and $y \in D_1$, the distance is large: $\mathrm{ed}(x, y) > n/\beta$.

We start by making the following core observation. Take two random strings $z_0, z_1 \in \{0,1\}^n$. Each $\mathcal{D}_i$, $i \in \{0,1\}$, is generated by applying a cyclic shift by a random displacement $r \in [1, n/100]$ to the corresponding z_i. We show that in order to discover, for an input string, from which $\mathcal{D}_i$ it came from, one has to make at least $\Omega(\log n)$ queries. Intuitively, this follows from the fact that if the number q of queries is small ($q = o(\log n)$) then the algorithm's view is close to the uniform distribution on $\{0,1\}^q$, no matter which positions are queried. Nevertheless, the edit distance between the two random strings is likely to be large, and a small shift will not change this significantly.

We then amplify the above query lower bound by applying the same idea recursively. In a string generated according to $\mathcal{D}_i$'s, we replace every symbol $a \in \{0,1\}$ by a random string selected independently from $\mathcal{D}_a$. This way we obtain two distributions on strings of length $n' = n^2$, that require $\Omega(\log^2 n) = \Omega(\log^2 n')$ queries to be told apart. We call the above operation of replacing symbols by strings that come from other distributions a *substitution product*. Strings created this way consist of n blocks of length n each. Intuitively, to distinguish from which of the new distributions an input string comes from, one has to discover for at least $\Omega(\log n)$ blocks which distribution $\mathcal{D}_a$ the respective block comes from. By applying the recursive step multiple times, we obtain a $2^{\Omega(\frac{\log n}{\log\log n})}$ lower bound for a polylogarithmic approximation factor.

To formally prove our result, we develop several tools. First, we need tools for analyzing the behavior of edit distance under the product substitution. It turns out that to control edit distance under the substitution product, we need to work with a large alphabet Σ. In the final step of the construction, we map the large alphabet to sufficiently long random binary strings, thereby extending the lower bound to the binary alphabet as well.

Second, we need tools for analyzing indistinguishability of our distributions under a small number of queries. For this, we introduce a notion of *similarity* of distributions. This notion smoothly composes with the substitution product operation, which amplifies the similarity. We also show that random acyclic shifts of random strings are likely to produce strings with high similarity. Finally, we show that if an algorithm is able to distinguish distributions meeting our similarity notion, then it must make many queries. We believe that these tools and ideas behind them may find applications in showing query lower bounds for other problems.

1.4 Future Directions

We study a new query model that seems to tap into 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 pave the way to significantly improved algorithms for other tasks involving edit distance, such as the nearest neighbor search. We mention below a few natural goals for future investigation.

Symmetric Model. Extend our results to the symmetric query model. A lower bound would show a separation between edit and Ulam distances in this model as well. It seems plausible that a variation of our hard distribution leads to a lower bound of the form $n^{1/2+\Omega(1/\log\log n)}$ for

polylogarithmic approximation. The current lower bound is of the form $\Omega(\sqrt{n/\alpha})$. A query upper bound would likely lead to improved sub-linear time algorithms.

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 [Mat07]. 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 $\Omega(\log n)$ [KR06].

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 Algorithms. Tighten the asymmetric query complexity upper bound to $n^{\frac{\varepsilon \log\log\log n}{\log\log n}}$ for approximation $(\log n)^{O(1/\varepsilon)}$, perhaps by a more careful subsampling procedure. In particular, it seems plausible that one may only sample $(\log\log n)^{O(1)}$ children at each node, instead of the present $(\log n)^{O(1)}$. This may ultimately lead to an algorithm that runs in time $n^{1+o(1)}$ and approximates edit distance within a factor of, say, $O(\log^2 n)$.

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

2 Outline of Our Results

We now sketch the proofs of our results.

2.1 Outline of the Upper Bound

In this section, we provide an overview of our algorithmic results, in particular of the proof of Theorem 1.2. Full statements and proofs of the results appear in Section 3.

Our proof has two major components. The first one is a characterization of edit distance by a different "distance", denoted $\mathcal{E}$, which approximates edit distance well. The second component is a sampling algorithm that approximates $\mathcal{E}$ up to a constant factor by making a small number of queries into x. We describe each of the components below. In the following, for a string x and integers $s, t \geq 1$, $x[s:t]$ denotes the substring of x comprising of $x[s], \ldots, x[t-1]$.

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

Our characterization of $\mathrm{ed}(x, y)$ may be viewed as computation on a tree, where the nodes correspond to substrings $x[s : s+l]$, for some start position $s \in [n]$ and length $l \in [n]$. The root is the entire string $x[1 : n+1]$. For a node $x[s : s+l]$, we obtain its children by partitioning $x[s : s+l]$ into b equal-length blocks, $x[s + j \cdot l/b : s + (j+1) \cdot l/b]$, where $j \in \{0, 1, \ldots b-1\}$. Hence $b \geq 2$ is the arity of the tree. The height of the tree is $h \stackrel{\text{def}}{=} \log_b n$. We also use the following notation: for level $i \in \{0, 1, \ldots h\}$, let $l_i \stackrel{\text{def}}{=} n/b^i$ be the length of strings at that level. Let $B_i \stackrel{\text{def}}{=} \{1, l_i + 1, 2l_i + 1, \ldots\}$ 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 like 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., $u + t_j$. Additionally, we 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 Figure 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 the shifts r_j are what differentiates the Hamming distance and $\mathcal{E}$-distance.

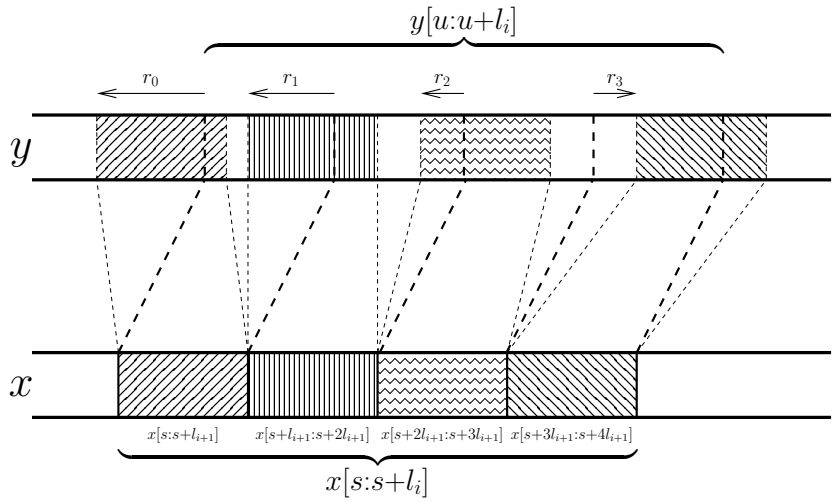

Figure 1: Illustration of the $\mathcal{E}$-distance $\mathcal{E}(i, s, u)$ for $b = 4$. The pairs of blocks of the same shading are the blocks whose $\mathcal{E}$-distance is used for computing $\mathcal{E}(i, s, u)$.

We prove that the $\mathcal{E}$-distance is a $O(bh) = O(\frac{b}{\log b} \log n)$ approximation to $\text{ed}(x, y)$ (see Theorem 3.3). For $b = 2$, the $\mathcal{E}$-distance is a $O(\log n)$ approximation to $\text{ed}(x, y)$, but unfortunately, we do not know how to compute it or approximate it well in better than quadratic time. It is also easy to observe that one can compute a $1 + \varepsilon$ approximation to $\mathcal{E}$-distance in $\tilde{O}_\varepsilon(n^2)$ time via a dynamic programming that considers only r_j's which are powers of $1 + \varepsilon$. Instead, we show that, using the query 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.

2.1.2 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 $\beta = n^{0.1}$ and $b = n^{0.01}$, although our main result actually follows by setting $b = (\log n)^{O(1/\varepsilon)}$.

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

A natural idea is to uniformly subsample $\operatorname{polylog} n$ out of b children at each node, and use Chernoff-type concentration bounds to argue that Equation (1) may be approximated only from the $\mathcal{E}$-distance estimates of the subsampled 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 subsample 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 subsample 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 subsample 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 3.11 for 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 $\rho = l_i = n/\beta$ and $\sigma = n/\beta$). However, we hope to prune the tree at the subsequent levels.

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 $j = 0, \ldots b-1$. 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 $a_j \leq l_{i+1} = (n/\beta)/b$. 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 subsample 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*. Specifically, for different nodes we will subsample 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 subsampling in the subtree rooted at v.

Technically, our main tool is a *Non-uniform Sampling Lemma*, which we use to assign the necessary precisions to nodes. It may be stated as follows (see Lemma 3.12 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., $|a_j - \hat{a}_j| \leq 1/w_j$. 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 subsampling 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, subsampling 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 subsampling is too expensive.

The above lemma is somewhat inspired by the sketching and streaming technique introduced by Indyk and Woodruff [IW05] (and used for the F_k moment estimation), where one partitions elements a_j by weight level, and then performs corresponding subsampling 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 technique). For completeness, we mention that the distribution $\mathcal{W}$ is essentially the distribution with probability distribution function $f(x) = \nu/x^2$ for $x \in [1, m^3]$ and a normalization constant ν. The algorithm R essentially uses the samples that were (in retrospect) well-approximated, i.e., $\hat{a}_j \gg 1/w_j$, in order to approximate σ.

In our $\mathcal{E}$-distance estimation algorithm, we use both uniform and non-uniform subsampling lemmas at each node to both prune the tree and assign the precisions to the subsampled 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 $\varepsilon = O(1/h)$.

2.2 Outline of the Lower Bound

In this section we outline the proof of Theorem 1.3. The full proof appears in Section 4. Here, we focus on the main ideas, skipping or simplifying some of the technical issues.

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 2.2.1. The full construction appears in Section 4.4.1. In Section 2.2.2, we sketch the machinery that we developed to prove that distinguishing these distributions requires many queries; the details appear in Section 4.2. We then sketch in Section 2.2.3 the tools needed to prove that the distributions are indeed very different in terms of edit distance; the detailed version appears in Section 4.3.

2.2.1 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 $\operatorname{supp}(\mathcal{D}_i)$, are close in edit distance. Second, every string in $\operatorname{supp}(\mathcal{D}_0)$ is far in edit distance from every string in $\operatorname{supp}(\mathcal{D}_1)$. 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 $\mathrm{supp}(\mathcal{D}_i)$ 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 $\mathrm{supp}(D_0)$ and every string in $\mathrm{supp}(\mathcal{D}_1)$ 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.

One could ask whether the $\Omega(\log n)$ lower bound for the number of queries in this construction can be improved. The answer is negative, because for a sufficiently large constant C, by querying any consecutive $C \log n$ symbols of z_1, one obtains a pattern that most likely does not occur in z_0, and therefore, can be used to distinguish between the distributions. This means that we need a different construction to show a superlogarithmic lower bound.

Substitution Product. We now introduce the *substitution product*, which plays an important role in 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. 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 $> c_2 n$, assuming $2c_1 < c_2$ [ACCL07]).

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.

2.2.2 Bounding the Number of Queries

We start with definitions. Let $\mathcal{D}_0, \ldots, \mathcal{D}_k$ be 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 key lemmas.

Uniform Similarity Implies a Lower Bound on the Number of Queries (Lemma 4.4). This lemma formalizes the ramifications of uniform α-similarity for a pair of distributions. It shows 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. The lemma 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 (Lemma 4.7). This lemma constructs block-distributions that are uniformly similar using cyclic shifts of random base

strings. It shows that if one takes n random base strings in Σ^n and creates n distributions by shifting each of the strings by a random number of 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 lemma 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 lemmas, one can show the result claimed earlier 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 the lemma that the distributions are likely to be uniformly $O(1/\log n)$-similar.

Substitution Product Amplifies Uniform Similarity (Lemma 4.8). 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 $(\Sigma')^{n'}$. The lemma states that $\mathcal{D}_1 \circledast \mathcal{E}, \ldots, \mathcal{D}_k \circledast \mathcal{E}$ are uniformly $\alpha\beta$-similar.

The main idea behind the proof of the lemma is the following. Querying q locations in a string that comes from $\mathcal{D}_i \circledast \mathcal{E}$, we can see 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 if q' blocks are revealed is bounded by $\alpha q'$.

The lemma can be used to prove the earlier claim that the two-level construction produces distributions that require $\Omega(\log^2 n)$ queries to be told apart.

2.2.3 Preserving Edit Distance

It now remains to describe our tools for analyzing the edit distance between strings generated by our distributions. All of these tools are collected in Section 4.3. In most cases we focus in our analysis on $\underline{\text{ed}}$, which is the version of edit distance that only allows for insertions and deletions. It clearly holds that $\text{ed}(x, y) \leq \underline{\text{ed}}(x, y) \leq 2 \cdot \text{ed}(x, y)$, and this connection is tight enough for our purposes. An additional advantage of $\underline{\text{ed}}$ is that for any strings x and y, $2\,\text{LCS}(x, y) + \underline{\text{ed}}(x, y) = |x| + |y|$.

We start by reproducing a well known bound on the longest common substring of randomly selected strings (Lemma 4.9). It gives a lower bound on $\text{LCS}(x, y)$ for two randomly chosen strings. The lower bound then implies that the distance between two strings chosen at random is large, especially for a large alphabet.

Theorem 4.10 shows how the edit distance between two strings in Σ^n changes 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. It is worth to highlight the following two issues:

- We do not need a special version of this theorem for distributions. It suffices to first bound edit distance for the recursive construction when instead of strings shifted at random, we use

strings themselves. Then it suffices to bound by how much the strings can change as a result of shifts (at all levels of the recursion) to obtain desired bounds.

- 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 $o(\log n)$.

Finally, Theorem 4.12 and Lemma 4.14 effectively reduce the alphabet size, because they 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. It turns out that it suffices to map every element of the large alphabet Σ to a random string of length $\Theta(\log|\Sigma|)$ over the binary alphabet.

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.

3 Fast Algorithms via Asymmetric Query Complexity

In this section we describe our near-linear time algorithm for estimating the edit distance between two strings. As we mentioned in the introduction, the algorithm is obtained from an efficient query algorithm.

The main result of this section is the following query complexity upper bound theorem, which is a full version of Theorem 1.2. It implies our near-linear time algorithm for polylogarithmic approximation (Theorem 1.1).

Theorem 3.1. *Let $n \geq 2$, $\beta = \beta(n) \geq 2$, and integer $b = b(n) \geq 2$ be such that $(\log_b n) \in \mathbb{N}$.*

There is an algorithm solving DTEP_β with approximation $\alpha = O(b \log_b n)$ and $\beta \cdot (\log n)^{O(\log_b n)}$ queries into x. The algorithm runs in $n \cdot (\log n)^{O(\log_b n)}$ time.

For every constant $\beta = O(1)$ and integer $t \geq 2$, there is an algorithm for solving DTEP_β with $O(n^{1/t})$ approximation and $O(\log n)^{t-1}$ queries. The algorithm runs in $\tilde{O}(n)$ time.

In particular, note that we obtain Theorem 1.1 by setting $b = (\log n)^{c/\varepsilon}$ for a suitably high constant $c > 1$.

The proof is partitioned in three stages. (The first stage corresponds to the first "major component" mentioned in Introduction, and Section 2.1, and the next two stages correspond to the second "major component".) In the first stage, we describe a characterization of edit distance by a different quantity, namely $\mathcal{E}$-distance, which approximates edit distance well. The characterization is parametrized by an integer parameter $b \geq 2$. A small b leads to a small approximation factor (in fact, as small as $O(\log n)$ for $b = 2$), whereas a large b leads to a faster algorithm. In the second stage, we show how one can design a sampling algorithm that approximates $\mathcal{E}$-distance for some setting of the parameter b, up to a constant factor, by making a small number of queries into x. In the third stage, we show how to use the query algorithm to obtain a near-linear time algorithm for edit distance approximation.

The three stages are described in the following three sections, and all together give the proof of Theorem 3.1.

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

Our characterization may be viewed as computation on a tree, where the nodes correspond to substrings $x[s : s + l]$, for some start position $s \in [n]$ and length[3] $l \in [n]$. The root is the entire string $x[1 : n + 1]$. For a node $x[s : s + l]$, the children are blocks $x[s + j \cdot l/b : s + (j + 1) \cdot l/b]$, where $j \in \{0, 1, \ldots b - 1\}$, and b is the arity of the tree. The $\mathcal{E}$-distance for the node $x[s : s + l]$ is defined recursively as a function of the distances of its children. Note that the characterization is asymmetric in the two strings.

Before giving the definition we establish further 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. Fix some tree level i for $0 \leq i \leq h$. Consider some substring $x[s : s + l_i]$ at level i, where $l_i \stackrel{\text{def}}{=} n/b^i$. Let $B_i \stackrel{\text{def}}{=} \{1, l_i + 1, 2l_i + 1, \ldots\}$ be the set of starting positions of blocks at level i.

Definition 3.2 ($\mathcal{E}$-distance)**.** *Consider two strings x, y of length $n \geq 2$. Fix $i \in \{0, 1, \ldots h\}$, $s \in B_i$, and a position $u \in \mathbb{Z}$.*

If $i = h$, then the $\mathcal{E}$-distance *of $x[s : s + l_i]$ to the position u is 1 if $u \notin [n]$ or $x[s] \neq y[u]$, and 0 otherwise.*

For $i \in \{0, 1, \ldots h - 1\}$, we recursively define the $\mathcal{E}$-distance *$\mathcal{E}_{x,y}(i, s, u)$ of $x[s : s + l_i]$ to the position u as follows. Partition $x[s : s + l_i]$ into b blocks of length $l_{i+1} = l_i/b$, starting at positions $s + jl_{i+1}$, where $j \in \{0, 1, \ldots b - 1\}$. Then*

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

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

We illustate the $\mathcal{E}$-distance for $b = 4$ in Figure 1. Since x and y will be clear from the context, we will just use the notation $\mathcal{E}(i, s, u)$ without indices x and y.

The main property of the $\mathcal{E}$-distance is that it gives a good approximation to the edit distance between x and y, as quantified in the following theorem, which we prove below.

Theorem 3.3 (Characterization)**.** *For evry $b \geq 2$ and two strings $x, y \in \Sigma^n$, the $\mathcal{E}$-distance between x and y is a $6 \cdot \frac{b}{\log b} \cdot \log n$ approximation to the edit distance between x and y.*

We also give an alternative, equivalent definition of the $\mathcal{E}$-distance between x and y. It is motivated by considering the matching (alignment) induced by the $\mathcal{E}$-distance when computing $\mathcal{E}(0, 1, 1)$. In particular, when computing $\mathcal{E}(0, 1, 1)$ recursively, we can consider all the "matching positions" (positions $u + jl_{i+1} + r_j$ for r_j's achieving the minimum). We denote by Z a vector of integers $z_{i,s}$, indexed by $i \in \{0, 1, \ldots h\}$ and $s \in B_i$, where $z_{0,1} = 1$ by convention. The coordinate $z_{i,s}$ should be understood as the position to which we match the substring $x[s : s + l_i]$ in the calculation of $\mathcal{E}(0, 1, 1)$. Then we define the cost of Z as

$$\text{cost}(Z) \stackrel{\text{def}}{=} \sum_{i=0}^{h-1} \sum_{s \in B_i} \sum_{j=0}^{b-1} |z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}}|.$$

[3] We remind that the notation $x[s : s + l]$ corresponds to characters $x[s], x[s + 1], \ldots x[s + l - 1]$. More generally, $[s : s + l]$ stands for the interval $\{s, s + 1, \ldots, s + l - 1\}$. This convention simplifies subsequent formulas.

The cost of Z can be seen as the sum of the displacements $|r_j|$ that appear in the calculation of the $\mathcal{E}$-distance from Definition 3.2. The following claim asserts an alternative definition of the $\mathcal{E}$-distance.

Claim 3.4 (Alternative definition of $\mathcal{E}$-distance)**.** *The $\mathcal{E}$-distance between x and y is the minimum of*

$$\mathrm{cost}(Z) + \sum_{s\in[n]} \mathrm{H}(x[s], y[z_{h,s}]) \tag{2}$$

over all choices of the vector $Z = (z_{i,s})_{i\in\{0,1,\ldots h\}, s\in B_i}$ with $z_{0,1} = 1$, where $\mathrm{H}(\cdot,\cdot)$ is the Hamming distance, namely $H(x[s], y[z_{h,s}])$ is 1 if $z_{h,s} \notin [n]$ or $x[s] \neq y[z_{h,s}]$, and 0 otherwise.

Proof. The quantity (2) simply unravels the recursive formula from Definition 3.2. The equivalence between them follows from the fact that $|z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}}|$ directly corresponds to quantities $|r_j|$ in the $\mathcal{E}_{x,y}(i, s, z_{i,s})$ definition, which appear in the computation on the tree, and the $\sum_{s\in[n]} \mathrm{H}(x[s], y[z_{h,s}])$ term corresponds to the summation of $\mathcal{E}_{x,y}(h, s, z_{h,s})$ over all $s \in [n]$. □

We are now ready to prove Theorem 3.3.

Proof of Theorem 3.3. Fix $n, b \geq 2$ and let $h \stackrel{\text{def}}{=} \log_b n$. We break the proof into two parts, an upper bound and a lower bound on the $\mathcal{E}$-distance (in terms of edit distance). They are captured by the following two lemmas, which we shall prove shortly.

Lemma 3.5. *The $\mathcal{E}$-distance between x and y is at most $3hb \cdot \mathrm{ed}(x, y)$.*

Lemma 3.6. *The edit distance $\mathrm{ed}(x, y)$ is at most twice the $\mathcal{E}$-distance between x and y.*

Combining these two lemmas gives $\frac{1}{2}\mathrm{ed}(x, y) \leq \mathcal{E}_{x,y}(0, 1, 1) \leq 5hb \cdot \mathrm{ed}(x, y)$, which proves Theorem 3.3. □

We proceed to prove these two lemmas.

Proof of Lemma 3.5. Let $A : [n] \to [n] \cup \{\perp\}$ be an optimal alignment from x to y. Namely A is such that:

- If $A(s) \neq \perp$, then $x[s] = y[A(s)]$.
- If $A(s_1) \neq \perp$, $A(s_2) \neq \perp$, and $s_1 < s_2$, then $A(s_1) < A(s_2)$.
- $L \stackrel{\text{def}}{=} |A^{-1}(\perp)|$ is minimized.

Note that $n - L$ is the length of the Longest Common Subsequence (LCS) of x and y. It clearly holds that $\frac{1}{2}\mathrm{ed}(x, y) \leq L \leq \mathrm{ed}(x, y)$.

To show an upper bound on the $\mathcal{E}$-distance, we use the alternative characterization from Claim 3.4. Specifically, we show how to construct a vector Z proving that the $\mathcal{E}$-distance is small.

At each level $i \in \{1, 2, \ldots h\}$, for each block $x[s : s + l_i]$ where $s \in B_i$, we set $z_{i,s} \stackrel{\text{def}}{=} A(j)$, where j is the smallest integer $j \in [s : s + l_i]$ such that $A(j) \neq \perp$ (i.e., to match a block we use the first in it that is aligned under the alignment A). If no such j exists, then $z_{i,s} \stackrel{\text{def}}{=} z_{i-1,s'} + (s - s')$, where $s' \stackrel{\text{def}}{=} l_{i-1} \cdot \lfloor (s-1)/l_{i-1} \rfloor + 1$, that is, s' is such that $x[s' : s' + l_{i-1}]$ is the parent of $x[s : s + l_i]$ in the tree.

Note that it follows from the definition of $z_{h,s}$ and L that $\sum_{s\in[n]} \mathrm{H}(x[s], y[z_{h,s}]) = L$. It remains to bound the other term $\mathrm{cost}(Z)$ in the alternative definition of $\mathcal{E}$-distance.

To accomplish this, for every $i \in \{0, 1, 2, \ldots, h-1\}$ and $s \in B_i$, we define $d_{i,s}$ as the maximum of $|z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}}|$ over $j \in \{0, \ldots, b-1\}$. Although we cannot bound each $d_{i,s}$ separately, we bound the sum of $d_{i,s}$ for each level i.

Claim 3.7. *For each $i \in \{0, 1, \ldots h\}$, we have that $\sum_{s\in B_i} d_{i,s} \le 2L$.*

Proof. We shall prove that each $d_{i,s}$ is bounded by $X_{i,s}+Y_{i,s}$, where $X_{i,s}$ and $Y_{i,s}$ are essentially the number of unmatched positions in x and in y, respectively, that contribute to $d_{i,s}$. We then argue that both $\sum_{s\in B_i} X_{i,s}$ and $\sum_{s\in B_i} Y_{i,s}$ are bounded by L, thus completing the proof of the claim.

Formally, let $X_{i,s}$ be the number of positions $j \in [s : s + l_i]$ such that $A(j) = \perp$. If $X_{i,s} = l_i$, then clearly $d_{i,s} = 0$. It is also easily verified that if $X_{i,s} = l_i - 1$, then $d_{i,s} \le l_i - 1$. In both cases, $d_{i,s} \le X_{i,s}$, and we also set $Y_{i,s} \stackrel{\text{def}}{=} 0$.

If $X_{i,s} \le l_i - 2$, let j' be the largest integer $j' \in [s : s + l_i]$ for which $A(j') \neq \perp$ (note that j' exists and it is different from the smallest such possible integer, which was called j when we defined $z_{i,s}$, because $X_{i,s} \le l_i - 2$). In this case, let $Y_{i,s}$ be $A(j') - z_{i,s} + 1 - (l_i - X_{i,s})$, which is the number of positions in y between $z_{i,s}$ and $A(j')$ (inclusive) that are not aligned under A. Let $\Delta_{i,s,j} \stackrel{\text{def}}{=} z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}}$ for $j \in \{0, \ldots, b-1\}$. By definition, it holds $d_{i,s} = \max_j |\Delta_{i,s,j}|$. Now fix j. If $\Delta_{i,s,j} \neq 0$, then there is an index $k \in [s + jl_{i+1} : s + (j + 1)l_{i+1}]$ such that $A(k) = z_{i+1,s+jl_{i+1}}$. If $\Delta_{i,s,j} > 0$ (which corresponds to a shift to the left), then at least $\Delta_{i,s,j}$ indices $j' \in [s : k]$ are such that $A(j') = \perp$, and therefore, $|\Delta_{i,s,j}| \le X_{i,s}$. If $\Delta_{i,s,j} < 0$ (which corresponds to a shift to the right), then at least $|\Delta_{i,s,j}|$ positions in y between $z_{i,s}$ and $z_{i+1,s+jl_{i+1}}$ are not aligned in A. Thus, $|\Delta_{i,s,j}| \le Y_{i,s}$.

In conclusion, for every $s \in B_i$, $d_{i,s} \le X_{i,s}+Y_{i,s}$. Observe that $\sum_{s\in B_i} X_{i,s} = L$ and $\sum_{s\in B_i} Y_{i,s} \le L$ (because they correspond to distinct positions in x and in y that are not aligned by A). Hence, we obtain that $\sum_{s\in B_i} d_{i,s} \le \sum_{s\in B_i} X_{i,s} + Y_{i,s} \le 2L$. □

We now claim that $\mathrm{cost}(Z) \le 2hbL$. Indeed, consider a block $x[s : s + l_i]$ for some $i \in \{0, 1, \ldots h-1\}$ and $s \in B_i$, and one of its children $x[s+jl_{i+1} : s+(j+1)l_{i+1}]$ for $j \in \{0, 1, \ldots b-1\}$. The contribution of this child to the sum $\mathrm{cost}(Z)$ is $|z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}}| \le d_{i,s}$ by definition. Hence, using Claim 3.7, we conclude that

$$\mathrm{cost}(Z) \le \sum_{i=0}^{h-1} \sum_{s\in B_i} \sum_{j=0}^{b-1} d_{i,s} \le \sum_{i=0}^{h-1} \sum_{s\in B_i} d_{i,s} \cdot b \le h \cdot 2L \cdot b.$$

Finally, by Claim 3.4, we have that the $\mathcal{E}$-distance between x and y is at most $2hbL + L \le 2hb \cdot \mathrm{ed}(x, y) + \mathrm{ed}(x, y) \le 3hb \cdot \mathrm{ed}(x, y)$. □

Proof of Lemma 3.6. We again use the alternative characterization given by Claim 3.4. Let Z be the vector obtaining the minimum of Equation (2). Define, for $i \in \{0, 1, \ldots h\}$ and $s \in B_i$,

$$\delta_{i,s} \stackrel{\text{def}}{=} \sum_{s'\in[s:s+l_i]} \mathrm{H}(x[s'], y[z_{h,s'}]) + \sum_{i':i\le i'<h} \sum_{s'\in B_{i'}\cap[s:s+l_i]} \sum_{j=0}^{b-1} \left| z_{i',s'} + jl_{i'+1} - z_{i'+1,s'+jl_{i'+1}} \right|.$$

Note that $\delta_{0,1}$ equals the $\mathcal{E}$-distance by Claim 3.4. Also, we have the following inductive equality for $i \in \{0, 1, \dots h-1\}$ and $s \in B_i$:

$$\delta_{i,s} = \sum_{j=0}^{b-1} \left(\delta_{i+1,s+jl_{i+1}} + |z_{i,s} + jl_{i+1} - z_{i+1,s+jl_{i+1}}| \right). \tag{3}$$

We now prove inductively for $i \in \{0, 1, 2 \dots h\}$ that for each $s \in B_i$, the length of the LCS of $x[s : s + l_i]$ and $y[z_{i,s} : z_{i,s} + l_i]$ is at least $l_i - \delta_{i,s}$.

For the base case, when $i = h$, the inductive hypothesis is trivially true. If $x[s] = y[z_{i,s}]$, then the LCS is of length 1 and $\delta_{h,s} = 0$. If $x[s] \neq y[z_{i,s}]$, then the LCS is of length 0 and $\delta_{h,s} = 1$.

Now we prove the inductive hypothesis for $i \in \{0, 1, \dots h-1\}$, assuming it holds for $i+1$. Fix a string $x[s : s + l_i]$, and let $s_j = s + jl_{i+1}$ for $j \in \{0, 1, \dots b-1\}$. By the inductive hypothesis, for each $j \in \{0, 1, \dots b-1\}$, the length of the LCS between $x[s_j : s_j + l_{i+1}]$ and $y[z_{i+1,s_j} : z_{i+1,s_j} + l_{i+1}]$ is at least $l_{i+1} - \delta_{i+1,s_j}$. In this case, the substring in y starting at $z_{i,s} + jl_{i+1}$, namely $y[z_{i,s} + jl_{i+1} : z_{i,s} + (j+1)l_{i+1}]$, has an LCS with $x[s_j : s_j + l_{i+1}]$ of length at least $l_{i+1} - \delta_{i+1,s_j} - |z_{i,s} + jl_{i+1} - z_{i+1,s_j}|$. Thus, by Equation (3), the LCS of $x[s : s + l_i]$ and $y[z_{i,s} : z_{i,s} + l_i]$ is of length at least

$$\sum_{j=0}^{b-1} \left(l_{i+1} - \delta_{i+1,s_j} - |z_{i,s} + jl_{i+1} - z_{i+1,s_j}| \right) = l_i - \delta_{i,s},$$

which finishes the proof of the inductive step.

For $i = 0$, this implies that $\mathrm{ed}(x, y) \leq 2\delta_{0,1} = 2\mathcal{E}_{x,y}(0, 1, 1)$. □

3.2 Sampling Algorithm

We now describe the sampling and estimation algorithms that are used to obtain our query complexity upper bounds. In particular, our algorithm approximates the $\mathcal{E}$-distance defined in the previous section. The guarantee of our algorithms is that the output $\hat{\mathcal{E}}$ satisfies $(1 - o(1))\mathcal{E}(0, 1, 1) - n/\beta \leq \hat{\mathcal{E}} \leq (1 + o(1))\mathcal{E}(0, 1, 1) + n/\beta$. This is clearly sufficient to distinguish between $\mathcal{E}(0, 1, 1) \leq n/\beta$ and $\mathcal{E}(0, 1, 1) \geq 4n/\beta$. After presenting the algorithm, we prove its correctness and prove that it only samples $\beta \cdot n^{O(\varepsilon)}$ positions of x in order to make the decision.

3.2.1 Algorithm Description

We now present our sampling algorithm, as well as the estimation algorithm, which given y and the sample of x, decides DTEP_β.

Sampling algorithm. To subsample x, we start by partitioning x recursively into blocks as defined in Definition 3.2. In particular, we fix a tree of arity b, indexed by pairs (i, s) for $i \in \{0, 1, \dots h\}$, and $s \in B_i$. At each level $i = 0, \dots h$, we have a subsampled set $C_i \subseteq B_i$ of vertices at that level of the tree. The set C_i is obtained from the previous one by extending C_{i-1} (considering all the children), and a careful subsampling procedure. In fact, for each element in C_i, we also assign a number $w \geq 1$, representing a "precision" and describing how well we want to estimate the $\mathcal{E}$ distance at that node, and hence governing the subsampling of the subtree rooted at the node.

Our sampling algorithm works as follows. We use a (continuous) distribution $\mathcal{W}$ on $[1, n^3]$, which we define later, in Lemma 3.12.

Algorithm 1: Sampling Algorithm

Take C_0 to be the root vertex (indexed $(i,s) = (0,1)$), with precision $w_{(0,1)} = \beta$.
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.

Estimation Algorithm. We compute a value $\tau(v, z)$, for each node $v \in \cup_i C_i$ and position $z \in [n]$, such that $\tau(v, z)$ is a good approximation ($1 + o(1)$ factor) to the $\mathcal{E}$-distance of the node v to position z.

We also use a "reconstruction algorithm" R, defined in Lemma 3.12. It takes as input at most b quantities, their precision, and outputs a positive number.

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.

3.2.2 Analysis Preliminaries: Approximators and a Concentration Bound

We use the following approximation notion that captures both an additive and a multiplicative error. For convenience, we work with factors e^{ε} instead of usual $1 + \varepsilon$.

Definition 3.8. *Fix* $\rho > 0$ *and some* $f \in [1, 2]$. *For a quantity* $\tau \geq 0$, *we call its* (ρ, f)–approximator *any quantity* $\hat{\tau}$ *such that* $\tau/f - \rho \leq \hat{\tau} \leq f\tau + \rho$.

It is immediate to note the following *additive property*: if $\hat{\tau}_1, \hat{\tau}_2$ are (ρ, f)-approximators to τ_1, τ_2 respectively, then $\hat{\tau}_1 + \hat{\tau}_2$ is a $(2\rho, f)$-approximator for $\tau_1 + \tau_2$. Also, there's a composion

property: if $\hat{\tau}'$ is an (ρ', f')-approximator to $\hat{\tau}$, which itself is a (ρ, f)-approximator to τ, then $\hat{\tau}'$ is a $(\rho' + f'\rho, ff')$-aproximator to τ.

The definition is motivated by the following concentration statement on the sum of random variables. The statement is an immediate application of the standard Chernoff/Hoeffding bounds.

Lemma 3.9 (Sum of random variables)**.** *Fix $n \in \mathbb{N}$, $\rho > 0$, and error probability δ. Let $Z_i \in [0, \rho]$ be independent random variables, and let $\zeta > 0$ be a sufficiently large absolute constant. Then for every $\varepsilon \in (0, 1)$, the summation $\sum_{i\in[n]} Z_i$ is a $(\zeta\rho\frac{\log 1/\delta}{\varepsilon^2}, e^{\varepsilon})$-approximator to $\mathbb{E}\left[\sum_{i\in[n]} Z_i\right]$, with probability $\geq 1 - \delta$.*

Proof of Lemma 3.9. By rescaling, it is sufficient to prove the claim for $\rho = 1$. Let $\mu = \mathbb{E}\left[\sum_{i\in[n]} Z_i\right]$. If $\mu > \frac{\zeta}{4} \cdot \frac{\log 1/\delta}{\varepsilon^2}$, then, a standard application of the Chernoff implies that $\sum_i Z_i$ is a e^{ε} approximation to μ, with $\geq 1 - \delta$ probability, for some sufficiently high $\zeta > 0$.

Now assume that $\mu \leq \frac{\zeta}{4} \cdot \frac{\log 1/\delta}{\varepsilon^2}$. We use the following variant of the Hoeffding inequality, which can be derived from [Hoe63].

Lemma 3.10 (Hoeffding bound)**.** *Let Z_i be n independent random variables such that $Z_i \in [0, 1]$, and $\mathbb{E}\left[\sum_{i\in[n]} Z_i\right] = \mu$. Then, for any $t > 0$, we have that $\Pr\left[\sum_i Z_i \geq t\right] \leq e^{-(t-2\mu)}$.*

We apply the above lemma for $t = \zeta \cdot \frac{\log 1/\delta}{\varepsilon^2}$. We obtain that $\Pr[\sum_i Z_i \geq t] \leq e^{-t/2} = e^{-\Omega(\log 1/\delta)} < \delta$, which completes the proof that $\sum_i Z_i$ is a $(\zeta\frac{\log 1/\delta}{\varepsilon^2}, e^{\varepsilon})$-approximator to μ (when $\rho = 1$). □

3.2.3 Main Analysis Tools: Uniform and Non-uniform Sampling Lemmas

We present our two main subsampling lemmas that are applied, recursively, at each node of the tree. The first lemma, on Uniform Sampling, is a simple Chernoff bound in a suitable regime.

The second lemma, called Non-uniform Sampling Lemma, is the heart of our sampling, and is inspired by a sketching/streaming technique introduced in [IW05] for optimal estimation of F_k moments in a stream. Although a relative of their method, our lemma is different both in intended context and actual technique. We shall use the constant $\zeta > 0$ coming from Lemma 3.9.

Lemma 3.11 (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 subsampling each $j \in [b]$ with probability $p_w = \min\{1, \frac{w}{b} \cdot \zeta\frac{\log 1/\delta}{\varepsilon^2}\}$. 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 $\sum_{j\in[b]} a_j$, and $|J| \leq O(w \cdot \frac{\log 1/\delta}{\varepsilon^2})$.*

Proof. If $p_w = 1$, then $J = [b]$ and there is nothing to prove; so assume that $p_w = \frac{w}{b} \cdot \zeta\frac{\log 1/\delta}{\varepsilon^2} < 1$ for the rest.

The bound on $|J|$ follows from a standard application of the Chernoff bound: $\mathbb{E}[|J|] = p_w b \leq O(w \cdot \frac{\log 1/\delta}{\varepsilon^2})$, hence the probability that $|J|$ exceeds twice the quantity is at most $e^{-\Omega(\log 1/\delta)} \leq \delta/2$.

We are going to apply Lemma 3.9 to the variables $Z_j = a_j/p_w \cdot \chi[j \in J]$, where the indicator variable $\chi[j \in J]$ is 1 iff $j \in J$. Note that $0 \leq Z_j \leq \frac{\varepsilon^2}{w \cdot \zeta \log 1/\delta}$. We thus obtain that $\sum_{j\in[b]} Z_j$ is a $(\frac{\zeta\varepsilon^{-2}\log 1/\delta}{w \cdot \zeta\varepsilon^{-2}\log 1/\delta}, e^{\varepsilon})$-approximator, and hence $(1/w, e^{\varepsilon})$-approximator, to $\mathbb{E}\left[\sum_j Z_j\right] = \sum_{j\in[b]} p_w \cdot \frac{a_j}{p_w} = \sum_{j\in[b]} a_j$. □

We now present and prove the Non-uniform Sampling Lemma.

Lemma 3.12 (Non-uniform Sampling)**.** *Fix integers $n \le N$, 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 $\mathbb{E}_{w\in\mathcal{W}}[w] \le O(\frac{1}{\rho}\cdot\frac{\log 1/\delta}{\varepsilon^3}\cdot\log N)$, 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 $1-\delta$.

For concreteness, we mention that $\mathcal{W}$ is the maximum of $O(\frac{1}{\rho}\cdot\frac{\log 1/\delta}{\varepsilon^3})$ copies of the (truncated) distribution $1/x^2$ (essentially equivalent to a distribution of x where the logarithm of x is distributed geometrically).

Proof. We start by describing the distribution $\mathcal{W}$ and the algorithm R. Fix $k = \frac{2\zeta}{\rho}\cdot\frac{\log 1/\delta}{(\varepsilon/2)^3}$. We first describe a related distribution: let $\mathcal{W}_1$ be distribution on x such that the pdf function is $p_1(x) = \nu/x^2$ for $1 \le x \le N^3$ and $p_1(x) = 0$ otherwise, where $\nu = (\int_1^\infty p_1(x)\,\mathrm{dx})^{-1} = (1-1/N^3)^{-1}$ is a normalization constant. Then $\mathcal{W}$ is the distribution of x where we choose k i.i.d. variables $x_1, \ldots x_k$ from $\mathcal{W}_1$ and then set $x = \max_{i\in[k]} x_i$. Note that the pdf of $\mathcal{W}$ is $p(x) = \nu^k \frac{k}{x^2}(1-1/x)^{k-1}$.

The algorithm R works as follows. For each $i \in [n]$, we define k "indicators" $s_{i,j} \in \{0, 1/k\}$ for $j \in [k]$. Specifically, we generate the set of random variables $w_{i,j} \in \mathcal{W}_1$, $j \in [k]$, conditioned on the fact that $\max_{j\in[k]} w_{i,j} = w_i$. Then, for each $i \in [n], j \in [k]$, we set $s_{i,j} = 1/k$ if $\hat{a}_i \ge t/w_i$ for $t = 3/\varepsilon$, and $s_{i,j} = 0$ otherwise. Finally, we set $s = \sum_{i\in[n],j\in[k]} s_{i,j}$ and the algorithm outputs $\hat{\sigma} = st/\nu$ (as an estimate for σ).

We note that the variables $w_{i,j}$ could be thought as being chosen i.i.d. from $\mathcal{W}_1$. For each, the value $\hat{a}_i$ is an $(1/w_{i,j}, f)$-approximator to a_i since $\hat{a}_i$ is a $(1/\max_j w_{i,j}, f)$-approximator to a_i.

It is now easy to bound $\mathbb{E}_{w\in\mathcal{W}}[w]$. Indeed, we have $\mathbb{E}_{w\in\mathcal{W}_1}[w] = \int_1^{N^3} x\cdot\nu/x^2\,\mathrm{dx} \le O(\log N)$. Hence $\mathbb{E}_{w\in\mathcal{W}}[w] \le \sum_{j\in[k]} \mathbb{E}_{w\in\mathcal{W}_1}[w] \le O(k\log N) = O(\frac{1}{\rho}\cdot\frac{\log 1/\delta}{\varepsilon^3}\cdot\log N)$.

We now need to prove that $\hat{\sigma}$ is an approximator to σ, with probability at least $1-\delta$. We first compute the expectation of $s_{i,j}$, for each $i \in [n], j \in [k]$. This expectation depends on the approximator values $\hat{a}_i$, which itself may depend on w_i. Hence we can only give upper and lower bounds on the expectation $\mathbb{E}[s_{i,j}]$. Later, we want to apply a concentration bound on the sum of $s_{i,j}$. Since $s_{i,j}$ may be interdependent, we will apply the concentration bound on the upper/lower bounds of $s_{i,j}$ to give bounds on $s = \sum s_{i,j}$.

Formally, we define random variables $\overline{s}_{i,j}, \underline{s}_{i,j} \in \{0, 1/k\}$. We set $\overline{s}_{i,j} = 1/k$ iff $w_{i,j} \ge (t-1)/(fa_i)$, and 0 otherwise. Similarly, we set $\underline{s}_{i,j} = 1/k$ iff $w_{i,j} < f(t+1)/a_i$, and 0 otherwise. We now claim that

$$\underline{s}_{i,j} \le s_{i,j} \le \overline{s}_{i,j}. \tag{4}$$

Indeed, if $s_{i,j} = 1/k$, then $\hat{a}_i \ge t/w_{i,j}$, and hence, using the fact that $\hat{a}_i$ is a $(1/w_{i,j}, f)$-approximator to a_i, we have $w_{i,j} \ge (t-1)/(fa_i)$, or $\overline{s}_{i,j} = 1/k$. Similarly, if $s_{i,j} = 0$, then $\hat{a}_i < t/w_{i,j}$, and hence $w_{i,j} < f(t+1)/a_i$, or $\underline{s}_{i,j} = 0$. Note that each collection $\{\overline{s}_{i,j}\}$ and $\{\underline{s}_{i,j}\}$ is a collection of independent random variables.

We now bound $\mathbb{E}[\overline{s}_{i,j}]$ and $\mathbb{E}[\underline{s}_{i,j}]$. For the first quantity, we have:

$$\mathbb{E}[\overline{s}_{i,j}] = \int_{(t-1)/(fa_i)}^{N^3} \tfrac{1}{k} p_1(x)\,\mathrm{dx} \le \tfrac{fa_i}{k(t-1)} \int_1^\infty \nu/x^2\,\mathrm{dx} = \nu/k\cdot\tfrac{fa_i}{t-1}.$$

For the second quantity, we have:

$$\mathbb{E}\left[\underline{s}_{i,j}\right] = \int_{f(t+1)/a_i}^{N^3} p_1(x)\,\mathrm{dx} = \nu/k \cdot (\tfrac{a_i/f}{t+1} - 1/N^3).$$

Finally, using Eqn. (4) and the fact that $\mathbb{E}[s] = \sum_{i,j}\mathbb{E}[s_{i,j}]$, we can bound $\mathbb{E}[\hat{\sigma}] = \mathbb{E}[st/\nu]$ as follows:

$$\tfrac{t}{f(t+1)}\sum_{i\in[n]} a_i - nt/N^3 \le \tfrac{t}{\nu}\sum_{i,j}\mathbb{E}\left[\underline{s}_{i,j}\right] \le \mathbb{E}[ts/\nu] \le \tfrac{t}{\nu}\sum_{i,j}\mathbb{E}\left[\overline{s}_{i,j}\right] \le f\sum_{i\in[n]} a_i \cdot \tfrac{t}{t-1}.$$

Since each $\overline{s}_{i,j}, \underline{s}_{i,j} \in [0, 1/k]$ for $k = O(\frac{t}{\rho}\cdot\frac{\log 1/\delta}{\varepsilon^2})$, we can apply Lemma 3.9 to obtain a high concentration bound. For the upper bound, we obtain, with probability at least $1-\delta/2$:

$$ts/\nu \le e^{\varepsilon/2}\cdot\mathbb{E}\left[t/\nu\cdot\sum_{i,j} s_{i,j}\right] + \rho \le e^{\varepsilon/2}\cdot f\sum a_i\cdot\tfrac{t}{t-1} + \rho \le e^{\varepsilon}\cdot f\cdot\sigma + \rho.$$

Similarly, for the lower bound, we obtain, with probability at least $1-\delta/2$:

$$ts/\nu \ge e^{-\varepsilon/2}\cdot(\sum a_i\cdot\tfrac{t}{f(t+1)} - nt/N^3) - \rho/2 \ge e^{-\varepsilon}/f\cdot\sigma - \rho,$$

using that $\rho/2 \ge nt/N^3$. This completes the proof that $\hat{\sigma}$ is a $(\rho, f\cdot e^{\varepsilon})$-approximator to σ, with probability at least $1-\delta$. □

3.2.4 Correctness and Sample Bound for the Main Algorithm

Now, we prove the correctness of the algorithms 1, 2 and bound its query complexity. We note that we use Lemmas 3.11 and 3.12 with $\delta = 1/n^3$, $\varepsilon = 1/\log n$, and $N = n$ (which in particular, completely determine the distribution $\mathcal{W}$ and algorithm R used in the algorithms 1 and 2).

Lemma 3.13 (Correctness)**.** *For $b = \omega(1)$, the output of the Algorithm 2 (Estimation), is a $(n/\beta, 1+o(1))$-approximator to the $\mathcal{E}$-distance from x to y, w.h.p.*

Proof. From a high level view, we prove inductively from $i = 0$ to $i = h$ that expanding/subsampling the current C_i gives a good approximator, namely a $e^{O((h-i)/\log n)}$ factor approximation, with probability at least $1 - i/n^{\Omega(1)}$. Specifically, at each step of the induction, we expand and subsample each node from the current C_i to form the set C_{i+1} and use Lemmas 3.11 and 3.12 to show that we don't loose on the approximation factor by more than $e^{O(1/\log n)}$.

In order to state our main inductive hypothesis, we define a hybrid distance, where the $\mathcal{E}$-distance of nodes at high levels (big i) is computed standardly (via Definition 3.2), and the $\mathcal{E}$-distance of the low-level nodes is estimated via sets C_i. Specifically, for fixed $f \in [1, 1.1]$, and $i \in \{0, 1, \dots h\}$, we define the following $(C_0, C_1 \dots C_i, f)$-*$\mathcal{E}$-distance*. For each vertex $v = (i, s)$ such that $v \in C_i$ has precision w_v, and $z \in [n]$, let $\tau_i(v, z)$ to be some $(l_i/w_v, f)$–approximator to the distance $\mathcal{E}(i, s, z)$. Then, iteratively for $i' = i-1, i-2, \dots, 0$, for all $v \in C_{i'}$ and $z \in [d]$, we compute $\tau_i(v, z)$ by applying the procedure $P(v, z)$ (defined in the Algorithm 2), using τ_i instead of τ.

We prove the following inductive hypothesis, for some suitable constants $t = 2$ and $r = \Theta(1)$ (sufficiently high r suffices).

IH$_i$: For any $f \in [1, 1.1]$, the $(C_0, C_1, \ldots C_i, f)$–$\mathcal{E}$–distance is a $(n/\beta, f \cdot e^{i \cdot t/\log n})$-approximator to the $\mathcal{E}$–distance from x to y, with probability at least $1 - i \cdot e^{-r \log n}$.

Base case is $i = 0$, namely that (C_0, f)-$\mathcal{E}$-distance is a $(n/\beta, f)$-approximator to the $\mathcal{E}$-distance between x and y. This case follows immediately from the definition of the (C_0, f)-$\mathcal{E}$-distance and the initialization step of the Sampling Algorithm.

Now we prove the inductive hypothesis IH$_{i+1}$, assuming IH$_i$ holds for some given $i \in \{0, 1, \ldots h-1\}$. We remind that we defined the quantity $\tau_{i+1}(v, z)$, for all $v \in C_{i+1} \subseteq \overline{C}_i$, where $\overline{C}_i = \{(i+1, s + jl_{i+1}) \mid (i, s) \in C_i, j \in \{0, \ldots b-1\}\}$ and $z \in [n]$, to be a $(l_{i+1}/w_v, f)$–approximator of the corresponding $\mathcal{E}$-distance, namely $\mathcal{E}(v, z)$. The plan is to prove that, for all $v \in C_i$ with precision w_v, the quantity $\tau_{i+1}(v, z)$ is a $(l_i/w_v, f \cdot e^{2/\log n})$–approximator to $\mathcal{E}(v, z)$ with good probability — which we do in the claim below. Then, by definition of τ_i and IH$_i$, this implies that $\tau_{i+1}((0, 1), 1)$ is equal to the $(C_0, \ldots C_i, f \cdot e^{2/\log n} \cdot e^{i \cdot t/\log n})$–$\mathcal{E}$–distance, and hence is a $(n/\beta, f \cdot e^{(2+it)/\log n})$–approximator to the $\mathcal{E}$–distance from x to y. This will complete the proof of IH$_{i+1}$. We now prove the main technical step of the above plan.

Claim 3.14. *Fix $v \in C_i$ with precision $w \stackrel{\text{def}}{=} w_v$, where $v = (i, s)$, and some $z \in [n]$. For $j \in \{0, \ldots b-1\}$, let v_j be the j^{th} child of v; i.e., $v_j = (i+1, s + jl_{i+1})$. For $v_j \in C_{i+1}$ with precision $w_j \stackrel{\text{def}}{=} w_{v_j}$, and $z' \in [n]$, let $\tau_{i+1}(v_j, z')$ be a $(l_{i+1}/w_j, f)$–approximator to $\mathcal{E}(v_j, z')$.*

Apply procedure $P(v, z)$ using $\tau_{i+1}(v_j, z')$ estimates, and let δ be the output. Then δ is a $(l_i/w, fe^{2/\log n})$–approximator to $\mathcal{E}(v, z)$, with probability at least $1 - e^{-\Omega(\log n)}$.

Proof. For each $v_j \in J_v$, where J_v is as defined in Algorithm 1, we define the following quantities:

$$\delta_{v_j} \stackrel{\text{def}}{=} \min_{k:|k| \le n} \mathcal{E}(v_j, z + jl_{i+1} + k) + |k| \qquad \hat{\delta}_{v_j} \stackrel{\text{def}}{=} \min_{k:|k| \le n} \tau_{i+1}(v_j, z + jl_{i+1} + k) + |k|.$$

It is immediate to see that $\hat{\delta}_{v_j}$ is a $(l_{i+1}/w_j, f)$–approximator to δ_{v_j} by the definition of τ_{i+1}.

If $p_v \ge 1$, then we have that $w_j = \frac{w}{b} \cdot O(\log^3 n)$ for all $v_j \in J_v$. Then, by the additive property of $(l_{i+1}/w_j, f)$–approximators, $\delta = \sum_{v_j \in J_v} \hat{\delta}_{v_j}$ is a $(l_i/w, f)$–approximator to $\sum_{v_j \in J_v} \delta_{v_j} = \mathcal{E}(v, z)$.

Now suppose $p_v < 1$. Then, by Lemma 3.11, $\delta' = \frac{1}{p_v} \sum_{v_j \in J_v} \delta_{v_j}$ is a $(l_i/2w, e^{1/\log n})$–approximator to $\sum_{j=0}^{b-1} \delta_{v_j} = \mathcal{E}(v, z)$, with high probability. Furthermore, by Lemma 3.12 for $\rho = 1$, since $w_j \in \mathcal{W}$ are i.i.d. and $\frac{\hat{\delta}_{v_j}}{l_{i+1}}$ are each an $(1/w_j, f)$–approximator to $\frac{\delta_{v_j}}{l_{i+1}}$ respectively, then R outputs a value δ'' that is a $(1, f \cdot e^{1/\log n})$–approximator to $\sum_{v_j \in J_v} \frac{\delta_{v_j}}{l_{i+1}} = \frac{p_v}{l_{i+1}} \delta'$. In other words, $\delta = \frac{l_{i+1}}{p_v} \delta''$ is a $(l_{i+1}/p_v, f \cdot e^{1/\log n})$–approximator to δ'. Since $l_{i+1}/p_v \le l_i/(3w)$, combining the two approximator guarantees, we obtain that δ is a $(l_i/w, f \cdot e^{2/\log n})$–approximator to $\mathcal{E}(v, z)$, w.h.p. □

We now apply a union bound over all $v \in C_i$ and $z \in [n]$, and use the above Claim 3.14. We now apply IH$_i$ to deduce that $\tau_{i+1}((0, 1), 1)$ is a $(n/\beta, f \cdot e^{ti/\log n} \cdot e^{2/\log n})$–approximator with probability at least

$$1 - ie^{-r \log n} - e^{-\Omega(\log n)} \ge 1 - (i+1)e^{-r \log n},$$

for some suitable $r = \Theta(1)$. This proves IH$_{i+1}$.

Finally we note that IH$_h$ implies that $(C_0, \ldots C_h, f)$–$\mathcal{E}$–distance is a $(n/\beta, f \cdot e^{th/\log n})$–approximator to the $\mathcal{E}$–distance between x and y. We conclude the lemma with the observation that our Estimation Algorithm 2 outputs precisely the $(C_0, \ldots C_h, 1)$–$\mathcal{E}$–distance. □

It remains to bound the number of positions that Algorithm 2 queries into x.

Lemma 3.15 (Sample size)**.** *The Sampling Algorithm queries $Q_b = \beta(\log n)^{O(\log_b n)}$ positions of x, with probability at least $1 - o(1)$. When $b = n^{1/t}$ for fixed constant $t \in \mathbb{N}$ and $\beta = O(1)$, we have $Q_b = (\log n)^{t-1}$ with probability at least 2/3.*

Proof. We prove by induction, from $i = 0$ to $i = h$, that $\mathbb{E}\left[|C_i|\right] \le \beta \cdot (\log n)^{ic}$, and $\mathbb{E}\left[\sum_{v \in C_i} w_v\right] \le \beta \cdot (\log n)^{ic+5}$ for a suitable $c = \Theta(1)$. The base case of $i = 0$ is immediate by the initialization of the Sampling Algorithm 1. Now we prove the inductive step for i, assuming the inductive hypothesis for $i-1$. By Lemma 3.11, $\mathbb{E}\left[|C_i|\right] \le \mathbb{E}\left[\sum_{v \in C_{i-1}} w_v\right] \cdot O(\log^3 n) \le \beta(\log n)^{ic}$ by the inductive hypothesis. Also, by Lemma 3.12, $\mathbb{E}\left[\sum_{v \in C_i} w_v\right] \le \mathbb{E}\left[|C_i|\right] \cdot O(\log^4 n) + \mathbb{E}\left[\sum_{v \in C_{i-1}} w_v\right] \cdot O(\log^3 n) \le \beta(\log n)^{ic+5}$. The bound then follows from an application of the Markov bound.

The second bound follows from a more careful use of the parameters of the two sampling lemmas, Lemmas 3.11 and 3.12. In fact, it suffices to apply these lemmas with $\varepsilon = e^{\Theta(1/t)}$ and $\delta = 0.1$ for the first level and $\delta = 1/n^3$ for subsequent levels. □

These lemmas, 3.13 and 3.15, together with the characterization theorem 3.3, almost complete the proof of Theorem 3.1. It remains to bound the run time of the resulting estimation algorithm, which we do in the next section.

3.3 Near-Linear Time Algorithm

We now discuss the time complexity of the algorithm, and show that the Algorithm 2 (Estimation) may be implemented in $n \cdot (\log n)^{O(h)}$ time. We note that as currently described in Algorithm 2, our reconstruction technique takes time $\tilde{O}(hQ_b \cdot n)$ time, where $Q_b = \beta(\log n)^{O(\log_b n)}$ is the sample complexity upper bound from Lemma 3.15 (note that, combined with the algorithm of [LMS98], this already gives a $n^{4/3+o(1)}$ time algorithm). The main issue is the computation of the quantities $\delta_{v'}$, as, naively, it requires to iterate over all $k \in [n]$.

To reduce the time complexity of the Algorithm 2, we define the following quantity, which replaces the quantity $\delta_{v'}$ in the description of the algorithm:

$$\delta'_{v'} = \min_{k = e^{i/\log n} : i \in [\log n \cdot \ln(3n/\beta)]} \left(|k| + \min_{k' : |k'| \le k} \tau(v', z + j l_{i+1} + k') \right).$$

Lemma 3.16. *If we use $\delta'_{v'}$ instead of $\delta_{v'}$ in Algorithm 2, the new algorithm outputs at most a $1 + o(1)$ factor higher value than the original algorithm.*

Proof. First we note that it is sufficient to consider only $k \in [-3n/\beta, 3n/\beta]$, since, if the algorithm uses some k with $|k| > 3n/\beta$, then the resulting output is guaranteed to be $> 3n/\beta$. Also, the estimate may only increase if one restricts the set of possible k's.

Second, if we consider k's that are integer powers of $e^{1/\log n}$, we increase the estimate by only a factor $e^{1/\log n}$. Over $h = O(\log_b n)$ levels, this factor accumulates to only $e^{h/\log n} \le 1 + o(1)$. □

Finally, we mention that computing all $\delta'_{v'}$ may be performed in $O(\log^2 n)$ time after we perform the following (standard) precomputation on the values $\tau(v', z')$ for $z' \in [n]$ and $v' \in C_{i+1}$. For each dyadic interval I, compute $\min_{z \in I} \tau(v, z)$. Then, for each (not necessarily dyadic) interval $I' \subset [n]$, computing $\min_{z' \in I'} \tau(v', z')$ may be done in $O(\log n)$ time. Hence, since we consider only $O(\log n)$ values of k, we obtain $O(\log^2 n)$ time per computation of $\delta'_{v'}$.

Total running time becomes $O(hQ_b \cdot n \cdot \log^2 n) = n \cdot (\log n)^{O(\log_b n)}$.

A more technical issue that we swept under the carpet is that distribution $\mathcal{W}$ defined in Lemma 3.12 is a continuous distribution on $[1, n^3]$. However this is not an issue since a $n^{-\Omega(1)}$ discretization suffices to obtain the same result, with only $O(\log n)$ loss in time complexity.

4 Query Complexity Lower Bound

We now give a full proof of our lower bound, Theorem 1.3. After some preliminaries, this section contains three rather technical parts: tools for analyzing indistinguishability, tools for analyzing edit distance behavior, and a finally a part where we put together all elements of the proof. The precise and most general forms of our lower bound appear in that final part as Theorem 4.15 and Theorem 4.16.

4.1 Preliminaries

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

$$\underline{\mathrm{ed}}(x, y) = |x| + |y| - 2\,\mathrm{LCS}(x, y), \tag{5}$$

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

Alignments. For two strings x, y of length n, an *alignment* is a function $A : [n] \to [n] \cup \{\perp\}$ that is monotonically increasing on $A^{-1}([n])$ and satisfies $x[i] = y[A(i)]$ for all $i \in A^{-1}([n])$. Observe that an alignment between x and y corresponds exactly to a common subsequence to x and y.

Projection. For a string $x \in \Sigma^n$ and $Q \subseteq [n]$, we write $x|_Q$ for the string that is the projection of x on the coordinates in Q. Clearly, $x|_Q \in \Sigma^{|Q|}$. Similarly, if $\mathcal{D}$ is a probability distribution over strings in Σ^n, we write $\mathcal{D}|_Q$ for the distribution that is the projection of $\mathcal{D}$ on the coordinates in Q. Clearly, $\mathcal{D}|_Q$ is a distribution over strings in $\Sigma^{|Q|}$.

Substitution Product. Suppose that we have a "mother" string $x \in \Sigma^n$ and a mapping $B : \Sigma \to (\Sigma')^{n'}$ of the original alphabet into strings of length n' over a new alphabet Σ'. Define the *substitution product* of x and B, denoted $x \circledast B$, to be the concatenation of $B(x_1), \cdots, B(x_n)$. Letting $B_a = B(a)$ for each $a \in \Sigma$ (i.e., B defines a collection of $|\Sigma|$ strings), we have

$$x \circledast B \stackrel{\text{def}}{=} B_{x_1} B_{x_2} \cdots B_{x_n} \in (\Sigma')^{nn'}.$$

Similarly, for each $a \in \Sigma$, let $\mathcal{D}_a$ be a probability distribution over strings in $(\Sigma')^{n'}$. The *substitution product* of x and $\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{D}_a)_{a \in \Sigma}$, denoted $x \circledast \mathcal{D}$, is defined as the probability distribution over strings in $(\Sigma')^{nn'}$ produced by replacing every symbol x_i, $1 \leq i \leq n$, in x by an independent sample B_i from $\mathcal{D}_{x_i}$.

Finally, let $\mathcal{E}$ be a "mother" probability distribution over strings in Σ^n, and for each $a \in \Sigma$, let $\mathcal{D}_a$ be a probability distribution over strings in $(\Sigma')^{n'}$. The *substitution product* of $\mathcal{E}$ and $\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{D}_a)_{a \in \Sigma}$,

denoted $\mathcal{E} \circledast \mathcal{D}$, is defined as the probability distribution over strings in $(\Sigma')^{nn'}$ produced as follows: first sample a string $x \sim \mathcal{E}$, then independently for each $i \in [n]$ sample $B_i \sim \mathcal{D}_{x_i}$, and report the concatenation $B_1 B_2 \dots B_n$.

Shift. For $x \in \Sigma^n$ and integer r, let $S^r(x)$ denote a cyclic shift of x (i.e. rotating x) to the left by r positions. Clearly, $S^r(x) \in \Sigma^n$. Similarly, let $\mathcal{S}_s(x)$ the distribution over strings in Σ^n produced by rotating x by a random offset in $[s]$, i.e. choose $r \in [s]$ uniformly at random and take $S^r(x)$.

For integers i, j, define $i +_n j$ to be the unique $z \in [n]$ such that $z = i + j \pmod n$. For a set Q of integers, let $Q +_n j = \{i +_n j : i \in Q\}$.

Fact 4.1. *Let $x \in \Sigma^n$ and $Q \subset [n]$. For every integer r, we have $S^r(x)|_Q = x|_{Q +_n r}$. Thus, for every integer s, the probability distribution $\mathcal{S}_s(x)|_Q$ is identical to $x|_{Q +_n r}$ for a random $r \in [s]$.*

4.2 Tools for Analyzing Indistinguishability

In this section, we introduce tools for analyzing indistinguishability of distributions we construct. We introduce a notion of uniform similarity, show what it implies for query complexity, give quantitative bounds on it for random cyclic shifts of random strings, and show how it composes under the substitution product.

4.2.1 Similarity of Distributions

We first define an auxiliary notion of similarity. Informally, a set of distributions on the same set are similar if the probability of every element in their support is the same up to a small multiplicative factor.

Definition 4.2. *Let $\mathcal{D}_1, \dots, \mathcal{D}_k$ be probability distributions on a finite set Ω. Let $p_i : \Omega \to [0,1]$, $1 \le i \le k$, be the probability mass function for $\mathcal{D}_i$. We say that the distributions are* α-similar *if for every $\omega \in \Omega$,*

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

We now define uniform similarity for distributions on strings. Uniform similarity captures how the similarity between distributions on strings changes as a function of the number of queries.

Definition 4.3. *Let $\mathcal{D}_1, \dots, \mathcal{D}_k$ be probability distributions on Σ^n. We say that they are* uniformly α-similar *if for every subset Q of $[n]$, the distributions $\mathcal{D}_1|_Q, \dots, \mathcal{D}_k|_Q$ are $\alpha|Q|$-similar.*

Finally, we show that if two distributions on strings are uniformly similar, then an algorithm distinguishing strings drawn from them has to make many queries.

Lemma 4.4. *Let $\mathcal{D}_0$ and $\mathcal{D}_1$ be uniformly μ-similar distributions on Σ^n. Let $\mathcal{A}$ be a randomized algorithm that makes q (adaptive) queries to symbols of a string selected according to either $\mathcal{D}_0$ or $\mathcal{D}_1$, and outputs either `0` or `1`. Let p_j, for $j \in \{0,1\}$, be the probability that $\mathcal{A}$ outputs `j` when the input is selected according to $\mathcal{D}_j$. Then*

$$\min\{p_0, p_1\} \le \frac{1+\mu q}{2}.$$

Proof. Once the random bits of $\mathcal{A}$ are fixed, $\mathcal{A}$ can be seen as a decision tree with depth q the following properties. Every internal node corresponds to a query to a specific position in the input string. Every internal node has $|\Sigma|$ children, and the $|\Sigma|$ edges outgoing to the children are labelled with distinct symbols from Σ. Each leaf is labelled with either `0` or `1`; this is the algorithm's output, i.e. the computation ends up in a leaf if and only if the sequence of queries on the path from the root to the leaf gives the sequence described by the edge labels on the path.

Fix for now $\mathcal{A}$'s random bits. Let t be the probability that $\mathcal{A}$ outputs `0` when the input is chosen from $\mathcal{D}_0$, and let t' be defined similarly for $\mathcal{D}_1$. We now show an upper bound on $t - t'$. t is the probability that the computation ends up in a leaf v labelled `0` for an input chosen according to $\mathcal{D}_0$. Consider a specific leaf v labelled with `0`. The probability of ending up in the leaf equals the probability of obtaining a specific sequence of symbols for a specific sequence of at most q queries. Let t_v be this probability when the input is selected according to $\mathcal{D}_0$. The same probability for $\mathcal{D}_1$ must be at least $(1 - q\mu)t_v$, due to the uniform μ-similarity of the distributions. By summing over all leaves v labelled with `0`, we have $t' \geq (1 - \mu q)t$, and therefore, $t - t' \leq q\mu \cdot t \leq q\mu$.

Note that p_0 is the expectation of t over the choice of $\mathcal{A}$'s random bits. Analogously, $1 - p_1$ is the expectation of t'. Since $t - t'$ is always at most μq, we have $p_0 - (1 - p_1) \leq \mu q$. This implies that $p_0 + p_1 \leq 1 + \mu q$, and $\min\{p_0, p_1\} \leq \frac{1+\mu q}{2}$. □

4.2.2 Random Shifts

In this section, we give quantitative bounds on uniform similarity between distributions created by random cyclic shifts of random strings.

Making a query into a cyclic shift of a string is equivalent to querying the original string in a position that is shifted, and thus, it is important to understanding what happens to a fixed set of q queries that undergoes different shifts. Our first lemma shows that a sufficiently large set of shifts of q queries can be partitioned into at most q^2 large sets, such that no two shifts in the same set intersect (in the sense that they query the same position).

Lemma 4.5. *Let Q be a subset of $[n]$ of size q, and let $Q_i \stackrel{\text{def}}{=} Q +_n i$ be its shift by i modulo n. Every $\mathcal{I} \subset [n]$ of size $t \geq 16q^4 \ln q$ admits a q^2-coloring $C : \mathcal{I} \to [q^2]$ with the following two properties:*

- *For all $i \neq j$ with $Q_i \cap Q_j \neq \emptyset$, we have $C(i) \neq C(j)$.*
- *For all $i \in [n]$, we have $|C^{-1}(i)| \geq n/(2q^4)$.*

Proof. Let $x \in [n]$. There are exactly q different indices i such that $x \in Q_i$. For every Q_i such that $x \in Q_i$, x is an image of a different $y \in Q$ after a cyclic shift. Therefore, each Q_i can intersect with at most $q(q-1)$ other sets Q_j.

Consider the following probabilistic construction of C. For consecutive $i \in \mathcal{I}$, we set $C(i)$ to be a random color in $[q^2]$ among those that were not yet assigned to sets Q_j that intersect Q_i. Each color $c \in [q^2]$ is considered at least t/q^2 times: each time c is selected it makes c not be considered for at most $q(q-1)$ other $i \in \mathcal{I}$. Each time c is considered, it is selected with probability at least $1/q^2$. By the Chernoff bound, the probability that a given color is selected less than $t/(2q^4)$ times is less than

$$\exp\left(-\frac{t}{q^4} \cdot \frac{1}{2^2} \cdot \frac{1}{2}\right) \leq \frac{1}{q^2}.$$

By the union bound, the probability of selecting the required coloring is greater than zero, so it exists. □

Fact 4.6. *Let n and k be integers such that $1 \leq k \leq n$. Then $\sum_{i=1}^{k} \binom{n}{i} \leq n^k$.*

The following lemma shows that random shifts of random strings are likely to result in uniformly similar distributions.

Lemma 4.7. *Let $n \in \mathbb{Z}_+$ be greater than 1. Let $k \leq n$ be a positive integer. Let x_i, $1 \leq i \leq k$, be uniformly and independently selected strings in Σ^n, where $2 \leq |\Sigma| \leq n$. With probability $2/3$ over the selection of x_i's, the distributions $\mathcal{S}_s(x_1)$, ..., $\mathcal{S}_s(x_k)$ are uniformly $\frac{1}{A}$-similar, for $A \stackrel{\text{def}}{=} \max\left\{\log_{|\Sigma|} \sqrt[6]{\frac{s}{400 \ln n}}, 1\right\}$.*

Proof. Let $p_{i,Q,\omega}$ be the probability of selecting a sequence $\omega \in \Sigma^{|Q|}$ from the distribution $S_s(x_i)|_Q$, where $Q \subseteq [n]$ and $1 \leq i \leq k$. We have to prove that with probability at least $2/3$ over the choice of x_i's, it holds that for every $Q \subseteq Q$ and every $\omega \in \Sigma^{|Q|}$,

$$(1 - |Q|/A) \cdot \max_{i=1,\ldots,k} p_{i,Q,\omega} \leq \min_{i=1,\ldots,k} p_{i,Q,\omega}.$$

The above inequality always holds when Q is empty or has at least A elements. Let $Q \subseteq [n]$ be any choice of queries, where $0 < |Q| < A$. By Fact 4.6, there are at most n^A such different choices of queries. Let $q \stackrel{\text{def}}{=} |Q|$. Note that $8q^4 \ln q \leq 8q^5 \leq 8A^5 \leq 8|\Sigma|^{5A} \leq 8 \cdot \frac{s}{400 \ln n} \leq s$. This implies that we can apply Lemma 4.5, which yields the following. We can partition all s shifts of Q over x_i that contribute to the distribution $S_s(x_i)|_Q$ into q^2 sets σ_j such that the shifts in each of the sets are disjoint, and each of the sets has size at least $s/(2q^4)$. For each of the sets σ_j, and for each $\omega \in \Sigma^q$, the probability that fewer than $(1 - \frac{q}{2A})|\sigma_j|/|\Sigma|^q$ shifts give ω is bounded by

$$\begin{aligned}
\exp\left(-\frac{1}{2} \cdot \left(\frac{q}{2A}\right)^2 \cdot \frac{|\sigma_j|}{|\Sigma|^q}\right) &\leq \exp\left(-\frac{s}{16q^2A^2|\Sigma|^q}\right) \\
&\leq \exp\left(-\frac{s}{16A^4|\Sigma|^A}\right) \\
&\leq \exp\left(-\frac{s}{16|\Sigma|^{5A}}\right) \\
&\leq \exp\left(-\frac{1}{16} \cdot \sqrt[6]{s \cdot (400 \ln n)^5}\right) \\
&\leq \exp\left(-9.2 \sqrt[6]{s(\ln n)^5}\right),
\end{aligned}$$

where the first bound follows from the Chernoff bound. Analogously, the probability that more than $(1 + \frac{q}{2A})|\sigma_j|/|\Sigma|^q$ shifts give ω is bounded by

$$\begin{aligned}
\exp\left(-\frac{1}{4} \cdot \left(\frac{q}{2A}\right)^2 \cdot \frac{|\sigma_j|}{|\Sigma|^q}\right) &\leq \exp\left(-\frac{s}{32q^2A^2|\Sigma|^q}\right) \\
&\leq \exp\left(-\frac{s}{32A^4|\Sigma|^A}\right) \\
&\leq \exp\left(-\frac{s}{32|\Sigma|^{5A}}\right) \\
&\leq \exp\left(-\frac{1}{32} \cdot \sqrt[6]{s \cdot (400 \ln n)^5}\right) \\
&\leq \exp\left(-4.6 \sqrt[6]{s(\ln n)^5}\right),
\end{aligned}$$

where the first inequality follows from the version of the Chernoff bound that uses the fact that $\frac{q}{2A} \leq \frac{1}{2} \leq 2e - 1$.

We now apply the union bound to all x_i, all choices of $Q \subseteq [n]$ with $|Q| < A$, all corresponding sets σ_j, and all settings of $\omega \in \Sigma^{|Q|}$ to bound the probability that $p_{i,Q,\omega}$ does not lie between $|\Sigma|^{-|Q|} \cdot (1 - \frac{q}{2A})$ and $|\Sigma|^{-|Q|} \cdot (1 + \frac{q}{2A})$. Assuming that $A > 1$ (otherwise, the lemma holds trivially), note first that

$$\begin{aligned} n \cdot n^A \cdot A^2 \cdot |\Sigma|^A &\leq n^{5A} \\ &\leq \exp(5A \ln n) \\ &\leq \exp\left(5|\Sigma|^A \ln n\right) \\ &\leq \exp\left(5\sqrt[6]{\frac{s(\ln n)^5}{400}}\right) \\ &\leq \exp\left(2.4 \cdot \sqrt[6]{s(\ln n)^5}\right). \end{aligned}$$

Our bound is

$$\exp\left(2.4 \cdot \sqrt[6]{s(\ln n)^5}\right) \cdot \left(\exp\left(-9.2 \cdot \sqrt[6]{s(\ln n)^5}\right) + \exp\left(-4.6 \cdot \sqrt[6]{s(\ln n)^5}\right)\right)$$
$$\leq \exp\left(-6.8 \cdot \sqrt[6]{s(\ln n)^5}\right) + \exp\left(-2.2 \cdot \sqrt[6]{s(\ln n)^5}\right) \leq 0.01 + 0.2 \leq 1/3.$$

Therefore, all $p_{i,Q,\omega}$ of interest lie in the desired range with probability at least 2/3. Then, we know that for any Q of size less than A, and any $\omega \in \Sigma^{|Q|}$, we have

$$\begin{aligned} \left(1 - \frac{|Q|}{A}\right) \cdot \max_{i=1,\ldots,k} p_{i,Q,\omega} &\leq \cdot \left(1 - \frac{|Q|}{A}\right) \cdot \left(1 + \frac{|Q|}{2A}\right) \cdot |\Sigma|^{-|Q|} \\ &= \left(1 - \frac{|Q|}{2A} - \frac{|Q|^2}{2A^2}\right) \cdot |\Sigma|^{-|Q|} \\ &\leq \left(1 - \frac{|Q|}{2A}\right) \cdot |\Sigma|^{-|Q|} \\ &\leq \min_{i=1,\ldots,k} p_{i,Q,\omega}. \end{aligned}$$

This implies that $\mathcal{S}_s(x_1), \ldots, \mathcal{S}_s(x_k)$ are uniformly $\frac{1}{A}$-similar with probability at least 2/3. □

4.2.3 Amplification of Uniform Similarity via Substitution Product

One of the key parts of our proof is the following lemma that shows that the substitution product of uniformly similar distributions amplifies uniform similarity.

Lemma 4.8. *Let $\mathcal{D}_a$ for $a \in \Sigma$, be uniformly α-similar distributions on $(\Sigma')^{n'}$. Let $\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{D}_a)_{a \in \Sigma}$. Let $\mathcal{E}_1, \ldots, \mathcal{E}_k$ be uniformly β-similar probability distributions on Σ^n, for some $\beta \in [0, 1]$. Then the k distributions $(\mathcal{E}_1 \circledast \mathcal{D}), \ldots, (\mathcal{E}_k \circledast \mathcal{D})$ are uniformly $\alpha\beta$-similar.*

Proof. Fix $t, t' \in [k]$, let X be a random sequence selected according to $\mathcal{E}_t \circledast \mathcal{D}$, and let Y be a random sequence selected according to $\mathcal{E}_{t'} \circledast \mathcal{D}$. Fix a set $S \subseteq [n \cdot n']$ of indices, and the corresponding sequence s of $|S|$ symbols from Σ'. To prove the lemma, it suffices to show that

$$\Pr[X|_S = s] \geq (1 - \alpha\beta|S|) \cdot \Pr[Y|_S = s], \tag{6}$$

since in particular the inequality holds for t that minimizes $\Pr[X|_S = s]$, and for t' that maximizes $\Pr[Y|_S = s]$.

Recall that each $(\mathcal{E}_j \circledast \mathcal{D})$ is generated by first selecting a string x according to $\mathcal{E}_j$, and then concatenating n blocks, where the i-th block is independently selected from $\mathcal{D}_{x_i}$. For $i \in [n]$ and $b \in \Sigma$, let $p_{i,b}$ be the probability of drawing from $\mathcal{D}_b$ a sequence that when used as the i-th block, matches s on the indices in S (if the block is not queried, set $p_{i,b} = 1$). Let q_i be the number of indices in S that belong to the i-th block. Since $\mathcal{D}_b$ for $b \in \Sigma$ are α-similar, for every $i \in [n]$, it holds that $(1 - \alpha q_i) \cdot \max_{b\in\Sigma} p_{i,b} \leq \min_{b\in\Sigma} p_{i,b}$. For every $i \in [n]$, define $\alpha_i^\star \stackrel{\text{def}}{=} \min_{b\in\Sigma} p_{i,b}$ and $\beta_i^\star \stackrel{\text{def}}{=} \max_{b\in\Sigma} p_{i,b}$. We thus have

$$(1 - \alpha q_i)\beta_i^\star \leq \alpha_i^\star. \tag{7}$$

The following process outputs `1` with probability $\Pr[Y|_S = s]$. Whenever we say that the process outputs a value, `0` or `1`, it also terminates. First, for every block $i \in [n]$, the process independently picks a random real r_i in $[0, 1]$. It also independently draws a random sequence $c \in \Sigma^n$ according to $\mathcal{E}_{t'}$. If $r_i > \beta_i^\star$ for at least one i, the process outputs `0`. Otherwise, let $Q = \{i \in [n] : \ r_i > \alpha_i^\star\}$. If $r_i \leq p_{i,c_i}$ for all $i \in Q$, the process outputs `1`. Otherwise, it outputs `0`. The correspondence between the probability of outputting `1` and $\Pr[Y|_S = s]$ directly follows from the fact that each of the random variables r_i simulates selecting a sequence that matches s on indices in S with the right success probability, i.e., p_{i,c_i}, and the fact that block substitutions are independent. The important difference, which we exploit later, is that not all symbols of c have always impact on whether the above process outputs `0` or `1`.

For every $Q \subseteq [n]$, let p'_Q be the probability that the above process selected Q. Furthermore, let $p''_{Q,c}$ be the conditional probability of outputting `1`, given that the process selected a given $Q \subseteq [n]$, and a given $c \in \Sigma^n$. It holds

$$\Pr\left[Y|_S = s\right] = \sum_{Q\subseteq[n]} p'_Q \cdot \mathbb{E}_{c\leftarrow\mathcal{E}_{t'}}\left[p''_{Q,c}\right].$$

Notice that for two different $c_1, c_2 \in \Sigma^n$, we have $p''_{Q,c_1} = p''_{Q,c_1}$ if $c_1|_Q = c_2|_Q$, since this probability only depends on the symbols at indices in Q. Thus, for $\tilde{c} \in \Sigma^{|Q|}$ we can define $\tilde{p}_{Q,\tilde{c}}$ to be equal to $p''_{Q,c}$ for any $c \in \Sigma$ such that $c|_Q = \tilde{c}$. We can now write

$$\Pr\left[Y|_S = s\right] = \sum_{Q\subseteq[n]} p'_Q \cdot \mathbb{E}_{\tilde{c}\leftarrow\mathcal{E}_{t'}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right],$$

and analogously,

$$\Pr\left[X|_S = s\right] = \sum_{Q\subseteq[n]} p'_Q \cdot \mathbb{E}_{\tilde{c}\leftarrow\mathcal{E}_{t}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right].$$

Due to the uniform β-similarity of $\mathcal{E}_{t'}$ and $\mathcal{E}_t$, we know that for every $Q \subset [n]$, the probability of selecting each $\tilde{c} \in \Sigma^{|Q|}$ from $\mathcal{E}_t|_Q$ is at least $(1 - \beta|Q|)$ times the probability of selecting the same $\tilde{c}$ from $\mathcal{E}_{t'}|_Q$. This implies that

$$\mathbb{E}_{\tilde{c}\leftarrow\mathcal{E}_{t}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right] \geq (1 - \beta|Q|) \cdot \mathbb{E}_{\tilde{c}\leftarrow\mathcal{E}_{t'}|_Q}\left[\tilde{p}_{Q,\tilde{c}}\right].$$

We obtain

$$\begin{aligned}
\Pr[Y|_S = s] - \Pr[X|_S = s] &= \sum_{Q\subseteq[n]} p'_Q \cdot \left(\mathbb{E}_{\tilde{c}\leftarrow\mathcal{E}_{t'|Q}}[\tilde{p}_{Q,\tilde{c}}] - \mathbb{E}_{\tilde{c}\leftarrow\mathcal{E}_{t|Q}}[\tilde{p}_{Q,\tilde{c}}]\right). \\
&\leq \sum_{Q\subseteq[n]} p'_Q \cdot \beta|Q| \cdot \mathbb{E}_{\tilde{c}\leftarrow\mathcal{E}_{t'|Q}}[\tilde{p}_{Q,\tilde{c}}] \\
&= \beta \cdot \sum_{Q\subseteq[n]} p'_Q \cdot |Q| \cdot \mathbb{E}_{c\leftarrow\mathcal{E}_{t'}}[p''_{Q,c}] \\
&= \beta \cdot \mathbb{E}_{c\leftarrow\mathcal{E}_{t'}}\left[\sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c} \cdot |Q|\right]. \qquad (8)
\end{aligned}$$

Fix now any $c \in \Sigma^n$ for which the process outputs `1` with positive probability. The expected size of Q for the fixed c, given that the process outputs `1`, can be written as

$$\mathbb{E}\left[|Q| \;\middle|\; \text{process outputs } \texttt{1}\right] = \frac{\sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c} \cdot |Q|}{\sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c}}$$

The probability that a given $i \in [n]$ belongs to Q for the fixed c, given that the process outputs `1` equals $\frac{p_{i,c_i}-\alpha^\star_i}{p_{i,c_i}}$. This follows from the two facts (a) if the process outputs `1` then r_i is uniformly distributed on $[0, p_{i,c_i}]$; and (b) $i \in Q$ if and only if $r_i \in (\alpha^\star_i, \beta^\star_i]$. We have

$$\frac{p_{i,c_i} - \alpha^\star_i}{p_{i,c_i}} \leq \frac{\beta^\star_i - \alpha^\star_i}{\beta^\star_i} \leq \frac{\alpha q_i \cdot \beta^\star_i}{\beta^\star_i} = \alpha q_i. \qquad (9)$$

By the linearity of expectation, the expected size of Q in this setting is at most $\sum_{i\in[n]} \alpha q_i = \alpha \cdot |S|$. Therefore,

$$\sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c} \cdot |Q| \leq \alpha \cdot |S| \cdot \sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c}. \qquad (10)$$

Note that the inequality trivially holds also for c for which the process always outputs `0`; both sides of the inequality equal 0.

By plugging (10) into (8), we obtain

$$\begin{aligned}
\Pr[Y|_S = s] - \Pr[X|_S = s] &\leq \beta \cdot \mathbb{E}_{c\leftarrow\mathcal{E}_{t'}}\left[\alpha \cdot |S| \cdot \sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c}\right] \\
&= \alpha\beta \cdot |S| \cdot \mathbb{E}_{c\leftarrow\mathcal{E}_{t'}}\left[\sum_{Q\subseteq[n]} p'_Q \cdot p''_{Q,c}\right] \\
&= \alpha\beta \cdot |S| \cdot \Pr[Y|_S = s].
\end{aligned}$$

This proves (6) and completes the proof of the lemma. □

4.3 Tools for Analyzing Edit Distance

This section provides tools to analyze how the edit distance changes under a under substitution product. We present two separate results with different guarantees, one is more useful for a large alphabet, the other for a small alphabet. The latter is used in the final step of reduction to binary alphabet.

4.3.1 Distance between random strings

The next bound is well-known, see also [CS75, BGNS99, Lue09]. We reproduce it here for completeness.

Lemma 4.9. *Let $x, y \in \Sigma^n$ be chosen uniformly at random. Then*

$$\Pr\left[\mathrm{LCS}(x,y) \geq 5n/\sqrt{|\Sigma|}\right] \leq e^{-5n/\sqrt{|\Sigma|}}.$$

Proof. Let $c \stackrel{\text{def}}{=} 5 > e^{1.5}$ and $t \stackrel{\text{def}}{=} cn/\sqrt{|\Sigma|}$. The number of potential alignments of size t between two strings of length n is at most $\binom{n}{t}^2 \leq (\frac{ne}{t})^{2t}$. Each of them indeed becomes an alignment of x, y (i.e. symbols that are supposed to align are equal) with probability at most $1/|\Sigma|^t$. Applying a union bound,

$$\Pr[\mathrm{LCS}(x,y) \geq t] \leq (\tfrac{ne}{t})^{2t}/|\Sigma|^t \leq (e^2 c^{-2}|\Sigma|)^t \cdot |\Sigma|^{-t} \leq e^{-t}. \qquad \square$$

4.3.2 Distance under substitution product (large alphabet)

We proceed to analyze how the edit distance between two strings, say $\mathrm{ed}(x,y)$, changes when we perform a substitution product, i.e. $\mathrm{ed}(x \circledast B, y \circledast B)$. The bounds we obtain are additive, and are thus most effective when the edit distance $\mathrm{ed}(x,y)$ is large (linear in the strings length). Furthermore, they depend on $\lambda_B \in [0,1]$, which denotes the maximum normalized LCS between distinct images of $B : \Sigma \to (\Sigma')^{n'}$, hence they are most effective when λ_B is small, essentially requiring a large alphabet Σ'.

Theorem 4.10. *Let $x, y \in \Sigma^n$ and $B : \Sigma \to (\Sigma')^{n'}$. Then*

$$n' \cdot \underline{\mathrm{ed}}(x,y) - 8nn'\sqrt{\lambda_B} \;\leq\; \underline{\mathrm{ed}}(x \circledast B, y \circledast B) \;\leq\; n' \cdot \underline{\mathrm{ed}}(x,y),$$

where $\lambda_B \stackrel{\text{def}}{=} \max\left\{\frac{\mathrm{LCS}(B(a),B(b))}{n'} : \; a \neq b \in \Sigma\right\}$.

Before proving the theorem, we state a corollary that will turn to be most useful. The corollary follows from Theorem 4.10 by letting $\Sigma' = \Sigma$, and using Lemma 4.9 together with a union bound over all pairs $B(a), B(b)$ (while assuming $n' \geq |\Sigma|$).

Corollary 4.11. *Assume $|\Sigma| \geq 2$ and $n' \geq |\Sigma|$ is sufficiently large (i.e. at least some absolute constant c'). Let $B : \Sigma \to (\Sigma)^{n'}$ be a random function, i.e. for each $a \in \Sigma$ choose $B(a)$ uniformly at random. Then with probability at least $1 - 2^{-n'/|\Sigma|}$, for all n and all $x, y \in \Sigma^n$,*

$$0 \;\leq\; n' \cdot \underline{\mathrm{ed}}(x,y) - \underline{\mathrm{ed}}(x \circledast B, y \circledast B) \;\leq\; O(nn'/|\Sigma|^{1/4}).$$

Proof of Theorem 4.10. By using the direct connection (5) between $\underline{\mathrm{ed}}(x,y)$ and $\mathrm{LCS}(x,y)$, it clearly suffices to prove

$$n' \cdot \mathrm{LCS}(x,y) \leq \mathrm{LCS}(x \circledast B, y \circledast B) \leq n' \cdot \mathrm{LCS}(x,y) + 4nn'\sqrt{\lambda_B}. \tag{11}$$

Throughout, we assume the natural partitioning of x, y into n blocks of length n'.

The first inequality above is immediate. Indeed, give an (optimal) alignment between x and y, do the following; for each (i,j) such that x_i is aligned with y_j, align the entire i-th block in $x \circledast B$

with the entire j-th block in $y \circledast B$. It is easily verified that the result is indeed an alignment and has size $n' \cdot \underline{\mathrm{ed}}(x, y)$.

To prove the second inequality above, fix an optimal alignment A between $x \circledast B$ and $y \circledast B$; we shall construct an $\hat{A}$ alignment for x, y in three stages, namely, first pruning A into A', then pruning it further into A'', and finally constructing $\hat{A}$. Define the *span* of a block b in either $x \circledast B$ or $y \circledast B$ (under the current alignment) to be the number of blocks in the other string to which it is aligned in at least one position (e.g. the span of block i in $x \circledast B$ is the number of blocks j for which at least one position p in block i satisfies that $A(p)$ is in block j.)

Now iterate the following step: "unalign" a block (in either $x \circledast B$ or $y \circledast B$) completely whenever its span is greater than $s \stackrel{\text{def}}{=} 2/\sqrt{\lambda_B}$. Let A' be the resulting alignment; its size is $|A'| \geq |A| - 4nn'/s$ because each iteration is triggered by a distinct block, the total span of all these blocks is at most $4n$, hence the total number of iterations is at most $4n/s$.

Next, iterate the following step (starting with A' as the current alignment): remove alignments between two blocks (one in $x \circledast B$ and one in $y \circledast B$) if, in one of the two blocks, at most $\lambda_B n'$ positions are aligned to the other block. Let A'' be the resulting alignment; its size is $|A''| \geq |A'| - ns \cdot \lambda_B n'$ because each iteration is triggered by a distinct pair of blocks, out of at most ns pairs (by the span bound above).

This alignment A'' has size $|A''| \geq |A| - 4nn'/s - nn's\lambda_B$. Furthermore, if between two blocks, say block i in $x \circledast B$ and block j in $y \circledast B$, the number of aligned positions is at least one, then this number is actually greater than $\lambda_B n'$ (by construction of A'') and thus $x[i] = y[j]$ (by definition of $\lambda_B n'$).

Finally, construct an alignment $\hat{A}$ between x and y, where initially, $\hat{A}(i) = \perp$ for all $i \in [n]$. Think of the alignment A'' as the set of aligned positions, namely $\{(p, q) \in [n] \times [n] : \ A''(p) = q\}$. Let $\mathrm{blk}_{x \circledast B}(p)$ denote the number of the block in $x \circledast B$ which contains p, and similarly for positions q in $y \circledast B$. Now scan A'', as a set of pairs, in lexicographic order. More specifically, initialize (p, q) to be the first edge in A'', and iterate the following step: assign $\hat{A}(\mathrm{blk}_{x \circledast B}(p)) = \mathrm{blk}_{y \circledast B}(q)$, and advance (p, q) according to the lexicographic order so that both coordinates now belong to new blocks, i.e. set it to be the next pair $(p', q') \in A''$ for which both $\mathrm{blk}_{x \circledast B}(p') > \mathrm{blk}_{x \circledast B}(p)$ and $\mathrm{blk}_{y \circledast B}(q') > \mathrm{blk}_{y \circledast B}(q)$. We claim that $\hat{A}$ is an alignment between x and y. To see this, consider the moment when we assign some $\hat{A}(i) = j$. Then the corresponding blocks in $x \circledast B$ and $y \circledast B$ contain at least one pair of positions that are aligned under A'', and thus, as argued above, $x[i] = y[j]$. In addition, all subsequent assignments of the form $\hat{A}(i') = j'$ satisfy that both $i' > i$ and $j' > j$. Hence $\hat{A}$ is indeed an alignment.

En route to bounding the size of $\hat{A}$, we claim that each iteration scans (i.e. advances the current pair by) at most n' pairs from A''. To see this, consider an iteration where we assign some $\hat{A}(i) = j$. Every pair $(p, q) \in A''$ that is scanned in this iteration satisfies that either $i = \mathrm{blk}_{x \circledast B}(p)$ or $j = \mathrm{blk}_{x \circledast B}(p)$. Each of these two requirements can be satisfied by at most n' pairs, and together at most $2n'$ pairs are scanned. By the fact that A'' is monotone, it can be easily verified that at least one of the two requirements must be satisfied by all scanned pairs, hence the total number of scanned pairs is at most n'.

Using the claim, we get that $|\hat{A}| \leq |A''|/n'$ (recall that each iteration also makes one assignment to $\hat{A}$). It immediately follows that

$$n' \cdot \mathrm{LCS}(x, y) \geq n' \cdot |\hat{A}| \geq |A''| \geq |A| - 4nn'/s - nn's\lambda_B = \mathrm{LCS}(x \circledast B, y \circledast B) - 4nn'\sqrt{\lambda_B},$$

which completes the proof of (11) and of Theorem 4.10. □

4.3.3 Distance under substitution product (any alphabet)

We give another analysis for how the edit distance between two strings, say $\mathrm{ed}(x, y)$, changes when we perform a substitution product, i.e. $\mathrm{ed}(x \circledast B, y \circledast B)$. The bounds we obtain here are multiplicative, and may be used as a final step of alphabet reduction (say, from a large alphabet to the binary one).

Theorem 4.12. *Let $B : \Sigma \to (\Sigma')^{n'}$, and suppose that (i) for every $a \neq b \in \Sigma$, we have*

$$\mathrm{LCS}(B_a, B_b) \leq \tfrac{15}{16} n';$$

and (ii) for every $a, b, c \in \Sigma$ (possibly equal), and every substring B' of (the concatenation) $B_b B_c$ that has length n' and overlaps each of B_b and B_c by at least $n'/10$, we have

$$\mathrm{LCS}(B_a, B') \leq 0.98 n'.$$

Then for all $x, y \in \Sigma^n$,

$$c_1 n' \cdot \underline{\mathrm{ed}}(x, y) \leq \underline{\mathrm{ed}}(x \circledast B, y \circledast B) \leq n' \cdot \underline{\mathrm{ed}}(x, y), \tag{12}$$

where $0 < c_1 < 1$ is an absolute constant.

Before proving the theorem, let us show that it is applicable for a random mapping B, by proving two extensions of Lemma 4.9. Unlike the latter, the lemmas below are effective also for small alphabet size.

Lemma 4.13. *Suppose $|\Sigma| \geq 2$ and let $x, y \in \Sigma^n$ be chosen uniformly at random. Then with probability at least $1 - |\Sigma|^{-l/8}$, the following holds: for every substring x' in x of length $l \geq 24$, and every length l substring y' in B_b, we have*

$$\mathrm{LCS}(x', y') \leq \tfrac{15}{16} l.$$

Proof. Set $\alpha \stackrel{\text{def}}{=} 1/16$. Fix l and the positions of x' inside x and of y' inside y. Then x' and y' are chosen at random from Σ^l, hence

$$\Pr[\mathrm{LCS}(x', y') \geq (1-\alpha)l] \leq \tbinom{l}{(1-\alpha)l}^2 |\Sigma|^{-(1-\alpha)l} \leq (\tfrac{e}{\alpha})^{2\alpha l} |\Sigma|^{-(1-\alpha)l} \leq |\Sigma|^{-l/4},$$

where the last inequality uses $|\Sigma| \geq 2$.

Now apply a union bound over all possible positions of x' and y' and all values of l. It follows that the probability that x and y contain length l substrings x' and y' (respectively) with $\mathrm{LCS}(x', y') \geq (1-\alpha)l$ is at most $|\Sigma|^3 \cdot |\Sigma|^{-l/4} \leq |\Sigma'|^{-l/8}$, if only l is sufficiently large. □

The next lemma is an easy consequence of Lemma 4.13. It follows by applying a union bound and observing that disjoint substrings of $B(a)$ are independent.

Lemma 4.14. *Let $B : \Sigma \to (\Sigma')^{n'}$ be chosen uniformly at random for $|\Sigma'| \geq 2$ and $n' \geq 1000 \log |\Sigma|$. Then with probability at least $1 - |\Sigma'|^{-\Omega(n')}$, B satisfies the properties (i) and (ii) described in Theorem 4.12.*

Proof of Theorem 4.12. The last inequality in (12) is straightforward. Indeed, whenever x_i is aligned against y_j, we have $x_i = y_j$ and $B(x_i) = B(y_j)$, hence we can align the corresponding blocks in $x \circledast B$ and $y \circledast B$. We immediately get that $\mathrm{LCS}(x \circledast B, y \circledast B) \geq n' \cdot \mathrm{LCS}(x, y)$.

Let us now prove the first inequality. Denote $R \stackrel{\text{def}}{=} \underline{\mathrm{ed}}(x \circledast B, y \circledast B)$, and fix a corresponding alignment between the two strings. The string $x \circledast B$ is naturally partitioned into n blocks of length n'. The total number of coordinates in $x \circledast B$ that are unaligned (to $y \circledast B$) is exactly $R/2$, which is $R/2n$ in an average block.

We now prune this alignment in two steps. First, "unaliagn" each block in $x \circledast B$ with at least $(nn'/100R) \cdot (R/2n) = n'/200$ unaligned coordinates. By averaging (or Markov's inequality), this step applies to at most $100R/nn'$-fraction of the n blocks.

Next, define the *gap* of a block in $x \circledast B$ to be the difference (in the positions) between the first and last positions in $y \circledast B$ that are aligned against a coordinate in $x \circledast B$. The second pruning step is to unalign every block in $x \circledast B$ whose gap is at least $1.01n'$. Every such block can be identified with a set of at least $n'/100$ unaligned positions in $y \circledast B$ (sandwiched inside the gap), hence these sets (for different blocks) are all disjoint, and the number of such blocks is at most $(R/2)/(n'/100) = 50R/n'$.

Now consider one of the remaining blocks (at least $n - 100R/n' - 50R/n'$ blocks). By our pruning, for each such block i we can find a corresponding substring of length n' in $y \circledast B$ with at least $n' - n'/200 - n'/100 > 0.98n'$ aligned pairs (between these two substrings). Using the property (ii) of B, the corresponding substring in $y \circledast B$ must have overlap of at least $0.9n'$ with some block of $y \circledast B$ (recall that $y \circledast B$ is also naturally partitioned into length n' blocks). Thus, for each such block i in $x \circledast B$ there is a corresponding block j in $y \circledast B$, such that these two blocks contain at least $0.9n' - 0.02n' = 0.88n'$ aligned pairs. By the property (i) of B, it follows that the corresponding coordinates in x and in y are equal, i.e. $x_i = y_j$. Observe that distinct blocks i in $x \circledast B$ are matched in this way to distinct blocks j in $y \circledast B$ (because the initial substrings in $y \circledast B$ were non-overlapping, and they each more than $n'/2$ overlap with a distinct block j).

It is easily verified that the above process gives an alignment between x and y. Recall that the number of coordinates in x that are not aligned in this process is at most $150R/n'$, hence $\underline{\mathrm{ed}}(x, y) \leq 300R/n'$, and this completes the proof. □

4.4 The Lower Bound

We now put all the elements of our proof together. We start by describing hard distributions, and then prove their properties. We also give a slightly more precise version of the lower bound for polynomial approximation factors in a separate subsection.

4.4.1 The Construction of Hard Distributions

We give a probabilistic construction for the hard distributions. We have two basic parameters, n which is roughly the length of strings, and α which is the approximation factor. We require that $2 < \alpha \ll n/\log n$. The strings length is actually smaller than n (for n large enough), but our query complexity lower bound hold also for length n, e.g., by a simple argument of padding by a fixed string.

We now define the hard distributions.

1. Fix an alphabet Σ of size $\lceil 5^2 \cdot 2^{16} \cdot \log_\alpha^4 n \rceil$.

2. Set:
   - $T \stackrel{\text{def}}{=} \lceil 1000 \cdot \log |\Sigma| \rceil$.
   - $\beta \stackrel{\text{def}}{=} \begin{cases} \alpha, & \text{if } \alpha < n^{1/3}, \\ \frac{n}{\alpha \ln n}, & \text{otherwise.} \end{cases}$
   - $s \stackrel{\text{def}}{=} \left\lceil 400\beta \ln n \cdot |\Sigma|^{12} \right\rceil$, thus $s = O(\beta \cdot \log n \cdot \log_\alpha^{48} n)$.
   - $B \stackrel{\text{def}}{=} \lceil 8\alpha s \cdot \log_\alpha n \rceil$, implying that $B = O(\alpha\beta \log n \cdot \log_\alpha^{49} n)$. Notice that $B < \frac{n}{T}$ for n large enough. If $\alpha < n^{1/3}$, then $B = \tilde{O}(n^{2/3})$. Otherwise, $\log_\alpha n \leq 3$, $\log |\Sigma| = O(1)$, and $B = o(n)$.
3. Select at random $|\Sigma|$ strings of length B, denoted x_a for $a \in \Sigma$.
4. Define $|\Sigma|$ corresponding distributions $\mathcal{D}_a$. For each $a \in \Sigma$, let
   $$\mathcal{D}_a \stackrel{\text{def}}{=} \mathcal{S}_s(x_a),$$
   and set
   $$\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{D}_a)_{a \in \Sigma}.$$
5. Define by induction on ia a collection of distributions $\mathcal{E}_{i,a}$ for $a \in \Sigma$. As the base case, set
   $$\mathcal{E}_{1,a} \stackrel{\text{def}}{=} \mathcal{D}_a.$$
   For $i > 1$, set
   $$\mathcal{E}_{i,a} \stackrel{\text{def}}{=} \mathcal{E}_{i-1,a} \circledast \mathcal{D}.$$
6. Let $i_\star \stackrel{\text{def}}{=} \left\lfloor \log_B \frac{n}{T} \right\rfloor$. Note that the distributions $\mathcal{E}_{i_\star,a}$ are defined on strings of length $B^{i_\star}$, which is is of course at most $\frac{n}{T}$, but due to an earlier observation, we also know that $i_\star \geq 1$, for n large enough.
7. Fix distinct $a_\star, b_\star \in \Sigma$. Let $\mathcal{F}_0 \stackrel{\text{def}}{=} \mathcal{E}_{i_\star,a_\star}$ and $\mathcal{F}_1 \stackrel{\text{def}}{=} \mathcal{E}_{i_\star,b_\star}$.
8. Pick a random mapping $R : \Sigma \to \{0,1\}^T$. Let $\mathcal{F}'_0 \stackrel{\text{def}}{=} \mathcal{F}_0 \circledast R$ and $\mathcal{F}'_1 \stackrel{\text{def}}{=} \mathcal{F}_1 \circledast R$. Note that the strings drawn from $\mathcal{F}'_0$ and $\mathcal{F}'_1$ are of length at most n.

Notice the construction is probabilistic only because of step #3 (the base strings x_a), and #8 (the randomized reduction to binary alphabet).

4.4.2 Proof of the Query Complexity Lower Bound

The next theorem shows that:

- Every two strings selected from the same distribution $\mathcal{F}_i$ are always close in edit distance.
- With non-zero probability (recall the construction is probabilistic), distribution $\mathcal{F}_0$ produces strings that are far, in edit distance, from strings produced by $\mathcal{F}_1$, yet distinguishing between these cases requires many queries.

Essentially the same properties hold also for $\mathcal{F}'_0$ and $\mathcal{F}'_1$.

Theorem 4.15. *Consider a randomized algorithm that is given full access to a string in Σ^n, and query access to another string in Σ^n. Let $2 < \alpha \leq o(n/\log n)$. If the algorithm distinguishes, with probability at least $2/3$, edit distance $\geq n/2$ from $\leq n/(4\alpha)$, then it makes*

$$\left(2 + \Omega\left(\frac{\log \alpha}{\log\log n}\right)\right)^{\max\left\{1, \Omega\left(\frac{\log n}{\log \alpha + \log\log n}\right)\right\}}$$

queries for $\alpha < n^{1/3}$, and $\Omega\left(\log \frac{n}{\alpha \ln n}\right)$ queries for $\alpha \geq n^{1/3}$. The bound holds even for $|\Sigma| = O(\log_\alpha^4 n)$.

For $\Sigma = \{0,1\}$, the same number of queries is required to distinguish edit distance $\geq c_1 n/2$ and $\leq c_1 n/(4\alpha)$, where $c_1 \in (0,1)$ is the constant from Theorem 4.12.

Proof. We use the construction described in Section 4.4.1. Recall that $i_\star \geq 1$, for n large enough, and that $i_\star \leq \log_B n$.

Let $F : \Sigma \to \Sigma^B$ be defined as $F(a) \stackrel{\text{def}}{=} x_a$ for every $a \in \Sigma$. We define $y_{i,a}$ inductively. Let $y_{1,a} \stackrel{\text{def}}{=} x_a$ for every $a \in \Sigma$, then for $i > 1$ define $y_{i,a} \stackrel{\text{def}}{=} y_{i-1,a} \circledast F$.

We now claim that for every word z with non-zero probability in $\mathcal{E}_{i,a}$ for $a \in \Sigma$, we have

$$\frac{\mathrm{ed}(z, y_{i,a})}{B^i} \leq \frac{i \cdot 2 \cdot s}{B} \leq \frac{i}{4\alpha \log_\alpha n}.$$

This follows by induction on i, since every rotation by s can be "reversed" with at most s insertions and s deletions. In particular,

$$\frac{\mathrm{ed}(z, y_{i_\star,a})}{B^{i_\star}} \leq \frac{\log_B n}{4\alpha \log_\alpha n} = \frac{\log \alpha}{4\alpha \log B} \leq \frac{1}{4\alpha},$$

where the last inequality is because $\alpha \leq B$.

It follows from Lemma 4.9 and the union bound that with probability

$$1 - |\Sigma|^2 \cdot e^{-5B/\sqrt{|\Sigma|}} \geq 1 - |\Sigma|^2 \cdot e^{-5|\Sigma|} \geq 1 - e^{-3|\Sigma|} \geq 1 - e^{-3} \geq 2/3$$

(over the choice of F, i.e. x_a for $a \in \Sigma$), that for all $a \neq b \in \Sigma$ we have $\mathrm{LCS}(x_a, x_b) \leq 5B/\sqrt{|\Sigma|}$, that is, the value corresponding to $\sqrt{\lambda_B}$ in Lemma 4.10 is at most $\sqrt{5/\sqrt{|\Sigma|}} \leq 1/(16 \log_\alpha n)$. We assume henceforth this event occurs. Then by Lemma 4.10 and induction, we have that for all $a \neq b$,

$$\underline{\mathrm{ed}}(y_{i,a}, y_{i,b}) \geq B^i \left(2 - \frac{i}{2\log_\alpha n}\right)$$

which gives

$$\begin{aligned}
\mathrm{ed}(y_{i_\star,a_\star}, y_{i_\star,b_\star}) &\geq \frac{1}{2}\underline{\mathrm{ed}}(y_{i_\star,a_\star}, y_{i_\star,b_\star}) \geq B^{i_\star}\left(1 - \frac{i_\star}{4\log_\alpha n}\right) \geq B^{i_\star}\left(1 - \frac{\log\alpha}{4\log B}\right) \\
&\geq B^{i_\star}\left(1 - \frac{1}{4}\right) = \frac{3}{4}B^{i_\star}.
\end{aligned}$$

Consider now an algorithm that is given full access to the string $y_{i_\star,a_\star}$ and query access to some other string z. If z comes from $\mathcal{F}_0 = \mathcal{E}_{i_\star,a_\star}$, then $\mathrm{ed}(y_{i_\star,a_\star}, z) \leq \frac{B^{i_\star}}{4\alpha}$. If z comes from $\mathcal{F}_1 = \mathcal{E}_{i_\star,b_\star}$, then $\mathrm{ed}(y_{i_\star,a_\star}, z) \geq \frac{3}{4}B^{i_\star} - \frac{1}{4\alpha}B^{i_\star} \geq \frac{1}{2}B^{i_\star}$ by the triangle inequality.

We now show that the algorithm has to make many queries to learn whether z is drawn from $\mathcal{F}_0$ or from $\mathcal{F}_1$. By Lemma 4.7, with probability at least 2/3 over the choice of x_a's, $\mathcal{E}_{1,a}$'s are uniformly $\frac{1}{A}$-similar, for

$$A \stackrel{\text{def}}{=} \log_{|\Sigma|} \sqrt[6]{\frac{s}{400 \ln B}} \geq \log_{|\Sigma|} \sqrt[6]{\beta \cdot |\Sigma|^{12}} = 2 + \frac{\log \beta}{6 \log |\Sigma|}.$$

Note that both the above statement regarding $\frac{1}{A}$-similarity as well as the earlier requirement that $\mathrm{LCS}(x_a, x_b)$ be small for all $a \neq b$, are satisfied with non-zero probability.

Observe that $\log |\Sigma| = \Theta(1 + \log(\frac{\log n}{\log \alpha}))$. For $\alpha < n^{1/3}$,

$$A = 2 + \Omega\left(\frac{\log \alpha}{1 + \log\left(\frac{\log n}{\log \alpha}\right)}\right) = 2 + \Omega\left(\frac{\log \alpha}{\log \log n}\right).$$

For $\alpha \geq n^{1/3}$,

$$A \geq 2 + \Omega\left(\frac{\log \frac{n}{\alpha \ln n}}{1 + \log\left(\frac{\log n}{\log \alpha}\right)}\right) \geq \Omega\left(\log \frac{n}{\alpha \ln n}\right),$$

where the last transition follows since $\frac{\log n}{\log \alpha} = \Theta(1)$ and $\alpha = o(n/\log n)$.

By using Lemma 4.8 over $\mathcal{E}_{i,a}$'s, we have that $\mathcal{E}_{i,a}$'s are uniformly $\frac{1}{A^i}$-similar. It now follows from Lemma 4.4 that an algorithm that distinguishes whether its input z is drawn from $\mathcal{F}_0 = \mathcal{E}_{i_\star, a_\star}$ or from $\mathcal{F}_1 = \mathcal{E}_{i_\star, b_\star}$ with probability at least 2/3, must make at least $A^{i_\star}/3$ queries to z. Consider first the case of $\alpha < n^{1/3}$. We have $i_\star = \Omega\left(\frac{\log n}{\log B}\right) = \Omega\left(\frac{\log n}{\log \alpha + \log \log n}\right)$. The number of queries we obtain is

$$\left(2 + \Omega\left(\frac{\log \alpha}{\log \log n}\right)\right)^{\max\left\{1, \Omega\left(\frac{\log n}{\log \alpha + \log \log n}\right)\right\}}.$$

For $\alpha \geq n^{1/3}$ we have $i_\star \geq 1$, and the algorithm must make $\Omega\left(\log \frac{n}{\alpha \ln n}\right)$ queries. This finishes the prove of the first part of the theorem, which states a lower bound for an alphabet of size $\Theta(\log_\alpha^4 n)$.

For the second part of the theorem regarding alphabet $\Sigma = \{0, 1\}$, we use the distributions from the first part, but we employ the mapping $\mathcal{R} : \Sigma \to \{0,1\}^T$ to replace every symbol in Σ with a binary string of length T. Lemma 4.14 and Theorem 4.12 state that if R is chosen at random, then with non-zero probability, R preserves (normalized) edit distance up to a multiplicative c_1. Using such a mapping R and α/c_1 instead of α in the entire proof, we obtain the desired gap in edit distance between $\mathcal{F}'_0$ and $\mathcal{F}'_1$. The number of required queries remains the same after the mapping, because every symbol in a string obtained from $\mathcal{F}'_0$ or $\mathcal{F}'_1$ is a function of a single symbol from a string obtained from $\mathcal{F}_0$ or $\mathcal{F}_1$, respectively. An algorithm using few queries to distinguish $\mathcal{F}'_0$ from $\mathcal{F}'_1$ would therefore imply an algorithm with similar query complexity to distinguish $\mathcal{F}_0$ from $\mathcal{F}_1$, which is not possible. □

4.4.3 A More Precise Lower Bound for Polynomial Approximation Factors

We now state a more precise statement that specifies the exponent for polynomial approximation factors.

Theorem 4.16. *Let λ be a fixed constant in $(0,1)$. Let t be the largest positive integer such that $\lambda \cdot t < 1$.*

Consider an algorithm that is given a string in Σ^n, and query access to another string in Σ^n. If the algorithm correctly distinguishes edit distance $\geq n/2$ and $\leq n/(4n^\lambda)$ with probability at least $2/3$, then it needs $\Omega(\log^t n)$ queries, even for $|\Sigma| = O(1)$.

For $\Sigma = \{0,1\}$, the same number of queries is required to distinguish edit distance $\geq c_1 n/2$ and $\leq c_1 n/(4n^\lambda)$, where $c_1 \in (0,1)$ is the constant from Theorem 4.12.

Proof. The proof is a modification of the proof of Lemma 4.15. We reuse the same construction with the following differences:

- We set $\alpha \stackrel{\text{def}}{=} n^\lambda$. This is our approximation factor.
- We set $\beta \stackrel{\text{def}}{=} n^{\frac{1}{2}\left(\frac{1}{t}-\lambda\right)}$. This is up to a logarithmic factor the shift at every level of recursion

T, s, B, $|\Sigma|$ are defined in the same way as functions of α and β. Note that $B = \Theta\left(n^{\frac{1}{2}\left(\frac{1}{t}+\lambda\right)} \log n\right)$ and $T = \Theta(1)$. This implies that for sufficiently large n, $i_\star = \lfloor \log_B \frac{n}{T} \rfloor = t$, because $B^t = \tilde{\Theta}\left(n^{\frac{1+\lambda t}{2}}\right) = o(n)$, and $B^{t+1} = \tilde{\Theta}\left(n^{\frac{1}{2}+\frac{1}{2t}+\frac{\lambda(t+1)}{2}}\right) = \tilde{\Omega}\left(n^{1+\frac{1}{2t}}\right) = \omega(n)$.

As in the proof of Lemma 4.15, we achieve the desired separation in edit distance. Recall that the number of queries an algorithm must make is $\Omega(A^{i_\star})$, where

$$A \geq 2 + \frac{\log \beta}{6 \log |\Sigma|} = \Omega(\log n).$$

Thus, the number of required queries equals $\Omega(\log^t n)$. □

References

[ACCL07] Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Estimating the distance to a monotone function. *Random Structures and Algorithms*, 31:371–383, 2007. Previously appeared in RANDOM'04.

[AJP10] Alexandr Andoni, T.S. Jayram, and Mihai Pătraşcu. Lower bounds for edit distance and product metrics via Poincaré-type inequalities. *Accepted to ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, 2010.

[AK10] Alexandr Andoni and Robert Krauthgamer. The computational hardness of estimating edit distance. *SIAM Journal on Computing*, 39(6):2398–2429, 2010. Previously appeared in FOCS'07.

[AN10] Alexandr Andoni and Huy L. Nguyen. Near-tight bounds for testing Ulam distance. *Accepted to ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, 2010.

[AO09] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 199–204, 2009.

[BEK+03] Tuğkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 316–324, 2003.

[BES06] Tuğkan Batu, Funda Ergün, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 792–801, 2006.

[BFC08] Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(28):486–496, 2008.

[BGNS99] R. A. Baeza-Yates, R. Gavaldà, G. Navarro, and R. Scheihing. Bounding the expected length of longest common subsequences and forests. *Theory Comput. Syst.*, 32(4):435–452, 1999.

[BJKK04] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 550–559, 2004.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[CM07] Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Trans. Algorithms*, 3(1), 2007. Special issue on SODA'02.

[Cor03] Graham Cormode. *Sequence Distance Embeddings*. Ph.D. Thesis, University of Warwick. 2003.

[CPSV00] Graham Cormode, Mike Paterson, Suleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 197–206, 2000.

[CS75] V. Chvatal and D. Sankoff. Longest common subsequences of two random sequences. *J. Appl. Probability*, 12:306–315, 1975.

[EKK+00] Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld, and Manesh Viswanathan. Spot-checkers. *J. Comput. Syst. Sci.*, 60(3):717–751, 2000.

[Gus97] Dan Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, Cambridge, 1997.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[IM03] Piotr Indyk and Jiří Matoušek. Low distortion embeddings of finite metric spaces. *CRC Handbook of Discrete and Computational Geometry*, 2003.

[Ind01] Piotr Indyk. Algorithmic aspects of geometric embeddings (tutorial). In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 10–33, 2001.

[IW05] Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. *Proceedings of the Symposium on Theory of Computing (STOC)*, 2005.

[KN06] Subhash Khot and Assaf Naor. Nonembeddability theorems via Fourier analysis. *Math. Ann.*, 334(4):821–852, 2006. Preliminary version appeared in FOCS'05.

[KOR00] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput.*, 30(2):457–474, 2000. Preliminary version appeared in STOC'98.

[KR06] Robert Krauthgamer and Yuval Rabani. Improved lower bounds for embeddings into L_1. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1010–1017, 2006.

[Lev65] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals (in russian). *Doklady Akademii Nauk SSSR*, 4(163):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.

[LMS98] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, 1998.

[Lue09] G. S. Lueker. Improved bounds on the average length of longest common subsequences. *J. ACM*, 56(3):1–38, 2009.

[Mat07] Jiří Matoušek. Collection of open problems on low-distortion embeddings of finite metric spaces. March 2007. Available online. Last access in August, 2007.

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

[MS00] S. Muthukrishnan and Cenk Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 416–424, 2000.

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

[OR07] Rafail Ostrovsky and Yuval Rabani. Low distortion embedding for edit distance. *J. ACM*, 54(5), 2007. Preliminary version appeared in STOC'05.

[Sah08] Süleyman Cenk Sahinalp. Edit distance under block operations. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008.

[SS02] Michael Saks and Xiaodong Sun. Space lower bounds for distance approximation in the data stream model. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 360–369, 2002.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168 – 173, 1974.