Improved Algorithms for Edit Distance and LCS: Beyond Worst Case

Mahdi Boroujeni* Masoud Seddighin† Saeed Seddighin‡

Abstract

Edit distance and longest common subsequence are among the most fundamental problems in combinatorial optimization. Recent developments have proven strong lower bounds against subquadratic time solutions for both problems. Moreover, the best approximation factors for subquadratic time solutions have been limited to 3 for edit distance and super constant for longest common subsequence. Improved approximation algorithms for these problems[1] are some of the biggest open questions in combinatorial optimization. In this work, we present improved algorithms for both edit distance and longest common subsequence. The running times are truly subquadratic, though we obtain $1+o(1)$ approximate solutions for both problems if the input satisfies a mild condition. In this setting, first, an adversary chooses one of the input strings. Next, this string is perturbed by a random procedure, and then the adversary chooses the second string **after observing the perturbed one**.

1 Introduction

Distance similarity measures are classic and well-studied problems in computer science. The notable examples are *edit distance* (ED) and *longest common subsequence* (LCS), which have been proposed to capture the notion of similarity for strings. These problems find their applications in various contexts, such as computational biology, text processing, compiler optimization, data analysis, image analysis, etc. Therefore, both edit distance and longest common subsequence have been subject to a plethora of studies in the past few decades (see *e.g.* [2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 42, 44, 45, 48, 53, 54, 55, 56]).

Edit distance is defined on two strings s and $\bar{s}$ and seeks the smallest number of character insertions, character deletions, and character substitutions to transform s into $\bar{s}$. While in edit distance the goal is to make a transformation, longest common subsequence asks for the largest string that appears as a subsequence in both s and $\bar{s}$. Both problems, almost universally, have been used as textbook examples of dynamic programming.

Although the quadratic time solutions for ED and LCS have been known for many decades, no substantial improvement was made to the algorithms. Apart from the logarithmic improvements of [45], not much was known for any of the problems in the worst case. The breakthroughs of Backurs and Indyk [13], Abboud, Backurs, and Williams [2], and Bringmann and Kunnemann [23] explain this failure. They show that a truly subquadratic time solution for either ED or LCS refutes a widely believed conjecture[2].

Approximate algorithms have also been persistently studied for the two problems. For edit distance, a series of works [44], [15], [16], and [12] improve the approximation factor culminating in the seminal work of Andoni, Krauthgamer, and Onak [11] that finally obtains a polylogarithmic approximation in near-linear time. Recent developments also answer the long-standing open question of whether a constant approximation factor can be obtained for ED in truly subquadratic time: first a quantum solution [17], next a breakthrough algorithm [24] obtaining constant factor with a classic computer, and finally near-linear time solutions for far strings [41, 50][3]. All these algorithms are based on a simple concept: edit distance satisfies triangle inequality, and thus, by losing a constant factor in the approximation, we can imply bounds on the edit distance between two strings whose distances from a fixed string are known. Although the hope is to improve the approximation factor to $2+\epsilon$ [40, 52], it seems quite unlikely that triangle inequality alone can be used to obtain significantly better bounds. The question of whether this bound can be improved beyond the current techniques is often discussed and raised as an important open question by experts in the community [6, 40, 49]. Our work makes a step forward in this direction.

LCS has also received tremendous attention in recent years [1, 4, 26, 35, 51, 52]. Only trivial solutions were known for LCS until very recently: a 2 approximate solution when the alphabet is 0/1 and an $O(\sqrt{n})$ ap-

*Sharif University of Technology

†Institute for Research in Fundamental Sciences (IPM), School of Computer Science

‡Harvard University

[1] $1+\epsilon$ approximation for ED or subpolynomial approximation for LCS in truly subquadratic time.

[2] The strong exponential time hypothesis SETH states that SAT cannot be solved in time $O(2^{(1-\epsilon)n})$ for any constant $\epsilon > 0$.

[3] The running times are near-linear if the distance between the two strings is $\Omega(n)$.

proximate solution for general alphabets in linear time. Both these bounds are recently improved by Hajiaghayi *et al.* [35] and Rubinstein and Song [51]. Also, a significant improvement to the approximation algorithms of LCS is given by Rubinstein *et al.* [52]. Despite recent works, our understanding of approximation algorithms for LCS is limited. Also, quite a few hardness results are obtained for both edit distance and LCS [1, 4, 26]. Still, it remains a mystery to what extent we can approximate LCS in truly subquadratic time with randomized algorithms.

In this work, we present approximate solutions for both problems that run in truly subquadratic time. We prove that the approximation factor of our algorithms is $1+o(1)$ if the instances adhere to our random setting. In other words, if the input is not delicately engineered to mislead our algorithm, then we expect to obtain a $1+o(1)$ approximation of the solution. In our setting, after an adversary chooses a string $\hat{s}$, the characters of $\hat{s}$ are displaced by a random procedure to generate the first string s. The adversary then chooses the second string $\overline{s}$ after observing s (along with changes made to $\hat{s}$).

We note that our improvements are not obtained because the random displacement changes the structure of the solution; if an instance $(t, \bar{t})$ is hard, the adversary can choose $\hat{s} = t$ as the first string, and after observing the changes made to $\hat{s}$, modify $\bar{t}$ to obtain $\overline{s}$ in a way that the solution of $(t, \bar{t})$ remains a valid solution for $(s, \overline{s})$ (by rearranging the order of the characters). Rather, our improvement is based on the fact that algorithm-specific bad-instances are ruled out in our random setting. This can be seen as practical solutions for typical (and not necessarily worst-case) instances of ED and LCS.

1.1 Related Work. Closely related to the present paper is the work of Kuszmaul [43] wherein the author presents a constant factor approximation algorithm to find the edit distance between a pseudo-random string s and an adversarially constructed string $\overline{s}$ in subquadratic time. The motivation behind their setting is the case that the first string is generated independently from a uniform distribution for which their running time improves to (almost) linear. We remark that *i.i.d.* generated s is a special case of our random setting[4], and thus, our algorithm immediately obtains a $1+o(1)$ approximation for this case. It is easy to verify that if the first string s is relatively balanced[5] the algorithm of Kuszmaul *et al.* obtains constant factor approximation in our setting in near linear time.

Furthermore, Andoni and Krauthgamer [10] study the smoothed complexity of edit distance where two binary strings are given as input by an adversary. Each character then is perturbed with a probability p with an additional condition that characters of a fixed longest common subsequence change similarly in both strings. Their work obtains an $O(n^{1+\epsilon})$ time algorithm, which approximates the edit distance between perturbed strings within a constant factor when p is constant. One stark difference between our setting and the setting of Andoni and Krauthgamer [10] is that our setting only modifies one of the strings, and as such, edge cases can still be constructed via a careful choice of the second string. In the setting of Andoni and Krauthgamer [10], however, random perturbation is applied to both strings, and therefore, many natural cases are ruled out. For instance, in their setting, the longest common subsequence of the two perturbed strings is at least an $\Omega(p)$ fraction of the length of the strings with high probability. In comparison, our setting is much stronger: one can easily construct instances for which the solution size is as small as 0 or as large as n for both ED and LCS. Also, characters of the solution may correspond to any subset of positions in the two strings.

To the best of our knowledge, no smoothed algorithm has been proposed for longest common subsequence, and the current best bounds are limited to those given for worst-case scenarios. For edit distance, there are several works that obtain constant factor approximation in subquadratic or near-linear time in the worst-case or under natural assumptions [10, 17, 41, 43, 50]. However, prior to our work, an approximation factor of $1+o(1)$ for edit distance was not known even under additional assumptions.

1.2 Preliminaries, Results, and Techniques. We start by formally defining the problems and our setting. In both ED and LCS, two strings are given as input. We denote the input strings by s and $\overline{s}$, respectively. Also, for simplicity, we assume w.l.o.g that both of the strings are of the same size n, though this assumption is not an inherent barrier to any of our algorithms. The edit distance between two strings is defined as follows:

[4]If we rearrange the characters of an *i.i.d.* drawn $\hat{s}$, the resulting string s would still be *i.i.d.* drawn from the same distribution.

[5]The number of the occurrences of all symbols are roughly the same

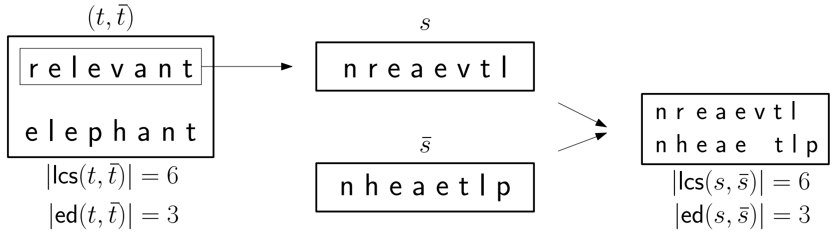

$(t, \bar{t})$
relevant
elephant
$|\mathsf{lcs}(t,\bar{t})| = 6$
$|\mathsf{ed}(t,\bar{t})| = 3$
s
nreaevtl
$\bar{s}$
nheaetlp
nreaevtl
nheae tlp
$|\mathsf{lcs}(s,\bar{s})| = 6$
$|\mathsf{ed}(s,\bar{s})| = 3$

Figure 1: In our setting, if an instance $(t, \bar{t})$ is hard, the adversary can choose $\hat{s} = t$ as the first string, and after observing the changes made to $\hat{s}$, modify $\bar{t}$ to obtain $\bar{s}$ in a way that the solution of $(t, \bar{t})$ remains a valid solution for $(s, \bar{s})$.

edit distance

input: Two strings s and $\bar{s}$, both of size n.

solution: The smallest set of operations that we need to perform on s to transform it into $\bar{s}$. We are allowed to (i) add a character at any position, (ii) remove a character, and (iii) change a character. Each operation incurs a cost of 1.

For instance, if s = elephant and $\bar{s}$ = relevant, we can transform s into $\bar{s}$ by performing three operations as follows:

$$\begin{aligned} \text{elephant} &\xRightarrow{\text{insert 'r' at position 0}} \text{relephant} \\ &\xRightarrow{\text{replace 'p' with 'v' at position 4}} \text{relevhant} \\ &\xRightarrow{\text{delete 'h' at position 5}} \text{relevant.} \end{aligned}$$

Therefore, $|\mathsf{ed}(\text{elephant}, \text{relevant})| = 3$. The other measure that we are interested in is the longest common subsequence. For two strings s and $\bar{s}$, $\mathsf{lcs}(s, \bar{s})$ is defined as the longest sequence of (not necessarily consecutive) characters that the two strings share.

longest common subsequence

input: Two strings s and $\bar{s}$, both of size n.

solution: The longest string $\bar{s}$ that appears as a subsequence in both s and $\bar{s}$.

As an example, when s = elephant and $\bar{s}$ = relevant, both strings have eleant as a subsequence and thus $|\mathsf{lcs}(\text{elephant}, \text{relevant})| = 6$.

In our setting, we assume that the adversary first provides a string $\hat{s}$ and s is constructed from $\hat{s}$ via a random procedure. Our main focus is on the case that s is a random displacement of $\hat{s}$, or in other words, characters of $\hat{s}$ are reshuffle by a permutation uniformly drawn from the space of all permutations.

After s is realized and observed by the adversary, she provides another string $\bar{s}$ as the second input of the problem. Notice that this comes with full knowledge of s, $\hat{s}$, and our algorithm[6]. An example of this process is shown in Figure 2.

Before we state our results and techniques, let us further explain how our setting compares to the worst-case and random settings. First, we would like to point out that our setting does not oversimplify the problem. To see this, consider a pair of strings t and $\bar{t}$ for which computing/approximating LCS is desirable. If we start with $\hat{s} = t$ in our setting setting, s would consist of characters of s shuffled by a random permutation. Now, with an appropriate displacement of characters of $\bar{t}$, one can construct a string $\bar{s}$ such that the solution of $(t, \bar{t})$ remains a (not necessarily optimal) solution (but shuffled) for s and $\bar{s}$.

Our setting deviates from the average or random setting in which both strings are generated/modified with random procedures. For instance, if the characters of each string are randomly drawn from a certain distribution over the alphabet, then we expect that the LCS or ED of the two strings follows from some pattern. In particular, for LCS, we expect that the positions of solution characters are distributed almost evenly over the strings. Indeed this is not the case for our setting as the adversary chooses $\bar{s}$ after observing s. In other words, if we show the longest common subsequence of the two strings by a matching from characters of s to the characters of $\bar{s}$, the position of the edges in the solution is as unpredictable as the one for the worst-case setting.

Our framework is inspired by recent developments for edit distance [17, 24] and LCS [52]. In particular, the algorithm of [24] is based on a 3-step framework[7]:

Step 0 (window-compatible solutions): In this step, they construct a set of windows for both strings, namely W_s and $W_{\bar{s}}$, with the promise that if the edit distances between the windows of W_s and $W_{\bar{s}}$ are available, one can recover a $1 + o(1)$ approximation

[6]Albeit, the adversary is not aware of our random bits.

[7]The reader can find more details about this framework in a blog post by Rubinstein [49].

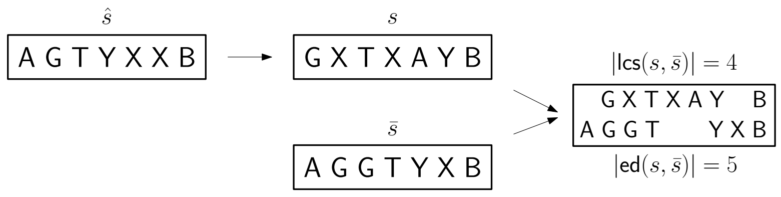


Figure 2: Our setting is is illustrated in this figure. First, an adversary chooses a string $\hat{s}$ of size n. Then, s is constructed by a random displacement of the characters of $\hat{s}$. Next, the adversary chooses another string $\bar{s}$ of size n. The LCS or ED for s and $\bar{s}$ is desired.

of $|\mathsf{ed}(s,\bar{s})|$ in subquadratic time. This step gives no improvement nor loss in the running time. That is, if we naively compute the solution for every pair of windows via the classic algorithm, the total running time would be quadratic. This step is useful in the following sense: the task of approximating the edit distance between two long strings reduces to the task of approximating edit distance between many smaller windows. The construction is shown in Figure 3.

Step 1 (sparsification via triangle inequality): The main improvement in the running time comes in this step. Due to Step 0, all one needs to approximate the solution for the two strings is to compute the edit distances of the windows. Triangle inequality is the key to this improvement. While computation of edit distance between all pairs gives a quadratic running time, one can use the following observation to compute the edit distance for only a few[8] window pairs: If $|\mathsf{ed}(w_i, w_j)| \leq d_1$ and $|\mathsf{ed}(w_i, w_k)| \leq d_2$ then $d_1 + d_2$ is a clear upper bound on the edit distance between w_j and w_k. Although this just gives an upper bound, Chakraborty *et al.* [24] show that except for $(|W_s||W_{\bar{s}}|)^{1-\epsilon}$[9] many pairs, the derived upper bound is a constant approximation of the actual distance. This step runs in truly subquadratic time and estimates the distances for almost all window pairs. Note that in this step, a constant factor loss in the approximation is inevitable due to the application of triangle inequality. An LCS analogue of the triangle inequality, called *birthday triangle inequality*, provides a solution for LCS [52]. All that remains is to devise a method to tolerate the error for the few undiscovered[10] distances.

Step 2 (discovering the edit distance between the remaining window pairs): While the number of undiscovered distances for window pairs is asymptotically smaller than the number of window pairs, detecting such pairs may still require quadratic time. Chakraborty *et al.* [24] make the following observation to address this issue: Since the optimal solution (as well any approximately optimal solution) incorporates a class of window pairs that adhere to certain structures, we only need to guarantee that most of the distances between window pairs of each class are discovered. In other words, we can tolerate some error so long as in each class, the error incurred by the undiscovered distances is negligible in comparison to the overall size of the solution. This then leads to the question of "whether we can guarantee that we discover most of the distances in each class of window-pairs correctly?" to which Chakraborty *et al.* [24] give a positive answer. In their algorithm, first, the undiscovered distances are detected for a sampled set of windows, and then neighborhoods close to the undiscovered distances are reexamined via the classic algorithm for edit distance. This is shown in Figure 5.

Both of our algorithms for edit distance and LCS are inspired by this framework. We too, make use of Step 0 in order to break the problem down to smaller windows. Our main novelty is an alternative sparsification algorithm to replace Step 1, explained later in the section. Indeed, our aim is to obtain a $1 + o(1)$ approximation algorithms for ED and LCS, and triangle inequality is too weak for this purpose. Current ED algorithms obtain approximation factor 3 [17, 24], and the belief is that the best bound that can be obtained via this technique is 2 [40, 52].

Our sparsification deviates from previous work in the following sense: all the previous algorithms [17, 24, 52] approximate the solutions for the windows in Step 1 and prove that except a few pairs, the rest of the values are estimated approximately correctly. For the remaining pairs, the estimated values are **worse** than the actual ones; for ED the values are larger, and for LCS, the values are smaller. Therefore, their only worry is that an optimal solution should remain approximately optimal by the estimations, which is addressed in Step 2. In our sparsification, we may miscalculate the solution for one pair of windows as a much smaller or a much

[8]Truly sublinear in terms of the number of window pairs.

[9]In this framework $1 > \epsilon > 0$ is always a constant number. This value, however, may vary depending on the sparsification technique.

[10]By undiscovered distances, we mean window pairs whose distance is overestimated by our algorithm.

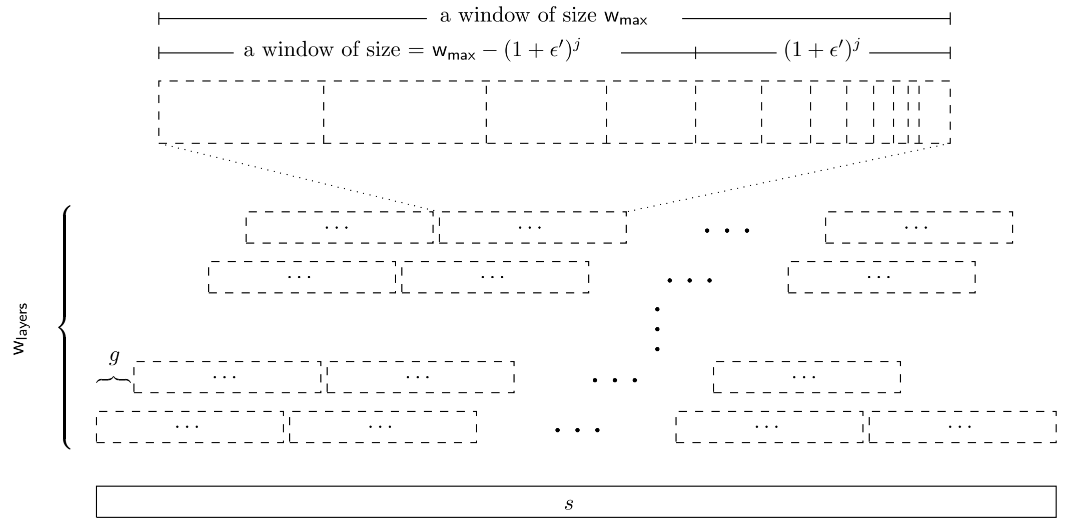
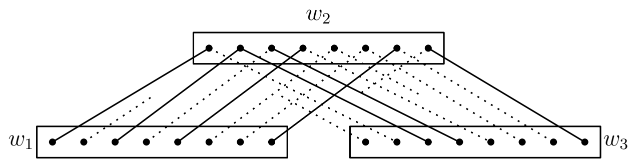
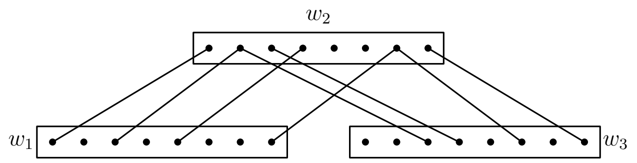

a window of size $\mathsf{w_{max}}$
a window of size $= \mathsf{w_{max}} - (1+\epsilon')^j$
$(1+\epsilon')^j$
$\mathsf{w_{layers}}$
g
s

Figure 3: s is shown with a solid rectangle and windows of W_s are depicted via dashed rectangles. Next to each window of size $\mathsf{w_{max}}$, we have $O(\log_{1+\epsilon'} \mathsf{w_{max}})$ windows of smaller sizes which is shown for one of the windows on the top of the figure. Windows of $\bar{s}$ are constructed similarly.



(a) If $|\mathsf{ed}(w_1, w_2)| \leq 3$ and $|\mathsf{ed}(w_2, w_3)| \leq 2$, due to triangle inequality, $|\mathsf{ed}(w_1, w_3)| \leq |\mathsf{ed}(w_1, w_2)| + |\mathsf{ed}(w_2, w_3)| = 5$ [17].



(b) If $|\mathsf{lcs}(w_1, w_2)| \geq 4$ and $|\mathsf{lcs}(w_2, w_3)| \geq 4$, due to birthday paradox, it is expected that $|\mathsf{lcs}(w_1, w_3)| \geq d(|\mathsf{lcs}(w_1, w_2)|/d)(|\mathsf{lcs}(w_2, w_3)|/d) = 4$ in most cases [52].

Figure 4: Let w_1, w_2, and w_3 be three windows of length $d = 8$. This figure illustrates how the triangle inequality and the birthday paradox triangle inequality are used to bound the ed or lcs of w_1 and w_3.

larger value. This adds extra difficulty to deal with the miscalculated values. On the one hand, we have to make sure an optimal solution remains approximately optimal by the estimated values. On the other hand, we should guarantee that our algorithm does not find fake solutions. That is, no inefficient solution should attain a much better quality due to wrong estimations. This requires additional consideration: previous works had to prove a bound for just one solution, however, we need to prove bounds for all inefficient solutions, of which there are exponentially many! For this purpose, we add an additional Step 3 to our algorithm. Similar to previous work, in Step 2, we try to detect pairs with worse estimates, and then in Step 3, we deal with cases whose estimates are better than the actual values.

Our main results are truly subquadratic time algorithms that obtain $1 + o(1)$ approximate solutions for both edit distance and longest common subsequence in the random setting.

Theorem 3.1 [restated informally]. *There exists an algorithm for* ED*, which computes a $1 + o(1)$ approximate solution and runs in truly subquadratic time in the random setting.*

Theorem 4.1 [restated informally]. *There exists an algorithm for* LCS*, which computes a $1 + o(1)$ approximate solution and runs in truly subquadratic time in the random setting.*

In the following, we bring some of the ideas and techniques of our algorithms. For notational convenience, we state the ideas for LCS, but similar techniques (with a more careful analysis) also apply to edit distance. Our framework consists of Steps 0 through 3. The main technical contribution of our framework is Step 1, which is a replacement for triangle inequality for our setting. Steps 0 and 2 are used from previous work, and Step 3 is an additional procedure designed to

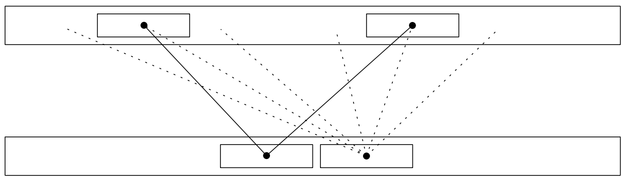


Figure 5: First a subset of windows of W_s are sampled and by computing their ED from all windows of $W_{\overline{s}}$ the undiscovered distances are detected for the sampled windows. Next, neighborhoods close to the undiscovered distances are reexamined via the classic algorithm for edit distance.

restrain the error made in Step 1.

Much like previous work, in Step 0, we construct two sets of windows W_s and $W_{\overline{s}}$ for the strings, and then our goal becomes approximating the LCS of every pair of windows between W_s and $W_{\overline{s}}$. For the purpose of our problems, the windows may have different sizes all in the range $[\mathsf{w_{min}}, \mathsf{w_{max}}]$. However, when the solution size is large (which is the only case we cannot solve the problem in truly subquadratic time), we have $|W_s||W_{\overline{s}}|{\mathsf{w_{max}}}^2 \simeq n^2$. That is, if we naïvely compute the LCS of every pair of windows, our running time would be quadratic.

Before we proceed to Step 1, we discretize the solutions. More precisely, define $\xi \in \{0, 1, 1+\epsilon, (1+\epsilon)^2, \ldots, \mathsf{w_{max}}\}$ to be a threshold and let $\mathsf{lcs}_\xi(w_i, \overline{w_j}) \rightarrow \{0, 1\}$ be a 0/1 function specifying whether $|\mathsf{lcs}(w_i, \overline{w_j})| \geq \xi$. Classic discretization techniques imply that if one determines $\mathsf{lcs}_\xi(w_i, w_j)$ for all $\xi \in \{0, 1, 1+\epsilon, (1+\epsilon)^2, \ldots, \mathsf{w_{max}}\}$ then a $1+\epsilon$ approximation of the solutions for window pairs would be available. Thus, for Step 1, we fix a $\xi \in \{0, 1, 1+\epsilon, (1+\epsilon)^2, \ldots, \mathsf{w_{max}}\}$, and assume that the goal is to determine lcs_ξ for all pairs of windows. For a fixed ξ, let us define the LCS-graph G_ξ to be a bipartite graph whose vertices are the windows and $(i, j) \in E(\mathsf{G}_\xi)$ iff $\mathsf{lcs}_\xi(w_i, \overline{w_j}) = 1$ ($w_i \in W_s$ and $\overline{w_j} \in W_{\overline{s}}$). Note that we only take into account the edges between the two parts and not within the parts.

1.2.1 Step 1: An Alternative to Triangle Inequality.

In Step 1, we make a connection between the smoothness of the setting and the shape of the LCS-graph. In general, the degrees of the vertices in G_ξ could be any number between 0 and $\max\{|W_s|, |W_{\overline{s}}|\}$. More precisely, for each vertex i corresponding to the windows of $W_{\overline{s}}$ and any integer $d \in [0, |W_s|]$, one can determine a threshold for which the degree of vertex i is equal to d in the LCS-graph. Our key observation in this step is the following: by relaxing the threshold ξ by a factor of $1+\epsilon$, we can be sure that either the degrees are very high (close to $|W_s|$) or very low (close to 0)!

Let us illustrate the above with a thought experiment. Assume for simplicity that all the windows are disjoint and let W_s be the set of windows constructed for s and let $\overline{w} \in W_{\overline{s}}$ be an arbitrary window of $W_{\overline{s}}$. Indeed, if $\overline{w}$ were constructed via a random procedure (similar to the windows of s) we would expect that the size of the longest common subsequence of any window in W_s and $\overline{w}$ is concentrated around a certain value. However, this may not be the case as $\overline{w}$ may be adversarially constructed to be very close to certain windows of W_s and far from others. In the extreme case, when $\overline{w}$ is exactly equal to one of the windows of W_s, the longest common subsequence for that pair is equal to $|\overline{w}|$, which may be much larger than the value it would obtain had we constructed $\overline{w}$ via a random procedure. However, we can then argue that since $\overline{w}$ is exactly equal to one of the windows of W_s and those characters are displaced according to a random procedure, then for the rest of the windows, the value of LCS for $\overline{w}$ is concentrated around a fixed value. In this case, $\overline{w}$ is only connected to one window of W_s in the LCS-graph when ξ is large enough.

Now consider two windows w_i and w_j of W_s. In order to make sure in the LCS-graph, $\overline{w}$ is only connected to two windows w_i and w_j of W_s, it suffices to divide each of w_i and w_j into two equal pieces and then concatenate one piece from each window to obtain $\overline{w}$. Still, we expect the size of the LCS between $\overline{w}$ and w_i and w_j to be large, while this is not the case for the rest of the windows. This argument can be generalized: For any subset $Q \subseteq W_s$ with a bounded size, one can construct $\overline{w}$ in a way that its corresponding vertex in the LCS-graph is only connected to the vertices of Q. However, there is a caveat to our construction: as the size of Q grows, the longest common subsequence of $\overline{w}$ and the windows of Q drops.

Let us step back and recall the construction of windows in Step 0. As aforementioned, our construction ensures that $|W_s||W_{\overline{s}}|{\mathsf{w_{max}}}^2 \simeq n^2$, which is essentially the time necessary to compute the LCS of every pair of windows. We point out here that when the value of the LCS for two windows is bounded by μ, a runtime of $O({\mathsf{w_{max}}}^2)$ is not required; we can actually find the

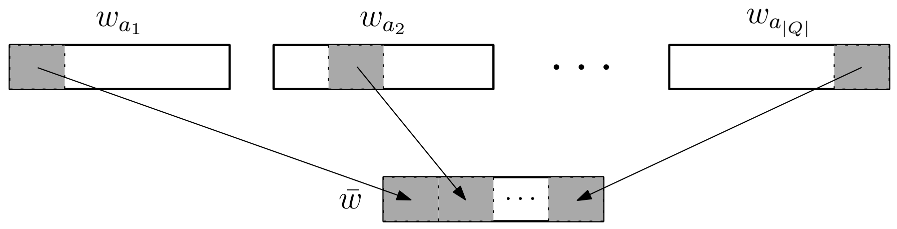


Figure 6: For any subset $Q \subseteq W_s$ with a bounded size, $Q = \{w_{a_1}, w_{a_2}, \ldots, w_{a_{|Q|}}\}$, one can construct $\overline{w}$ in a way that its corresponding vertex in the LCS-graph is only connected to the vertices of Q. However, as the size of Q grows, the longest common subsequence of $\overline{w}$ and the windows of Q drops.

longest common subsequence in time $O(\mathsf{w}_{\max}\mu)$. Thus, when μ is asymptotically smaller than $\mathsf{w}_{\max}$[11] we can find the LCS's in truly subquadratic time. Therefore, in the above construction, when $|Q|$ is large enough (say $|Q| = \sqrt{|W_s|}$), the constructed instance is not very hard to solve. Therefore, there is a limit to which a window $\overline{w}$ of $W_{\overline{s}}$ can have edges to only an arbitrary subset of W_s.

Of course, the above is just an attempt to make instances for which detecting the edges of the LCS-graph is challenging. The failure alone does not imply that hard instances do not exist. One technical challenge that we face in this paper is to formally prove that the degrees of the vertices in our setting cannot be arbitrary. More formally, we show this in Lemma 1.1.

LEMMA 1.1. (STATED INFORMALLY) *Let $(s, \overline{s})$ be a random instance of LCS and W_s and $W_{\overline{s}}$ be constructed accordingly. There exists a $0 < \kappa < 1$ such that for any $0 \le \xi \le \mathsf{w}_{\max}$, any $0 < \epsilon < 1$, and any window $\overline{w}$ of $W_{\overline{s}}$, either the degree of window $\overline{w}$ in $\mathsf{G}_{(1-\epsilon)\xi}$ is at least $|W_s| - |W_s|^{1-\kappa}$ or the degree of $\overline{w}$ in $\mathsf{G}_{(1+\epsilon)\xi}$ is at most $|W_s|^{1-\kappa}$.*

One thing to keep in mind before outlining the proof of Lemma 1.1 is that one needs to relax ξ by a factor of $1+\epsilon$ to obtain Lemma 1.1. Otherwise, trivial counterexamples can be made. Moreover, in our construction for W_s and $W_{\overline{s}}$, we may very well have overlapping windows or windows with different sizes that further make the argument difficult to prove.

To prove Lemma 1.1, we first simplify the problem by defining another string s' which shares a lot of similarity with s, yet it has a simpler structure. Let Σ be the alphabet of the strings and D be a distribution over the symbols of Σ. More precisely, D returns symbol $\sigma \in \Sigma$ with probability $\mathsf{fr}_{\hat{s}}(\sigma)/n$, where $\mathsf{fr}_{\hat{s}}(\sigma)$ is the frequency of symbol σ in $\hat{s}$. We construct string s' by realizing n characters from distribution D independently.

Construct $W_{s'}$ the same way as W_s and assume for simplicity that $W_{s'}$ only contains non-overlapping windows of size n^β and that $|W_{s'}| = n^\alpha$ $(\alpha + \beta = 1)$. Let $\overline{w} \in W_{\overline{s}}$ be a window of $W_{\overline{s}}$. In Section 2.1, we show that for each window w' of $W_{s'}$, with high probability $|\mathsf{lcs}(w', \overline{w})|$ is close to

$$\mathsf{AVGL}_{\overline{w}} = \mathbb{E}_{r \sim \mathsf{D}^{n^\beta}}\left[|\mathsf{lcs}(r, \overline{w})|\right]^{12}$$

The key to proving this fact is that LCS can be seen as a bounded function since replacing a character in a string by another symbol changes the size of the longest common subsequence by at most one. Thus, we can use the result of Boucheron *et al.* [19] to bound the value of $|\mathsf{lcs}(w', \overline{w}) - \mathsf{AVGL}_{\overline{w}}|$[13].

In the second step, we roughly prove that for a given threshold n^τ, a constant ϵ, and a large enough n, either (I) for almost all the windows $w' \in W_{s'}$ we have $|\mathsf{lcs}(w', \overline{w})| > n^\tau/(1+\epsilon)$ or (II) for almost all the windows $w' \in W_{s'}$ we have $|\mathsf{lcs}(w', \overline{w})| \le n^\tau(1+\epsilon)$. Suppose $|\overline{w}| = n^\zeta$ and let Ω be the set of all the strings of length n^ζ constructed by characters of Σ. We use a probabilistic method to show that with a high probability, for no string in Ω both conditions (I) and (II) are violated. In fact, we show that if we select a string $r \in \Omega$ uniformly at random, the probability that r violates both conditions (I) and (II) is $O(1/|\Omega|^2)$, which in turn implies that with a high probability, no such string exists.

In the third step, we prove that although s is made via a different procedure than s', yet the same argument holds for it. The challenge is that there is a correlation between the windows of s (for s' all the windows were completely independent). Hence, the concentration bounds proved for the windows of s' are not directly applicable to the windows of s. However,

[11]$\mu = \mathsf{w}_{\max}{}^{1-\epsilon}$ for some constant $\epsilon > 0$.

[12]Term $r \sim \mathsf{D}^{n^\beta}$ indicates that r is constructed by realizing n^β characters from distribution D.

[13]Due to some structural differences of LCS and ED, we use another concentration bound (McDiarmid [47]) for edit distance

in Section 2.3, we introduce a two-phase procedure to shuffle $\hat{s}$ that enables us to show similar concentration bounds for s. In this procedure, instead of directly permuting the characters of $\hat{s}$, we first initialize s by realizing n independent characters from D and then modify a subset of the characters to make sure s and $\hat{s}$ have the same multiset of characters. We then show that the symbols whose frequencies are significantly changed during the perturbation phase do not have a considerable impact on the size of LCS. Consequently, we prove a statement similar to the statement in Section 2.2 for the windows of W_s.

1.2.2 Steps 2 and 3: A $1 + o(1)$ Approximation Algorithm for ED and LCS Lemma 1.1 gives us a strong tool to approximate the solutions for the window pairs. For a given ξ and a window $\overline{w} \in W_{\overline{s}}$, we can sample a few windows of W_s and compute the longest common subsequence of $\overline{w}$ and the sampled windows. Based on the outcome, we can decide which case of Lemma 1.1 holds for $\overline{w}$. If $\overline{w}$ is high-degree, we report all edges and we can be sure that the error is small, otherwise, we simply report no edges for $\overline{w}$ and can be sure that we leave a few edges behind. This is similar to the sparsification of Chakraborty *et al.* [24] for obtaining a constant approximation algorithm for edit distance.

What differs between the two algorithms is that in the algorithm of Chakraborty *et al.* [24], when the value of $|\mathsf{ed}(w_i, \overline{w_j})|$ is miscalculated, we are sure that the estimated value is an upper bound on the edit distance in which case may not necessarily approximate the correct distance. In our algorithm, however, the miscalculated value could be much larger or much smaller. For this reason, Step 2 alone does not guarantee an approximation factor of $1 + o(1)$. More precisely, when overestimates are also given for LCS, (or equivalently underestimates are made for ED), any inefficient algorithm can attain a value which may be better than that of the optimal. Thus, in addition to proving a lower bound on the quality of the optimal solution, we need to prove an upper bound on the quality of any solution in which case their count is exponentially large.

We do use Step 2 of the algorithm of Chakraborty *et al.* [24] but we complement that with our own Step 3. In Step 2, we address the error of underestimates and make sure the value of the optimal solution is not hurt significantly. In Step 3, we deal with overestimates. The idea is based on the following simple observation: an overestimation may only incur an error to our solution if the overestimated pair of windows is used in our solution.

Let us be more specific about the algorithm. After Step 2, the estimates for the LCS of the window pairs are stored in a table T. Next, a dynamic program finds an optimal alignment between the windows of s and windows $\overline{s}$ that maximizes the LCS. Denote by $D[i][j]$ the longest common subsequence for the prefix of s ending at the last character of w_i and the prefix of $\overline{s}$ ending at the last character of $\overline{w}_j$. This is then updated by considering the previous pair that takes part in the solution. This runs in time $\widetilde{O}(|W_s||W_{\overline{s}}|)$ and finds the optimal alignment.

In Step 3, we run this DP and find an optimal alignment. However, keep in mind that overestimates may give us a fake solution. To address this issue, after solving the DP, we compute the LCS of every pair of windows that contribute to the found solution and find out what is the actual quality of the solution we found. If the quality of the solution does not change by much, we report the solution and can be sure that it is a $1+o(1)$ approximation of the optimal solution. Otherwise, we correct the values that made us find a fake solution and update table T accordingly. We next run the same DP to find a solution according to the same recursive formula, but this time we use the updated table as the estimates for the LCS of the windows. We continue on with this routine: (1) we find the optimal alignment, (2) verify if the solution is fake or not, and (3) if fake, we update the table and run the dynamic program again.

The crux of the argument is that at some point our algorithm terminates with a solution which is not fake and since the number of overestimates is significantly smaller than the number of window pairs, the total running time is truly subquadratic for this step. This is formally proven in Section 4.

2 Step 1: An Alternative to Triangle Inequality

Let α and β be two constants such that $\alpha + \beta = 1$ and let $W = \{w_1, w_2, \ldots, w_{n^\alpha}\}$ be a subset of W_s that partitions s into n^α non-overlapping windows of size n^β. Furthermore, let $\overline{w}$ be a window of $W_{\overline{s}}$, and suppose $|\overline{w}| = n^\zeta$. Our goal in this section is to prove Lemmas 2.1 and 2.2, which state roughly that for a small constant ϵ, and a threshold n^τ, with a high probability we have

- Either for almost all windows $w_i \in W$ we have $|\mathsf{lcs}(\overline{w}, w_i)| \geq n^\tau/(1+\epsilon)$, or for almost all windows $w_i \in W$ we have $|\mathsf{lcs}(\overline{w}, w_i)| \leq n^\tau(1+\epsilon)$.

- Either for almost all windows $w_i \in W$ we have $|\mathsf{ed}(\overline{w}, w_i)| \geq n^\tau/(1+\epsilon)$, or for almost all windows $w_i \in W$ we have $|\mathsf{ed}(\overline{w}, w_i)| \leq n^\tau(1+\epsilon)$.

LEMMA 2.1. *Let κ be a constant such that $\kappa > (\zeta - \tau)+$*

$2(\beta-\tau)$ and assume

$$n > \max\left\{\begin{array}{c} \sqrt[\tau]{\frac{16e\alpha}{3\epsilon^2\tau}\ln\frac{16e\alpha}{3\epsilon^2\tau}} \\ \sqrt[\eta_1]{\frac{32e}{3\epsilon^2\eta_1}\ln\frac{32e}{3\epsilon^2\eta_1}} \\ \sqrt[\eta_2]{\frac{32e}{\epsilon^2\eta_2}\ln\frac{32e}{\epsilon^2\eta_2}} \end{array}\right\},$$

where ϵ is a small constant, $\eta_1 = \kappa - (\zeta - \tau)$, and $\eta_2 = \kappa-(\zeta-\tau)-2(\beta-\tau)$. Then, with a high probability, either $|\mathsf{lcs}(w_i,\overline{w})| \geq n^\tau(1+\epsilon)$ holds for at most n^κ windows $w_i \in W$, or $|\mathsf{lcs}(w_i,\overline{w})| < n^\tau/(1+\epsilon)$ holds for at most n^κ windows $w_i \in W$.

Lemma 2.2. *Let κ be a constant such that $\kappa > (\zeta-\tau)+2(\beta-\tau)$ and assume*

$$n > \max\left\{\begin{array}{c} \sqrt[2\tau-\beta]{\frac{e\alpha}{\epsilon^2(2\tau-\beta)}\ln\frac{e\alpha}{\epsilon^2(2\tau-\beta)}} \\ \sqrt[\eta_1]{\frac{2e}{\epsilon^2\eta_1}\ln\frac{2e}{\epsilon^2\eta_1}} \\ \sqrt[\eta_2]{\frac{32e}{\epsilon^2\eta_2}\ln\frac{32e}{\epsilon^2\eta_2}} \end{array}\right\},$$

where ϵ is a small constant, $\eta_1 = \kappa-(\zeta-\tau)-(\beta-\tau)$, and $\eta_2 = \kappa-(\zeta-\tau)-2(\beta-\tau)$. Then, , with a high probability, either $|\mathsf{ed}(w_i,\overline{w})| \geq n^\tau(1+\epsilon)$ holds for at most n^κ windows $w_i \in W$, or $|\mathsf{ed}(w_i,\overline{w})| < n^\tau/(1+\epsilon)$ holds for at most n^κ windows $w_i \in W$.

We prove Lemmas 2.1 and 2.2 in three steps. In the first step, in Section 2.1, we provide two concentration bounds for LCS and ED. Next, in Section 2.2, we prove two statements similar to Lemmas 2.1 and 2.2 for the case that the characters of each window are independently drawn from an arbitrary distribution. Finally, in the third step, in Section 2.3, we extend these results to the case of the windows in W.

2.1 Concentration of LCS and ED. Let $\Sigma = \{\sigma_1,\sigma_2,\ldots,\sigma_k\}$ be a set of alphabets and let D be a distribution over the symbols in Σ. Throughout this section, we prove two concentration bounds for LCS and ED. These two bounds imply that for a window $\overline{w} \in W_{\overline{s}}$ and a string w whose every character is randomly drawn from D, values of $|\mathsf{lcs}(w,\overline{w})|$ and $|\mathsf{ed}(w,\overline{w})|$ are highly concentrated around their mean values.

Suppose w is a string of length n^β, constructed by realizing n^β independent characters from D (we denote this by $w \sim \mathsf{D}^{n^\beta}$) and let $\overline{w} \in W_{\overline{s}}$ be a window of length n^ζ. Define $\mathsf{AVGE}_{\overline{w}}$ and $\mathsf{AVGL}_{\overline{w}}$ as

$$\text{(2.1)} \qquad \mathsf{AVGE}_{\overline{w}} = \mathbb{E}_{w'\sim\mathsf{D}^{n^\beta}}|\mathsf{ed}(w',\overline{w})|$$

and

$$\text{(2.2)} \qquad \mathsf{AVGL}_{\overline{w}} = \mathbb{E}_{w'\sim\mathsf{D}^{n^\beta}}|\mathsf{lcs}(w',\overline{w})|.$$

Our goal in this section is to prove Lemmas 2.3 and 2.4. Notice that the bound provided by Lemma 2.4 is tighter than Lemma 2.3, which is due to some structural advantages of LCS over ED.

Lemma 2.3. *The probability that*

$$\Big||\mathsf{ed}(w,\overline{w})| - \mathsf{AVGE}_{\overline{w}}\Big| \geq q$$

is upper-bounded by $2\exp(-\frac{2q^2}{n^\beta})$.

Lemma 2.4. *The probability that*

$$\Big||\mathsf{lcs}(w,\overline{w})| - \mathsf{AVGL}_{\overline{w}}\Big| \geq q$$

is upper-bounded by $\exp(-\frac{q^2}{2\mathsf{AVGL}_{\overline{w}}+2q/3})$.

We start by proving Lemma 2.3. To this end, we use McDiarmid's Inequality [46]. A formal statement of this inequality is given in Theorem 2.1 .

Theorem 2.1. (McDiarmid [46]) *Let $X_1,\ldots,X_k \in \mathcal{X}$ be k independent random variables and assume that we have a function $f : \mathcal{X}^k \to \mathbb{R}$ such that for every $i \in [k]$*

$$\text{(2.3)} \quad \sup_{x_1,\ldots,x_k,\hat{x}_i}\big|f(x_1,x_2,\ldots,x_k)-f(x_1,x_2,\ldots,x_{i-1},\hat{x}_i,x_{i+1},\ldots,x_k)\big| \leq c_i.$$

Then for every $q > 0$ we have:

$$\mathbb{P}(\big|\mathbb{E}[f(X_1,\ldots,X_k)] - f(X_1,\ldots,X_k)\big| \geq q) \leq 2\exp\big(-\frac{2q^2}{\sum_{i=1}^k c_i^2}\big).$$

Inequality (2.3) states roughly that for every i, replacing the i'th coordinate by some other value while keeping the rest of the coordinates intact, changes the value f by at most c_i. It can be easily verified that for both $f(\cdot) = |\mathsf{ed}(\cdot,\overline{w})|$ and $f(\cdot) = |\mathsf{lcs}(\cdot,\overline{w})|$, Inequality (2.3) holds for every $1 \leq i \leq n^\beta$ and $c_i = 1$. More formally, assume $|\mathsf{lcs}(\cdot,\overline{w})|$ and $|\mathsf{ed}(\cdot,\overline{w})|$ are two functions which receive a string r consisted of n^β characters as input and return $|\mathsf{lcs}(r,\overline{w})|$ and $|\mathsf{ed}(r,\overline{w})|$ respectively. Then, we have

$$\text{(2.4)} \qquad \Big||\mathsf{lcs}(r,\overline{w})| - |\mathsf{lcs}(r^i,\overline{w})|\Big| \leq 1$$

and

$$\text{(2.5)} \qquad \Big||\mathsf{ed}(r,\overline{w})| - |\mathsf{ed}(r^i,\overline{w})|\Big| \leq 1,$$

where r^i is a string similar to r, except that the i'th character is replaced by another symbol. The reason that these inequalities hold is the trivial observation that changing a character in a string, can change the size of the longest common subsequence and edit distance by at most one. Thus, for these cases, we have an upper bound of n^β for $\sum_i c_i^2$. Combining Theorem 2.1 and Inequalities (2.3), (2.4) and (2.5), we conclude that Inequalities (2.6) and (2.7) hold for LCS and ED (recall that w is a string randomly drawn from D^{n^β}):

$$\mathbb{P}\Big(\Big||\mathsf{lcs}(w,\overline{w})| - \mathsf{AVGL}_{\overline{w}}\Big| \geq q\Big) \leq 2\exp(-\frac{2q^2}{n^\beta}), \tag{2.6}$$

$$\mathbb{P}\Big(\Big||\mathsf{ed}(w,\overline{w})| - \mathsf{AVGE}_{\overline{w}}\Big| \geq q\Big) \leq 2\exp(-\frac{2q^2}{n^\beta}). \tag{2.7}$$

Inequality (2.7) directly implies Lemma 2.3. However, as mentioned earlier, the bound of Lemma 2.4 is tighter. To prove Lemma 2.4, we use the result of Boucheron, Lagosi, and Massart [19] for bounded functions.

Theorem 2.2. (Boucheron *et al.* [19]) *Suppose $X_1, X_2, \ldots, X_k \in \mathcal{X}$ are k independent random variables and assume that we have a function $f : \mathcal{X}^k \to \mathbb{R}$ such that for every $1 \leq i \leq k$ and $x_1, x_2, \ldots, x_k$ the conditions*

$$\sup_{\hat{x}_i} f(x_1,x_2,\ldots,x_k) - f(x_1,x_2,\ldots,x_{i-1},\hat{x}_i,x_{i+1},\ldots,x_k) \leq 1,$$

and

$$\sum_{i=1}^{k} \sup_{\hat{x}_i} f(x_1,\ldots,x_k) - f(x_1,\ldots,x_{i-1},\hat{x}_i,x_{i+1},\ldots,x_k) \leq f(x_1,\ldots,x_k)$$

hold. Then

$$\mathbb{P}(\big|\mathbb{E}[f(X_1,X_2,\ldots,X_n)] - f(X_1,X_2,\ldots,X_k)\big| \geq t) \leq 2\exp\big(-\frac{2q^2}{2\mathbb{E}[f(X_1,X_2,\ldots,X_k)] + 2q/3}\big).$$

To prove Lemma 2.4, it suffices to show that the conditions of Theorem 2.2 hold for LCS. As discussed earlier, the first condition of Theorem 2.2 is valid for LCS. Thus, it only remains to verify the second condition, which we show in Lemma 2.5.

Lemma 2.5. *Suppose r is a string, and for every i, let r^i be a string similar to r, except that the i'th character is replaced by some other symbol, such that $|\mathsf{lcs}(r^i,\overline{w})|$ is minimized. We have*

$$\sum_i |\mathsf{lcs}(r,\overline{w})| - |\mathsf{lcs}(r^i,\overline{w})| \leq |\mathsf{lcs}(r,\overline{w})|.$$

Proof. For every i, we say the i'th character of r (denoted by $r[i]$) is *excess*, if there exists a common subsequence l of r and $\overline{w}$ such that $|l| = |\mathsf{lcs}(r,\overline{w})|$ (that is, l is one of the longest common subsequences of r and $\overline{w}$), and $r[i] \notin l$. Trivially, if $r[i]$ is excess, we have $|\mathsf{lcs}(r,\overline{w})| - |\mathsf{lcs}(r^i,\overline{w})| = 0$. On the other hand, at least $n^\beta - |\mathsf{lcs}(r,\overline{w})|$ of the characters in r are excess. For the rest of the characters, we know that $|\mathsf{lcs}(r,\overline{w})| - |\mathsf{lcs}(r^i,\overline{w})| \leq 1$. Hence, we have

$$\sum_i |\mathsf{lcs}(r,\overline{w})| - |\mathsf{lcs}(r^i,\overline{w})| \leq |\mathsf{lcs}(r,\overline{w})|.$$

□

Combining Theorem 2.2 and Lemma 2.5 implies Lemma 2.4.

It is worth to mention that the reason that we could not provide a better bound for ED was that the condition of Lemma 2.5 does not necessarily hold for ED. To see why, consider the strings in Figure 7. For this example, $\overline{w}$ consists of $n^{\beta-\phi}$ blocks of size n^ϕ, and r consists of $n^{\beta-\phi}+1$ blocks of size n^ϕ, and $\Sigma = \{a, b\}$. Each block of $\overline{w}$ and r only differs in the first character. Note that for this instance, we have $\mathsf{ed}(\overline{w}, r) = n^\phi + n^{\beta-\phi}$. Furthermore, it is easy to observe that for every $1 \leq i \leq n^\beta$, changing $r[i]$ to b decreases the size of $\mathsf{ed}(\overline{w}, r)$ exactly by one. Hence, for every i we have $|\mathsf{ed}(r,\overline{w})| - |\mathsf{ed}(r^i,\overline{w})| = 1$, and consequently

$$\sum_i |\mathsf{ed}(r,\overline{w})| - |\mathsf{ed}(r^i,\overline{w})| = n^\beta.$$

2.2 Independent Random Strings.

In this section, we consider the following setting: suppose $W' = \{w_1, w_2, \ldots, w_{n^\alpha}\}$ are n^α strings of length n^β, each constructed by realizing n^β independent characters from a distribution D over alphabet Σ, and let $\overline{w} \in W_{\overline{s}}$ be a window of length n^ζ. In Section 2.1, we showed that with a high probability, for each w_i, the sizes of $\mathsf{lcs}(w_i,\overline{w})$ and $\mathsf{ed}(w_i,\overline{w})$ are close to $\mathsf{AVGL}_{\overline{w}}$ and $\mathsf{AVGE}_{\overline{w}}$. Using this fact, here we prove Lemmas 2.6 and 2.7 for the windows in W'.

Lemma 2.6. *Let κ be a constant such that $\kappa > \zeta - \tau$ and assume that*

$$n > \max\left\{\begin{array}{c} \sqrt[\tau]{\frac{16e\alpha}{3\epsilon^2\tau}\ln\frac{16e\alpha}{3\epsilon^2\tau}}, \\ \sqrt[\eta_1]{\frac{32e}{3\epsilon^2\eta_1}\ln\frac{32e}{3\epsilon^2\eta_1}} \end{array}\right\},$$

where ϵ is a small constant and $\eta_1 = \kappa - (\zeta - \tau)$. Then, with high probability, either $|\mathsf{lcs}(w_i,\overline{w})| \geq (1+\epsilon)n^\tau$ holds for at most n^κ windows $w_i \in W'$, or $|\mathsf{lcs}(w_i,\overline{w})| < n^\tau/(1+\epsilon)$ holds for at most n^κ windows $w_i \in W'$.

Lemma 2.7. *Let κ be a constant such that $\kappa > (\zeta-\tau)+(\beta-\tau)$ and assume that*

$$n > \max\left\{\begin{array}{c} \sqrt[2\tau-\beta]{\frac{e\alpha}{\epsilon^2(2\tau-\beta)}\ln\frac{e\alpha}{\epsilon^2(2\tau-\beta)}}, \\ \sqrt[\eta_1]{\frac{2e}{\epsilon^2\eta_1}\ln\frac{2e}{\epsilon^2\eta_1}} \end{array}\right\},$$

where ϵ is a small constant and $\eta_1 = \kappa - (\zeta - \tau) - (\beta - \tau)$. Then, with high probability, either $|\mathsf{ed}(w_i,\overline{w})| \geq (1+\epsilon)n^\tau$ holds for at most n^κ windows $w_i \in W'$, or $|\mathsf{ed}(w_i,\overline{w})| < n^\tau/(1+\epsilon)$ holds for at most n^κ windows $w_i \in W'$.

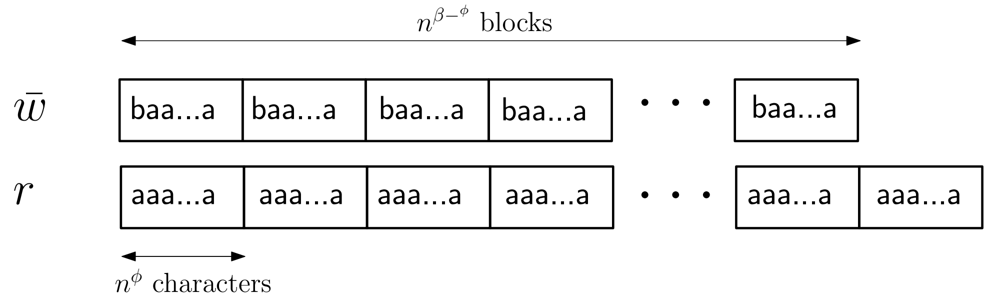


Figure 7: For strings $\overline{w}$ and r we have $\mathsf{ed}(\overline{w}, r) = n^{\phi} + n^{\beta-\phi}$. Furthermore, except the characters in the last block of r, changing every other character of r to b decreases the size of $\mathsf{ed}(\overline{w}, r)$ exactly by one.

Before proving Lemmas 2.6 and 2.7, we would like to note that choosing $q = \epsilon\mathsf{AVGE}_{\overline{w}}$ and $q = \epsilon\mathsf{AVGL}_{\overline{w}}$ in Lemmas 2.3 and 2.4, implies the following two inequalities:

(2.8)
$$\mathbb{P}\Big(\Big||\mathsf{ed}(w,\overline{w})| - \mathsf{AVGE}_{\overline{w}}\Big| \geq \epsilon\mathsf{AVGE}_{\overline{w}}\Big) \leq 2\exp(-\frac{2\epsilon^2\mathsf{AVGE}_{\overline{w}}^2}{n^{\beta}})$$

and

(2.9)
$$\mathbb{P}\Big(\Big||\mathsf{lcs}(w,\overline{w})| - \mathsf{AVGL}_{\overline{w}}\Big| \geq \epsilon\mathsf{AVGL}_{\overline{w}}\Big) \leq \exp(-\frac{\epsilon^2\mathsf{AVGL}_{\overline{w}}}{2+2\epsilon/3}).$$

In addition, we use Lemma 2.8 to provide a lower bound on the value of n in the proof of both Lemmas 2.6 and 2.7.

Lemma 2.8. *Let $z > 1$ and $0 < p < 1$ be two constants and let $k = \sqrt[p]{(ez/p)\ln(ez/p)}$. We have $\frac{k^p}{\ln k} \geq z$.*

We prove Lemmas 2.6 and 2.7 in Sections 2.2.1 and 2.2.2. However, the proof of both lemmas has the same structure.

2.2.1 Proof of Lemma 2.6.

Let Ω be the set of all strings of size n^{ζ} made by characters in Σ. Divide Ω into two categories L and R. A string ℓ belongs to L if

$$\mathbb{E}_{w\sim\mathsf{D}^{n^{\beta}}}\Big[|\mathsf{lcs}(\ell, w)|\Big] \leq n^{\tau}$$

and belongs to R otherwise. Notice that $|\Omega| = |\mathsf{L}| + |\mathsf{R}| \leq n^{n^{\zeta}}$.

Let ℓ be a string randomly drawn from L. First, we use Inequality (2.9) to provide an upper bound on the probability that $|\mathsf{lcs}(\ell, w_i)| > (1+\epsilon)n^{\tau}$ holds for ℓ.

Lemma 2.9. *Let ℓ be a string randomly drawn from L, and let $w_i \in W'$ be a window. We have*

$$\mathsf{Pr}\Big[|\mathsf{lcs}(\ell, w_i)| > (1+\epsilon)n^{\tau}\Big] \leq \exp -\Big[\frac{n^{\tau}\epsilon^2}{8/3}\Big].$$

Define E_ℓ to be the event that for at least $n^{\alpha-\eta}$ of the windows in W', $|\mathsf{lcs}(\ell, w_i)| > (1+\epsilon)n^{\tau}$ holds. Using the upper bound we obtained in Lemma 2.9 and union bound, In Lemma 2.10 we bound the probability that event E_ℓ occurs ($\mathsf{Pr}[E_\ell]$).

Lemma 2.10. *Let ℓ be a string randomly drawn from L, and let E_ℓ be the event that for at least $n^{\alpha-\eta}$ windows in W', $|\mathsf{lcs}(\ell, w_i)| > (1+\epsilon)n^{\tau}$ holds. We have*

$$\mathsf{Pr}[E_\ell] \leq n^{\alpha n^{\alpha-\eta}} \cdot \exp -\Big[\frac{\epsilon^2 n^{\alpha+\tau-\eta}}{8/3}\Big].$$

Note that we have $|L| \leq n^{n^{\zeta}}$. Now, if $|L|\cdot\mathsf{Pr}[E_\ell] = o(1)$, then we can claim that for a large enough n, with a high probability, for no string $\ell \in \mathsf{L}$, event E_ℓ occurs. Therefore, we adjust η in a way that:

(2.10)
$$|L|\cdot\mathsf{Pr}[E_\ell] = n^{n^{\zeta}} \cdot n^{\alpha n^{\alpha-\eta}} \cdot \exp -\Big[\frac{\epsilon^2 n^{\alpha+\tau-\eta}}{8/3}\Big] < 1/n^{n^{\zeta}},$$

which means

$$\ln n(2n^{\zeta} + \alpha n^{\alpha-\eta}) < c\epsilon^2 n^{\tau+\alpha-\eta}$$
$$2n^{\zeta}\ln n < (c\epsilon^2 n^{\tau} - \alpha\ln n)n^{\alpha-\eta}$$
$$\frac{2n^{\zeta}\ln n}{c\epsilon^2 n^{\tau} - \alpha\ln n} < n^{\alpha-\eta}, \tag{2.11}$$

where $c = 3/8$.

Lemma 2.11. *In order for Inequality (2.11) to hold, it suffices to*

- *Choose a small enough η such that $\alpha - \eta > \zeta - \tau$.*
- *Choose a large enough n, such that*

$$\alpha\ln n < c\epsilon^2 n^{\tau}/2. \tag{2.12}$$

and

$$\frac{32}{3\epsilon^2} < n^{(\alpha-\eta)-(\zeta-\tau)}/\ln n. \tag{2.13}$$

With a similar argument, we can prove the same bounds as in Lemma 2.14 for the strings in R.

Suppose that we chose η and n so that the conditions of Lemma 2.14 hold, and let $\kappa = \alpha - \eta$. String $\overline{w}$ either belongs to L or R. If $\overline{w} \in \mathsf{L}$, then, with a high probability, $|\mathsf{lcs}(\overline{w}, w_i)| > n^{\tau}(1+\epsilon)$ holds for at most n^{κ} windows $w_i \in W'$. If $\overline{w} \in \mathsf{R}$, then with a high probability, $|\mathsf{lcs}(\overline{w}, w_i)| < n^{\tau}/(1+\epsilon)$ holds for at most n^{κ} windows in $w_i \in W'$.

Finally, using Lemma 2.8 we can provide a lower bound on n according to Inequalities (2.12) and (2.13). This, implies Lemma 2.6.

LEMMA 2.6. *Let κ be a constant such that $\kappa > \zeta - \tau$ and assume that*

$$n > \max\left\{\begin{array}{c} \sqrt[\tau]{\frac{16e\alpha}{3\epsilon^2\tau}\ln\frac{16e\alpha}{3\epsilon^2\tau}}, \\ \sqrt[\eta_1]{\frac{32e}{3\epsilon^2\eta_1}\ln\frac{32e}{3\epsilon^2\eta_1}} \end{array}\right\},$$

where ϵ is a small constant and $\eta_1 = \kappa - (\zeta - \tau)$. Then, with high probability, either $|\mathsf{lcs}(w_i, \overline{w})| \geq (1+\epsilon)n^\tau$ holds for at most n^κ windows $w_i \in W'$, or $|\mathsf{lcs}(w_i, \overline{w})| < n^\tau/(1+\epsilon)$ holds for at most n^κ windows $w_i \in W'$.

2.2.2 Proof of Lemma 2.7

The structure of the proof of Lemma 2.7 is very similar to the proof of Lemma 2.6. Again, let Ω be the set of all strings of size n^ζ made by characters in Σ. Divide Ω into two subsets L and R. A string ℓ belongs to L if

$$\mathbb{E}_{w\sim \mathsf{D}^{n^\beta}}\left[|\mathsf{ed}(\ell, w)|\right] \leq n^\tau$$

and belongs to R otherwise ($|\Omega| = |\mathsf{L}| + |\mathsf{R}| \leq n^{n^\zeta}$).

Let ℓ be a string randomly drawn from L. We use Inequality (2.9) to provide an upper bound on the probability that $|\mathsf{ed}(\ell, w_i)| > (1+\epsilon)n^\tau$ holds for ℓ.

LEMMA 2.12. *Let ℓ be a string randomly drawn from L, and let $w_i \in W'$ be a window. We have*

$$\mathsf{Pr}\left[|\mathsf{ed}(\ell, w_i)| > (1+\epsilon)n^\tau\right] \leq \exp -\left[\frac{2n^{2\tau}\epsilon^2}{n^\beta}\right].$$

Using a similar argument as Lemma 2.12 for R, we show that for a string r selected uniformly at random from R we have

$$\mathsf{Pr}\left[|\mathsf{lcs}(r, w_i)| < (1-\epsilon)n^\tau\right] \leq \exp -\left[\frac{2n^{2\tau}\epsilon^2}{n^\beta}\right].$$

LEMMA 2.13. *Let ℓ be a string randomly drawn from L, and let E_ℓ be the event that for at least $n^{\alpha-\eta}$ windows in W', $|\mathsf{ed}(\ell, w_i)| > (1+\epsilon)n^\tau$ holds. We have*

$$\mathsf{Pr}[E_\ell] \leq n^{\alpha n^{\alpha-\eta}} \cdot \exp -\left[\frac{2\epsilon^2 n^{\alpha+2\tau-\eta}}{n^\beta}\right].$$

Now, if $|L|\cdot\mathsf{Pr}[E_\ell] = o(1)$, then we claim that for a large enough n, with a high probability for no string $\ell \in \mathsf{L}$, event E_ℓ occurs. We have $|L| \leq n^{n^\zeta}$. Thus, we must adjust η in a way that:

$$|L|\cdot\mathsf{Pr}[E_\ell] = n^{n^\zeta}\cdot n^{\alpha n^{\alpha-\eta}}\cdot\exp -\left[\frac{2\epsilon^2 n^{\alpha+2\tau-\eta}}{n^\beta}\right] < 1/n^{n^\zeta}, \tag{2.14}$$

which means

$$(2n^\zeta + \alpha n^{\alpha-\eta})\ln n < c\epsilon^2 n^{2\tau+\alpha-\eta-\beta}$$

$$2n^\zeta \ln n < (c\epsilon^2 n^{2\tau-\beta} - \alpha\ln n)n^{\alpha-\eta}$$

$$\frac{2n^\zeta \ln n}{c\epsilon^2 n^{2\tau-\beta} - \alpha\ln n} < n^{\alpha-\eta}, \tag{2.15}$$

where $c = 2$.

LEMMA 2.14. *In order for Inequality (2.15) to hold, it suffices to*

- *Choose a small enough η such that $\alpha - \eta > (\zeta - \tau) + (\beta - \tau)$.*
- *Choose a large enough n, such that*

$$\alpha \ln n < \epsilon^2 n^{2\tau-\beta}, \tag{2.16}$$

and

$$\frac{2}{\epsilon^2} < n^{(\alpha-\eta)-(\zeta-\tau)-(\beta-\tau)}/\ln n. \tag{2.17}$$

Finally, suppose that $\alpha - \eta > (\zeta - \tau) + (\beta - \tau)$ and n is large enough such that both Inequalities (2.16) and (2.17) hold, and let $\kappa = \alpha - \eta$. String $\overline{w}$ either belongs to L or R. If $\overline{w} \in \mathsf{L}$, then $|\mathsf{ed}(\overline{w}, w_i)| > n^\tau(1+\epsilon)$ holds for at most n^κ windows $w_i \in W$. If $\overline{w} \in \mathsf{R}$, then $|\mathsf{ed}(\overline{w}, w_i)| < n^\tau/(1+\epsilon)$ holds for at most n^κ windows in w_i. Moreover, using Lemma 2.8 we can provide a lower bound on n according to Inequalities (2.16) and (2.17). This, implies Lemma 2.7.

LEMMA 2.7. *Let κ be a constant such that $\kappa > (\zeta-\tau)+(\beta-\tau)$ and assume that*

$$n > \max\left\{\begin{array}{c} \sqrt[2\tau-\beta]{\frac{e\alpha}{\epsilon^2(2\tau-\beta)}\ln\frac{e\alpha}{\epsilon^2(2\tau-\beta)}}, \\ \sqrt[\eta_1]{\frac{2e}{\epsilon^2\eta_1}\ln\frac{2e}{\epsilon^2\eta_1}} \end{array}\right\},$$

where ϵ is a small constant and $\eta_1 = \kappa - (\zeta - \tau) - (\beta - \tau)$. Then, with high probability, either $|\mathsf{ed}(w_i, \overline{w})| \geq (1+\epsilon)n^\tau$ holds for at most n^κ windows $w_i \in W'$, or $|\mathsf{ed}(w_i, \overline{w})| < n^\tau/(1+\epsilon)$ holds for at most n^κ windows $w_i \in W'$.

2.3 Windows of a Random Permutation.

In Section 2.2, we proved Lemmas 2.6 and 2.7 for the windows in W'. In this section, we present generalized versions of these lemmas for the windows in W. Recall that the strings in W are windows of s, and s is constructed by a random shuffle on $\hat{s}$. Unfortunately, the windows in W do not admit the nice random structure of the windows in W'. However, we show that the structure of the windows in W and W' are not too different.

To provide a tool to compare the windows in W and W', let us introduce a probabilistic procedure to generate a random shuffle of the characters in $\hat{s}$. In this section, we assume that s is constructed via the following process:

- Let D be a distribution of probabilities over the symbols in Σ, which selects character σ with probability $\mathsf{fr}_{\hat{s}}(\sigma)/|\hat{s}|$. Draw a random string $s' \sim \mathsf{D}^n$.
- Initially set $s = s'$. Call a symbol $\sigma \in \Sigma$ *overly-frequent*, if $\mathsf{fr}_s(\sigma) > \mathsf{fr}_{\hat{s}}(\sigma)$ and *infrequent* if $\mathsf{fr}_s(\sigma) < \mathsf{fr}_{\hat{s}}(\sigma)$. While s is not a permutation of $\hat{s}$, repeat the following update process.
- (Update process): select an infrequent symbol σ and choose a random character in s whose corresponding symbol is overly-frequent and change it to σ.

It is clear that at the end of the process, s is a random shuffle of $\hat{s}$. According to the above process for producing s, we attribute every character of s a label which shows whether or not that character is changed during the update process.

Definition 1. *We say a character in s is stable, if it does not change during the update process.*

Trivially, stable characters are the same in s' and s, while unstable characters are not. Intuitively, stable characters show how similar s is to a random string drawn from $\mathsf{D}^{n^{\alpha+\beta}}$. In Lemma 2.15, we compare the frequency of the characters in s and s'. This lemma states that the ratio of unstable characters for high-frequency symbols is small.

Lemma 2.15. *Let σ be a symbol with frequency at least $n^{2(\beta-\tau)+\eta}$ in s, where $\eta > 0$ is a constant. We can say that for $\gamma = \sqrt{8\ln(n)}n^{-(\beta-\tau+\eta/2)}$, with probability at least $1 - 1/n^3$, the number of unstable characters corresponding to symbol σ is at most $\gamma \mathsf{fr}_{\hat{s}}(\sigma)$.*

Corollary 2.1. (of Lemma 2.15) *Since unstable characters are selected uniformly at random, according to Lemma 2.15, we can say that for a character c whose corresponding symbol has a frequency at least $n^{2(\beta-\tau)+\eta}$, the probability that c would be unstable is less than γ.*

Now, we show that the number of unstable characters in s is not large enough to have a significant impact on $|\mathsf{lcs}(w_i, \overline{w})|$ and $|\mathsf{ed}(w_i, \overline{w})|$. To this end, we define *negligible* symbols.

Definition 2. *Let $w_i \in W$ be a window of s and let w_i' be its corresponding window in s'. For a given threshold n^τ and a small constant ϵ', we say that a set C of symbols is negligible for window w_i, if the total number of the unstable characters in w_i with symbols in C is at most $\epsilon' n^\tau$.*

Intuitively, if a set C of symbols is negligible for some window w_i, we can ignore the unstable characters corresponding to C in that window, while losing a factor at most ϵ' of the longest common subsequence and edit distance of w_i and $\overline{w}$. Denote by H the set of symbols with frequency at least $n^{2(\beta-\tau)+\eta}$ in $\hat{s}$. In Lemma 2.16 we show that with a high probability, symbols in H are negligible for any window w_i.

Lemma 2.16. *For every window w_i, with probability at least $1 - 1/n^3$, set H is negligible.*

By Lemma 2.16, the expected number of the windows that H is not negligible for them is less than $n^\alpha \cdot n^{-3} = n^{\alpha-3}$. Using Chernoff bound we can conclude that the number of such windows is concentrated around its mean value, which implies that with a high probability, no such window exists.

Corollary 2.2. *With probability at least $1 - 1/n^3$, the number of the windows in W for which set H is not negligible is $o(1)$.*

Finally, consider the symbols with a frequency less than $n^{2(\beta-\tau)+\eta}$. Note that $\overline{w}$ contains at most n^ζ number of such symbols, and each one of these symbols has frequency less than $n^{2(\beta-\tau)+\eta}$ in s. Thus, the total number of such characters in s is at most $n^{\zeta+2(\beta-\tau)+\eta}$.

Observation 1. *Let L be the set of the symbols in $\overline{w}$ with frequency less than $n^{2(\beta-\tau)+\eta}$ in s. The number of the windows in W_s for which L is non-negligible is at most $n^{\zeta+2(\beta-\tau)+\eta}/\epsilon' n^\tau$.*

According to Observation 1, the number of windows for which low-frequency characters are not negligible is bounded by $n^{\zeta-\tau+2(\beta-\tau)+\eta}/\epsilon'$. On the other hand, by Lemma 2.16, high-frequency symbols are negligible for the rest of the windows. Recall from Section 2.1 that any unstable character can increase the value of $|\mathsf{lcs}(\overline{w}, w_i)| - |\mathsf{lcs}(\overline{w}, w_i')|$ and $|\mathsf{ed}(\overline{w}, w_i)| - |\mathsf{ed}(\overline{w}, w_i')|$ by at most one. Therefore, for these windows, we have $|\mathsf{lcs}(\overline{w}, w_i)| - |\mathsf{lcs}(\overline{w}, w_i')| < \epsilon' n^\tau$ and $|\mathsf{ed}(\overline{w}, w_i)| - |\mathsf{ed}(\overline{w}, w_i')| < \epsilon' n^\tau$. This along with Lemma 2.16 yields Lemmas 2.1 and 2.2.

Lemma 2.1. *Let κ be a constant such that $\kappa > (\zeta - \tau) + 2(\beta - \tau)$ and assume*

$$n > \max \left\{ \begin{array}{c} \sqrt[\tau]{\frac{16e\alpha}{3\epsilon^2\tau} \ln \frac{16e\alpha}{3\epsilon^2\tau}} \\ \sqrt[\eta_1]{\frac{32e}{3\epsilon^2\eta_1} \ln \frac{32e}{3\epsilon^2\eta_1}} \\ \sqrt[\eta_2]{\frac{32e}{\epsilon^2\eta_2} \ln \frac{32e}{\epsilon^2\eta_2}} \end{array} \right\},$$

where ϵ is a small constant, $\eta_1 = \kappa - (\zeta - \tau)$, and $\eta_2 = \kappa - (\zeta - \tau) - 2(\beta - \tau)$. Then, with a high probability, either $|\mathsf{lcs}(w_i, \overline{w})| \geq n^\tau(1+\epsilon)$ holds for at most n^κ windows $w_i \in W$, or $|\mathsf{lcs}(w_i, \overline{w})| < n^\tau/(1+\epsilon)$ holds for at most n^κ windows $w_i \in W$.

Lemma 2.2. *Let κ be a constant such that $\kappa > (\zeta - \tau) + 2(\beta - \tau)$ and assume*

$$n > \max \left\{ \begin{array}{c} \sqrt[2\tau-\beta]{\frac{e\alpha}{\epsilon^2(2\tau-\beta)} \ln \frac{e\alpha}{\epsilon^2(2\tau-\beta)}} \\ \sqrt[\eta_1]{\frac{2e}{\epsilon^2\eta_1} \ln \frac{2e}{\epsilon^2\eta_1}} \\ \sqrt[\eta_2]{\frac{32e}{\epsilon^2\eta_2} \ln \frac{32e}{\epsilon^2\eta_2}} \end{array} \right\},$$

where ϵ is a small constant, $\eta_1 = \kappa - (\zeta - \tau) - (\beta - \tau)$, and $\eta_2 = \kappa - (\zeta - \tau) - 2(\beta - \tau)$. Then, , with a high probability, either $|\mathsf{ed}(w_i, \overline{w})| \geq n^\tau(1+\epsilon)$ holds for at most n^κ windows $w_i \in W$, or $|\mathsf{ed}(w_i, \overline{w})| < n^\tau/(1+\epsilon)$ holds for at most n^κ windows $w_i \in W$.

3 $1 + o(1)$ Approximation Algorithm for Edit Distance

In this section, we present a $1 + o(1)$ approximation truly subquadratic time algorithm for ED in the random setting. Although we denote the approximation factor by $1 + O(\epsilon)$, since the dependence of the running time on $1/\epsilon$ is polynomial, choosing $\epsilon = 1/\log n$ results in a $1 + o(1)$ approximation factor and only incurs a $\mathsf{poly}(\log n)$ term to the running time.

We assume an estimate of the solution, denoted by n^δ, is given to our algorithm. It is shown that such an assumption only imposes a $1+\epsilon$ multiplicative term to the approximation factor and an $O((1/\epsilon)\log n)$ multiplicative term to the running time of our algorithm [17]. For a small solution size, we use the algorithm of Landau *et al.* [44] which finds the exact solution in time $O(n + |\mathsf{ed}|^2)$ where $|\mathsf{ed}|$ is the size of the solution. In this case, the running time would be bounded by $O(n + n^{2\delta})$. Hence, we only deal with cases with a large solution size. In Section 1.2, we discussed the outline of our algorithm. Here, we present Steps 0, 1, 2, and 3 in more details.

3.1 Step 0.

In Step 0, two sets of windows W_s and $W_{\bar{s}}$ are constructed for both s and $\bar{s}$. This construction has the property that if the edit distances between the windows of W_s and $W_{\bar{s}}$ are known, one can find a $1 + 4\epsilon$ approximation solution of $|\mathsf{ed}(s, \bar{s})|$ in truly subquadratic time. Therefore, the task of approximating the edit distance between s and $\bar{s}$ reduces to approximating edit distance between many smaller windows. The construction we use is similar to the construction of Boroujeni *et al.* [17]. You can find more details about this construction in the full version of the paper. In the construction of windows, for an arbitrary parameter $0 < x < 1$,

- the total number of windows is equal to $|W_s| + |W_{\bar{s}}| = \widetilde{O}_\epsilon(n^{x+(1-\delta)})$,
- the minimum size of a window is $\mathsf{w_{min}} = \epsilon n^{1-x}$,
- the maximum size of a window is $\mathsf{w_{max}} = n^{1-x}$,
- the number of different window sizes is $\mathsf{w_{sizes}} = \widetilde{O}(1/\epsilon)$, and
- the ratio of the maximum window size over the minimum window size is bounded by $\mathsf{w_{gap}} = \mathsf{w_{max}}/\mathsf{w_{min}} = \widetilde{O}(1/\epsilon)$.

Moreover, for each window, we only need to know its distance to $\widetilde{O}_\epsilon(n^x)$ other windows. We also assume all pairwise distances between windows are at least $\epsilon n^{\delta - x}$. This assumption only imposes a $1 + \epsilon$ term to the approximation factor.

Let opt be an optimal solution, and opt' be an optimal window-compatible solution. A window $\overline{w} \in W_{\bar{s}}$ with respect to a threshold ξ and a window size λ is called:

- *high degree* if the ed's between $\overline{w}$ and almost all windows of W_s with size n^β are less than $\xi \cdot (1 + \epsilon)$, or is called
- *low degree* if the ed's between $\overline{w}$ and almost all windows of W_s with size n^β are more than $\xi/(1 + \epsilon)$.

We show in Lemma 2.2 that if n is large enough, every window is either high degree or low degree (or both) with high probability for arbitrary ξ and λ. Since a $1 + O(\epsilon)$ approximation of the solution suffices for our algorithm, we use a discretization technique and consider only $\log_{1+\epsilon} \mathsf{w_{max}}$ possible values for ξ. More precisely, we consider all $\xi \in \{0, 1, 1 + \epsilon, (1 + \epsilon)^2, \ldots, \mathsf{w_{max}}\}$ except ξ's smaller than $\epsilon n^{\delta - x}$. Furthermore, we consider all $\mathsf{w_{sizes}}$ possibilites for λ. The total number of such values for ξ and λ is at most $(\log_{1+\epsilon} \mathsf{w_{max}}) \cdot \mathsf{w_{sizes}} = \widetilde{O}_\epsilon(1)$.

Before Step 1 (sparsification), our algorithm must decide whether each window is high degree or low degree with respect to each threshold and window size. This part of our algorithm is presented in Section 3.1.1.

3.1.1 High Degree and Low Degree Windows.

In this section, our algorithm decides whether a window $\overline{w} \in W_{\bar{s}}$ is high degree or low degree with regards to a threshold ξ and a window size λ via random sampling. Let k be the number of samples which we fix later. We sample k non-overlapping windows of W_s of length λ and compute the ed between $\overline{w}$ and the sampled windows. Afterward, we decide whether $\overline{w}$ is high degree or low degree according to the median of the computed ed's. The computed edit distance between window $\overline{w}$ and a sampled window w has one of the following situations.

- It is smaller than or equal to $\xi/(1 + \epsilon)$: in this case, we are almost sure that $\overline{w}$ is high degree since for a low degree window, the probability of sampling one string of size λ and a ed of no more than $\xi/(1 + \epsilon)$ is $\frac{n^\kappa}{n/\lambda}$ where n^κ is bounded by $(1/\epsilon^3)n^{3(1-\delta)+\rho}$ in Lemma 3.2. Moreover, n/λ is the number of possible non-overlapping windows of size λ. Therefore, the probability is bounded by
$$\frac{n^\kappa}{n/\lambda} \le \frac{(1/\epsilon^3)n^{3(1-\delta)+\rho}}{\frac{n}{\mathsf{w_{min}}}} = (1/\epsilon^4)n^{-x+3(1-\delta)+\rho}.$$
Assuming $x > 3(1-\delta)+\rho$, we define $c = x-3(1-\delta)-\rho > 0$. Therefore, the probability of $\overline{w}$ being low degree is at most $(1/\epsilon^4)n^{-c}$. Therefore, the window is high degree with a probability of at least $1 - (1/\epsilon^4)n^{-c}$.
- Similarly, if the ed is more than or equal to $\xi \cdot (1 + \epsilon)$, the window is low degree with a probability of at least $1 - (1/\epsilon^4)n^{-c}$.
- If the ed is in range $(\xi/(1 + \epsilon), \xi \cdot (1 + \epsilon))$, we take a slightly different ed threshold $\xi' = \xi \cdot (1 + \epsilon)^2$ and claim that $\overline{w}$ is high degree according to the new threshold since the probability of such an ed of a sampled string for a low degree window is at most $(1/\epsilon^4)n^{-c}$. Note that
$$\xi = \xi \cdot (1 + \epsilon)^2 = n^{\tau + 2\log_n(1+\epsilon)} = n^{\tau + 2(\epsilon + O(\epsilon^2))/\log n}.$$
Therefore, we can hide the difference between τ and $\tau + 2(\epsilon + O(\epsilon^2))/\log n$, which is $o(1)$, in ρ for a large enough n.

The median of the ed's between $\overline{w}$ and the k sampled windows also belongs to one of the three cases explained above which is used for deciding whether $\overline{w}$ is low degree or high degree. In the following, we prove that the overall probability of an incorrect decision for even one window is bounded by n^{-3}.

LEMMA 3.1. *We can decide whether every window $\overline{w}$ of $W_{\overline{s}}$ with regards to all thresholds and window sizes are low degree or high degree in time $\widetilde{O}_\epsilon(n^{2-x+(1-\delta)})$, within an approximation factor of $(1+\epsilon)^3$, and with a probability of at least $1-n^{-3}$.*

In Step 1, our algorithm handles high degree windows. Step 2 deals with low degree windows.

3.2 Step 1.

Lemma 2.2 shows that high degree windows of $W_{\overline{s}}$, with respect to a threshold ξ and a window size λ, are connected to almost all windows of W_s with size λ. However, we cannot apply this lemma for windows of size λ all at once since they overlap. Therefore, we apply Lemma 2.2 several times, each time on a subset of non-overlapping windows of size λ. Lemma 3.2 is concluded directly from Lemma 2.2 and partitioning the windows of size λ into non-overlapping subsets.

LEMMA 3.2. *If a window $\overline{w} \in W_{\overline{s}}$ is high degree with respect to a threshold ξ and a window size λ, the number of windows of W_s with size λ and which have an ed from $\overline{w}$ more than $\xi \cdot (1+\epsilon)$ is bounded by $\widetilde{O}_\epsilon(n^{4(1-\delta)+\rho})$, with high probability, where $\rho > 0$ is an arbitrarily small constant.*

Therefore, if $\overline{w}$ is high-degree with regard to a ξ and a λ, we report all edges for it and considering Lemma 3.2, we can be sure that the error is small. With this intention, Lemma 3.3 shows an upper bound on the error.

LEMMA 3.3. *If we report all edges for all high degree windows with respect to all thresholds and window sizes, the total number of false positives edges is bounded by $\widetilde{O}_\epsilon(n^{x+5(1-\delta)+\rho})$, with high probability, where $\rho > 0$ is an arbitrarily small constant.*

3.3 Step 2.

In this section, for a low degree window $\overline{s}$ with respect to ξ and λ, we find almost all windows of size λ in W_s with an ed less than ξ. We use a random sampling technique similar to that of [24]. Here, we only prove the running time since the correctness is already proven in [24].

The idea is to sample a limited number of low degree windows and use them to restrict the number of relevant windows for other low degree windows near the sampled ones. To explain what we mean when we say two windows are near each other, suppose we hypothetically construct larger windows by using a smaller parameter x'. Now, we say two windows are nearby if they lie on the same larger window. The properties of the new construction such as $\mathsf{w_{min}}'$ and $\mathsf{w_{max}}'$ are defined similarly.

If the number of low degree windows inside a larger window is at most $\epsilon^2 n^{x-x'-(1-\delta)}$, we can ignore these low degree windows since ignoring them imposes a total error of at most

$$\frac{n}{\mathsf{w_{min}}'} \cdot \epsilon^2 n^{x-x'-(1-\delta)} \cdot n^{1-x} = \epsilon n^\delta$$

to the overall solution. Large windows with few low degree windows are called *type one.*

On the other hand, if the number of such low degree windows in a larger window is more than $\epsilon^2 n^{x-x'-(1-\delta)}$, we call the larger window *type two.* For a larger window of type two, we hit at least one of its low degree windows using a random sampling method with high probability. We sample each window of $W_{\overline{s}}$ with a probability of

$$p = \frac{1}{\epsilon^2 n^{x-x'-(1-\delta)}} \cdot 4\log n.$$

The sampling is repeated for all ξ's and λ's. Therefore, we find at least one low degree window of each larger window of type two with the right ξ and λ with high probability. The expected size of the hitting set is

$$\begin{aligned}\mathbb{E}[|\text{hitting set}|] &= |W_{\overline{s}}| \cdot p \cdot \log_{1+\epsilon} \mathsf{w_{max}} \cdot \mathsf{w_{sizes}} \\ &= \widetilde{O}_\epsilon(n^{x'+2(1-\delta)}).\end{aligned}$$

We then find the ed of each sampled window with all of the windows of W_s. Note that only $\widetilde{O}_\epsilon(n^x)$ windows of W_s are possibly matching a window. The running time of this part of our algorithm and the number of windows pairs found are shown in Lemma 3.4.

LEMMA 3.4. *Our algorithm for sampled low degree windows runs in time $\widetilde{O}_\epsilon(n^{2+x'-x+2(1-\delta)})$ and outputs an expected number of $\widetilde{O}_\epsilon(n^{x'+5(1-\delta)+\rho})$ pairs of windows such that for every larger window of type two, at least one low degree window with the right ξ and λ exists in the output. Moreover, this part of our algorithm works correctly with a probability of at least $1-n^{-3}$.*

If a pair of windows such as $(w, \overline{w}) \in W_s \times W_{\overline{s}}$ is found, we conclude that if this pair is used in opt', windows near $\overline{w}$ are also matched to windows near w. Therefore, every pair is extended to at most

$$\begin{aligned}(\frac{d'}{g} \cdot \log_{1+\epsilon} \mathsf{w_{max}}')^2 &= (\frac{d'}{d}\mathsf{w_{layers}} \cdot \widetilde{O}_\epsilon(1))^2 \\ &\leq \widetilde{O}_\epsilon(n^{x-x'}\mathsf{w_{layers}})^2 \\ &= \widetilde{O}_\epsilon(n^{2(x-x')+2(1-\delta)})\end{aligned}$$

pairs. In Lemma 3.5, all pairs found in the first part of Step 2 are extended and their corresponding ed's are computed.

LEMMA 3.5. *The extension phase of our algorithm takes expected time $\widetilde{O}_\epsilon(n^{2-x'+9(1-\delta)+\rho})$ and produces expected $\widetilde{O}_\epsilon(n^{2x-x'+7(1-\delta)+\rho})$ tuples containing a pair of windows and an ed size, such that for each larger window of type two, a tuple relatively close to its match in opt is found.*

3.4 Step 3.

A total solution can be found from partial solution between windows in $\widetilde{O}_\epsilon(n^{2x+(1-\delta)})$ time via a dynamic program similar to [17] and [24]. However, the total solution may be fake since it may incorporate many false-positive pairs. More precisely, the total solution is not fake if it contains at most $\epsilon n^{x-(1-\delta)}$ false-positive pairs. Note that this number of false-positive pairs produce an error of at most $\epsilon n^{x-(1-\delta)} \cdot \mathsf{w_{max}} = \epsilon n^\delta$. If the solution is fake, it contains several false-positive pairs. In this case, we remove the discovered false-positive pairs and *restart* the dynamic program. To find whether if a solution is fake or not, we should check all partial solutions used in the total solution. This check is done in time

$$\frac{n}{\mathsf{w}_{\mathsf{min}}} \cdot O({\mathsf{w}_{\mathsf{max}}}^2) = \widetilde{O}_\epsilon(n^{2-x})$$

Using Lemma 3.3 we immediately show an upper bound on the number of restarts.

Corollary 3.1. *The dynamic program for* ED *restarts at most* $\frac{\widetilde{O}_\epsilon(n^{x+5(1-\delta)+\rho})}{\epsilon n^{x-(1-\delta)}} = \widetilde{O}_\epsilon(n^{6(1-\delta)+\rho})$ *times.*

The running time of Step 3 is therefore, bounded by the the number of restarts times the running time of the dynamic program.

Corollary 3.2. *Step 3 of our algorithm runs in time* $\widetilde{O}_\epsilon(n^{6(1-\delta)+\rho}) \cdot \widetilde{O}_\epsilon(n^{2x+(1-\delta)} + n^{2-x}) = \widetilde{O}_\epsilon(n^{2x+7(1-\delta)+\rho} + n^{2-x+6(1-\delta)+\rho})$.

3.4.1 A $1 + o(1)$ Approximation Algorithm for ED.

In this section, we conclude our algorithm by setting the parameters and showing that the running time is truly subquadratic. We denote the running time of our algorithm by $\widetilde{O}_\epsilon(n^{z+\rho})$ and set the parameters x, x', and δ to optimize the running time of our algorithm. Here, n^δ is a critical value such that instances with a solution size smaller than n^δ are solved via the algorithm of Landau *et al.* [44], and instances with a solution size larger than n^δ are solved via the main part our algorithm. Also, note that ρ is an arbitrarily small positive constant; therefore, we ignore it in the linear program.

(3.18)

$$\begin{aligned} &\text{minimize } z \\ &\text{subject to } 2\delta^* \le z && \text{(Landau } et\ al. \text{ [44])} \\ &1 \le 2\delta^* && \text{(Landau } et\ al. \text{ [44])} \\ &2 - x + (1-\delta^*) \le z && \text{(Lemma 3.1)} \\ &2 + x' - x + 2(1-\delta^*) \le z && \text{(Lemma 3.4)} \\ &2 - x' + 9(1-\delta^*) \le z && \text{(Lemma 3.5)} \\ &2x + 7(1-\delta^*) \le z && \text{(Corollary 3.2)} \\ &2 - x + 6(1-\delta^*) \le z && \text{(Corollary 3.2)} \\ &0 < x' < x < 1 \\ &0 < \delta^* < 1 \end{aligned}$$

This linear program is minimized for $x' = 22/39$, $x = 30/39$, and $\delta^* = 37/39$ where $z = 2-4/39 \approx 1.897436$. By rewriting the time complexity as $O(n^{1.898})$, we hide all $\mathsf{poly}(1/\epsilon)$, $\mathsf{polylog}(n)$, and n^ρ terms. This gives us the desired truly subquadratic time complexity of our algorithm.

Theorem 3.1. *There exists a randomized algorithm which runs in expected time* $O(n^{1.898})$ *and approximates random* ED *within a factor of* $1 + o(1)$.

The following corollary of Theorem 3.1 shows that a solution can be found with high probability.

Corollary 3.3. *There exists a randomized algorithm which runs in time* $O(n^{1.898})$ *and approximates random* ED *within a factor of* $1 + \epsilon$ *with probability at least* $1 - n^{-3}$.

4 A $1 + o(1)$ Approximation Algorithm for LCS

In this section, we present a $1 + o(1)$ approximation truly subquadratic time algorithm for LCS in our random setting. Similar to Section 3, we denote the approximation factor by $1 + O(\epsilon)$ and use $\epsilon = 1/\log n$ to achieve a $1 + o(1)$ approximation factor. Moreover, similar to Section 3, we assume a value of $n^{1-\delta}$ is given as input and we must decide whether the solution size is at most $n^{1-\delta}$ or it is much larger than $n^{1-\delta}$. Here, for a small solution size, we use a classic dynamic program which finds the exact solution in time $O(n^{2-\delta})$.

Lemma 4.1. *Let s and $\bar{s}$ be two arbitrary strings of size n and $\delta > 0$ be an arbitrary constant. If* $|\mathsf{lcs}(s, \bar{s})| \le n^{1-\delta}$, *then there exists an algorithm that computes* $|\mathsf{lcs}(s, \bar{s})|$ *in time* $\widetilde{O}(n^{2-\delta})$.

The main challenge of our algorithm is to deal with the Instances with a larger solution size. In Section 1.2 we illustrated the overall structure of the algorithm. In the following sections, we describe the steps of our algorithm in more detail.

4.1 Step 0.

In Step 0, two sets of windows W_s and $W_{\bar{s}}$ are constructed for both s and $\bar{s}$. This construction has the property that if the lcs of the windows of W_s and $W_{\bar{s}}$ are known, one can find a $1 + 8\epsilon$ approximation solution of $|\mathsf{lcs}(s, \bar{s})|$ in truly subquadratic time. Therefore, the task of approximating the lcs between s and $\bar{s}$ reduces to approximating lcs's between many smaller windows. The construction we use is similar to the construction of [52].

Note that in this construction for an arbitrary $0 < x < 1$, the following properties hold.

- The total number of windows is equal to $|W_s| + |W_{\bar{s}}| = \widetilde{O}_\epsilon((n^{2\delta})n^x)$.
- The maximum size of the windows is equal to $\mathsf{w}_{\mathsf{max}} = \widetilde{O}_\epsilon(n^{1-x+\delta})$.
- The minimum size of the windows is equal to $\mathsf{w}_{\mathsf{min}} = n^{1-x}$.
- The ratio of the maximum window size over the minimum window size is bounded by $\mathsf{w}_{\mathsf{gap}} = \mathsf{w}_{\mathsf{max}}/\mathsf{w}_{\mathsf{min}} = \widetilde{O}_\epsilon(n^\delta)$.
- The number of different window sizes is equal to $\mathsf{w}_{\mathsf{sizes}} = \widetilde{O}_\epsilon(n^\delta)$.

Moreover, we ignore all distances less than $\epsilon n^{(1-\delta)-x}$ between windows.

Let opt be an optimal solution and opt' be an optimal window-compatible solution. A window $\overline{w} \in W_{\bar{s}}$ with respect to a threshold ξ and a window size λ is called

- *high degree* if the lcs's of $\overline{w}$ and almost all of the windows of W_s with size λ are at least $\xi/(1+\epsilon)$, or is called
- *low degree* if the lcs's of $\overline{w}$ and almost all of the windows of W_s with size n^β are at most $\xi \cdot (1+\epsilon)$.

Recall that we treat high degree and low degree windows differently in our algorithm. Therefore, in Section 4.1.1 we determine each window is high degree or low degree.

4.1.1 High Degree and Low Degree Windows.

Next, our algorithm decides whether a window $\mathcal{B}$ is high degree or low degree with regards to a specified lcs threshold ξ and a window size λ via random sampling. Let k be the number of samples which we fix later. For each window $\bar{s} \in W_{\bar{s}}$, we sample k non-overlapping windows of W_s with length λ and compute the lcs between $\overline{w}$ and the sampled windows. Afterward, we decide whether $\overline{w}$ is high degree or low degree according to the median of the computed lcs's. Note that each lcs has one of these situations:

- It is smaller than or equal to $\xi/(1+\epsilon)$: in this case, the window is low degree with a high probability since for a high degree window, the probability of sampling even one string of size λ and a lcs of no more than $\xi/(1+\epsilon)$ according to Lemma 4.3 is
$$\frac{\widetilde{O}_\epsilon(n^{7\delta+\rho})}{n/\mathsf{w_{max}}} = \widetilde{O}_\epsilon(n^{-x+8\delta+\rho}).$$
Assuming $x > 8\delta + \rho$ and $c = x - 8\delta - \rho > 0$, the probability of the event that the lcs of a sampled string for a high degree window being less than or equal to $\xi/(1+\epsilon)$ is at most $\widetilde{O}_\epsilon(n^{-c})$. Therefore, the window is low degree with a probability of at least $1 - \widetilde{O}_\epsilon(n^{-c})$.
- Similarly, if the lcs is more than or equal to $(1+\epsilon)n^\tau$, the window is high degree with a probability of at least $1 - \widetilde{O}_\epsilon(n^{-c})$.
- If the lcs is in range $(n^\tau/(1+\epsilon), (1+\epsilon)n^\tau)$, we take a slightly different lcs threshold $\xi' = \xi/(1+\epsilon)^2$ and claim that $\overline{w}$ is high degree according to the new threshold since the probability of such an lcs of a sampled string for a low degree window is at most $\widetilde{O}_\epsilon(n^{-c})$. Note that since $\xi' = n^{\tau - 2\log_n(1+\epsilon)} = n^{\tau - O(\epsilon/\log n)}$, we can hide the difference between ξ' and ξ in ρ.

The median of the lcs of the k samples also has one of the three cases explained above which is the outcome of our decision process. In Lemma 4.2, we prove the probability of an incorrect decision is bounded by n^{-3}.

Lemma 4.2. *The time complexity of deciding each window is high degree or low degree within an approximation factor of $(1+\epsilon)^3$ is $\widetilde{O}_\epsilon(n^{2-x+3\delta})$ and we decide all windows correctly with a probability of at least $1 - n^{-3}$.*

4.2 Step 1.

Lemma 2.1 states that every window with respect to a threshold ξ and a window size λ is either high degree or low degree (or both) with high probability. Note that even if we know opt' uses which windows of $W_{\bar{s}}$, we do not know their contribution to the solution (ξ) and the size of their matching window λ. Therefore, our algorithm uses all possible values for both ξ and λ. To be more specific, we consider $\log_{1+\epsilon} \mathsf{w_{max}}$ possible values for ξ, and $\mathsf{w_{sizes}}$ values for λ. The number of all possibilities is bounded by
$$(\log_{1+\epsilon} \mathsf{w_{max}}) \cdot \mathsf{w_{sizes}} = \widetilde{O}(1/\epsilon) \cdot \widetilde{O}_\epsilon(n^\delta) = \widetilde{O}_\epsilon(n^\delta).$$

4.2.1 High Degree Windows.

Lemma 2.1 states that high degree windows are connected to almost all of the windows of W_s except a small subset of them. However, to apply this theorem, we should resolve the correlation between overlapping windows. We resolve this issue by applying Theorem 2.1 several times on a set of non-overlapping windows. Note that windows of W_s with an specific size can be partitioned into $\mathsf{w_{layers}} = \frac{d_i}{g_i} = \widetilde{O}_\epsilon(n^\delta)$ layers of non-overlapping windows.

Lemma 4.3. *If a window $\bar{s} \in W_{\bar{s}}$ is high degree for a threshold ξ and a window size λ, the number of windows of size λ in W_s which have a lcs less than $\xi/(1+\epsilon)$ is bounded by $\widetilde{O}_\epsilon(n^{7\delta+\rho})$.*

In Lemma 4.4 we show how many false-positives in total may occur by assuming high degree windows are connected to all of the windows of W_s.

Lemma 4.4. *The total number of false-positives for all windows of $W_{\bar{s}}$, all thresholds and all window sizes is bounded by $\widetilde{O}((1/\epsilon^6)n^{x+9\delta+\rho})$.*

4.3 Step 2.

In this section, we find the lcs of low degree windows and the windows of W_s. The intuition of this part is to sample a limited number of low degree windows and use them to restrict the number of relevant windows for nearby low degree windows. To implement this idea, we first define when two low degree windows are nearby. Note that the construction of the windows can be done for another parameter $x' < x$ which results in larger window sizes. The properties of the construction of larger windows are defined similarly such as $\mathsf{w_{min}}'$ and $\mathsf{w_{max}}'$. We say two windows are nearby if the lie in a larger window. Note that we fix the parameter x' later in Section 4.4.1.

If the number of low degree windows inside a larger window is at most $(\epsilon/n^\delta)n^{x-x'}$, we can ignore these low degree windows since their neglection impose an error of at most
$$(n/\mathsf{w_{min}}') \cdot (\epsilon/n^\delta)n^{x-x'} \cdot n^{1-x} = \epsilon n^{1-\delta}.$$
to the total solution. Large windows with few low degree windows are called *type one*.

On the other hand, if the number of such low degree window in a larger window is more than $(\epsilon/n^\delta)n^{x-x'}$, we call the larger window *type two*. For larger windows of type two, we hit at least one of their low degree windows using a random sampling method with high probability. We sample each window with a probability of
$$p = \frac{1}{\epsilon n^{x-x'-\delta}} \cdot 4\log n.$$
The expected size of the hitting set is
$$\mathbb{E}[|\text{hitting set}|] = |W_{\bar{s}}| \cdot p \cdot \log_{1+\epsilon} \mathsf{w_{max}} \cdot \mathsf{w_{sizes}} = \widetilde{O}_\epsilon(n^{x'+4\delta}).$$
We then find the lcs of each sampled window with all of the windows of W_s. By assuming all lcs thresholds, we find at least one low degree window of each window of type two with the right matched window size and the right lcs threshold with high probability. We prove this claim in Lemma 4.5.

Lemma 4.5. *There is an algorithm that finds an expected number of $\widetilde{O}_\epsilon(n^{x'+11\delta+\rho})$ potential pairs of windows and their* lcs *and runs in expected time $\widetilde{O}_\epsilon(n^{2+x'-x+6\delta})$ such that at least one window with the right window size and the right* lcs *is found for every window of type two with a probability of at least $1-n^{-3}$.*

Therefore, it only remains to compute the lcs between low degree windows of $W_{\bar{s}}$ and a restricted subset of windows of W_s. An output pair of Lemma 4.5 is extended to at most

$$(\frac{\mathsf{w_{max}}'}{g_i} \cdot \mathsf{w_{sizes}})^2 = \widetilde{O}_\epsilon(n^{2(x-x')+6\delta}).$$

pairs.

Lemma 4.6. *The extension phase of our algorithm takes expected time $\widetilde{O}_\epsilon(n^{2-x'+17\delta+\rho})$ and produces expected $\widetilde{O}_\epsilon(n^{2x-x'+17\delta+\rho})$ tuples, each containing a pair of windows and their* lcs*, such that for each window of type two, a tuple relatively close to its match in* opt′ *is found.*

4.4 Step 3.

Recall that Step 0 of our algorithm ensures that partial solutions for windows can be merged into a total solution using a dynamic program which runs in time $\widetilde{O}_\epsilon(n^{2x+6\delta})$. However, the dynamic program may find a fake solution which incorporates too many false-positive pairs. More precisely, the algorithm either finds a total solution that uses at most $\epsilon n^{x-\delta}$ false-positive pairs which impose an error of at most $\epsilon n^{x-\delta} \cdot \mathsf{w_{min}} = \epsilon n^{1-\delta}$ and finished successfully, or fails. In the case of failure, the dynamic program detects at least $\epsilon n^{x-\delta}$ false-positive pairs which can be removed for the next run of the dynamic program and the algorithm restarts. Using Lemma 4.4 we immediately conclude that the number of restarts is at most $\widetilde{O}_\epsilon(n^{11\delta+\rho})$.

Corollary 4.1. *The last phase of our algorithm for* LCS *restarts at most $\widetilde{O}_\epsilon(n^{11\delta+\rho})$ times.*

The following lemma shows the running time of this phase of our algorithm.

Lemma 4.7. *The last phase of our algorithm runs in time $\widetilde{O}(n^{2x+17\delta+\rho} + n^{2-x+11\delta+\rho})$ and finds a solution within an approximation factor of $1+o(1)$.*

4.4.1 A $1+o(1)$ Approximation Algorithm for LCS.

In this section, we conclude the algorithm by setting the parameters and calculate the running time by solving a linear program. We denote the running time of our algorithm by $\widetilde{O}_\epsilon(n^{z+\rho})$ and set the parameters x, x', and δ^* to optimize the running time of our algorithm. Note that $n^{1-\delta^*}$ is the threshold where solutions smaller than it are solved via the dynamic program and solutions larger than it are solved via our main algorithm. Since the time complexities of both of these algorithms are maximized for $n^{1-\delta^*}$, we use δ^* in our linear program instead of δ. Moreover, ρ is an arbitrarily small positive constant; therefore, we ignore it in the linear program.

$$\begin{aligned}
&\text{minimize } z \\
&\text{subject to } 2-\delta^* \le z && \text{(Lemma 4.1)}\\
&2-x+3\delta^* \le z && \text{(Lemma 4.2)}\\
&x < 1-\delta^* && \text{(Lemma 4.5)}\\
(4.19)\quad &2+x'-x+6\delta^* \le z && \text{(Lemma 4.5)}\\
&2-x'+17\delta^* \le z && \text{(Lemma 4.6)}\\
&2x+17\delta^* \le z && \text{(Lemma 4.7)}\\
&2-x+11\delta^* \le z && \text{(Lemma 4.7)}\\
&0 \le x' < x \le 1 \\
&0 \le \delta^* \le 1
\end{aligned}$$

This linear program is minimized for $x' = 18/34$, $x = 25/34$, and $\delta^* = 1/34$ where $z = 2-1/34 \approx 1.970589$. Note that the running time of our algorithm is $\widetilde{O}_\epsilon(n^{2-1/34+\rho})$ which we can rewrite as $O(n^{1.971})$ and hide all $\mathsf{poly}(1/\epsilon)$, $\mathsf{polylog}(n)$, and n^ρ terms by rounding up the exponent to the third decimal digit.

Theorem 4.1. *There exists a randomized algorithm which runs in expected time $O(n^{1.971})$ and approximates random* LCS *within a factor of $1+o(1)$.*

The following corollary of Theorem 4.1 shows that a solution can be found with high probability.

Corollary 4.2. *There exists a randomized algorithm which runs in time $O(n^{1.971})$ and approximates random* LCS *within a factor of $1+o(1)$ with probability at least $1-n^{-3}$.*

References

[1] A. Abboud and A. Backurs. Towards hardness of approximation for polynomial time problems. In *ITCS*, volume 67 of *LIPIcs*, pages 11:1–11:26. Dagstuhl, 2017.

[2] A. Abboud, A. Backurs, and V. V. Williams. Tight hardness results for LCS and other sequence similarity measures. In *FOCS*, pages 59–78. IEEE, 2015.

[3] A. Abboud, T. D. Hansen, V. V. Williams, and R. Williams. Simulating branching programs with edit distance and friends or: a polylog shaved is a lower bound made. In *STOC*, pages 375–388. ACM, 2016.

[4] A. Abboud and A. Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In *ITCS 2018*, volume 94 of *LIPIcs*, pages 35:1–35:14. Dagstuhl, 2018.

[5] C. E. Alves, E. N. Cáceres, and S. W. Song. A coarse-grained parallel algorithm for the all-substrings longest common subsequence problem. *Algorithmica*, 45(3):301–335, 2006.

[6] A. Andoni. Private Communication, 2018.

[7] A. Andoni, M. Deza, A. Gupta, P. Indyk, and S. Raskhodnikova. Lower bounds for embedding edit distance into normed spaces. In *SODA*, pages 523–526. SIAM, 2003.

[8] A. Andoni, A. Goldberger, A. McGregor, and E. Porat. Homomorphic fingerprints under misalignments: Sketching edit and shift distances. In *STOC*, pages 931–940. ACM, 2013.

[9] A. Andoni and R. Krauthgamer. The computational hardness of estimating edit distance. In *FOCS*, pages 724–734. IEEE, 2007.

[10] A. Andoni and R. Krauthgamer. The smoothed complexity of edit distance. In *ICALP*, pages 357–369. Springer, 2008.

[11] A. Andoni, R. Krauthgamer, and K. Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *FOCS*, pages 377–386. IEEE, 2010.

[12] A. Andoni and K. Onak. Approximating edit distance in near-linear time. In *STOC*, pages 199–204. ACM, 2009.

[13] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *STOC*, pages 51–58. ACM, 2015.

[14] N. Bansal, M. Lewenstein, B. Ma, and K. Zhang. On the longest common rigid subsequence problem. *Algorithmica*, 56(2):270–280, Feb. 2010.

[15] Z. Bar-Yossef, T. Jayram, R. Krauthgamer, and R. Kumar. Approximating edit distance efficiently. In *FOCS*, pages 550–559. IEEE, 2004.

[16] T. Batu, F. Ergun, and C. Sahinalp. Oblivious string embeddings and edit distance approximations. In *SODA*, pages 792–801. SIAM, 2006.

[17] M. Boroujeni, S. Ehsani, M. Ghodsi, M. HajiAghayi, and S. Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. In *SODA*, pages 1170–1189. SIAM, 2018.

[18] M. Boroujeni and S. Seddighin. Improved MPC algorithms for edit distance and Ulam distance. In *SPAA*, pages 31–40. ACM, 2019.

[19] S. Boucheron, G. Lugosi, and P. Massart. A sharp concentration inequality with applications. *Random Structures & Algorithms*, 16(3):277–292, 2000.

[20] V. Braverman, M. Charikar, W. Kuszmaul, D. P. Woodruff, and L. F. Yang. The one-way communication complexity of dynamic time warping distance. In *SoCG*, volume 129 of *LIPIcs*, pages 16:1–16:15. Dagstuhl, 2019.

[21] K. Bringman and M. Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *SODA*, pages 1216–1235. SIAM, 2018.

[22] K. Bringmann, F. Grandoni, B. Saha, and V. V. Williams. Truly sub-cubic algorithms for language edit distance and RNA-folding via fast bounded-difference min-plus product. In *FOCS*, pages 375–384. IEEE, 2016.

[23] K. Bringmann and M. Kunnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS*, pages 79–97. IEEE, 2015.

[24] D. Chakraborty, D. Das, E. Goldenberg, M. Koucky, and M. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *FOCS*, pages 979–990. IEEE, 2018.

[25] M. Charikar, O. Geri, M. P. Kim, and W. Kuszmaul. On estimating edit distance: Alignment, dimension reduction, and embeddings. In *ICALP*, pages 34:1–34:14. Dagstuhl, 2018.

[26] L. Chen, S. Goldwasser, K. Lyu, G. N. Rothblum, and A. Rubinstein. Fine-grained complexity meets IP=PSPACE. In *SODA*, pages 1–20. SIAM, 2019.

[27] M. Crochemore, C. S. Iliopoulos, Y. J. Pinzon, and J. F. Reid. A fast and practical bit-vector algorithm for the longest common subsequence problem. *Information Processing Letters*, 80(6):279–285, 2001.

[28] M. Crochemore, G. M. Landau, and M. Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32(6):1654–1673, 2003.

[29] J. B. de Monvel. Extensive simulations for longest common subsequences. *The European Physical Journal B-Condensed Matter and Complex Systems*, 7(2):293–308, 1999.

[30] M. Garofalakis and A. Kumar. Correlating XML data streams using tree-edit distance embeddings. In *PODS*, pages 143–154. ACM, 2003.

[31] O. Gold and M. Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. In *ICALP*. Dagstuhl, 2017.

[32] E. Goldenberg, R. Krauthgamer, and B. Saha. Sublinear algorithms for gap edit distance. In *FOCS*. IEEE, 2019. in press.

[33] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

[34] B. Haeupler, A. Rubinstein, and A. Shahrasbi. Near-linear time insertion-deletion codes and $(1+\epsilon)$-approximating edit distance via indexing. In *STOC*, pages 697–708, 2019.

[35] M. Hajiaghayi, M. Seddighin, S. Seddighin, and X. Sun. Approximating LCS in linear time: Beating the $\sqrt{n}$ barrier. In *SODA*, pages 1181–1200. SIAM, 2019.

[36] M. Hajiaghayi, S. Seddighin, and X. Sun. Massively parallel approximation algorithms for edit distance and longest common subsequence. In *SODA*, pages 1654–1672. SIAM, 2019.

[37] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

[38] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *FOCS*, pages 10–33. IEEE, 2001.

[39] R. Jayaram and B. Saha. Approximating language edit distance beyond fast matrix multiplication: Ultralinear grammars are where parsing becomes hard! In *ICALP*, pages 19:1–19:15. Dagstuhl, 2017.

[40] M. Koucky. Approximating edit distance within constant factor in truly sub-quadratic time. TCS+, 2018.

[41] M. Koucky and M. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. *arXiv preprint arXiv:1904.05459*, 2019.

[42] W. Kuszmaul. Dynamic time warping in strongly subquadratic time: Algorithms for the low-distance regime and approximate evaluation. *arXiv preprint arXiv:1904.09690*, 2019.

[43] W. Kuszmaul. Efficiently approximating edit distance between pseudorandom strings. In *SODA*, pages 1165–1180. SIAM, 2019.

[44] G. M. Landau, E. W. Myers, and J. P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.

[45] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[46] C. McDiarmid. On the method of bounded differences. *Surveys in Combinatorics*, 141(1):148–188, 1989.

[47] C. McDiarmid. Concentration for independent permutations. *Combinatorics, Probability and Computing*, 11(2):163–178, 2002.

[48] R. Ostrovsky and Y. Rabani. Low distortion embeddings for edit distance. In *STOC*, pages 218–224. ACM, 2005.

[49] A. Rubinstein. Approximating edit distance. `https://theorydish.blog/2018/07/20/approximating-edit-distance`, 2018.

[50] A. Rubinstein and J. Brakensiek. Constant-factor approximation of near-linear edit distance in near-linear time. *arXiv preprint arXiv:1904.05390*, 2019.

[51] A. Rubinstein and Z. Song. Reducing approximate longest common subsequence to approximate edit distance. *arXiv preprint arXiv:1904.05451*, 2019.

[52] A. Runbinstein, S. Seddighin, Z. Song, and X. Sun. Approximation algorithms for LCS and LIS with truly improved running times. In *FOCS*. IEEE, 2019. in press.

[53] B. Saha. Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems. In *FOCS*, pages 118–135. IEEE, 2015.

[54] B. Saha. Fast & space-efficient approximations of language edit distance and RNA folding: An amnesic dynamic programming approach. In *FOCS*, pages 295–306. IEEE, 2017.

[55] M. E. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *SODA*, pages 1698–1709, 2013.

[56] X. Sun and D. P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *SODA*, pages 336–345. SIAM, 2007.