Approximation Algorithms for LCS and LIS with Truly Improved Running Times

Aviad Rubinstein*, Saeed Seddighin†, Zhao Song‡, Xiaorui Sun§

*Computer Science Department, Stanford University, Stanford, California, USA, aviad@cs.stanford.edu

†SEAS, Harvard University, Cambridge, Massachusetts, USA, saeedreza.seddighin@gmail.com

‡ Simons Institute for the Theory of Computing, Berkeley, California, USA, magic.linuxkde@gmail.com

§Computer Science Department, University of Illinois at Chicago, Chicago, Illinois, USA, xiaorui@uic.edu

***Abstract*—Longest common subsequence (LCS) is a classic and central problem in combinatorial optimization. While LCS admits a quadratic time solution, recent evidence suggests that solving the problem may be impossible in truly subquadratic time. A special case of LCS wherein each character appears at most once in every string is equivalent to the longest increasing subsequence problem (LIS) which can be solved in quasilinear time. In this work, we present novel algorithms for approximating LCS in truly subquadratic time and LIS in truly sublinear time. Our approximation factors depend on the ratio of the optimal solution size over the input size. We denote this ratio by λ and obtain the following results for LCS and LIS without any prior knowledge of λ.**

- **A truly subquadratic time algorithm for LCS with approximation factor $O(\lambda^3)$.**
- **A truly sublinear time algorithm for LIS with approximation factor $O(\lambda^3)$.**

Triangle inequality was recently used by Boroujeni *et al.* [1] and Chakraborty *et al.* [2] to present new approximation algorithms for edit distance. Our techniques for LCS extend the notion of triangle inequality to non-metric settings.

***Keywords*-approximation; LCS; LIS; subquadratic time;**

I. Introduction

In this paper, we consider three of the most classic problems in combinatorial optimization: the longest common subsequence (LCS), the edit distance (ED), and the longest increasing subsequence (LIS). The LCS of two strings A and B is simply their longest (not necessarily contiguous) common subsequence. The edit distance is defined as the minimum number of character deletions, insertions, and substitutions required to transform A into B. For the purpose of our discussion, we consider a more restricted definition of edit distance where substitutions are not allowed[1]. Longest increasing subsequence is equivalent to a special case of LCS where the input strings are permutations. All three problems are very fundamental and have been subject to a plethora of studies in the past few decades and specially in recent years [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [1], [2], [15], [16], [17].

If the strings have length n, both LCS and ED can be solved in quadratic time ($O(n^2)$) with dynamic programming. These running times are slightly improved to $O(n^2/\log^2(n))$ by Masek and Paterson [18], however, efforts to improve the running time to $O(n^{2-\Omega(1)})$ for either edit distance or LCS were all futile.

In recent years, our understanding of the source of complexity for these problems tremendously improved thanks to a sequence of fine-grained complexity developments [10], [11]. We now know that assuming the strong exponential time hypothesis (SETH) [10], or even weaker assumptions such as the orthogonal vectors conjecture (OVC) [10] or branching-program-SETH [11], there are no truly subquadratic[2] time algorithms for LCS. Similar results also hold for edit distance [9].

Indeed, the classic approach to break the quadratic barrier for these problems is approximation algorithms. Note that for (multiplicative) approximations, LCS and edit distance are no longer equivalent (much like we have a 2-approximation algorithm for Vertex Cover, but Independent Set is NP-hard to approximate within near-linear factors).

For edit distance, an $\widetilde{O}(n + \Delta^2)$-time algorithm of [3] (where Δ is the true edit distance between the strings) implies a linear-time $\sqrt{n}$-approximation algorithm. The approximation factor has been significantly improved in a series of works to $O(n^{3/7})$ [4], to $O(n^{0.34})$ [5], to $O(2^{\widetilde{O}(\sqrt{\log n})})$ [6][3], and finally to polylogarithmic [7]. A recent work of Boroujeni *et al.* [1] obtains a constant factor approximation quantum algorithm for edit distance that runs in truly subquadratic time. Finally, the breakthrough of Chakraborty *et al.* [2] gave a classic (randomized) constant factor approximation for edit distance in truly subquadratic time. A key component in both of the latest constant factor approximation algorithms is the application of triangle inequality (for edit distance between certain substrings of the input). A particular challenge in extending these ideas to LCS is that LCS is not a metric and in particular does

[1] Alternatively, the cost of a substitution is doubled as it requires a deletion and an insertion.

[2] By *truly sub-quadratic* we mean $O(n^{2-\epsilon})$, for any constant $\epsilon > 0$

[3] We define $\widetilde{O}(f)$ to be $f \cdot \log^{O(1)}(f)$.

2575-8454/19/$31.00 ©2019 IEEE
DOI 10.1109/FOCS.2019.00071

not satisfy the triangle inequality.

Our understanding of the complexity of approximate solutions for LCS is embarrassingly limited. For general strings, there are several linear-time $1/\sqrt{n}$-approximation algorithms based on sampling techniques. For alphabet size $|\Sigma|$, there is a trivial $1/|\Sigma|$-approximation algorithm that runs in linear time. Whether or not these approximation factors can be improved by keeping the running time subquadratic is one of the central problems in fine-grained complexity. Very recently, both the general $1/\sqrt{n}$-approximation factor, and, for binary strings, the $1/2$-approximation factor, have been slightly improved ([15] and [19], respectively). These works give improved algorithm for the two extreme cases where the size of the alphabet is very small or very large. In comparison, our approximation guarantee depends on the solution size rather than the size of the alphabet. Also, for the special case of balanced strings, we improve upon the result of [19] by obtaining an $o(|\Sigma|)$ approximate solution in subquadratic time. There are a few fine-grained complexity results for approximate LCS, but they only hold against deterministic algorithms, and rely on very strong assumptions [12], [13], [14].

A. *Our Results*

For simplicity, we use $\mathsf{lcs}(A, B)$ to denote the size (not the whole sequence) of the longest common subsequence for two strings A and B. Similarly, we use $\mathsf{ed}(A, B)$ to denote the edit distance and $\mathsf{lis}(A)$ for the size of the longest common subsequence. We sometimes normalize the solution by the length of the strings so that the size of the solution remains in the interval $[0, 1]$. We refer to the normalized solutions by $||\mathsf{lcs}(A, B)|| = \mathsf{lcs}(A, B)/n$ and $||\,\mathsf{ed}(A, B)|| = \mathsf{ed}(A, B)/2n$ (here both strings have equal length n), and $||\mathsf{lis}(A)|| = \mathsf{lis}(A)/n$. In this way, $||\,\mathsf{ed}(A, B)|| + ||\mathsf{lcs}(A, B)|| = 1$ (assuming both strings have equal length).

As mentioned earlier, recent developments for edit distance are based on a simple but rather useful observation. Edit distance satisfies triangle inequality, or in other words, given three strings A_1, A_2, A_3 of length n such that $||\,\mathsf{ed}(A_1, A_2)|| \leq \delta$ and $||\,\mathsf{ed}(A_2, A_3)|| \leq \delta$ hold, we can easily imply that $||\,\mathsf{ed}(A_1, A_3)|| \leq 2\delta$. While lcs does not satisfy the triangle inequality in any meaningful way, it does, *on average*, satisfy the following birthday-paradox-like property that we call *birthday triangle inequality*.

Property (birthday triangle inequality)**.** *Consider three equal-length strings A_1, A_2, and A_3 such that $||\mathsf{lcs}(A_1, A_2)|| \geq \lambda$ and $||\mathsf{lcs}(A_2, A_3)|| \geq \lambda$. If the common subsequences correspond to random indices of each string, we expect that $||\mathsf{lcs}(A_1, A_3)|| \geq \lambda^2$.*

Of course, this is not necessarily the case in general. More precisely, it is easy to construct examples[4] in which $||\mathsf{lcs}(A_1, A_2)|| = 1/2$ and $||\mathsf{lcs}(A_2, A_3)|| = 1/2$, but $||\mathsf{lcs}(A_1, A_3)|| = 0$. Our first main result shows that while it only holds on average, we can algorithmically replace the triangle inequality for edit distance with the birthday triangle inequality *on worst case inputs*.

Theorem (Main Theorem, formally stated as Theorem II.1)**.** *Given strings A, B both of length n such that $||\,\mathsf{lcs}(A, B)|| = \lambda$, we can approximate the length of the LCS between the two strings within an $O(\lambda^3)$ factor in subquadratic time. The approximation factor improves to $(1-\epsilon)\lambda^2$ when $1/\lambda$ is constant.*

We remark that our algorithm is actually able to output the whole sequence of the solution, but we only focus on estimating the size of the solution for simplicity. We begin by comparing our main theorem to previous work on edit distance. In this case, $1/\lambda$ is constant w.l.o.g.[5] and therefore the approximation factor of our algorithm is $(1-\epsilon)\lambda^2$. If $\delta = ||\,\mathsf{ed}(A, B)||$, then our LCS algorithm outputs a transformation from A to B using at most $2n(1-(1-\epsilon)(1-\delta)^3)$ operations. Observe that when the strings are not overly close and $\delta = \Omega(1)$ by scaling ϵ, we already recover a $(3+\epsilon')$-approximation for edit distance in truly subquadratic time. For mildly far strings, say $\delta = 0.1$, a more careful look at the expansion of $(1-\delta)^3$ reveals that we save an additive $\Theta(\delta^2)$ in the approximation factor. For example, with $\delta = 0.1$ our approximation factor for edit distance is 2.71 instead of 3.

An interesting implication of our main result is for LCS over a large alphabet Σ, where the optimum $||\,\mathsf{lcs}(A, B)||$ may be much smaller than 1. This is believed to be the hardest regime for approximation algorithms (and indeed the only one for which we have any conditional hardness of approximation results [12], [13], [14]). Here, we consider instances that satisfy a mild balance assumption: we assume that there is a character that appears with frequency at least $1/|\Sigma|$ in both strings[6]. Then, our main theorem implies an $O(1/|\Sigma|^{3/4})$-approximate solution in truly subquadratic time (the first improvement over the trivial $1/|\Sigma|$ approximation in this regime).

Corollary (LCS, formally stated as Corollary II.3)**.** *Given a pair of strings (A, B) of length n over alphabet Σ that satisfy the balance condition, we can approximate their LCS within an $O(|\Sigma|^{3/4})$ factor in truly subquadratic time.*

Next, we show that a similar result can be obtained for LIS. Perhaps coincidentally, the approximation factor of our algorithm is also $O(\lambda^3)$ which is same to LCS,

[4] For example, $A_1 = 0^{n/2}0^{n/2}, A_2 = 0^{n/2}1^{n/2}, A_3 = 1^{n/2}1^{n/2}$.

[5] When we use our solution to approximate edit distance, we can safely assume that $||\mathsf{lcs}(A, B)|| = \Omega(1)$ since otherwise the edit distance of the two strings is very close to $2n$.

[6] Note that in every instance in each string there is a character that appears with frequency at least $1/|\Sigma|$, but in general that may not be the same character.

but the technique is completely different. Although LIS can be solved exactly in time $O(n \log n)$, there have been several attempts to approximate the size of LIS and related problems in sublinear time [20], [21], [22], [23], [24], [25], [8]. The best known solution is due to the work of Saks and Seshadhri [8] that obtains a $(1+\epsilon)$-approximate algorithm for LIS in polylogarithmic time, when the solution size is at least a constant fraction of the input size [7]. In other words, if $||\mathsf{lis}(A)|| = \lambda$ and $1/\lambda$ is constant, their algorithm approximates $\mathsf{lis}(A)$ in polylogarithmic time. However, this only works if $1/\lambda$ is constant and even if $1/\lambda$ is logarithmically large, their method fails to run in sublinear time[8]. We complement the work of Saks and Seshadhri [8] by presenting a result for LIS similar to our result for LCS. More precisely, we show that when $||\mathsf{lis}(A)|| = \lambda$, an $O(\lambda^3)$ approximation of LIS can be obtained in truly sublinear time. Although our approximation factor is worse than that of [8], our result works for any (not necessarily constant) λ.

Theorem (LIS, formally stated as Theorem IV.8)**.** *Given an array A of n integer numbers such that $||\mathsf{lis}(A)|| = \lambda$. We can approximate the length of the LIS for A in sublinear time within a factor $O(\lambda^3)$.*

If one favors the running time over the approximation factor, it is possible to improve the exponent of n in the running time down to any constant $\kappa > 0$ at the expense of incurring a larger multiplicative factor to the approximation.

B. Preliminaries

In LCS or edit distance, we are given two strings A and B as input. We assume for simplicity that the two strings have equal length and refer to that by n. In LCS, the goal is to find the largest subsequence of the characters which is shared between the two strings. In edit distance, the goal is to remove as few characters as possible from the two strings such that the remainders for the two strings are the same. We use $\mathsf{lcs}(A,B)$ and $\mathsf{ed}(A,B)$ to denote the size of the longest common subsequence and the edit distance of two strings A and B.

In LIS, the input contains an array A of n integer numbers and the goal is to find a sequence of elements of A whose values (strictly) increase as their indices increase. For LIS, we denote the solution size for an array A by $\mathsf{lis}(A)$. We also use $\mathsf{lis}^{[\alpha,\beta]}(A)$ to denote the size of the longest increasing subsequence subject to elements whose values lie in range $[\alpha, \beta]$. Longest increasing subsequence is equivalent to LCS when the inputs are two permutations of n distinct characters.

Finally, we define a notation to denote the normalized solution sizes. For LCS, we denote the normalized solution size by $||\mathsf{lcs}(A,B)|| = \mathsf{lcs}(A,B)/\sqrt{|A||B|}$ for A and B and we use $||\,\mathsf{ed}(A,B)|| = \mathsf{ed}(A,B)/(2\sqrt{|A||B|})$ for edit distance. Note that, when the two strings have equal length we have $||\,\mathsf{ed}(A,B)|| + ||\mathsf{lcs}(A,B)|| = 1$. Similarly, for longest increasing subsequence, we denote by $||\mathsf{lis}(A)|| = \mathsf{lis}(A)/|A|$ the normalized solution size. We usually refer to the size of the input array by n.

Throughout this paper, we call an algorithm $f(\lambda)$-approximation for LCS if it is able to distinguish the following two cases: i) $||\mathsf{lcs}(A,B)|| \geq \lambda$ or ii) $||\mathsf{lcs}(A,B)|| < \lambda f(\lambda)$. A similar definition carries over to LIS. Once an $f(\lambda)$-approximation algorithm is provided for either LCS or LIS, one can turn it into an algorithm that outputs a solution with size $f(\lambda)(1-\epsilon)\lambda n$ provided that the optimal solution has a size λn. The algorithm is not aware of the value of λ but will start with $\lambda_0 = 1$ and iteratively multiply λ_0 by $1-\epsilon$ until a solution is found.

C. Techniques Overview

Our algorithm for LCS is closely related to the recent developments for edit distance [1], [2]. We begin by briefly explaining the previous ideas for approximating edit distance and then we show how we use these techniques to obtain a solution for LCS. Finally, in Section I-C2 we outline our algorithm for LIS.

1) Summary of Previous ED Techniques: Indeed, edit distance is hard only if the two strings are far ($||\,\mathsf{ed}(A,B)|| = \delta$ and $\delta = n^{-o(1)}$) otherwise the $O(n + (n\delta)^2)$ algorithm of Landau *et al.* [3] computes the solution in truly subquadratic time. The algorithm of Chakraborty *et al.* [2] for edit distance has three main steps that we briefly discuss in the following.

Step 0 (window-compatible solutions):: In the first step, they construct a set of windows W_A for string A and a set of windows W_B for string B. Each window is essentially a substring of the given string. Let k denote the total number of windows of $W_A \cup W_B$. For simplicity, let all the windows have the same size d and $n \simeq O(kd)$[9]. The construction features two key properties: 1) provided that the edit distances of the windows between W_A and W_B are available, one can recover a $1+\epsilon$ approximation of edit distance in time $\widetilde{O}(n+k^2)$ via dynamic programming. 2) $k^2 \times d^2 \simeq O(n^2)$. That is, if we naively compute the edit distance of every pair of windows, the overall running time would still asymptotically be the same as that of the classic algorithm.

In order to obtain a solution for edit distance, it suffices to know the distances between the windows. However, Chakraborty *et al.* [2] show that knowing the distances between most of the window pairs is enough to obtain an approximately optimal solution for edit distance. More precisely, if the distances are not correctly approximated for

[7] Their algorithm obtains an additive error of δn in time $2^{\widetilde{O}(1/\delta)}$. When the solution size is bounded by λn, one needs to set $\delta < \lambda$ in order to guarantee a multiplicative factor approximation.

[8] There is a term $(1/\lambda)^{1/\lambda}$ in the running time.

[9] The equality holds if we assume $\delta = \Omega(1)$.

at most $O(k^{2-\Omega(1)})$ pairs, we can still obtain an approximate solution for edit distance. Step 1 provides estimates for the distances of the windows which is approximately correct except for $O(k^{2-\Omega(1)})$ many pairs and Step 2 shows how this can be used to obtain a solution for edit distance. Discretization simplifies the problem substantially. For a fixed $0 \leq \delta \leq 1$, they introduce a graph G_δ where the nodes correspond to the windows and an edge between window $w_i \in W_A$ and window $w_j \in W_B$ means that $||\,\mathsf{ed}(w_i, w_j)|| \leq \delta$. If we are able to construct G_δ for logarithmically different choices of δ, we can as well estimate the distances within a $1+\epsilon$ factor for the windows. Therefore the problem boils down to constructing G_δ for a fixed given δ without computing the edit distance between all pairs of windows.

Step 1 (sparsification via triangle inequality):: This step is the heart of the algorithm. Suppose we choose a high-degree vertex v from G_δ and discover all its incident edges by computing its edit distance to the rest of the windows. Triangle inequality implies that every pair of windows in $N(v)$ has a distance bounded by 2δ. Therefore by losing a factor 2 in the approximation, one can put all these edges in G_δ and not compute the edit distances explicitly. Although this does save some running time, in order to make sure the running time is truly subquadratic, we need to make a similar argument for paths of length 3 and thereby lose a factor 3 in the approximation. This method sparsifies the graph and what remains is to discover the edges of a sparse graph.

Step 2 (discovering the edges of the sparse graph):: Step 1 uses triangle inequality and discovers many edges between the vertices of G_δ. However, it may not discover all the edges completely. When in the remainder graph, the degrees are small (and hence the graph is sparse) triangle inequality does not offer an improvement and thus a different approach is required. Roughly speaking, Chakraborty *et al.* [2] subsample the windows of W_A into a smaller set S and discover all pairs of windows $w_i \in S$ and $w_j \in W_B$ such that edge (i, j) is not discovered in Step 1. Next, they compute the edit distance of each pair of windows $(w_i, w_j), w_i \in W_A, w_j \in W_B$ such that there exist two nearby windows (w_a, w_b) satisfying $w_a \in S, w_b \in W_B$ and the edge between w_a and w_b was missed in Step 1. The key observation is that even though this procedure does not discover all the edges, the approximated distances lead to an approximate solution for edit distance.

2) LCS: Our algorithm for LCS mimics the same guideline. In addition to this, Steps 0 and 2 of our algorithm are LCS analogues of the ones used by Chakraborty *et al.* [2]. The main novelty of our algorithm is Step 1 which is a replacement for triangle inequality. Recall that unlike edit distance, triangle inequality does not hold for LCS.

Challenge I.1. *How can we introduce a notion similar to triangle inequality to a non-metric setting such as LCS?*

We introduce the notion of birthday triangle inequality to overcome the above difficulty. Given windows w_1, w_2, and w_3 of size d such that $||\mathsf{lcs}(w_1, w_2)|| \geq \lambda$ and $||\mathsf{lcs}(w_2, w_3)|| \geq \lambda$ hold, what can we say about the LCS of w_1 and w_3? In general, nothing! $||\mathsf{lcs}(w_1, w_3)||$ could be as small as 0. However, let us add some randomness to the setting. Think of the LCS of w_1 and w_2 as a matching from the characters of w_1 to w_2 and similarly the LCS for w_2 and w_3 as another matching between characters of w_2 and w_3. Assume (for the sake of the thought experiment) that the characters of w_2 appear randomly in each matching. Since $||\mathsf{lcs}(w_1, w_2)|| \geq \lambda$, each character of w_2 appears with probability at least λ in the matching between w_1 and w_2. A similar argument implies that each character of w_2 appears with probability λ in the matching of w_2 and w_3. Thus, (assuming independence), each character of w_2 appears in both matchings with probability λ^2. This means that in expectation, there are $\lambda^2 d$ paths of length 2 between w_1 and w_3 which suggests $||\mathsf{lcs}(w_1, w_3)|| \geq \lambda^2$ as shown in Figure 1. This is basically birthday paradox used for the sake of triangle inequality.

Replacing triangle inequality by birthday triangle inequality is particularly challenging since birthday triangle inequality only holds on average. In contrast, triangle inequality holds for any tuple of arbitrary strings. Most of our technical discussions is dedicated to proving that we can algorithmically use birthday triangle inequality to obtain a solution for the worst-case scenarios. The most inconvenient step of our analysis is to show that our algorithm estimates the LCS of most window pairs in the sparsification phase. While this is straightforward for edit distance, birthday triangle inequality requires a deeper analysis of the underlying graph. In particular, we need to prove that if the undiscovered edges are too many, then birthday triangle inequality can be applied to certain neighborhoods of the graph.

There are two difficulties that we face here. On one hand, in order to apply birthday triangle inequality to a subgraph, we need to have enough structure for that subgraph to show the implication can be made. On the other hand, our assumptions cannot be too strong, otherwise such neighborhoods may not cover the edges of the graph. Therefore, the first challenge that we need to overcome is characterizing subgraphs in which birthday triangle inequality is guaranteed to be applicable. Our suggestion is the *bi-cliques* structure. Although combinatorial techniques seem unlikely to prove this, we use the Blakley-Roy inequality to show that in a large enough bi-clique, we can use birthday triangle inequality to imply a bound on the LCS of certain pairs. The second challenge is to prove that if the underlying graph is dense enough, the graph contains many bi-cliques that cover almost all the edges that we plan to discover. This is again a challenging graph theoretic problem. We leverage extremal

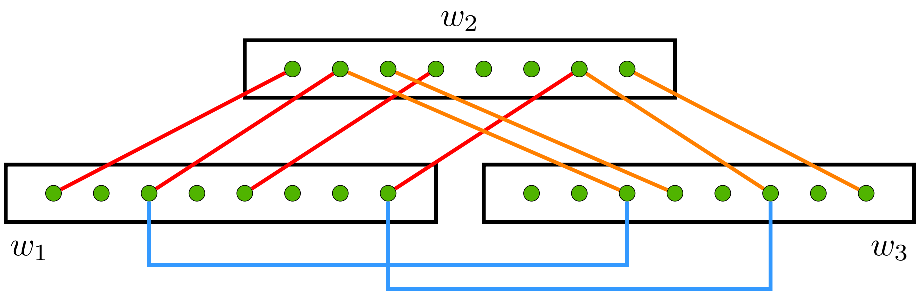


Figure 1: Birthday paradox for triangle inequality: let w_1, w_2, w_3 be three windows of length $d = 8$ and assume $\lambda = 1/2$. The LCS between w_1 and w_2 is $\lambda d = 4$ and the LCS between w_2 and w_3 is $\lambda d = 4$. Finally due to birthday paradox, we expect that the LCS between w_1 and w_3 is $\lambda^2 d = 2$.

graph theory tools such as Turan's theorem for cliques and bi-cliques to obtain this bound.

Similar to edit distance, we construct a set $W = W_A \cup W_B$ of k windows in Step 0 and aim to sparsify the edges of the lcs-graph in Step 1. Our construction ensures that $kd \simeq \Theta(n)$ and that knowing the LCS of the window pairs suffices to approximate the LCS of the two strings. For a threshold $0 \leq \lambda \leq 1$, define a matrix $\mathsf{O} : [k] \times [k] \to \{0,1\}$ to be a matrix which identifies whether $||\mathsf{lcs}(w_i, w_j)|| \geq \lambda$. In other words, $\mathsf{O}[i][j] = 1 \iff ||\mathsf{lcs}(w_i, w_j)|| \geq \lambda$. For an $0 < \alpha \leq 1$, we call a matrix O_α an α approximation of O if it meets the following two conditions:

$$\mathsf{O}_\alpha[i][j] = 0 \implies \|\mathsf{lcs}(w_i, w_j)\| < \lambda$$

and

$$\mathsf{O}_\alpha[i][j] = 1 \implies \|\mathsf{lcs}(w_i, w_j)\| \geq \alpha \cdot \lambda$$

Notice that O_α gives more flexibility than O for the cases that $\lambda\alpha \leq \mathsf{lcs}(w_i, w_j) < \lambda$. That is, both 0 and 1 are acceptable in these cases. Indeed an α approximation algorithm for the above problem is enough to obtain an α approximation algorithm for LCS. However, this is not necessary as Step 2 allows for incorrect approximation for up to $k^{2-\Omega(1)}$ many window pairs. Therefore, the problem of approximating LCS essentially boils down to approximating O for a given basket of windows $W = W_a \cup W_b$ and a fixed λ by allowing sufficiently small error in the output. A naive solution is to iterate over all pairs w_i and w_j and compute $\mathsf{lcs}(w_i, w_j)$ in time $O(d^2)$ and determine O accordingly. However, this amounts to a total running time of $O(k^2 d^2)$ which is quadratic and not desirable. In order to save time, we need to compute the LCS of fewer than k^2 pairs of windows. To make this possible, we allow our algorithm to miss up to $O(k^{2-\Omega(1)})$ edges of the graph. Step 2 ensures that this does not hurt the approximation factor significantly.

We construct a graph from the windows wherein each vertex corresponds to a window and each edge identifies a pair with a large LCS (in terms of λ). Let us call this graph the lcs-graph and denote it by G_λ. The goal is to detect the edges of the graph by allowing false-positive. As we discussed earlier, the hard instances of the problem are the cases where the lcs-graph is dense for which we need a sparsifier. Roughly speaking, in our sparsification technique, our algorithm constructs another graph $\widehat{\mathsf{G}}_\lambda$ such that $\widehat{\mathsf{G}}_\lambda$ is valid in the sense that the edges of $\widehat{\mathsf{G}}_\lambda$ correspond to pairs of windows with large enough LCS. In addition to this, our algorithm guarantees that after the removal of the edges of $\widehat{\mathsf{G}}_\lambda$ from G_λ the remainder is sparse. In other words, $|E(\mathsf{G}_\lambda) \setminus E(\widehat{\mathsf{G}}_\lambda)| = O(|V(\mathsf{G}_\lambda)|^{2-\Omega(1)})$. Of course, if the overall running time of the sparsification phase is truly subquadratic, the error of undiscovered edges can be addressed by the techniques of [2] in Step 2.

Below, we bring a formal definition for sparsification.

sparsification
input: Windows $w_1, w_2, \ldots, w_k$, parameters λ, and α.
solution: A matrix $\widehat{\mathsf{O}}_\alpha \in \{0,1\}^{k \times k}$ such that:

- $\widehat{\mathsf{O}}_\alpha[i][j] = 1 \implies ||\mathsf{lcs}(w_i, w_j)|| \geq \alpha \cdot \lambda$
- $\left|\left\{(i,j) \mid ||\mathsf{lcs}(w_i, w_j)|| \geq \lambda \text{ and } \widehat{\mathsf{O}}_\alpha[i][j] = 0\right\}\right| = k^{2-\Omega(1)}$

We present two sparsification techniques for LCS. The first one (Section III-A), has an approximation factor of $(1-\epsilon)\cdot\lambda^2$. In Section III-B we present another sparsification technique that has a worse approximation factor $O(\lambda^3)$ but leaves fewer edges behind. Although the second sparsification technique has a worse approximation factor, it has the advantage that the number of edges that remain in the sparse graph is truly subquadratic regardless of the value of λ and therefore it extends our solution to the case that $\lambda = o(1)$ (see Section III-B for a detailed discussion).

Let us note one last algorithmic challenge to keep in mind before we begin to describe our sparsification techniques. For edit distance, if window pairs (w_1, w_2) and (w_2, w_3) are close, we are *guaranteed* that w_1 and w_3 are also close; for longest common subsequence, we will argue that (w_1, w_3) are *likely* to be have a long LCS (for a "random" choice of (w_1, w_3)). Nonetheless, in order to add (w_1, w_3) as an edge to our graph we have to *verify* that their LCS is indeed long.

If we were to verify an edge naively, we would need as much time as computing the LCS between (w_1, w_3) from scratch!

Sparsification 1, $(1-\epsilon)\lambda^2$-approximation:: Similar to edit distance, applying birthday triangle inequality to paths of length 2 for LCS does not improve the running time significantly. Therefore, we need to use birthday triangle inequality for paths of length 3. To this end, we define the notion of constructive tuples as follows: a tuple $\langle w_i, w_a, w_b, w_j\rangle$ is an (ϵ, λ)-constructive tuple, if we have $||\mathsf{lcs}(w_i, w_a)|| \geq \lambda$, $||\mathsf{lcs}(w_i, w_j)|| \geq \lambda$, $||\mathsf{lcs}(w_b, w_j)|| \geq \lambda$ and by taking the intersection of the three LCS matchings, we are able to imply $||\mathsf{lcs}(w_a, w_b)|| \geq (1-\epsilon)\lambda^3$ (see Figure 2 for an example). Taking the intersection of the matchings can be done in linear time which is faster than computing the LCS.

Our sparsification technique here is simple but the analysis is very intricate. We subsample a set S of windows and compute the LCS of every window in S and all other windows. We set $|S| = k^\gamma \log k$, where $\gamma \in (0,1)$. At this point, for some pairs, we already know their LCS. However, if neither w_i nor w_j is in S, we do not know if $||\mathsf{lcs}(w_i, w_j)|| \geq \lambda$ or not. Therefore, for such pairs, we try to find windows $w_a, w_b \in S$ such that $\langle w_i, w_a, w_b, w_j\rangle$ is constructive. If such a constructive tuple is found for a pair of windows, then we conclude that their normalized LCS is at least $(1-\epsilon)\lambda^3$.

All that remains is to argue that this method discovers almost all the edges of the lcs-graph G_λ and the number of undiscovered edges is $k^{2-\Omega(1)}$. This is the most difficult part of the analysis. We note that proving the existence of **only one** constructive tuple is already non-trivial **even when G_λ is complete**. However, our goal is to show almost all the edges are discovered via constructive tuples when G_λ is dense (and of course not necessarily complete).

Define a pair of windows w_i and w_j to be *well-connected*, if there are at least $k^{2-\gamma}$ different (w_a, w_b) pairs such that $\langle w_i, w_a, w_b, w_j\rangle$ is (ϵ, λ)-constructive. Since each window appears in S with probability $k^{\gamma-1}\log k$, for each well-connected pair we find one constructive tuple via our algorithm with high probablity. Therefore, we need to prove that the total number of pairs (w_i, w_j) such that (w_i, w_j) is not well-connected but $||\mathsf{lcs}(w_i, w_j)|| \geq \lambda$ is subquadratic. Let us put these edges in a new graph NG_λ whose vertices are all the windows. We first leverage the Blakley-Roy inequality and a double counting technique to prove that if NG_λ has a large complete bipartite subgraph, then there is one constructive tuple which includes only the vertices of this subgraph (Lemma III.5). Next, we apply the Turan's theorem to show that if NG_λ is dense, then it has a lot of large complete bipartite subgraphs. Finally, we use a probabilistic method to conclude that NG_λ cannot be too dense otherwise there are a lot of constructive tuples in the graph which implies that at least one edge (w_i, w_j) in NG_λ is well-connected. This is not possible since all the well-connected pairs are detected in our sparsification algorithm with high probability.

The above argument proves that if we sparsify our graph using our sparsification algorithm, the remainder graph would have a subquadratic number of edges. Therefore after plugging Step 2 into the algorithm, the running time remains subquadratic. However, since Turan theorem gives us a weak bound, the running time of the algorithm using this sparsification is $O(n^{2-\Omega(\lambda)})$ and is only truly subquadratic when $1/\lambda$ is constant.

Sparsification 2, $O(\lambda^3)$-approximation:: In Section III-A, we present another sparsification method that although gives us a slightly worse approximation factor $O(\lambda^3)$ it always leaves a truly subquadratic number of edges behind and therefore the running time of the algorithm would be truly subquadratic regardless of the parameter λ. This sparsification is based on a novel data structure.

Let $\mathsf{opt}_{i,a}$ denote the longest common subsequence of w_i and w_a (with some fixed tie-breaking rule, e.g. lexicographically first). Define $\mathsf{lcs}_{w_a}(w_i, w_j)$ to be the size of the longest common subsequence between $\mathsf{opt}_{i,a}$ and w_j. Notice that this definition is no longer symmetric. Let $||\mathsf{lcs}_{w_a}(w_i, w_j)||$ denote the relative value, i.e., $||\mathsf{lcs}_{w_a}(w_i, w_j)|| = \mathsf{lcs}_{w_a}(w_i, w_j)/\sqrt{|w_i| \cdot |w_j|}$. The first ingredient of the algorithm is a data-structure, namely $\mathsf{lcs\text{-}cmp}$. After a preprocess of time $O(|w_a|\sum_{i\in S}|w_i|)$, $\mathsf{lcs\text{-}cmp}$ is able to answer queries of the following type in time $O(|w_i| + |w_j|)$:

- "for a $0 \leq \widetilde{\lambda} \leq 1$ either certify that $\|\mathsf{lcs}_{w_a}(w_i, w_j)\| \geq \Omega(\widetilde{\lambda}^2)$ or report that $\|\mathsf{lcs}_{w_a}(w_i, w_j)\| < O(\widetilde{\lambda})$".

In our sparsification, we repeat the following procedure k^γ times, where $\gamma \in (0,1)$. We sample a window w_a uniformly at random and construct $\mathsf{lcs\text{-}cmp}(w_a, S)$ for $S = \{w_i | i \neq a \text{ and } |w_i| \geq |w_a|\}$. After the preprocessing step, we make a query for every pair of windows (w_i, w_j) such that $w_i, w_j \in S$ and determine if $\mathsf{lcs}_{w_a}(w_i, w_j)$ is at least $\Omega(\lambda^4)$ or upper bounded by $O(\lambda^2)$ (here $\widetilde{\lambda} = \lambda^2$). If their LCS is at least $\Omega(\lambda^4)$ we report this pair as an edge in our lcs-graph. Finally, we use the Turan theorem to prove that the number of remaining edges in our graph is small.

To be more precise, we first construct a graph NG_λ that reflects the edges that are not detected via our sparsification. We give directions to the edges based on the length of the windows. If NG_λ is dense enough, then there is one vertex v in NG_λ with a large enough outgoing degree. We use the neighbors of v to construct another graph NF_λ with vertex set $N(v)$. An edge exists in NF_λ if $\max\{\|\mathsf{lcs}_{w_v(w_i,w_j)}\|, \|\mathsf{lcs}_{w_v(w_j,w_i)}\|\} \geq \Omega(\lambda^2)$. Edges of NF_λ are directed the same way as NG_λ. We prove that NF_λ has no large independent set. In other words, if we select a large enough set of vertices in NF_λ, then there is at least one edges between them. Next, we apply the Turan theorem to prove that NF_λ is dense. Finally, we imply that since

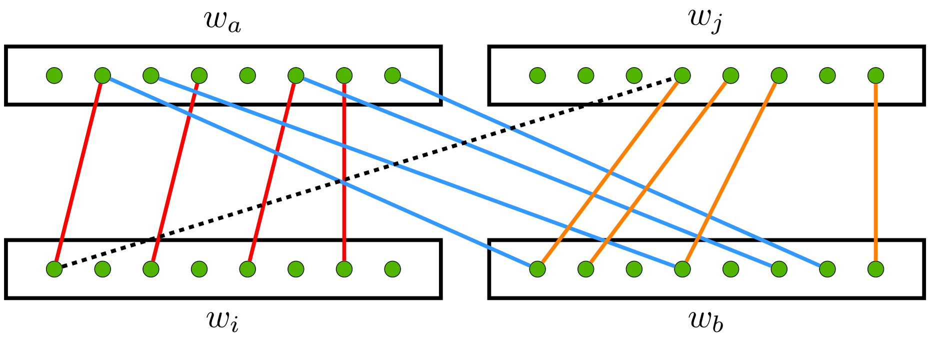


Figure 2: Let w_i, w_a, w_b and w_j denote four windows and each of them has length $d = 8$. This figure shows how the intersection of the edges of three windows are taken in order to construct a solution for the LCS of w_i and w_j. If the size of the intersection is large, then such a tuple is called constructive. The solid lines represent LCS between two strings, and the dashed line represents the intersection of the three LCSs.

NF_λ is dense, there is one vertex u in the neighbors of v such that there are a lot of 2-paths between v and u. This implies that the edge (u, v) should have been detected in our sparsification and therefore must not exist in NG_λ. This contradiction implies that NG_λ is sparse in the first place.

3) LIS: In this section, we present our result for longest increasing subsequence. More precisely, we show that when the solution size is lower bounded by $n\lambda$ ($\lambda \in [0, 1]$), one can approximate the solution within a factor $O(\lambda^3)$ in time $\widetilde{O}(\sqrt{n}/\lambda^7)$. This married with a simple sampling algorithm for the cases that $\lambda < n^{-\Omega(1)}$, provides an $O(\lambda^3)$-approximate algorithm with running time of $\widetilde{O}(n^{0.85})$ (without further dependence on λ). We further extend this result to reduce the running time to $\widetilde{O}(n^\kappa \mathsf{poly}(1/\lambda))$ for any $\kappa > 0$ by imposing a multiplicative factor of $\mathsf{poly}(1/\lambda)$[10] to the approximation.

Our algorithm heavily relies on sampling random elements of the array for which longest increasing subsequence is desired. Denote the input sequence by $A = \langle a_1, a_2, \ldots, a_n \rangle$. A naive approach to approximate the solution is to randomly subsample the elements of A to obtain a smaller array B and then compute the longest increasing subsequence of B to estimate the solution size for A. Let us first show why this approach alone fails to provide a decent approximation factor. First, consider an array $A = \langle 1, 2, \ldots, n \rangle$ which is strictly increasing. Based on A, we construct two inputs A' and A'' in the following way:

- A' is exactly equal to A except that a p fraction of the elements in A' are replaced by 0.
- A'' is exactly equal to A except that every block of length $\sqrt{n}$ is reversed in A''. In other words, $A'' = \langle \sqrt{n}, \sqrt{n}-1, \sqrt{n}-2, \ldots, 1, 2\sqrt{n}, 2\sqrt{n}-1, \ldots, \sqrt{n}+1, \ldots, n, n-1, n-2, \ldots, n-\sqrt{n}+1 \rangle$.

We subsample the two arrays A' and A'' with a rate of $1/\sqrt{n}$ to obtain two smaller arrays B' and B'' of size roughly $O(\sqrt{n})$. It is easy to prove that $\mathsf{lis}(B') = \Omega(\sqrt{n})$ and $\mathsf{lis}(B'') = \Omega(\sqrt{n})$ both hold even though $\mathsf{lis}(A') = \Omega(n)$ but $\mathsf{lis}(A'') = O(\sqrt{n})$. By setting $p = 1/e$[11] we can also make sure that $\mathsf{lis}(B')$ and $\mathsf{lis}(B'')$ are within a small multiplicative range even though the gap between $\mathsf{lis}(A')$ and $\mathsf{lis}(A'')$ is substantial.

The above observation shows that the problem is very elusive when random sampling is involved. We bring a remedy to this issue in the following. Divide the input array into $\sqrt{n}$ subarrays of size $\sqrt{n}$. We denote the subarrays by $\mathsf{sa}_1, \mathsf{sa}_2, \ldots, \mathsf{sa}_{\sqrt{n}}$ and fix an optimal solution opt for the longest increasing subsequence of A. Define $\mathsf{sm}(\mathsf{sa}_i)$ to be the smallest number in sa_i that contributes to opt and $\mathsf{lg}(\mathsf{sa}_i)$ to be the largest number in sa_i that contributes to opt. Moreover, define $\mathsf{lis}^{[\ell,r]}$ to be the longest increasing subsequence of an array subject to the elements whose values lie within the interval $[\ell, r]$. This immediately implies

$$\mathsf{lis}(A) = \sum_{i=1}^{\sqrt{n}} \mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i).$$

Another observation that we make here is that since we assume $||\mathsf{lis}(A)|| \geq \lambda$ and the size of each subarray is bounded by $\sqrt{n}$, then we have

$$\frac{\mathsf{lis}(A)}{\max_i \mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i)} \geq \sqrt{n}\lambda$$

which essentially means that in order to approximate $\mathsf{lis}(A)$ it suffices to compute $\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i)$ for $\widetilde{O}(1/\lambda)$ many randomly sampled subarrays. This is quite helpful since this shows that we only need to sample $\widetilde{O}(1/\lambda)$ many subarrays and solve the problem for them. However, we do not know the values of $\mathsf{sm}(\mathsf{sa}_i)$ and $\mathsf{lg}(\mathsf{sa}_i)$ in advance. Therefore, the

[10] The exponent of $1/\lambda$ depends exponentially on $1/\kappa$.

[11] $e \simeq 2.7182$.

main challenge is to predict the values of $\mathsf{sm}(\mathsf{sa}_i)$ and $\mathsf{lg}(\mathsf{sa}_i)$ before we sample the subarrays.

Indeed, one needs to read the entire array to correctly compute $\mathsf{sm}(\mathsf{sa}_i)$ and $\mathsf{lg}(\mathsf{sa}_i)$ for each of the subarrays. However, we devise a method to approximately guess these values without losing too much in the size of the solution. Roughly speaking, we show that if we sample $k = O(1/(\lambda\epsilon))$ different elements from a subarray sa_i for some constant ϵ and denote them by $a_{j_1}, a_{j_2}, \ldots, a_{j_k}$, then for at least one pair (α, β), $[a_{j_\alpha}, a_{j_\beta}]$ is approximately close to $[\mathsf{sm}(\mathsf{sa}_i), \mathsf{lg}(\mathsf{sa}_i)]$ up to a $(1-\epsilon)$ factor.

The above argument provides $O((1/(\lambda\epsilon))^2)$ candidate domain intervals for each sa_i. However, this does not provide a solution since we do not know which candidate domain interval approximates $[\mathsf{sm}(\mathsf{sa}_i), \mathsf{lg}(\mathsf{sa}_i)]$ for each sa_i. Of course, if we were to randomly choose one candidate interval for every subarray, we would make a correct guess for at least $O(\sqrt{n}(\lambda\epsilon)^2)$ subarrays which provides an approximation guarantee of $O(\lambda^2)$ for our algorithm. However, our assignments have to be monotone too. More precisely, let $[\widetilde{\mathsf{sm}}(\mathsf{sa}_i), \widetilde{\mathsf{lg}}(\mathsf{sa}_i)]$ be the guesses that our algorithm makes, then we should have

$$\begin{aligned}\widetilde{\mathsf{sm}}(\mathsf{sa}_1) \leq \widetilde{\mathsf{lg}}(\mathsf{sa}_1) &\leq \widetilde{\mathsf{sm}}(\mathsf{sa}_2) \leq \widetilde{\mathsf{lg}}(\mathsf{sa}_2)\\ &\leq \cdots\\ &\leq \widetilde{\mathsf{sm}}(\mathsf{sa}_{\sqrt{n}}) \leq \widetilde{\mathsf{lg}}(\mathsf{sa}_{\sqrt{n}}).\end{aligned}$$

Random sampling does not guarantee that the sampled intervals are monotone. To address this issue, we introduce the notion of *pseudo-solutions*. A pseudo-solution is an assignment of monotone intervals to subarrays in order to approximate $\mathsf{sm}(\mathsf{sa}_i)$ and $\mathsf{lg}(\mathsf{sa}_i)$. The *quality* of a pseudo solution with intervals $[\ell_1, r_1], [\ell_2, r_2], \ldots, [\ell_{\sqrt{n}}, r_{\sqrt{n}}]$ is equal to $\sum_i \mathsf{lis}^{[\ell_i, r_i]}(\mathsf{sa}_i)$. For a fixed pseudo-solution, this can be easily approximated via random sampling. Thus, our goal is to construct a pseudo-solution whose quality is at least an $O(\lambda^3)$ approximation of the size of the optimal solution. To this end, we present a greedy method in Section IV-B to construct the desired pseudo-solution.

Finally, in Section IV-D, we show how the above ideas can be generalized to improve the running time down to $\widetilde{O}(n^\kappa \mathsf{poly}(1/\lambda))$ for any arbitrarily small $\kappa > 0$ by imposing a factor $\mathsf{poly}(1/\lambda)$ to the approximation guarantee.

II. Organization of the Paper

Our algorithm for LCS contains 3 steps (Steps 0-2) that are explained in details in the full-version. Since Steps 0 and 2 follow from previous work, we only bring Step 1 in this version and defer the rest to the full-version.

In both our results for LCS and LIS, we assume that the goal is to find approximate solutions, provided that the solution size is at least $\lambda_0 n$. After the algorithms terminate, if the output is smaller than what we expect, we realize that the solution is smaller than $\lambda_0 n$. Therefore, we begin by setting $\lambda_0 = 1$ and iteratively multiply λ_0 by a $1-\epsilon$ factor until we obtain a solution. This only adds a multiplicative factor of $\log 1/\lambda$ to the running time and a multiplicative factor of $1-\epsilon$ to the approximation. Since we present two different sparsification techniques, we obtain two theorems: one is Theorem II.1 and the other is Theorem II.2.

Theorem II.1. *Given strings A, B of length $|A| = |B| = n$ with $||\mathsf{lcs}(A, B)|| = \lambda$, we can approximate the length of the LCS between the two strings within a factor $O(\lambda^3)$ in time $\widetilde{O}(n^{39/20})$.*

Proof: If $\lambda \leq n^{-20}$ we run the classic $O(n^2\lambda)$ time algorithm and get an exact solution in the desired time. Otherwise, we begin by setting $\lambda_0 = 1$ and iteratively multiply λ_0 by a factor $1-\epsilon$ until a solution is found with size at least $\Omega(\lambda^3)n$. This adds an overhead of $O(\log 1/\lambda)$ to the running time. By Choosing $d = \sqrt{n}$ we can bound the total running time of Steps 0, 1, and 2 by $\widetilde{O}(n^{13/10}\lambda^{-13}) \leq \widetilde{O}(n^{39/20})$. ■

Theorem II.2. *Given strings A, B of length $|A| = |B| = n$ with $||\mathsf{lcs}(A, B)|| = \lambda$, we can approximate the length of the LCS between the two strings within a factor $(1-\epsilon)\lambda^2$ in $n^{2-\Omega(\epsilon\lambda)}$ time for any $\epsilon > 0$.*

Proof: The proof is identical to Theorem II.1, we omit the details here. ■

As an immediate corollary of Theorem II.1, we present an algorithm that beats the $1/|\Sigma|$ approximation factor in truly subquadratic time, when the strings are balanced.

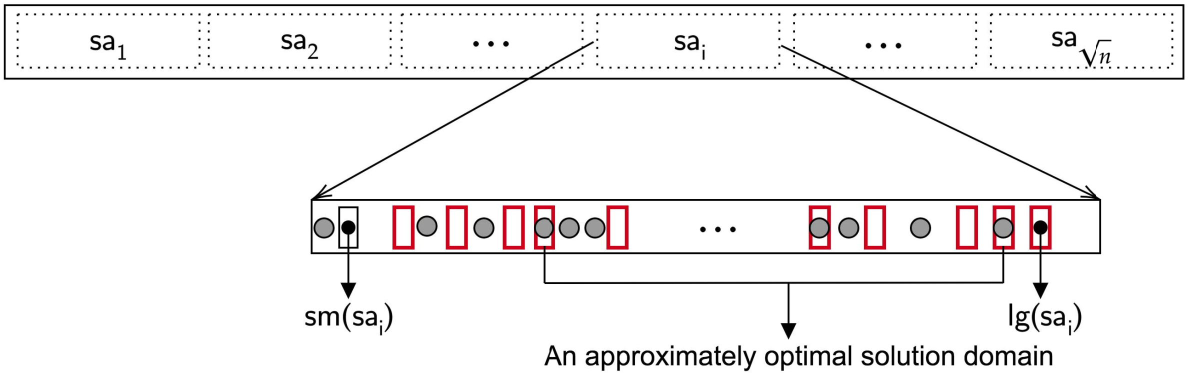
Corollary II.3. *Given a pair of strings (A, B) of length n over alphabet Σ that satisfy the balance condition, we can approximate their LCS within an $O(|\Sigma|^{3/4})$ factor in time $O(n^{39/20})$.*

Proof: Since A and B are balanced, there is a character $\sigma \in \Sigma$ that appears at least $n/|\Sigma|$ times in both strings. Indeed, finding a solution of size $n/|\Sigma|$ by restricting our attention to only character σ can be done in time $O(n)$. If $\mathsf{lcs}(A, B) \leq n/|\Sigma|^{1/4}$ this already gives us an $O(|\Sigma|^{3/4})$ approximate solution. Otherwise, $||\mathsf{lcs}(A, B)|| > 1/|\Sigma|^{1/4}$ and the approximation factor of our $O(\lambda^3)$-approximation algorithm would be bounded by $O(|\Sigma|^{3/4})$. ■

Finally, we bring our results for LIS in Section IV. We show that

Theorem II.4. *Given a length-n sequence A with $\mathsf{lis}(A) = n\lambda$. We can approximate the length of the LIS within a factor of $O(\lambda^3)$ in time $\widetilde{O}(n^{17/20})$.*

Proof: If $\lambda < n^{-1/20}$ we sample the array with a rate of $n^{-3/20}$ and compute the LIS for the sampled array. The running time of the algorithm is $\widetilde{O}(n^{17/20})$. The approximation factor is $O(n^{-3/20}) \geq O(\lambda^3)$. Otherwise, by Theorem IV.8, we estimate the size of LIS up to an $O(\lambda^3)$



sa_1
sa_2
$\cdots$
sa_i
$\cdots$
$\mathsf{sa}_{\sqrt{n}}$
$\cdots$
$\mathsf{sm}(\mathsf{sa}_i)$
$\mathsf{lg}(\mathsf{sa}_i)$
An approximately optimal solution domain

Figure 3: Red rectangles show the elements of sa_i that contribute to $\mathsf{lis}(A)$ and gray circles show the elements of sa that are sampled via our algorithm.

approximation factor in time $\widetilde{O}(\lambda^{-7}\sqrt{n}) \leq \widetilde{O}(n^{17/20})$. ■

III. LCS Step 1: Sparsification via Birthday Triangle Inequality

Recall that we are given two sets of windows W_A and W_B for the strings and our goal is to approximate the LCS of every pair of windows between W_A and W_B. For simplicity, we put all the windows in the same basket $W = W_A \cup W_B$ and denote the windows by $w_1, w_2, \ldots, w_k$ where k is the total number of windows. Since the windows have different lengths, we define $\mathsf{w}_{\mathsf{max}} = \max_{i \in [k]} |w_i|$ to be the maximum length of the windows. Similarly, we also define $\mathsf{w}_{\mathsf{min}} = \min_{i \in [k]} |w_i|$ to be the minimum length of the windows. Let $\mathsf{w}_{\mathsf{gap}} = \mathsf{w}_{\mathsf{max}}/\mathsf{w}_{\mathsf{min}}$. Let $\mathsf{w}_{\mathsf{layers}}$ denote the number of different window sizes. Notations $\mathsf{w}_{\mathsf{gap}}$ and $\mathsf{w}_{\mathsf{layers}}$ will be used in the later analysis.

In order to approximate the LCS's we fix a $\lambda \in \{\epsilon\lambda_0, (1+\epsilon)\epsilon\lambda_0, (1+\epsilon)^2\epsilon\lambda_0, \ldots, 1\}$ and sparsify graph G_λ. In Section III-A, we present a sparsification algorithm (Algorithm 1) which provides λ^2-approximation when λ is constant. The formal guarantee of the algorithm is provided in Theorem III.11. In Section III-B, we present a sparsification which provides $O(\lambda^3)$-approximation for any (potentially super-constant) λ.

A. Sparsification for O_{λ^2}

In our solution, we fix an arbitrary LCS for every pair of windows and refer to that as $\mathsf{opt}_{i,j}$ for two windows w_i and w_j. Note that we do not explicitly compute $\mathsf{opt}_{i,j}$ in our algorithm. Let us for simplicity, think of each $\mathsf{opt}_{i,j}$ as a matching between the characters of the two windows. Also, denote by $(\mathsf{opt}_{i,a} \cap \mathsf{opt}_{a,b} \cap \mathsf{opt}_{b,j})$ a solution which is constructed for windows w_i and w_j by taking the intersection of $(\mathsf{opt}_{i,a} \cap \mathsf{opt}_{a,b} \cap \mathsf{opt}_{b,j})$. More precisely, we have

$$(x,y) \in (\mathsf{opt}_{i,a} \cap \mathsf{opt}_{a,b} \cap \mathsf{opt}_{b,j}) \Longleftrightarrow \exists \alpha, \beta \text{ such that } \begin{array}{l} (x,\alpha) \in \mathsf{opt}_{i,a} \text{ and} \\ (\alpha,\beta) \in \mathsf{opt}_{a,b} \text{ and} \\ (\beta,y) \in \mathsf{opt}_{b,j}. \end{array}$$

Let $\|(\mathsf{opt}_{i,a} \cap \mathsf{opt}_{a,b} \cap \mathsf{opt}_{b,j})\| = \frac{|(\mathsf{opt}_{i,a} \cap \mathsf{opt}_{a,b} \cap \mathsf{opt}_{b,j})|}{\sqrt{|w_i||w_j|}}$.

Definition III.1 ((ϵ,λ)-constructive)**.** *We call a tuple $\langle w_i, w_a, w_b, w_j \rangle$ ($w_i \neq w_a \neq w_b \neq w_j$) a constructive tuple, if $\|(\mathsf{opt}_{i,a} \cap \mathsf{opt}_{a,b} \cap \mathsf{opt}_{b,j})\| \geq (1-\epsilon)\lambda^3$.*

The advantage of a constructive tuple is that if $\mathsf{opt}_{i,a}, \mathsf{opt}_{a,b}$, and $\mathsf{opt}_{b,j}$ are provided, one can construct a desirable solution for $\mathsf{opt}_{i,j}$ in linear time by taking the intersection of the given matchings. Our algorithm is actually based on the above observation. We bring our algorithm in the following:

We parameterize our algorithm by a value $0 < \gamma < 1$ to be set later. One may optimize the runtime of the algorithm by setting the value of γ in terms of the number of windows and the length of the windows. We first sample a set S of $O(k^\gamma \log k)$ windows. Next, we compute $\mathsf{opt}_{i,j}$ of every window $w_i \in S$ and every other window w_j (not necessarily in S). Finally, we find all the constructive tuples $\langle w_i, w_a, w_b, w_j \rangle$ such that $w_a, w_b \in S$ and update $\widehat{\mathsf{O}}_{\lambda^2}$ accordingly. This is shown in Algorithm 1.

The running time of our algorithm is equal to $O(k|S|\mathsf{w}_{\mathsf{max}}^2 + k^2|S|^2\mathsf{w}_{\mathsf{max}})$. The rest of this section is dedicated to proving that what remains in the lcs-graph is sparse.

We first introduce the notion of well-connected pairs.

Definition III.2 (well-connected pair)**.** *We say a pair of windows (w_i, w_j) is well-connected, if there are at least $k^{2-\gamma}$ pairs of windows (w_a, w_b) such that $\langle w_i, w_a, w_b, w_j \rangle$ is (ϵ,λ)-constructive.*

Algorithm 1 Sparsification for O_{λ^2}

```
procedure QuadraticSparsification($w_1, w_2, \ldots, w_k, \lambda, \epsilon$)   ▷ Theorem III.11
    $\gamma \leftarrow 0.1$
    $S \leftarrow 40k^{\gamma}\log k$ i.i.d. samples of $[k]$
    $\widehat{\mathsf{O}}_{\lambda^2} \leftarrow \{0\}^{k\times k}$
    for $w_i \in S$ do   ▷ Takes $|S|k\mathsf{w}_{\mathsf{max}}^2$ time
        for $j \leftarrow 1$ to $k$ do
            $\mathsf{opt}_{i,j}, \mathsf{opt}_{j,i} \leftarrow \mathsf{lcs}(w_i, w_j)$
        end for
    end for
    for $w_i \in S$ do   ▷ Takes $|S|k\mathsf{w}_{\mathsf{max}}$ time
        for $j \leftarrow 1$ to $k$ do
            if $|\mathsf{opt}_{i,j}| > \lambda$ then
                $\widehat{\mathsf{O}}_{\lambda^2}[i][j] \leftarrow 1$
            end if
        end for
    end for
    for $i \leftarrow 1$ to $k$ do   ▷ Takes $k^2|S|^2\mathsf{w}_{\mathsf{max}}$ time
        for $j \leftarrow 1$ to $k$ do
            for $w_a \in S$ do
                for $w_b \in S$ do
                    if $\left\|\left(\mathsf{opt}_{i,a} \cap \mathsf{opt}_{a,b} \cap \mathsf{opt}_{b,j}\right)\right\| \geq (1-\epsilon)\lambda^3$ then
                        $\widehat{\mathsf{O}}_{\lambda^2}[i][j] \leftarrow 1$
                    end if
                end for
            end for
        end for
    end for
    return $\widehat{\mathsf{O}}_{\lambda^2}$
end procedure
```

It follows from the definition that well-connected pairs are detected in our algorithm with high probability.

Lemma III.3. *Let $\widehat{\mathsf{O}}_{\lambda^2} \in \{0,1\}^{k\times k}$ denote the output of Algorithm 1. With probability at least $1 - 1/\mathsf{poly}(k)$, for all $(i,j) \in [k]\times[k]$ such that (w_i, w_j) is a well-connected pair (Definition III.2), we have $\widehat{\mathsf{O}}_{\lambda^2}[i][j] = 1$.*

Proof: We consider a fixed (i,j) such that pair (w_i, w_j) is well-connected. Let

$$Q_{i,j} = \left\{(a,b) \;\middle|\; \left\|\mathsf{opt}_{i,a} \cap \mathsf{opt}_{a,b} \cap \mathsf{opt}_{b,j}\right\| \geq (1-\epsilon)\lambda^3\right\}.$$

Conceptually, we divide the process of sampling S into two phases: we sample $20k^{\gamma}\log k$ windows in the first phase, and then we sample another $20k^{\gamma}\log k$ windows in the second phase.

For each $a \in [k]$, let $Q_{i,j,a} = \{b : (a,b) \in Q_{i,j}\}$. Since $\sum_{a\in[k]} |Q_{i,j,a}| = |Q_{i,j}| \geq k^{2-\gamma}$, there are at least

$$\frac{k^{2-\gamma} - k\cdot k^{1-\gamma}/2}{k} = \frac{k^{1-\gamma}}{2}$$

different number a's in $[k]$ such that $|Q_{i,j,a}| \geq \frac{k^{1-\gamma}}{2}$. Hence, in the first phase, there is a sampled number q such that $|Q_{i,j,q}| \geq k^{1-\gamma}/2$ with probability at least

$$1 - \left(1 - \frac{k^{1-\gamma}/2}{k}\right)^{20k^{\gamma}\log k} < \frac{1}{k^5}.$$

We fix such a q. In the second phase, there is a sampled number r such that $r \in Q_{i,j,q}$ with probability at least

$$1 - \left(1 - \frac{k^{1-\gamma}/2}{k}\right)^{20k^{\gamma}\log k} < \frac{1}{k^5}.$$

The lemma is obtained by a union bound on all the well-connected pairs (w_i, w_j). ■

In what follows, we bound $|\{(i,j) \mid \|\mathsf{lcs}(w_i,w_j)\| \geq \lambda \text{ and } \widehat{\mathsf{O}}_{\lambda^2}[i][j] = 0\}|$. In other words, we prove an upper bound on the number of edges that remain in the graph.

Definition III.4 (NG_λ). *Define a graph NG_λ with k vertices and the following edges:*

$$E(\mathsf{NG}_\lambda) = \left\{(i,j) \;\middle|\; \|\mathsf{lcs}(w_i,w_j)\| \geq \lambda \quad \text{and} \quad \widehat{\mathsf{O}}_{\lambda^2}[i][j] = 0\right\}.$$

We first prove that every $K_{\Omega(\mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda^3)),\Omega(\mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda^3))}$ subgraph of NG_λ corresponds to at least one constructive tuple.

Lemma III.5. *Let X and Y be two sets of windows such that for every $w_i \in X$ and $w_j \in Y$ there is an (i,j) edge in $E(\mathsf{NG}_\lambda)$, for every $w_i, w_{i'} \in X$, $|w_i| = |w_{i'}|$ and for every $w_j, w_{j'} \in Y$, $|w_j| = |w_{j'}|$. If $|X| \geq \Omega(\mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda^3))$ and $|Y| \geq \Omega(\mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda^3))$, then there exist $w_i, w_a, w_b, w_j \in X \cup Y$ such that $\langle w_i, w_a, w_b, w_j\rangle$ is (ϵ,λ)-constructive, i.e.,*

$$\left\|\mathsf{opt}_{i,a} \cap \mathsf{opt}_{a,b} \cap \mathsf{opt}_{b,j}\right\| \geq (1-\epsilon)\lambda^3.$$

Proof: By assumption, we know that $|X|, |Y| \geq \Omega(\mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda^3))$. Let t_1 denote the window size for each window in X, and t_2 denote the window size for each window in Y.

We carefully select $X' \subseteq X$ and $Y' \subseteq Y$ such that

1) $|X'| \geq \Omega(1/(\epsilon\lambda^3))$.
2) $|Y'| \geq \Omega(1/(\epsilon\lambda^3))$.
3) $|X'|t_1 = (1 \pm O(\epsilon))|Y'|t_2$.

To do this, if $|X|t_1 > (1+O(\epsilon))|Y|t_2$, then we set $Y' = Y$ and select X' as an arbitrary subset of X with size $\left\lceil \frac{|Y|t_2}{t_1} \right\rceil$. if $|X|t_1 < (1-O(\epsilon))|Y|t_2$, then we set $X' = X$ and select Y' an arbitrary subset of Y with size $\left\lceil \frac{|X|t_1}{t_2} \right\rceil$.

We define a window-based (bipartite) graph $G_W = (V_W, E_W)$ to be the subgraph of NG_λ on $V_W = X' \cup Y'$ removing all the edges within X' and all the edges within

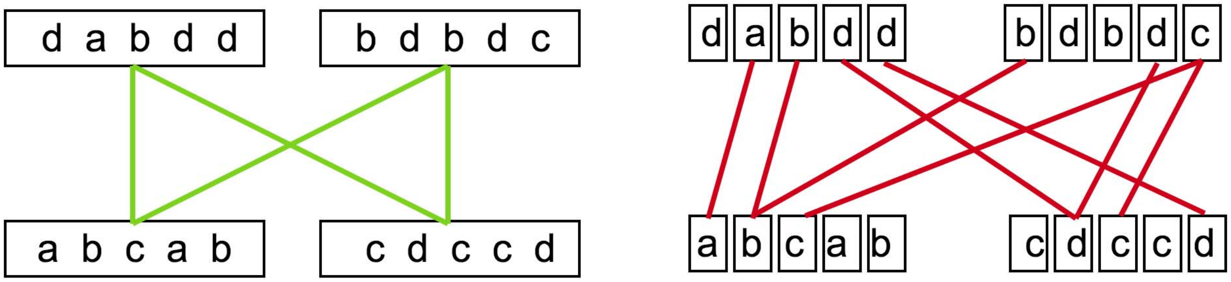


Figure 4: The graph on the left is an example of the string-based graph and the graph on the right is an example of the character-based graph.

Y'. Let $l_1 = |X'|$ and $l_2 = |Y'|$. The total number of nodes in window-based graph is $l_1 + l_2$, and the window-based graph is a bi-clique.

We also define a character-based (bipartite) graph $G_C = (V_C, E_C)$ as follows: Each window of X' has t_1 nodes in the character-based graph such that each node represents a character of the window. Similarly, each window of Y' has t_2 nodes in the character-based graph. Two nodes x, y in the character-based graph are adjacent iff $(x, y) \in \mathsf{opt}_{i,j}$ where w_i is the window containing character x and w_j is the window containing character y. Hence, the total number of nodes in character-based graph is $l_1 t_1 + l_2 t_2$. The total number of edges in character-based graph satisfy

$$l_1 l_2 \lambda \sqrt{t_1 t_2} \leq |E(G_C)| \leq l_1 l_2 \min\{t_1, t_2\}.$$

The average degree of the character-based graph is at least $2 l_1 l_2 \lambda \sqrt{t_1 t_2}/(l_1 t_1 + l_2 t_2)$. By Blakley-Roy inequality (Lemma V.3), the number of walks of length 3 in the character-based graph is at least

$$\begin{aligned} & |V(G_C)| \cdot \left(2 \frac{l_1 l_2 \lambda \sqrt{t_1 t_2}}{l_1 t_1 + l_2 t_2}\right)^3 \\ = \ & (l_1 t_1 + l_2 t_2) \cdot \left(2 \frac{l_1 l_2 \lambda \sqrt{t_1 t_2}}{l_1 t_1 + l_2 t_2}\right)^3 \\ = \ & 8 \lambda^3 \frac{(l_1 l_2 \sqrt{t_1 t_2})^3}{(l_1 t_1 + l_2 t_2)^2}. \end{aligned}$$

But many of the 3-walks are degenerate. For the number of 2-walks, we can upper bound it by

$$\begin{aligned} \#\text{2-walks} & \leq (\text{max degree of } G_W) \cdot 2|E(G_C)| \\ & \leq \max\{l_1, l_2\} \cdot 2(l_1 l_2 \min\{t_1, t_2\}) \\ & = 2(1 + O(\epsilon)) l_1^2 l_2 t_1. \end{aligned}$$

Note that the #1-walks is $2|E(G_C)|$. We have

$$\begin{aligned} & \frac{\#\text{2-walks} + \#\text{1-walks}}{\#\text{3-walks}} \\ \leq & \frac{2(1 + O(\epsilon)) l_1^2 l_2 t_1 + 2 l_1 l_2 \min\{t_1, t_2\}}{\left(8 \lambda^3 \frac{(l_1 l_2 \sqrt{t_1 t_2})^3}{(l_1 t_1 + l_2 t_2)^2}\right)} \\ < & \frac{(l_1 t_1 + l_2 t_2)^2 l_1^2 l_2 t_1}{2 \lambda^3 l_1^3 l_2^3 t_1^{1.5} t_2^{1.5}} \\ \leq & \frac{(4 + O(\epsilon)) l_1^4 l_2 t_1^3}{2 \lambda^3 l_1^{4.5} l_2^{1.5} t_1^3} \\ = & \frac{4 + O(\epsilon)}{2 \lambda^3 l_1^{0.5} l_2^{0.5}} \\ \leq & \ O(\epsilon), \end{aligned}$$

where the second step and third step follow from $l_1 t_1 = (1 \pm O(\epsilon)) l_2 t_2$, and the last step follows from $l_1 = \Omega(1/(\epsilon \lambda^3)), l_2 = \Omega(1/(\epsilon \lambda^3))$. Hence, the total number of 3-paths (3-walks that are not degenerate) is at least

$$8(1 - O(\epsilon)) \lambda^3 \frac{(l_1 l_2 \sqrt{t_1 t_2})^3}{(l_1 t_1 + l_2 t_2)^2}.$$

On the other hand, the number of 4-tuples in window-based graph is $8 \binom{l_1}{2} \binom{l_2}{2} \leq 2 l_1^2 l_2^2$. Thus, there must exist a 4-tuple containing at least

$$\begin{aligned} 8(1 - O(\epsilon)) \lambda^3 \frac{(l_1 l_2 \sqrt{t_1 t_2})^3}{(l_1 t_1 + l_2 t_2)^2} \cdot \frac{1}{2 l_1^2 l_2^2} & = \\ 4(1 - O(\epsilon)) \lambda^3 \sqrt{t_1 t_2} \cdot \frac{l_1 l_2 t_1 t_2}{(l_1 t_1 + l_2 t_2)^2} & = \\ 4(1 - O(\epsilon)) \lambda^3 \sqrt{t_1 t_2} \cdot \frac{1}{\frac{l_1 t_1}{l_2 t_2} + 2 + \frac{l_2 t_2}{l_1 t_1}} & \geq \\ 4(1 - O(\epsilon)) \lambda^3 \sqrt{t_1 t_2} \cdot \frac{1}{4 + O(\epsilon)} & \geq \\ (1 - O(\epsilon)) \lambda^3 \sqrt{t_1 t_2}, & \end{aligned}$$

many 3-walks where the third step follows from $l_1 t_1 = (1 \pm O(\epsilon)) l_2 t_2$. Rescaling the factor of ϵ for $|X|$ and $|Y|$ completes the proof. ■

Definition III.6 (reconstructive tuple)**.** *We say a tuple $\langle w_i, w_a, w_b, w_j \rangle$ is reconstructive if it is (ϵ, λ)-constructive and also $(i, a), (a, b), (b, j), (i, j) \in E(\mathsf{NG}_\lambda)$.*

Before proving Lemma III.10, we need to define a new graph NF_λ,

Definition III.7 (NF_λ)**.** *We construct a graph NF_λ in the following way: for each node $v \in \mathsf{NG}_\lambda$ we keep v in NF_λ with probability $p = k^{-\alpha}/4$ where $\alpha = 1 - \gamma/4$. For each $u, v \in \mathsf{NF}_\lambda$, if $(u, v) \in \mathsf{NG}_\lambda$ then we also draw an edge u, v in NF_λ.*

It is obvious that $V(\mathsf{NF}_\lambda) \subseteq V(\mathsf{NG}_\lambda)$ and $E(\mathsf{NF}_\lambda) \subseteq E(\mathsf{NG}_\lambda)$.

Based on definition of NF_λ, we are able to show that if NG_λ is dense, so is NF_λ.

Claim III.8 (NF_λ is a dense graph)**.** *Let $\mathsf{w}_{\mathsf{layers}}$ be the number of different window sizes. Let NG_λ denote a graph such that $|E(\mathsf{NG}_\lambda)| \geq k^{2-\beta} \cdot \mathsf{w}_{\mathsf{layers}}^2$ for some β and NF_λ be defined as Definition III.7. If $\gamma/4 - \beta = \Omega(1)$, then with probability at least 0.98, we have*

$$|E(\mathsf{NF}_\lambda)| = \Omega(|V(\mathsf{NF}_\lambda)|^{2-4\beta/\gamma} \cdot \mathsf{w}_{\mathsf{layers}}^2)$$

Proof: Based on the sampling rate, we know that the following hold in expectation:

$$\mathbb{E}[|V(\mathsf{NF}_\lambda)|] = \frac{1}{4k^\alpha}|V(\mathsf{NG}_\lambda)|,$$

and

$$\mathbb{E}[|E(\mathsf{NF}_\lambda)|] = \frac{1}{4^2 k^{2\alpha}}|E(\mathsf{NG}_\lambda)|.$$

Using standard Chernoff bound, we have with probability 0.99,

$$\begin{aligned} |V(\mathsf{NF}_\lambda)| \in [1/2, 2] \cdot \frac{1}{4k^\alpha}|V(\mathsf{NG}_\lambda)| &= [1/2, 2] \cdot \frac{1}{4}k^{1-\alpha} \\ &= [1/2, 2] \cdot \frac{1}{4}k^{\gamma/4}, \end{aligned}$$

To show the concentration of $|E(\mathsf{NF}_\lambda)|$, we define a random variable $x_{u,v}$ such that $x_{u,v} = 1$ if $(u, v) \in E(\mathsf{NF}_\lambda)$, 0 otherwise. Then we have

$$\begin{aligned} &\mathbb{E}\left[|E(\mathsf{NF}_\lambda)|^2\right] \\ =&\ \mathbb{E}\left[\left(\sum_{(u,v)\in E(\mathsf{NG}_\lambda)} x_{u,v}\right)^2\right] \\ =&\ \mathbb{E}\left[\sum_{(u,v)\in E(\mathsf{NG}_\lambda)} x_{u,v}^2\right] \\ &+\mathbb{E}\left[\sum_{(u,v),(u,v')\in E(\mathsf{NG}_\lambda)} \mathbf{1}_{v\neq v'} x_{u,v} x_{u,v'}\right] \\ &+\mathbb{E}\left[\sum_{(u,v)\in E(\mathsf{NG}_\lambda)} \sum_{(u',v')\in E(\mathsf{NG}_\lambda)} \mathbf{1}_{u\neq v\neq u'\neq v'} x_{u,v} x_{u',v'}\right] \\ \leq&\ p^2|E(\mathsf{NG}_\lambda)| + p^3 \cdot \#\text{2-walks} \\ &+p^4 \cdot \left(|E(\mathsf{NG}_\lambda)|^2 - \#\text{2-walks} - |E(\mathsf{NG}_\lambda)|\right). \end{aligned}$$

Since $\mathbb{E}[|E(\mathsf{NF}_\lambda)|] = p^2|E(\mathsf{NG}_\lambda)|$. Thus,

$$\begin{aligned} \mathbf{Var}[|E(\mathsf{NF}_\lambda)|] &= \mathbb{E}\left[(|E(\mathsf{NF}_\lambda)|)^2\right] - (\mathbb{E}[|E(\mathsf{NF}_\lambda)|])^2 \\ &\leq p^2|E(\mathsf{NG}_\lambda)| + p^3 \cdot \#\text{2-walks} \\ &\leq p^2|E(\mathsf{NG}_\lambda)| + p^3|V(\mathsf{NG}_\lambda)| \cdot |E(\mathsf{NG}_\lambda)| \\ &\leq 2p^3|V(\mathsf{NG}_\lambda)| \cdot |E(\mathsf{NG}_\lambda)|. \end{aligned}$$

In order to use Chebyshev's inequality, we need to make sure

$$p^2|E(\mathsf{NG}_\lambda)| > 1,$$

and

$$p^2|E(\mathsf{NG}_\lambda)| \geq 100(2p^3|V(\mathsf{NG}_\lambda)||E(\mathsf{NG}_\lambda)|)^{1/2}.$$

By $p = k^{\gamma/4-1}/4$ and $|E(\mathsf{NG}_\lambda)| \geq k^{2-\beta}$, we need

$$k^{\gamma/2-\beta}/16 > 1,$$

and the second condition is equivalent to

$$p|E(\mathsf{NG}_\lambda)| \geq 20000|V(\mathsf{NG}_\lambda)|.$$

We just need $k^{\gamma/4-\beta} \geq 80000$.

Using Chebyshev's inequality, we have

$$\begin{aligned} &\Pr\left[\Big||E(\mathsf{NF}_\lambda)| - \mathbb{E}[|E(\mathsf{NF}_\lambda)|]\Big| \geq \tau\sqrt{\mathbf{Var}[|E(\mathsf{NF}_\lambda)|]}\right] \\ \leq&\frac{1}{\tau^2} \end{aligned}$$

combining with $\sqrt{\mathbf{Var}[|E(\mathsf{NF}_\lambda)|]} \leq \frac{1}{100}\mathbb{E}[|E(\mathsf{NF}_\lambda)|]$ together implies

$$\begin{aligned} &\Pr\left[\Big||E(\mathsf{NF}_\lambda)| - \mathbb{E}[|E(\mathsf{NF}_\lambda)|]\Big| \geq \tau\frac{1}{100}\mathbb{E}[|E(\mathsf{NF}_\lambda)|]\right] \\ \leq&\frac{1}{\tau^2}. \end{aligned}$$

Choosing $\tau = 10$, we have with probability 0.99 that $|E(\mathsf{NF}_\lambda)| > 0.9\mathbb{E}[|E(\mathsf{NF}_\lambda)|]$, and consequently

$$\begin{aligned}|E(\mathsf{NF}_\lambda)| &> 0.9\mathbb{E}[|E(\mathsf{NF}_\lambda)|]\\ &\geq 0.9\frac{1}{4^2 k^{2\alpha}}|E(\mathsf{NG}_\lambda)|\\ &\geq 0.9\frac{1}{4^2}k^{\gamma/2-2}k^{2-\beta}\cdot \mathsf{w}_{\mathsf{layers}}^2\\ &= 0.9\frac{1}{4^2}k^{\gamma/2-\beta}\cdot \mathsf{w}_{\mathsf{layers}}^2\\ &\geq 0.9\frac{1}{4^2}k^{\frac{\gamma}{4}(2-4\beta/\gamma)}\cdot \mathsf{w}_{\mathsf{layers}}^2\\ &\geq \Omega(|V(\mathsf{NF}_\lambda)|^{2-4\beta/\gamma}\cdot \mathsf{w}_{\mathsf{layers}}^2).\end{aligned}$$

Thus, we complete the proof. ■

Next, we show

Claim III.9 (NF_λ has no reconstructive tuple)**.** *Let NF_λ be defined as Definition III.7. With probability at least 0.99, there is no reconstructive tuple in NF_λ.*

Proof: By Lemma III.3 and the definition of NG_λ, there is no pair of windows (w_i, w_j) in NG_λ such that pair (i, j) is well-connected. Formally speaking, for all pairs of windows (w_i, w_j), there are at most $k^{2-\gamma}$ pairs of windows (w_a, w_b) such that $\langle w_i, w_a, w_b, w_j\rangle$ is (ϵ, λ)-constructive.

For a fixed (i, a, b, j), the probability that we keep it in NF_λ is $(k^{-\alpha}/4)^4$. We can compute the expected number of constructive tuples in NF_λ,

$$\begin{aligned}\mathbb{E}[\#\text{constructive tuple}] &\leq k^2 k^{2-\gamma}\cdot(k^{-\alpha}/4)^4\\ &= k^{4-\gamma-4\alpha}/256 = 1/256\end{aligned}$$

where the last step follows from $\alpha = 1 - \gamma/4$.

By Markov's inequality, we have

$$\begin{aligned}&\Pr[\#\text{constructive tuple} \geq 1/2]\\ \leq&\frac{\mathbb{E}[\#\text{constructive tuple}]}{1/2}\\ \leq&1/100.\end{aligned}$$

Thus, with probability 0.99, there is no reconstructive tuple in NF_λ. ■

Finally, we use Lemma III.5 to prove an upper bound on the number of edges in NG_λ.

Lemma III.10. *Let NG_λ be as defined in Definition III.4. Then with probability at least 0.9, NG_λ has at most $k^{2-\Omega(\gamma\lambda\epsilon/\mathsf{w}_{\mathsf{gap}})}\cdot \mathsf{w}_{\mathsf{layers}}^2$ edges.*

Proof: We start with assuming

$$|E(\mathsf{NG}_\lambda)| \geq k^{2-\beta}\cdot \mathsf{w}_{\mathsf{layers}}^2,$$

and will get the contradiction at the end.

We construct a graph NF_λ based on Definition III.7. Using Claim III.9, we know that NF_λ has no reconstructive tuple. Next, we are going to show two facts, and they are contradicting to each other by some choice of β.

(Fact A). Note that NF_λ does not satisfy the assumption in Lemma III.5, thus we cannot apply Lemma III.5 to NF_λ directly. Let $\mathsf{w}_{\mathsf{layers}}$ denote the number of different window sizes in total, we can decompose NF_λ into $\mathsf{w}_{\mathsf{layers}}^2$ graphs such that $\mathsf{NF}_\lambda^{i,j}$ only involves size i and j, $\forall i, j \in [\mathsf{w}_{\mathsf{layers}}] \times [\mathsf{w}_{\mathsf{layers}}]$. Now, applying Lemma III.5, for every i, j, we imply that if $\mathsf{NF}_\lambda^{i,j}$ has no reconstructive tuple, then $\mathsf{NF}_\lambda^{i,j}$ has no bipartite graph of certain size, i.e., $K_{\Omega(\mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda^3)),\Omega(\mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda^3))}$.

(Fact B). Using the Turán's Theorem (Lemma V.4), we know for any integer $s \geq 2$, a graph G with n vertices and $\Omega(n^{2-1/s})$ edges has at least one $K_{s,s}$ subgraph. We consider graph NF_λ. Since

$$|E(\mathsf{NF}_\lambda)| = \Omega(|V(\mathsf{NF}_\lambda)|^{2-4\beta/\gamma}\cdot \mathsf{w}_{\mathsf{layers}}^2),$$

by the pigeonhole principle, there exist i and j such that $|E(\mathsf{NF}_\lambda^{i,j})| = \Omega(|V(\mathsf{NF}_\lambda)|^{2-4\beta/\gamma})$. By Turán's Theorem, such an $\mathsf{NF}_\lambda^{i,j}$ contains a large complete bipartite subgraph $K_{\gamma/(4\beta),\gamma/(4\beta)}$.

In order to get a contradiction, we just need

$$\gamma/(4\beta) \geq c\cdot \mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda^3).$$

for some sufficiently large constant $c > 1$ related to Lemma III.5. Thus, we have

$$\beta \leq \gamma\epsilon\lambda^3/(4c\cdot \mathsf{w}_{\mathsf{gap}}).$$

■

Theorem III.11 (quadratic sparsification for constant λ)**.** *Given k windows $w_1, \cdots, w_k$. Let $\mathsf{w}_{\mathsf{max}} = \max_{i\in[k]}|w_i|$, $\mathsf{w}_{\mathsf{min}} = \min_{i\in[k]}|w_i|$ and $\mathsf{w}_{\mathsf{gap}} = \mathsf{w}_{\mathsf{max}}/\mathsf{w}_{\mathsf{min}}$. Let the number of different window sizes be $\mathsf{w}_{\mathsf{layers}}$. For any $\lambda \in (0, 1)$ and $\epsilon \in (0, 1/10)$, there is a randomized algorithm (Algorithm 1) that runs in time*

$$O(\mathsf{w}_{\mathsf{max}}^2 k^{1.1}\log k + \mathsf{w}_{\mathsf{max}} k^{2.2}\log^2 k),$$

outputs a table $\widehat{\mathsf{O}}_{\lambda^2} \in \{0, 1\}^{k\times k}$ such that

$$||\mathsf{lcs}(w_i, w_j)|| \geq (1-\epsilon)\lambda^3, \text{ if } \widehat{\mathsf{O}}_{\lambda^2}[i][j] = 1$$

and

$$\begin{aligned}&\left|\left\{(i,j)\;\middle|\;||\mathsf{lcs}(w_i, w_j)|| \geq \lambda, \quad\text{and}\quad \widehat{\mathsf{O}}_{\lambda^2}[i][j] = 0\right\}\right|\\ =&O\left(k^{2-\Omega(\lambda\epsilon/\mathsf{w}_{\mathsf{gap}})}\cdot \mathsf{w}_{\mathsf{layers}}^2\right).\end{aligned}$$

The algorithm has success probability at least $1-1/\mathsf{poly}(k)$.

Proof: The overall running time is

$$\begin{aligned}&O(k|S|\mathsf{w}_{\mathsf{max}}^2 + k^2|S|^2\mathsf{w}_{\mathsf{max}})\\ =&O(k\cdot k^\gamma\log k\cdot \mathsf{w}_{\mathsf{max}}^2 + k^2\cdot k^{2\gamma}\log^2 k\cdot \mathsf{w}_{\mathsf{max}})\\ =&O(k^{1.1}\log k\cdot \mathsf{w}_{\mathsf{max}}^2 + k^{2.2}\log^2 k\cdot \mathsf{w}_{\mathsf{max}})\end{aligned}$$

where the first step follows from $|S| = O(k^\gamma\log k)$, the second step follows from $\gamma = 0.1$.

The guarantee of table $\widehat{\mathsf{O}}_{\lambda^2}$ follows from properties of graph NG_λ (Algorithm 1 provides the first property of table in Theorem statement, Lemma III.10 provides the second property of table in Theorem statement). ■

B. Sparsification for O_{λ^3}

One shortcoming of Lemma III.10 is that the number of remaining edges is only truly subquadratic if λ is constant. As we discuss in Section III-A, the overall running time of the algorithm depends on the number of edges in the remaining graph and in order for the running time to be truly subquadratic, we need to reduce the number of edges to truly subquadratic. In this section, we show how one can obtain this bound even when λ is super-constant. However, instead of losing a factor λ^2 in the approximation, our technique loses a factor of $O(\lambda^3)$.

Definition III.12 ($\mathsf{lcs}_{w_a}(w_i, w_j)$)**.** *For two windows w_i and w_j, and a window w_a, define $\mathsf{lcs}_{w_a}(w_i, w_j)$ as the length of LCS of $\mathsf{opt}_{i,a}$ and w_j, where $\mathsf{opt}_{i,a}$ denotes the LCS of w_i and w_a.*

Notice that unlike lcs, this new definition is not symmetric. Similar to lcs, we also normalize the size of lcs_s by the geometric mean of the lengths of the two windows. That is, we divide the size of the common string by $\sqrt{|w_i||w_j|}$. In what follows, we first give an algorithm for detecting close pairs of windows, and then prove that the number of remaining pairs whose lcs is at least λ is truly subquadratic.

Our algorithm is based on a data structure which we call $\mathsf{lcs\text{-}cmp}(w_a, S, \widetilde{\lambda})$. Roughly speaking, we will choose $\widetilde{\lambda} = \lambda^2$ when we use it. Let us fix a threshold $\widetilde{\lambda}$ and a window w_a. $\mathsf{lcs\text{-}cmp}(w_a, S, \widetilde{\lambda})$ receives a set S of windows as input such that the size of each window of S is at least $|w_a|$. Upon receiving the windows, it makes a preprocess in time $O(\widetilde{\lambda}^{-2}|w_a|\sum_{w_i \in S}|w_i|)$. Next, $\mathsf{lcs\text{-}cmp}(w_a, S, \widetilde{\lambda})$ would be able to answer each query of the following form in almost linear time:

- For two windows $w_i, w_j \in S$, either certify that $||\mathsf{lcs}_{w_a}(w_i, w_j)|| < \widetilde{\lambda}/2$ or find a solution for $||\mathsf{lcs}_{w_a}(w_i, w_j)||$ with a size of at least $\widetilde{\lambda}^2/8$.

We first show in Section III-B1, how $\mathsf{lcs\text{-}cmp}$ gives us a sparsification in truly subquadratic time and then discuss the algorithm for $\mathsf{lcs\text{-}cmp}$ in Section III-B2.

1) λ^3 Sparsification using $\mathsf{lcs\text{-}cmp}$*:* Similar to what we did in Section III-A, we again use a parameter γ in our algorithm and in the end, we adjust γ to minimize the total running time. In our algorithm, we repeat the following procedure $k^\gamma \log k$ times: sample a window w_a uniformly at random and let S be the set of all the other windows whose length is not smaller than $|w_a|$. Next, we obtain $\mathsf{lcs\text{-}cmp}(w_a, S, \widetilde{\lambda})$ via running the preprocessing step. Finally, for each pair of windows $w_i, w_j \in S$, we make a query to $\mathsf{lcs\text{-}cmp}$ to verify one of the following two possibilities:

- $||\mathsf{lcs}_{w_a}(w_i, w_j)|| < \lambda^2/2$;
- $||\mathsf{lcs}_{w_a}(w_i, w_j)|| \geq \lambda^4/8$.

If the latter is verified we set $\widehat{\mathsf{O}}_{\lambda^3/8}[i][j]$ to 1 otherwise we take no action. In what follows, we prove that after the above sparsification, the number of edges in the remaining graph is truly subquadratic.

Algorithm 2 Sparsification for O_{λ^3}

```
procedure CubicSparsification($w_1, w_2, \ldots, w_k, \lambda$)    ▷ Theorem III.19
    $\gamma \gets 0.1$
    $\widetilde{\lambda} \gets \lambda^2$
    for counter $= 1 \to k^\gamma \log k$ do
        Sample $a \sim [k]$ uniformly at random
        $S \gets \emptyset$
        for $i = 1 \to k$ do
            if $i \neq a$ and $|w_i| \geq |w_a|$ then
                $S \gets S \cup \{w_i\}$
            end if
        end for
        lcs-cmp.Initial($w_a, S, \widetilde{\lambda}$)    ▷ Algorithm 3, Lemma III.20
        for $w_i \in S$ do
            for $w_j \in S$ do
                if lcs-cmp.Query($w_i, w_j$) outputs accept then    ▷ Algorithm 3, Lemma III.21
                    $\widehat{\mathsf{O}}_{\lambda^3/8}[i][j] \gets 1$    ▷ $||\mathsf{lcs}_{w_a}(w_i, w_j)|| \geq \widetilde{\lambda}^2/8$
                end if
            end for
        end for
    end for
    return $\widehat{\mathsf{O}}_{\lambda^3/8}$
end procedure
```

In the rest of this section, we bound $|\{(i,j) \mid ||\mathsf{lcs}(w_i, w_j)|| \geq \lambda \text{ and } \widehat{\mathsf{O}}_{\lambda^3/8}[i][j] = 0\}|$. In other words, we prove an upper bound on the number of edges that remain in the graph. Define a graph NG_λ such that each vertex of NG_λ corresponds to a window and an edge (i,j) means that $||\mathsf{lcs}(w_i, w_j)|| \geq \lambda$ but $\widehat{\mathsf{O}}_{\lambda^3/8}[i][j] = 0$. The goal is to prove an upper bound on the number of edges of NG_λ. We formally prove this in Lemma III.18.

We define a notation called "close" which is similar to "well-connected" in Section III-A. In order to avoid confusion, we use "close" instead of well-connected.

Definition III.13 (close)**.** *Let $\gamma \in (0,1)$ and $\lambda \in (0,1)$. We say a pair (w_i, w_j) of windows is close, if there are at least $k^{1-\gamma}$ windows w_a such that $|w_a| \leq \min\{|w_i|, |w_j|\}$ and $||\mathsf{lcs}_{w_a}(w_i, w_j)|| \geq \lambda^2$.*

Our first observation is that Algorithm 2 detects all the close pairs with high probability.

Lemma III.14. *Let $\widehat{\mathsf{O}}_{\lambda^3/8} \in \{0,1\}^{k\times k}$ be the output of Algorithm 2. For each $(i,j) \in [k]\times[k]$, if (w_i,w_j) is close (Definition III.13) and $||\mathsf{lcs}(w_i,w_j)|| \geq \lambda$, then $\widehat{\mathsf{O}}_{\lambda^3/8}[i][j] = 1$ holds with probability at least $1-1/\mathsf{poly}(k)$.*

Proof: We consider a fixed (i,j) such that (w_i,w_j) is close. By Definition III.13, there are at least $k^{1-\gamma}$ w_a such that $|w_a| \leq \min\{|w_i|,|w_j|\}$ and $||\mathsf{lcs}_{w_a}(w_i,w_j)|| \geq \lambda^2$. Then the probability that none of these windows is sampled is at most

$$\begin{aligned}\left(1-\frac{k^{1-\gamma}}{k}\right)^t &= \left(1-\frac{1}{k^\gamma}\right)^t\\ &= \left(1-\frac{1}{k^\gamma}\right)^{10k^\gamma \log k}\\ &\leq 1/\mathsf{poly}(k).\end{aligned}$$

Taking a union over at most k^2 pairs completes the proof. ■

Before we proceed to Lemma III.18, we bring Lemma III.15 as an auxiliary observation.

Lemma III.15 (existence of a correlated pair)**.** *Let $\epsilon \in (0,1)$, $\lambda \in (0,1)$. Given a window w_a and a set of windows T. Let $\mathsf{w}_{\mathsf{gap}}$ denote the maximum size of windows divided by the minimum size of windows. Let T contain at most $\mathsf{w}_{\mathsf{layers}}$ different sizes and $|T| = \Omega(\mathsf{w}_{\mathsf{layers}}\cdot \mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda))$. If for each $w_i \in T$ we have $||\mathsf{lcs}(w_a,w_i)|| \geq \lambda$, then there exist two windows $w_i,w_j \in T$ such that $||\mathsf{lcs}_{w_a}(w_i,w_j)|| \geq (1-\epsilon)\lambda^2$.*

Proof: Since T has size $\Omega(\mathsf{w}_{\mathsf{layers}}\cdot \mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda))$ and the windows of T have at most $\mathsf{w}_{\mathsf{layers}}$ different sizes, there must exists a subset $T' \subseteq T$ such that $|T'| = \Omega(\mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda))$ and all the windows in T' have the same size. In the rest of the proof, we will focus on w_a and T'.

Let t_0 be the size of w_a and t be the size of all the windows in T'. We consider a window-based bipartite graph, on one side it has one node w_a, and on the other side it has $|T'|$ nodes. Then we can expand the window-based bipartite graph into a character-based bipartite graph, on one side it has t_0 nodes, and on the other side it has $t|T'|$ nodes. Two nodes x,y in the character-based graph are adjacent iff $(x,y) \in \mathsf{opt}_{a,i}$ where w_a is the window containing character x and w_i is the window containing character y. Since $||\mathsf{lcs}(w_a,w_i)|| \geq \lambda$ for every $w_i \in T'$, the number of edges in character-based bipartite graph is at least $\lambda|T'|\sqrt{t_0 t}$.

For i-th character in window w_a, we use D_i to denote the degree of the corresponding node in the character-based bipartite graph. The number of 2-walks is at least

$$\begin{aligned}\sum_{i=1}^{t_0} D_i(D_i-1) &= \left(\sum_{i=1}^{t_0} D_i^2 - \sum_{i=1}^{t_0} D_i\right)\\ &\geq \left(\frac{1}{t_0}(\sum_{i=1}^{t_0} D_i)^2 - \sum_{i=1}^{t_0} D_i\right)\\ &\geq \left(\frac{1}{t_0}(\lambda|T'|\sqrt{t_0 t})^2 - (\lambda|T'|\sqrt{t_0 t})\right)\\ &= (\lambda^2 t|T'|^2 - \lambda|T'|\sqrt{t_0 t})\\ &\geq (1-\epsilon)\lambda^2 t|T'|^2\end{aligned}$$

It remains to show Eq. (1). Since the number of edges in the character-based bipartite graph is at least $\lambda|T'|\sqrt{t_0 t}$, we have

$$\sum_{i=1}^{t_0} D_i \geq \lambda|T'|\sqrt{t_0 t}.$$

By $|T'| = \Omega(\mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda))$, we have

$$\lambda|T'|\sqrt{t_0 t} \geq t_0.$$

Due to a simple fact : $x(x-z) > y(y-z)$ as long as $x > y > z > 0$, we have

$$\frac{1}{t_0}(\sum_{i=1}^{t_0} D_i)^2 - \sum_{i=1}^{t_0} D_i \geq \frac{1}{t_0}(\lambda|T'|\sqrt{t_0 t})^2 - (\lambda|T'|\sqrt{t_0 t}). \tag{1}$$

The number of 3-tuple (w_i,w_a,w_j) is at most $|T'|^2$. Thus, there must exists a pair such that

$$||\mathsf{lcs}_{w_a}(w_i,w_j)|| \geq (1-\epsilon)\lambda^2$$

■

We define graph NG_λ and NF_λ as follows:

Definition III.16 (NG_λ)**.** *We assume that the edges of NG_λ are directed in the following way: For an edge (i,j) if $|w_i| \neq |w_j|$ the starting point of the edge would be the vertex corresponding to the longer window and the ending point of the edge would be the one corresponding to the shorter window. If both corresponding windows have the same lengths, then we use an arbitrary direction for (i,j).*

Definition III.17 (NF_λ)**.** *We construct graph NF_λ in the following way : let a denote the node in $V(\mathsf{NG}_\lambda)$ that has the highest outgoing degree, let $V(\mathsf{NF}_\lambda) = N(a)$. We add an edge between vertices (i,j) in NF_λ if $||\mathsf{lcs}_{w_a}(w_i,w_j)|| \geq (1-\epsilon)\lambda^2$. Give directions to the edges of NF_λ the same way we did it for NG_λ.*

Now, we are ready to prove the main observation of this section.

Lemma III.18 (upper bound on $E(\mathsf{NG}_\lambda)$)**.** *Let NG_λ be defined as Definition III.4. Then*

$$|E(\mathsf{NG}_\lambda)| = O(k^{2-\gamma}\cdot \mathsf{w}_{\mathsf{layers}}\cdot \mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda)).$$

holds with probability $1 - 1/\mathsf{poly}(k)$.

Proof: We start with assuming

$$|E(\mathsf{NG}_\lambda)| \geq \Omega(k^{2-\beta} \cdot \mathsf{w}_{\mathsf{layers}} \cdot \mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda)).$$

We construct a graph NF_λ based on Definition III.17. By Definition III.17, we know that $|V(\mathsf{NF}_\lambda)| \geq \Omega(k^{1-\beta} \cdot \mathsf{w}_{\mathsf{layers}} \cdot \mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda))$. By choosing some $\beta = \gamma$ we will make a contradiction.

Using Lemma III.15, we have for each set $T \subseteq V(\mathsf{NF}_\lambda)$ with $|T| = \Omega(\mathsf{w}_{\mathsf{layers}} \cdot \mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda))$, there exist two nodes u and v in T such that edge $(u, v) \in \mathsf{NF}_\lambda$.

If we look at the complement of graph NF_λ, we know there is no clique K_{r+1} where $r = O(\mathsf{w}_{\mathsf{layers}} \cdot \mathsf{w}_{\mathsf{gap}}/(\epsilon\lambda))$. Using Turan's theorem (Lemma V.5) we know that complement of graph NF_λ has at most $(1 - \frac{1}{r})\frac{q^2}{2}$ edges, where $q = |V(\mathsf{NF}_\lambda)|$. Then we have

$$|E(\mathsf{NF}_\lambda)| \geq \frac{q(q-1)}{2} - (1 - \frac{1}{r})\frac{q^2}{2} = \frac{q^2}{2r} - \frac{q}{2}.$$

This implies that there is one vertex j whose outgoing degree in NF_λ is at least

$$\begin{aligned}\frac{|V(\mathsf{NF}_\lambda)|}{2r} - \frac{1}{2} \geq & |V(\mathsf{NF}_\lambda)| \cdot \Omega\left(\frac{\epsilon\lambda}{\mathsf{w}_{\mathsf{layers}} \cdot \mathsf{w}_{\mathsf{gap}}}\right) \\ \geq & k^{1-\beta} \\ \geq & k^{1-\gamma}.\end{aligned}$$

Thus, there is a node in NG_λ that has a lot of outgoing edges, which means NG_λ has a close pair.

On the other hand, Using definition of NG_λ and Lemma III.14, we know that NG_λ should not contain any close pair. Thus, we get a contradiction. ■

Theorem III.19 (cubic sparsification for arbitrary λ)**.** *Given k windows $w_1, \cdots, w_k$. Let $\mathsf{w}_{\mathsf{layers}}$ denote the number of different sizes for windows. Let $\mathsf{w}_{\max} = \max_{i\in[k]} |w_i|$, $\mathsf{w}_{\min} = \min_{i\in[k]} |w_i|$ and $\mathsf{w}_{\mathsf{gap}} = \mathsf{w}_{\max}/\mathsf{w}_{\min}$. For any $\lambda \in (0, 1), \gamma \in (0, 1)$, there is a randomized algorithm (Algorithm 2) that runs in time*

$$O(\lambda^{-4}k^{1+\gamma}\mathsf{w}_{\max}^2 \log k + k^{2+\gamma}\mathsf{w}_{\max} \log k)$$

and outputs a table $\widehat{\mathsf{O}}_{\lambda^3} \in \{0, 1\}^{k\times k}$ such that

$$||\mathsf{lcs}(w_i, w_j)|| \geq \lambda^4/8, \text{ if } \widehat{\mathsf{O}}_{\lambda^3}[i][j] = 1$$

and

$$\begin{aligned}&\left|\left\{(i, j) \mid ||\mathsf{lcs}(w_i, w_j)|| \geq \lambda, \quad \text{and} \quad \widehat{\mathsf{O}}_{\lambda^3}[i][j] = 0\right\}\right| \\ =&O(k^{2-\gamma}\mathsf{w}_{\mathsf{layers}}\mathsf{w}_{\mathsf{gap}}/\lambda).\end{aligned}$$

The algorithm has success probability $1 - 1/\mathsf{poly}(k)$.

Proof: The running time of each round of Algorithm 2 is

$$\begin{aligned}&= O(\text{ constructing } S \text{ time} \\ &+\mathsf{lcs\text{-}cmp}.\textsc{Initial} \text{ time} \\ &+|S|^2 \cdot (\mathsf{lcs\text{-}cmp}.\textsc{Query} \text{ time})) \\ &= O(k + \widetilde{\lambda}^{-2}|S|\mathsf{w}_{\max}^2 + |S|^2\mathsf{w}_{\max}) \\ &= O(\lambda^{-4}k\mathsf{w}_{\max}^2 + k^2\mathsf{w}_{\max})\end{aligned}$$

where the last step follows by $\widetilde{\lambda} = \lambda^2$.

Since it repeats $O(k^\gamma \log k)$ rounds, thus the overall running time is

$$\begin{aligned}&O(k^\gamma \log k) \cdot O(\lambda^{-4}k\mathsf{w}_{\max}^2 + k^2\mathsf{w}_{\max}) \\ =&O(\lambda^{-4}k^{1+\gamma}\mathsf{w}_{\max}^2 \log k + k^{2+\gamma}\mathsf{w}_{\max} \log k).\end{aligned}$$

■

2) Implementation of $\mathsf{lcs\text{-}cmp}$*:* Given $\widetilde{\lambda}$, w_a and $w_1, w_2, \ldots, w_s$, we present an $O(\widetilde{\lambda}^{-1}|w_a|\sum_{i=1}^s |w_i|)$ time preprocessing algorithm and $O(|w_a|)$ time query algorithm for $\mathsf{lcs\text{-}cmp}$ such that for any $i, j \in [s]$

1) if $||\mathsf{lcs}_{w_a}(w_i, w_j)|| > \widetilde{\lambda}/2$, then the query algorithm outputs accept;
2) if $||\mathsf{lcs}_{w_a}(w_i, w_j)|| \leq \widetilde{\lambda}^2/8$, then the query algorithm outputs reject.

Algorithm 3 is our $\mathsf{lcs\text{-}cmp}$ preprocessing and query algorithm. The high level idea is to compute $\mathsf{opt}_{a,i}$ for every $i \in [s]$ and a set of at most $2/\lambda$ common subsequences between w_a and w_i for each i such that every character of w_a appears at most once among the sequences for a fixed i. $\mathsf{lcs}_{w_a}(w_i, w_j)$ is approximated by taking the longest intersection of $\mathsf{opt}_{a,i}$ and some sequence in the set of window w_j. Also see Figure 5 for the intuition.

Now we prove that Algorithm 3 implements $\mathsf{lcs\text{-}cmp}$.

Lemma III.20 (Initial)**.** *Given parameter $\widetilde{\lambda}$, a window w_a and a set of windows $w_1, \cdots, w_s$. The* Initial *of data structure* $\mathsf{lcs\text{-}cmp}$ *(Algorithm 3) takes $O(\widetilde{\lambda}^{-1}|w_a|\sum_{i=1}^s |w_i|)$ time, and outputs $\{X_{a,i}\}_{i\in[s]}$ and $\{Y_{a,i}\}_{i\in[s]}$ such that*

1) *$X_{a,i}$ corresponds to indices of a longest common subsequence between w_a and w_i for every $i \in [s]$.*
2) *$Y_{a,i}$ corresponds to indices of at most $2/\widetilde{\lambda}$ common subsequence between w_a and w_i for every $i \in [s]$.*
3) *If none of the element indices of a common subsequence between w_a and w_i is in $Y_{a,i}$, then the length of this common subsequence is less than $\widetilde{\lambda}|w_a|/2$.*

Proof: The first property follows from the definition of the algorithm.

For the second and third property, if an $\mathsf{opt}'_{a,i}$ obtained in Line 9 has length less than $\widetilde{\lambda}|w_a|/2$, then the element indices of $\mathsf{opt}'_{a,i}$ are not in $Y_{a,i}$. Hence, the third property holds. Also, it means that every time that Line 13 is executed, the size of $Y_{a,i}$ increases by at least $\widetilde{\lambda}|w_a|/2$. For a fixed

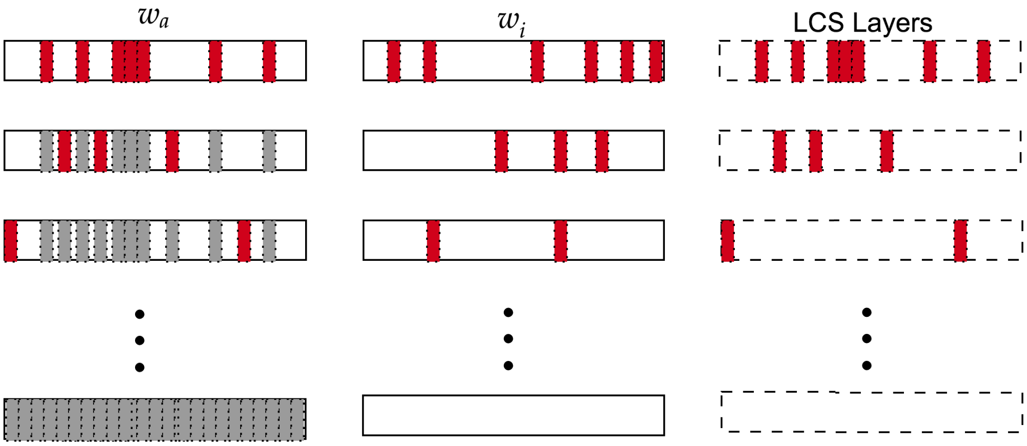


Figure 5: Computing a set of common subsequences between w_a and w_i for each i such that every character of w_a appears at most once among the sequences for a fixed i.

$i \in [s]$, Line 13 is executed at most $2/\widetilde{\lambda}$ times. Thus, the second property holds.

The running time is also obtained by the fact that Line 13 is executed at most $2/\widetilde{\lambda}$ times for every fixed $i \in [s]$. ■

Lemma III.21 (QUERY)**.** *For any $(i, j) \in [s] \times [s]$, the* QUERY *of data structure* $\mathsf{lcs\text{-}cmp}$ *(Algorithm 3) runs in time $O(|w_a|)$ with the following properties:*

1) *If $\|\mathsf{lcs}_{w_a}(w_i, w_j)\| > \widetilde{\lambda}$, then it outputs accept.*
2) *If $\|\mathsf{lcs}_{w_a}(w_i, w_j)\| \leq \widetilde{\lambda}^2/8$, then it outputs reject.*

Proof: Let $\mathsf{opt}_{a,i,j}$ be the LCS between $\mathsf{opt}_{a,i}$ and w_j. By the definition, we have

$$\|\mathsf{lcs}_{w_a}(w_i, w_j)\| = \frac{|\mathsf{opt}_{a,i,j}|}{\sqrt{|w_i| \cdot |w_j|}}$$

Consider the case of $\|\mathsf{lcs}_{w_a}(w_i, w_j)\| \geq \widetilde{\lambda}$. By the third property of Lemma III.20, there are less than $\widetilde{\lambda}|w_a|/2$ elements of $\mathsf{opt}_{a,i,j}$ with indices not in $Y_{a,j}$. So we have

$$|X_{a,i} \cap Y_{a,j}| > \widetilde{\lambda}\sqrt{|w_i||w_j|} - \frac{1}{2}\widetilde{\lambda}|w_a| \geq \frac{1}{2}\widetilde{\lambda}\sqrt{|w_i||w_j|}$$

where the last inequality follows from the fact that $|w_a|$ is smaller than or equal to $|w_i|$ for every $i \in [s]$. Hence the algorithm accepts.

Consider the case of $\|\mathsf{lcs}_{w_a}(w_i, w_j)\| \leq \widetilde{\lambda}^2/8$. By the second property of Lemma III.20, $Y_{a,j}$ corresponds to at most $2/\widetilde{\lambda}$ mutually non-overlapping common subsequence between w_a and w_j. Since $\|\mathsf{lcs}_{w_a}(w_i, w_j)\| \leq \widetilde{\lambda}^2/8$, the set of indices of every such a common subsequence has an intersection with $X_{a,i}$ of size at most $\widetilde{\lambda}^2\sqrt{|w_i||w_j|}/8$. Thus, we have

$$|X_{a,i} \cap Y_{a,j}| \leq \frac{1}{8}\widetilde{\lambda}^2\sqrt{|w_i||w_j|} \cdot \frac{2}{\widetilde{\lambda}} \leq \frac{1}{4}\widetilde{\lambda}\sqrt{|w_i||w_j|}.$$

Hence the algorithm rejects for this case. ■

IV. Longest Increasing Subsequence

We outlined the algorithm in Section I-C3. Here, we bring the details for each step of the algorithm. In the interest of space, we omit some of the proofs and bring them in the full-version. In Section IV-A we discuss the solution domains and show how we construct them. Next, in Section IV-B we discuss the details of constructing pseudo-solutions and finally in Section IV-C we show how we can obtain an approximate solution from the pseudo-solutions. Also, in Section IV-D we bring an improvement to the running time at the expense of having a larger approximation factor for the algorithm.

A. Solution Domains

We assume from now on that $\mathsf{lis}(A) > n\lambda$ holds. As mentioned earlier, we divide the input array into $\sqrt{n}$ subarrays of size $\sqrt{n}$ and denote them by $\mathsf{sa}_1, \mathsf{sa}_2, \ldots, \mathsf{sa}_{\sqrt{n}}$. For a fixed optimal solution opt, our goal is to approximate the smallest and the largest number of each subarray that contributes to opt. Let us refer to these numbers as the *domain* of each subarray. Let $\epsilon = 1/1000$ be accuracy. For a subarray sa_i, we sample k (will be decided later) different elements and refer to them by $a_{j_1}, a_{j_2}, \ldots, a_{j_k}$.

We first prove that,

Lemma IV.1 (constructing candidate domains)**.** *Let $\lambda \in (0, 1)$, $\epsilon \in (0, 1/2)$ and $\delta \in (0, 1/10)$. Let sa_i be a length-$\sqrt{n}$ subarray whose contribution to the optimal solution is at least $\epsilon\sqrt{n}\lambda$, i.e., $\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i), \mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i) \geq \epsilon\sqrt{n}\lambda$. If we uniformly sample $k = 20\log(1/\delta)/(\lambda\epsilon^2)$ elements $a_{j_1}, a_{j_2}, \ldots, a_{j_k}$ from sa_i, then with probability at least*

Algorithm 3 lcs-cmp data structure

```
data structure lcs-cmp

members
    $X_{a,1}, \cdots, X_{a,s}$
    $Y_{a,1}, \cdots, Y_{a,s}$
    $\widetilde{\lambda} \in (0,1)$
end members

procedure INITIAL($w_a, \{w_i\}_{i \in [s]}, \widetilde{\lambda}$)    ▷ Lemma III.20
    for $i \in [s]$ do
        compute $\mathsf{opt}_{a,i}$, a LCS between $w_a$ and $w_i$
        let $X_{a,i}$ be the set of indices of all the element of $\mathsf{opt}_{a,i}$ with respect to $w_a$
    end for
    for $i \in [s]$ do
        let $Y_{a,i}$ be an empty set
        while true do
            compute $\mathsf{opt}'_{a,i}$, a LCS between $w_a$ removing the elements with index in $Y_{a,i}$ and $w_i$
            if the length of $\mathsf{opt}'_{a,i}$ is less than $\widetilde{\lambda}|w_a|/2$ then
                break
            else
                put the indices of all the elements of $\mathsf{opt}'_{a,i}$ with respect to $w_a$ into $Y_{a,i}$
            end if
        end while
    end for
    return $\{X_{a,i}\}_{i \in [s]}$, $\{Y_{a,i}\}_{i \in [s]}$
end procedure

procedure QUERY($w_i, w_j$)    ▷ Lemma III.21
    if $|X_{a,i} \cap Y_{a,j}| > \widetilde{\lambda}\sqrt{|w_i| \cdot |w_j|}/4$ then
        return accept
    else
        return reject
    end if
end procedure

end data structure
```

Algorithm 4 Constructing the candidate domains

```
procedure CONSTRUCTCANDIDATEDOMAINS($\mathsf{sa}_i$)    ▷ Lemma IV.1
                ▷ Given random access to a subarray $\mathsf{sa}_i$
    $k \leftarrow 20\log(1/\delta)/(\lambda\epsilon^2)$
    Sample $k$ elements from $\mathsf{sa}_i$, and denote the sampled elements by $a_{j_1}, a_{j_2}, \ldots, a_{j_k}$
    for $\alpha$ in $[k]$ do
        for $\beta$ in $[k]$ do
            If $a_{j_\alpha} \le a_{j_\beta}$, then construct a candidate domain $[a_{j_\alpha}, a_{j_\beta}]$
        end for
    end for
    return all the constructed candidate domains
end procedure
```

$1-\delta$, there exists a pair $(\alpha, \beta) \in [k] \times [k]$ such that the following two conditions hold

1) $\mathsf{sm}(\mathsf{sa}_i) \le a_{j_\alpha} \le a_{j_\beta} \le \mathsf{lg}(\mathsf{sa}_i)$,
2) $\mathsf{lis}^{[a_{j_\alpha}, a_{j_\beta}]}(\mathsf{sa}_i) \ge (1-\epsilon)\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i), \mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i)$.

Proof: At least $\epsilon\sqrt{n}\lambda$ elements of sa_i appear in opt. Let us put all these elements in an array b in the same order that they appear in sa_i. Then it is obvious that b has at least $\epsilon\sqrt{n}\lambda$ elements. To prove the lemma, we bound the probability that none of the first $\epsilon/2$ fraction of the elements of b is sampled in our algorithm.

$$\begin{aligned}
&\Pr[\text{none first } \epsilon/2 \text{ fraction sampled}] \\
\le\ & \left(1 - \frac{\epsilon}{2} \cdot \epsilon\sqrt{n}\lambda \cdot \frac{1}{\sqrt{n}}\right)^k \\
=\ & \left(1 - \frac{\epsilon^2\lambda}{2}\right)^{\frac{2}{\epsilon^2\lambda} \cdot 10\log(1/\delta)} \\
\le\ & e^{-10\log(1/\delta)} \\
\le\ & \delta/2,
\end{aligned}$$

where the first step follows from the fact that b contains at least $\epsilon\lambda\sqrt{n}$ elements, the second step follows from $k = 20\log(1/\delta)/(\lambda\epsilon^2)$, and the third step follows from the fact that the $(1-1/x)^x \le 1/e$ for $\forall x \ge 4$. Hence, with probability at least $1-\delta/2$, at least one of the elements in the first $\epsilon/2$ fraction of b are sampled.

With the same analysis, one can prove that with probability at least $1-\delta/2$ at least one of the elements in the last $\epsilon/2$ fraction of b are also sampled.

Taking a union bound of two events, with probability at least $1-\delta$, at least one of the elements in the first and at least one of the elements in the last $\epsilon/2$ fraction of b are sampled.

Therefore, the lis of sa_i subject to this interval is at least a $1-\epsilon$ fraction of $\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i), \mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i)$.

■

Notice that the average contribution of each subarray to opt is $\sqrt{n}\lambda$ and Lemma IV.1 applies to a subarray if its contribution to opt is at least an ϵ fraction of this value. Therefore Lemma IV.1 implies that a considerable fraction of the solution is covered by the candidate domains.

Corollary IV.2 (existence of a desirable solution)**.** *Let $\lambda \in (0,1)$ such that $\mathsf{lis}(A) \ge n\lambda$ and $\epsilon \in (0, 1/4)$. If we run Algorithm 4 with parameter $\delta = \epsilon$ on every subarray independently, then with probability at least*

$1 - \exp(-\Omega(\epsilon^2\sqrt{n}\lambda))$*, there exist a set* $T \subseteq [\sqrt{n}]$ *and elements* α_i *and* β_i *sampled from* sa_i *for each* $i \in T$ *such that the following conditions hold:*

1) *For any* $i \in T$*,* $\alpha_i \leq \beta_i$*.*
2) *For any* $i, j \in T$ *satisfying* $i < j$*,* $\beta_i < \alpha_j$*.*
3) $\sum_{i\in T} \mathsf{lis}^{[\alpha_i,\beta_i]}(\mathsf{sa}_i) \geq (1-4\epsilon)\mathsf{lis}(A)$*.*

Proof: Lemma IV.1 holds for all subarrays whose contribution to opt is at least $\epsilon\sqrt{n}\lambda$. Let $S \subseteq [\sqrt{n}]$ denote the set of coordinates such that for each $i \in S$

$$\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i) \geq \epsilon\sqrt{n}\lambda.$$

Since $\sum_{i=1}^{\sqrt{n}} \mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i) \geq n\lambda$ and $\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i) \leq \sqrt{n}$, we have

$$\begin{aligned}&\sum_{i\in S} \mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i) \qquad (2)\\ \geq &\sum_{i=1}^{\sqrt{n}} \mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i) - \sqrt{n}\cdot\epsilon\sqrt{n}\lambda \geq \mathsf{lis}(A) - \epsilon n\lambda.\end{aligned}$$

Let $T \subseteq S$ denote the set of coordinates such that for each $i \in T$,

$$\mathsf{lis}^{[\alpha_i,\beta_i]}(\mathsf{sa}_i) \geq (1-\epsilon)\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i),$$

and

$$\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i) \geq \epsilon\sqrt{n}\lambda.$$

Now we show that with probability at least $1 - \exp(-\Omega(\epsilon^2\sqrt{n}\lambda))$,

$$\sum_{i\in T} \mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i) \geq (1-2\epsilon)\sum_{i\in S} \mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i). \tag{3}$$

For each $i \in S$, let X_i denote a random variable such that

$$X_i = \begin{cases}\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i), & \text{with probability of } i \in T;\\ 0, & \text{with probability of } i \notin T,\end{cases}$$

and $X = \sum_{i\in S} X_i$. By Lemma IV.1 (with $\delta = \epsilon$), We have $\mathbb{E}[X] \geq (1-\epsilon)\sum_{i\in S} \mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i)$. By Hoeffding bound (Theorem V.2),

$$\begin{aligned}&\Pr[X - \mathbb{E}[X] \geq \epsilon\mathbb{E}[X]]\\ \leq\ &2\exp\left(-\frac{2\epsilon^2(\mathbb{E}[X])^2}{\sum_{i\in S}\left(\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i)\right)^2}\right)\\ \leq\ &2\exp\left(-\frac{2\epsilon^2(1-\epsilon)^2\left(\sum_{i\in S}\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i)\right)^2}{n^{3/2}}\right)\\ \leq\ &2\exp(-2\epsilon^2(1-\epsilon)^4\sqrt{n}\lambda)\\ \leq\ &\exp(-\Omega(\epsilon^2\sqrt{n}\lambda)).\end{aligned}$$

Hence, Equation (3) holds with probability at least $1 - \exp(-\Omega(\epsilon^2\sqrt{n}\lambda))$.

Conditioned on Equation (3), we have

$$\begin{aligned}&\sum_{i\in T} \mathsf{lis}^{[\alpha_i,\beta_i]}(\mathsf{sa}_i)\\ \geq &\sum_{i\in T}(1-\epsilon)\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i)\\ \geq &(1-\epsilon)(1-2\epsilon)\sum_{i\in S}\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]}(\mathsf{sa}_i)\\ \geq &(1-\epsilon)(1-2\epsilon)(\mathsf{lis}(A) - \epsilon n\lambda)\\ \geq &(1-4\epsilon)\mathsf{lis}(A)\end{aligned} \tag{4}$$

where the first inequality follows from the definition of T, the second inequality follows from Equation (3), the third inequality follows from Equation (2) and the last inequality follows from $\mathsf{lis}(A) \geq n\lambda$.

Finally, by Equation (3) and (4), we have

$$\begin{aligned}&\Pr\left[\sum_{i\in T}\mathsf{lis}^{[\alpha_i,a_{\beta_i}]}(\mathsf{sa}_i) \geq (1-4\epsilon)\mathsf{lis}(A)\right]\\ \geq&1 - \exp(-\Omega(\epsilon^2\sqrt{n}\lambda)).\end{aligned}$$

■

B. *Constructing Approximately Optimal Pseudo-solutions*

We call a sequence of $\sqrt{n}$ intervals $[\ell_1, r_1], [\ell_2, r_2], \ldots, [\ell_{\sqrt{n}}, r_{\sqrt{n}}]$ a pseudo-solution if all of the intervals are monotone. That is $\ell_1 \leq r_1 < \ell_2 \leq r_2 < \ell_3 \leq r_3 \leq \ldots \leq \ell_{\sqrt{n}} \leq r_{\sqrt{n}}$. These intervals denote solution-domains for the subarrays. We also may decide not assign any solution domain to a subarray in which case we show the corresponding interval by $\emptyset$. Indeed, $\emptyset$ does not break the monotonicity of a pseudo-solution. The quality of a pseudo-solution is defined as $\sum_i \mathsf{lis}^{[\ell_i,r_i]}(\mathsf{sa}_i)$. We denote the quality of a pseudo-solution ps by $\mathsf{q}(\mathsf{ps})$.

Another way to interpret Corollary IV.2 is that one can construct a pseudo-solution using the sampled elements whose quality is at least a $1 - \epsilon$ fraction of $\mathsf{lis}(A)$. In this section, we present an algorithm to construct a small set of pseudo-solutions with the promise that at least one of them has a quality of at least $\mathsf{lis}(A)/t$, where t is the number of pseudo-solutions. Finally, in Section IV-C, we present a method to approximate the size of the optimal solution using pseudo-solutions.

We construct the pseudo-solutions via Algorithm 5. The input of Algorithm 5 is the set of candidate domain intervals obtained by Algorithm 4 on every subarray. We first find an assignment of candidate solution domains to the subarrays which is monotone and has the largest number of candidate domain intervals. (This step can be implemented by dynamic programming.) We make a pseudo-solution out of this assignment and update the set of candidate intervals by removing the ones which are used in our pseudo-solution. We then repeat the same procedure to construct the second

Algorithm 5 Constructing the pseudo solutions

```
procedure ConstructPseudoSolutions($\mathsf{cdi}_1, \ldots, \mathsf{cdi}_{\sqrt{n}}$)    ▷ Lemma IV.3,IV.4,IV.5
        ▷ $\{\mathsf{cdi}_i\}_{i\in[\sqrt{n}]}$ is $\sqrt{n}$ sets of candidate domain intervals
    pseudo-solutions ← ∅
    while true do
        assg ← largest assigment of candidate domain intervals to subarrays which is monotone
        if assg contains less than $\epsilon\sqrt{n}\lambda$ non-empty candidate domain intervals then
            break
        else
            Add assg to pseudo-solutions
            for $i \leftarrow 1$ to $\sqrt{n}$ do
                if assg contains a candidate domain interval for subarray $\mathsf{sa}_i$ then
                    remove the corresponding candidate domain interval from $\mathsf{cdi}_i$
                end if
            end for
        end if
    end while
    return pseudo-solutions $\mathsf{ps}_1, \mathsf{ps}_2, \ldots, \mathsf{ps}_t$
end procedure
```

pseudo-solution and update the candidate solution domains accordingly. We continue on, until the number of solution domains used in our pseudo-solution drops below $\epsilon\sqrt{n}$ in which case we stop.

We first prove in Lemma IV.3 that the number of pseudo-solutions constructed in Algorithm 5 is bounded by $O(k^2/(\lambda\epsilon))$. Next, we show in Lemma IV.4 that at least one of the pseudo-solutions constructed by Algorithm 5 has a quality of at least $\Omega(\mathsf{lis}(A)/t)$ where t is the number of pseudo-solutions. Finally we prove in Lemma IV.5 that the running time of Algorithm 5 is $O(tk^2\sqrt{n}\log n)$.

Lemma IV.3 (number of pseudo-solutions)**.** *For each $i \in [\sqrt{n}]$, let cdi_i be a set of at most k^2 candidate domain intervals. Let t denote the number of pseudo-solutions constructed in Algorithm 5. Then, we have $t \le k^2/(\lambda\epsilon)$.*

Proof: Note that for each subarray we have at most k^2 candidate domain intervals. Since there are $\sqrt{n}$ subarrays, then in total we have $\sqrt{n}k^2$ candidate domain intervals. Each time we construct a pseudo-solution, the total number of the candidate domain intervals is decreased by at least $\epsilon\sqrt{n}\lambda$. Thus, the total number of pseudo-solutions t can be upper bounded,

$$t \le \frac{\sqrt{n}k^2}{\epsilon\sqrt{n}\lambda} \le \frac{k^2}{\lambda\epsilon}.$$

■

Lemma IV.4 (quality of pseudo-solutions)**.** *Let $\mathsf{ps}_1, \mathsf{ps}_2, \ldots, \mathsf{ps}_t$ be the pseudo-solutions constructed by Algorithm 5. If $\mathsf{lis}(A) \ge n\lambda$ holds, then with probability at least $1 - \exp(-\Omega(\sqrt{n}\lambda))$, there exists an $i \in [t]$ such that*

$$\mathsf{q}(\mathsf{ps}_i) \ge \frac{\mathsf{lis}(A)}{2t}.$$

Proof: Let us focus again on the actual solution domains $[\mathsf{sm}(\mathsf{sa}_1), \mathsf{lg}(\mathsf{sa}_1)]$,$[\mathsf{sm}(\mathsf{sa}_2), \mathsf{lg}(\mathsf{sa}_2)]$, $\ldots$, $[\mathsf{sm}(\mathsf{sa}_{\sqrt{n}}), \mathsf{lg}(\mathsf{sa}_{\sqrt{n}})]$.

We define set $S \subseteq [\sqrt{n}]$ such that

$$\mathsf{lis}^{[\mathsf{sm}(\mathsf{sa}_i),\mathsf{lg}(\mathsf{sa}_i)]} \ge \epsilon\sqrt{n}\lambda, \forall i \in S.$$

Using Corollary IV.2 with $\epsilon = 1/10$, we know that there is a monotone pseudo-solution $[\alpha_i, \beta_i]_{i\in T}$ ($T \subseteq S$) such that $[\alpha_i, \beta_i]$ are candidate domain intervals and

$$\sum_{i\in T} \mathsf{lis}^{[\alpha_i,\beta_i]}(\mathsf{sa}_i) \ge (1 - 4\epsilon)\mathsf{lis}(A).$$

Denote this pseudo-solution as sol.

At the time we terminate Algorithm 5, there are at most $\epsilon\sqrt{n}\lambda$ candidate domain intervals of sol does not belongs to any pseudo-solution of the pseudo-solution set. Also, since each candidate domain interval contributes at most $\sqrt{n}$ to the quality of the pseudo-solution containing the interval, we have

$$\begin{aligned}\sum_{i=1}^{t} \mathsf{q}(\mathsf{ps}_i) &\ge (1 - 4\epsilon)\mathsf{lis}(A) - \epsilon\sqrt{n}\lambda \cdot \sqrt{n} \\ &\ge (1 - 4\epsilon)\mathsf{lis}(A) - \epsilon\mathsf{lis}(A) \\ &= (1 - 5\epsilon)\mathsf{lis}(A).\end{aligned}$$

Thus, there exists an $i \in [t]$ such that

$$\mathsf{q}(\mathsf{ps}_i) \ge (1 - 5\epsilon)\mathsf{lis}(A)/t = \mathsf{lis}(A)/(2t).$$

■

Lemma IV.5 (running time)**.** *For each $i \in [\sqrt{n}]$, let cdi_i be a set of at most k^2 candidate domain intervals. Let t denote the number of pseudo-solutions. The running time of Algorithm 5 is bounded by $O(tk^2\sqrt{n}\log n)$.*

Proof: Lemma IV.3 states that Algorithm 5 terminates after constructing t pseudo-solutions.

Now we show that constructing each pseudo-solution takes time $O(k^2\sqrt{n}\log n)$. Our solution is based on a dynamic programming technique. Let $D : [\sqrt{n}] \times [k^2] \to \mathbb{N}$ be an array such that $D[i][j]$ stores the size of the largest monotone pseudo-solution for the first i subarrays which ends with the j'th candidate domain interval of sa_i. Using classic segment-tree data structure (this data structure can be found in many textbooks, e.g. [26]), one can compute

Algorithm 6 Evaluate the pseudo solutions

```
procedure EvaluatePseudoSolutions($\mathsf{ps}_1, \ldots, \mathsf{ps}_t$)    ▷ Lemma IV.6
                                ▷ $\{\mathsf{ps}_i\}_{i \in [t]}$ is a set of pseudo solutions
    $p \leftarrow \frac{1000t \log^4 n}{\epsilon^4 \lambda \sqrt{n}}$
    Randomly sample each $i \in [\sqrt{n}]$ with probability p, and put all the samples in a set $W$
    for each $\mathsf{ps}_j$ do
        $\widetilde{\mathsf{q}}(\mathsf{ps}_j) \leftarrow 0$
        for each interval $[\ell_i, r_i]$ in $\mathsf{ps}_j$ do
            if $i \in W$ then
                $\widetilde{\mathsf{q}}(\mathsf{ps}_j) \leftarrow \widetilde{\mathsf{q}}(\mathsf{ps}_j) + \mathsf{lis}^{[\ell_i, r_i]}(\mathsf{sa}_i)/p$
            end if
        end for
    end for
    return largest $\widetilde{\mathsf{q}}(\mathsf{ps}_j)$ for all $j \in [t]$
end procedure
```

the value of $D[i][j]$ in time $O(\log n)$ from the previously computed elements of the array.

Thus, the total running is bounded by $O(tk^2\sqrt{n}\log n)$. ■

C. Evaluating the Pseudo-solutions

We finally use a concentration bound to show that the quality of a pseudo-solution can be approximated well by sampling a small number of subarrays. Since a pseudo-solution specifies the range of the numbers used in every subarray, the quality of a pseudo-solution, or in other words, the size of the corresponding increasing subsequence of a pseudo-solution can be formulated as

$$\mathsf{q}(\mathsf{ps}) = \sum_{i=1}^{\sqrt{n}} \mathsf{lis}^{[\ell_i, r_i]}(\mathsf{sa}_i)$$

where $[\ell_i, r_i]$ denotes the corresponding solution domain of ps for sa_i.

In Lemma IV.6, we prove that by sampling $O(\log^{O(1)} n/\lambda^4)$ many subarrays and computing $\mathsf{lis}^{[\ell_i, r_i]}(\mathsf{sa}_i)$ for them, one can approximate the quality of a pseudo-solution pretty accurately.

Lemma IV.6 (the quality of pseudo-solution)**.** *Let $\lambda \in (0, 1)$ and ϵ be a constant in $(0, 1/100)$. Let $\mathsf{ps}_1, \mathsf{ps}_2, \cdots, \mathsf{ps}_t$ be a set of t pseudo-solutions. With probability at least $1 - \exp(-\Omega(\log^2 n))$, Algorithm 6 runs in time $O(t^2\sqrt{n}\log^{O(1)} n/\lambda)$ such that ,*

1) *If there exists an $i \in [t]$, $\mathsf{q}(\mathsf{ps}_i) \geq \frac{\lambda n}{2t}$, then the algorithm outputs an estimation at least $\frac{\lambda n}{4t}$.*
2) *If $\mathsf{q}(\mathsf{ps}_i) < \frac{\lambda n}{8t}$ for al the $i \in [t]$, then the algorithm outputs an estimation smaller than $\frac{\lambda n}{4t}$.*

Putting all the previous Lemmas together gives the following result:

Corollary IV.7 (algorithm for LIS decision problem)**.** *Given a length-n sequence A, let $\lambda \in [1/n, 1]$. There is a randomized algorithm that runs in time $O(\lambda^{-7}\sqrt{n}\log^{O(1)} n)$ such that with probability $1 - 1/\mathsf{poly}(n)$*

- *The algorithm accepts if $\mathsf{lis}(A) > n\lambda$.*
- *The algorithm rejects if $\mathsf{lis}(A) = O(n\lambda^4)$.*

Proof: The correctness follows from Lemma IV.6, Lemma IV.4, and Lemma IV.5.

Running time: The running time is

$$\begin{aligned} \text{time} &= \underbrace{O(tk^2\sqrt{n}\log^{O(1)} n)}_{\text{Lemma IV.5}} + \underbrace{O(t^2\lambda^{-1}\sqrt{n}\log^{O(1)} n)}_{\text{Lemma IV.6}} \\ &= O(t^2\lambda^{-1}\sqrt{n}\log^{O(1)} n) \\ &= O(k^4\lambda^{-3}\sqrt{n}\log^{O(1)} n) \\ &= O(\lambda^{-7}\sqrt{n}\log^{O(1)} n) \end{aligned}$$

Thus, we complete the proof. ■

Finally, by starting with $\lambda = 1$ and iteratively multiplying λ by a $1/(1+\epsilon)$ factor until a solution is found, we can approximate $\mathsf{lis}(A)$ within an approximation factor of $O(\lambda^3)$.

Theorem IV.8 (polynomial approximation for LIS)**.** *Given a length-n sequence A such that $\mathsf{lis}(A) = n\lambda$ where $\lambda \in [1/n, 1]$ is unknown to the algorithm. There is an algorithm that runs in time $\widetilde{O}(\lambda^{-7}\sqrt{n})$ and outputs a number est such that*

$$\Omega(\mathsf{lis}(A)\lambda^3) \leq \mathsf{est} \leq O(\mathsf{lis}(A)).$$

with probability at least $1 - 1/\mathsf{poly}(n)$.

We remark that one can turn Theorem IV.8 into an algorithm with running time $\widetilde{O}(n^{17/20})$ by considering two cases separately. If $\lambda < n^{-1/20}$ we sample the array with a rate of $n^{-3/20}$ and compute the LIS for the sampled array. Otherwise, the running time of the algorithm is already bounded by $\widetilde{O}(n^{17/20})$.

D. An $O(n^\kappa)$ Time Algorithm via Bootstrapping

Let us move a step backward and analyze the previous algorithm for obtaining an $O(\lambda^3)$ approximate solution. We first divide the input array into $\sqrt{n}$ subarrays of size $\sqrt{n}$ and after constructing the pseudo-solutions, we sample $O(\lambda^4)$ subarrays to estimate the size of the solution for pseudo-solutions. The reason we set the size of the subarrays to $\sqrt{n}$ is that there is a trade-off between the first and the last steps of the algorithm. More precisely, if we have more than $\sqrt{n}$ subarrays then the number of samples we draw in the beginning would exceed $O_\lambda(\sqrt{n})$. On the other hand, having fewer than $\sqrt{n}$ subarrays results in larger subarrays which would be costly in the last step.

If one favors the running time over the approximation factor, the following improvement can be applied to the algorithm: In the last step of the algorithm, instead of sampling the entire subarrays and computing lis for every

Algorithm 7 Recursive Estimate LIS with Oracle

```
 1: procedure RecursiveEstimationWithOracle(Oracle, $A, \lambda, \ell, r$)      ▷ Lemma IV.9
 2:     ▷ input: sequence $A$, parameter $\lambda$, domain interval $[\ell, r]$
 3:     ▷ assume $\mathsf{sa}_1, \mathsf{sa}_2, \ldots, \mathsf{sa}_{n^\kappa}$ are subarrays of $A$
 4:     ▷ subroutine Oracle approximate LIS for subarrays with approximation factor $f(\lambda)$
 5:     for $i \in [\zeta]$ do
 6:         $\mathsf{cdi}_i \leftarrow$ ConstructCandidateDomains($\mathsf{sa}_i$)
 7:         discard all the intervals which are not in $[\ell, r]$ from $\mathsf{cdi}_i$
 8:     end for
 9:     $\{\mathsf{ps}_1, \ldots, \mathsf{ps}_t\} \leftarrow$ ConstructPseudoSolutions($\mathsf{cdi}_1, \ldots, \mathsf{cdi}_\zeta$)
10:     $\lambda_0 \leftarrow \left(\frac{\lambda}{2^8}\right)^4$
11:     $p \leftarrow \frac{20 \log^4 n}{\lambda_0 \zeta}$
12:     randomly sample each $i \in [\zeta]$ with probability $p$, and put all the samples in a set $Q$
13:     for $j \in [t]$ do
14:         $\mathsf{c} \leftarrow 0$
15:         for $i \in W$ do
16:             if $\exists [\ell_i, r_i] \in \mathsf{ps}_j$ and Oracle($\mathsf{sa}_i, \lambda_0, \ell_i, r_i$) accepts then
17:                 $\mathsf{c} \leftarrow \mathsf{c} + 1$
18:             end if
19:         end for
20:         if $\mathsf{c} \geq 3\lambda_0 p \zeta / 4$ then
21:             return accept
22:         end if
23:     end for
24:     return reject
25: end procedure
```

pseudo-solution, we recursively call the same procedure to approximate the size of the solution for each subarray. This way, having large subarrays would no longer be an issue and therefore we can have fewer subarrays to improve the number of samples we draw in the first step of the algorithm.

More formally, in order to obtain a running time of $O(\mathsf{poly}(\lambda) n^\kappa)$ we set the size of each subarray to $n^{1-\kappa}$ and therefore after constructing the pseudo-solutions, the problem boils down to approximating the solution for $\mathsf{poly}(\lambda)$ many subarrays of length $n^{1-\kappa}$. By running the same algorithm, we would have n^κ subarrays of length $n^{1-2\kappa}$ in the second iteration. After $1/\kappa - 1$ iterations, the subarrays are small enough and we can access all their elements in time $O(\mathsf{poly}(\lambda) n^\kappa)$. Of course, this imposes a factor of $\mathsf{poly}(\lambda)$ to the approximation.

By generalizing the ideas from previous subsections, we show that if there is an algorithm for LIS with approximation factor $f(\lambda)$, then we can get a $\left(f\left(\frac{\lambda^4}{2^{32}}\right) \cdot \frac{\lambda^4}{2^{33}}\right)$-approximate LIS algorithm with better running time using the $f(\lambda)$-approximate algorithm as a subroutine.

Lemma IV.9. *Assume we partition the sequence into ζ subarrays, where ζ is polynomially related to the length of the input sequence. For parameter $\lambda \in (0, 1)$, let* Oracle *be a $f(\lambda)$-approximate LIS algorithm (with respect to a domain interval) with running time $g(n, \lambda)$ and success probability $1 - \exp(-\Omega(\log^2 n))$ where n is the length of the input sequence. Then Algorithm 7 using* Oracle *as a subroutine is a $\left(f\left(\frac{\lambda^4}{2^{32}}\right) \cdot \frac{\lambda^4}{2^{33}}\right)$-approximate LIS algorithm with*

$$O\left(\lambda^{-4} g\left(\frac{n}{\zeta}, \frac{\lambda^4}{2^{32}}\right) \log^{O(1)} n + \lambda^{-7} \zeta \log^{O(1)} n\right)$$

running time and success probability $1 - \exp(-\Omega(\log^2 n))$, where ζ is the number of subarrays.

Proof: We first prove the correctness of the algorithm. Let A be a sequence of length n, and $\mathsf{sa}_1, \ldots, \mathsf{sa}_\zeta$ be the subarrays.

Consider the case of $\mathsf{lis}^{[\ell,r]}(A) \geq \lambda n$. By Corollary IV.2 and Lemma IV.4 with $\epsilon = \delta = 1/10$, with probability $1 - \exp(-\Omega(\zeta\lambda))$, there exists a pseudo-solution ps_j within interval $[\ell, r]$ satisfying

$$\begin{aligned} \mathsf{q}(\mathsf{ps}_j) &\geq \frac{\mathsf{lis}^{[\ell,r]}(A)}{2t} \\ &\geq \frac{\mathsf{lis}^{[\ell,r]}(A)\lambda\epsilon}{2k^2} \\ &\geq \frac{\mathsf{lis}^{[\ell,r]}(A)\lambda\epsilon}{2 \cdot 20^2 \log^2(1/\delta)/(\lambda^2\epsilon^4)} \\ &\geq \frac{\epsilon^5 \lambda^4}{800 \log^2(1/\delta)} n \geq \frac{\lambda^4}{2^{31}} n. \end{aligned}$$

Let α denote the number of subarrays sa_i such that $\mathsf{lis}^{[\ell_i,r_i]}(\mathsf{sa}_i) \geq \lambda_0 n/\zeta$ where $[\ell_i, r_i]$ is the interval for subarray sa_i in ps_j. We have

$$\alpha \geq \frac{\mathsf{q}(\mathsf{ps}_j) - \lambda^4 n/2^{32}}{n/\zeta} \geq \frac{\lambda^4 \zeta}{2^{32}}.$$

By Chernoff bound, Step 14 to Step 22 of Algorithm 7 accepts on ps_j with probability at least $1 - \exp(-\Omega(\log^2 n))$.

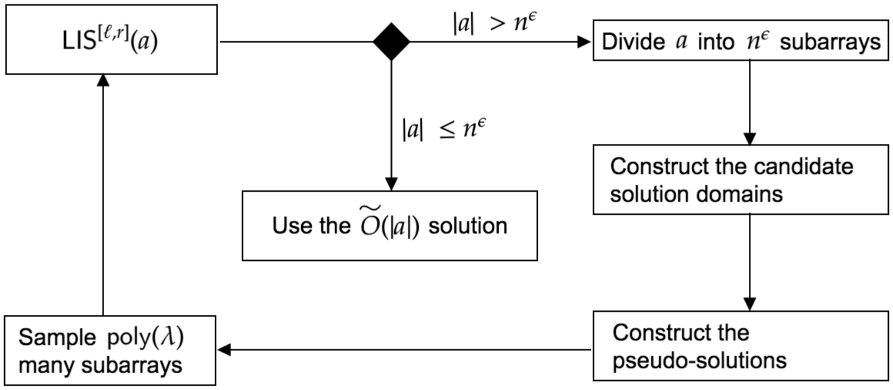
Consider the case of

$$\mathsf{lis}^{[\ell,r]}(A) \leq f\left(\frac{\lambda^4}{2^{32}}\right) \frac{\lambda^4}{2^{33}} n.$$

Then for any pseudo-solution ps_j, we have

$$\mathsf{q}(\mathsf{ps}_j) \leq f\left(\frac{\lambda^4}{2^{32}}\right) \frac{\lambda^4}{2^{33}} n.$$

Let β denote the number of subarrays sa_i such that $\mathsf{lis}^{[\ell_i,r_i]}(\mathsf{sa}_i) \geq f(\lambda^4/2^{32}) n/\zeta$ where $[\ell_i, r_i]$ is the interval



LIS$^{[\ell,r]}(a)$
$|a| > n^\epsilon$
Divide a into n^ϵ subarrays
$|a| \le n^\epsilon$
Construct the candidate solution domains
Use the $\widetilde{O}(|a|)$ solution
Construct the pseudo-solutions
Sample poly(λ) many subarrays

Figure 6: The flowchart of the $O_\lambda(n^\epsilon)$ time algorithm is shown.

for subarray sa_i in ps_j. We have

$$\beta \le \frac{\mathsf{q}(\mathsf{ps}_j)}{f(\lambda^4/2^{32})n/\zeta} \le \frac{\lambda^4}{2^{33}}\zeta.$$

By Chernoff bound, Step 14 to Step 22 of Algorithm 7 do not accept on ps_j with probability at least $1-\exp(-\Omega(\log^2 n))$. By union bound, Algorithm 7 rejects with probability at least $1-\exp(-\Omega(\log^2 n))$.

Hence, Algorithm 7 is a $\left(f\left(\frac{\lambda^4}{2^{32}}\right)\cdot\frac{\lambda^4}{2^{33}}\right)$-approximate algorithm for LIS of length n with success probability at least $1-\exp(-\Omega(\log^2 n))$.

Finally, we discuss the running time of Algorithm 7. By the definition of procedures CONSTRUCTCANDIDATEDOMAINS and CONSTRUCTPSEUDOSOLUTIONS, the running time of Step 5 to Step 9 of Algorithm 7 is $O(\lambda^{-9}\zeta\log^{O(1)} n)$. By Lemma IV.3, $t = O(\lambda^{-3})$, and by Chernoff bound with probability $1-\exp(-\Omega(\log^2 n))$ the size of Q is at most $O(\lambda^{-4}\log^4 n)$. Hence, the running time of Step 12 to Step 24 of Algorithm 7 is $O(\lambda^{-7}g(n/\zeta,\lambda^4/2^{32})\log^4 n)$. ■

By definition, we have the following basic fact about approximate ratio.

Fact IV.10. *Let f and f' be two functions mapping $(0,1)$ to $(0,1)$ such that $f(\lambda)\ge f'(\lambda)$ for any $\lambda\in(0,1)$. If there is a $f(\lambda)$-approximate LIS algorithm, then the algorithm is also $f'(\lambda)$-approximate.*

Now we present algorithm to approximate LIS using $\widetilde{O}(n^\kappa\mathsf{poly}(\lambda^{-1}))$ space by applying the pseudo-solution construction-evaluation framework recursively. In particular, we use the same algorithm on subarrays as an oracle and apply Lemma IV.9 recursively to approximate the entire sequence with slightly worse approximation ratio (compared with approximation ratio of the oracle).

Lemma IV.11. *Let κ be a constant of $(0,1)$ and $\lambda\in(0,1)$. Algorithm 8 approximates LIS with approximation ratio*

$$\frac{\lambda^{2\cdot 4^{(\lceil 1/\kappa\rceil-1)}}}{256^{3\cdot 4^{(\lceil 1/\kappa\rceil-1)}}}$$

and running time $O(n^\kappa\cdot\lambda^{-4^{O(1/\kappa)}}\log^{O(1)} n)$ and success probability $1-\exp(-\Omega(\log^3 n))$.

Algorithm 8 Recursive Estimate LIS

```
procedure RECURSIVELIS($A, \lambda, \ell, r$)    ▷ Lemma IV.11
    ▷ input: sequence $A$, parameter $\lambda$, domain interval $[\ell, r]$
        ▷ assume $\mathsf{sa}_1, \mathsf{sa}_2, \ldots, \mathsf{sa}_{n^\kappa}$ are subarrays of $A$
    if the length of $A$ is greater than $n^{2\kappa}$ then
        return RECURSIVELISWITHORACLE(RECURSIVELIS, $A, \lambda, \ell, r$) with $\zeta = n^\kappa$
    else
        for $i \in [n^\kappa]$ do
            $\mathsf{cdi}_i$ ← CONSTRUCTCANDIDATEDOMAINS($\mathsf{sa}_i$)
            discard all the intervals which are not in $[\ell, r]$ from $\mathsf{cdi}_i$
        end for
        $\{\mathsf{ps}_1, \ldots, \mathsf{ps}_t\}$ ← CONSTRUCTPSEUDOSOLUTIONS($\mathsf{cdi}_1, \ldots, \mathsf{cdi}_{n^\kappa}$)
        if EVALUATEPSEUDOSOLUTIONS($\mathsf{ps}_1, \ldots, \mathsf{ps}_t$) $\ge \lambda|A|$ then
            return accept
        else
            return reject
        end if
    end if
end procedure
```

Proof: We first prove the correctness of the algorithm by induction. Without loss of generality, we assume $1/\kappa$ is an integer.

For $i \in \{2, 3, \ldots, 1/\kappa\}$, denote

$$h_i(\lambda) = \frac{\lambda^{2\cdot 4^{(i-1)}-4}}{256^{2\cdot 4^{(i-1)}+3\cdot 4^{(i-2)}-7}}.$$

We show that if the length of the input sequence is $n^{i\cdot\kappa}$ then Algorithm 8 is $h_i(\lambda)$-approximate.

If the length of input sequence is $n^{2\kappa}$, then $h_2(\lambda) = \frac{\lambda^4}{2^{32}}$. By Corollary IV.2, Lemma IV.4, and Lemma IV.6, Algorithm 8 is $h_2(\lambda)$-approximate.

In the induction step, for an integer $2 \leq i < 1/\kappa$, we assume Algorithm 8 is $h_i(\lambda)$-approximate for input instance of length $n^{i\cdot\kappa}$. By Lemma IV.9, Algorithm 8 is $\left(h_i\left(\frac{\lambda^4}{2^{32}}\right)\frac{\lambda^4}{2^{33}}\right)$-approximate for input instance of length $n^{(i+1)\cdot\kappa}$. Since

$$\begin{aligned}
&h_i\left(\frac{\lambda^4}{2^{32}}\right)\frac{\lambda^4}{2^{33}}\\
=&\frac{\lambda^{4\cdot(2\cdot 4^{(i-1)}-4)}\cdot\lambda^4}{256^{4\cdot(2\cdot 4^{(i-1)}-4)}\cdot 256^{2\cdot 4^{(i-1)}+3\cdot 4^{(i-2)}-7}\cdot 2^{33}}\\
=&\frac{\lambda^{2\cdot 4^i-12}}{256^{2\cdot 4^i+2\cdot 4^{(i-1)}+3\cdot 4^{(i-2)}-18.875}}\\
>&\frac{\lambda^{2\cdot 4^i-4}}{256^{2\cdot 4^i+3\cdot 4^{(i-1)}-7}}\\
=&h_{i+1}(\lambda),
\end{aligned}$$

by Fact IV.10, Algorithm 8 is $h_{i+1}(\lambda)$-approximate for input instance of length $n^{(i+1)\cdot\kappa}$. Since

$$h_{1/\kappa}(\lambda) > \frac{\lambda^{2\cdot 4^{((1/\kappa)-1)}}}{256^{3\cdot 4^{((1/\kappa)-1)}}},$$

by Fact IV.10, Algorithm 8 is $\left(\frac{\lambda^{2\cdot 4^{((1/\kappa)-1)}}}{256^{3\cdot 4^{((1/\kappa)-1)}}}\right)$-approximate for input instance of length n.

By Corollary IV.2, Lemma IV.4, Lemma IV.6 and Lemma IV.9, we have the desired running time. The success probability is obtained by same corollaries/lemmas and union bound. ■

Finally, by starting with $\lambda = 1$ and iteratively multiplying λ by a $1/(1+\epsilon)$ factor until a solution is found, we can approximate $\mathsf{lis}(A)$ within an approximation factor of $\lambda^{O(4^{1/\kappa})}$.

Theorem IV.12. *Let κ be a constant of $(0, 1)$ and $\lambda \in (0, 1)$. There exists a $\widetilde{O}(n^{\kappa}\cdot\lambda^{-4^{O(1/\kappa)}})$ time algorithm for lis with approximation factor $\lambda^{4^{O(1/\kappa)}}$ and success probability $1 - \exp(-\Omega(\log^3 n))$.*

V. Probability and Graph Tools

In this section, we restate probability and graph tools that we use throughout this paper. All these theorems are proven in previous work.

Theorem V.1 (Chernoff Bounds). *Let $X = \sum_{i=1}^n X_i$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$, and all X_i are independent. Let $\mu = \mathbb{E}[X] = \sum_{i=1}^n p_i$. Then*
1. $\Pr[X \geq (1+\delta)\mu] \leq \exp(-\delta^2\mu/3)$, $\forall\delta > 0$;
2. $\Pr[X \leq (1-\delta)\mu] \leq \exp(-\delta^2\mu/2)$, $\forall 0 < \delta < 1$.

Theorem V.2 (Hoeffding bound). *Let $X_1, \cdots, X_n$ denote n independent bounded variables in $[a_i, b_i]$. Let $X = \sum_{i=1}^n X_i$, then we have*

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq 2\exp\left(-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right)$$

Theorem V.3 (Blakley-Roy inequality, [27], see also Proposition 3.1 in [28]). *Let G denote a graph that has n vertices and average degree d. The number of walks of length k in graph G is at least nd^k.*

Theorem V.4 (Turán theorem for bipartite graphs, [29], see also [30]). *For a graph G the Turán number $\mathrm{ex}(G, n)$ is the maximum number of edges that a graph on n vertices can have without containing a copy of G. For any $s \leq t$, $\mathrm{ex}(K_{s,t}, n) \leq \frac{1}{2}(t-1)^{1/s}n^{2-1/s} + o(n^{2-1/s})$*

Theorem V.5 (Turán theorem for cliques [31]). *Let G be any graph with n vertices, such that G is K_{r+1}-free. Then the number of edges in G is at most*

$$(1 - \frac{1}{r}) \cdot \frac{n^2}{2}.$$

Corollary V.6 (of Theorem V.5). *Let G be any graph with n vertices, such that G has no independent set of size $r+1$. Then the number of edges in G is at least*

$$\binom{n}{2} - (1 - \frac{1}{r}) \cdot \frac{n^2}{2} = \frac{nr + n^2}{2r}.$$

References

[1] M. Boroujeni, S. Ehsani, M. Ghodsi, M. HajiAghayi, and S. Seddighin, "Approximating edit distance in truly sub-quadratic time: Quantum and mapreduce," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2018.

[2] D. Charkraborty, D. Das, E. Goldenberg, M. Koucky, and M. Saks, "Approximating edit distance within constant factor in truly sub-quadratic time," in *Proceedings of the Fifty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, 2018.

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

[4] Z. Bar-Yossef, T. Jayram, R. Krauthgamer, and R. Kumar, "Approximating edit distance efficiently," in *Proceedings of the Forty-Fifth Annual IEEE Symposium on Foundations of Computer Science*, 2004.

[5] T. Batu, F. Ergun, and C. Sahinalp, "Oblivious string embeddings and edit distance approximations," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, 2006.

[6] A. Andoni and K. Onak, "Approximating edit distance in near-linear time," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, 2009.

[7] A. Andoni, R. Krauthgamer, and K. Onak, "Polylogarithmic approximation for edit distance and the asymmetric query complexity," in *Proceedings of the Fifty-First IEEE Annual Symposium on Foundations of Computer Science*, 2010.

[8] M. Saks and C. Seshadhri, "Estimating the longest increasing sequence in polylogarithmic time," in *Proceedings of the Fifty-First Annual IEEE Symposium on Foundations of Computer Science*, 2010.

[9] A. Backurs and P. Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)," in *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, 2015.

[10] A. Abboud, A. Backurs, and V. V. Williams, "Tight hardness results for LCS and other sequence similarity measures," in *Proceedings of the Fifty-Sixth IEEE Annual Symposium on Foundations of Computer Science*, 2015.

[11] A. Abboud, T. D. Hansen, V. V. Williams, and R. Williams, "Simulating branching programs with edit distance and friends," in *Proceedings of the Forth-Eighth Annual ACM SIGACT Symposium on Theory of Computing*, 2016.

[12] A. Abboud and A. Backurs, "Towards hardness of approximation for polynomial time problems," in *Proceedings of the Eighth Innovations in Theoretical Computer Science Conference*, 2017.

[13] A. Abboud and A. Rubinstein, "Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds," in *Proceedings of the Ninth Innovations in Theoretical Computer Science Conference*, 2018.

[14] L. Chen, S. Goldwasser, K. Lyu, G. N. Rothblum, and A. Rubinstein, "Fine-grained complexity meets IP=PSPACE," *Proceedings of the Thirtieth Annual ACM-SIAM symposium on Discrete Algorithm*, 2019.

[15] M. Hajiaghayi, M. Seddighin, S. Seddighin, and X. Sun, "Approximating LCS in linear time: Beating the barrier," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2019.

[16] M. Boroujeni and S. Seddighin, "Improved mpc algorithms for edit distance and ulam distance," in *SPAA*, 2019.

[17] M. Hajiaghayi, S. Seddighin, and X. Sun, "Massively parallel approximation algorithms for edit distance and longest common subsequence," in *SODA*. Society for Industrial and Applied Mathematics, 2019.

[18] W. J. Masek and M. S. Paterson, "A faster algorithm computing string edit distances," *Journal of Computer and System Sciences*, vol. 20, no. 1, pp. 18–31, 1980.

[19] A. Rubinstein and Z. Song, "Reducing approximate longest common subsequence to approximate edit distance," in *SODA*. https://arxiv.org/pdf/1904.05451, 2020.

[20] C. Schensted, "Longest increasing and decreasing subsequences," *Canadian Journal of Mathematics*, vol. 13, pp. 179–191, 1961.

[21] M. L. Fredman, "On computing the length of longest increasing subsequences," *Discrete Mathematics*, vol. 11, no. 1, pp. 29–35, 1975.

[22] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky, "Improved testing algorithms for monotonicity," in *Randomization, Approximation, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 1999, pp. 97–108.

[23] F. Ergün, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan, "Spot-checkers," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 717–751, 2000.

[24] E. Fischer, "The art of uninformed decisions: A primer to property testing," *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, vol. 1, pp. 229–264, 2004.

[25] N. Ailon, B. Chazelle, S. Comandur, and D. Liu, "Estimating the distance to a monotone function," *Random Structures & Algorithms*, vol. 31, no. 3, pp. 371–383, 2007.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT press, 2009.

[27] G. R. Blakley and P. Roy, "A hölder type inequality for symmetric matrices with nonnegative entries," *Proceedings of the American Mathematical Society*, vol. 16, no. 6, pp. 1244–1245, 1965.

[28] P. Keevash, B. Sudakov, and J. Verstraëte, "On a conjecture of erdős and simonovits: Even cycles," *Combinatorica*, vol. 33, no. 6, pp. 699–732, 2013.

[29] T. Kovári, V. Sós, and P. Turán, "On a problem of k. zarankiewicz," in *Colloquium Mathematicum*, vol. 1, no. 3, 1954, pp. 50–57.

[30] P. V. Blagojević, B. Bukh, and R. Karasev, "Turán numbers for $k_{s,t}$-free graphs: Topological obstructions and algebraic constructions," *Israel Journal of Mathematics*, vol. 197, no. 1, pp. 199–214, 2013.

[31] P. Turán, "On an external problem in graph theory," *Mat. Fiz. Lapok*, vol. 48, pp. 436–452, 1941.