Approximating Binary Longest Common Subsequence in Almost-Linear Time*

Xiaoyu He
xiaoyuh@princeton.edu
Princeton University
USA

Ray Li
rayyli@berkeley.edu
UC Berkeley
USA

ABSTRACT

The Longest Common Subsequence (LCS) is a fundamental string similarity measure, and computing the LCS of two strings is a classic algorithms question. A textbook dynamic programming algorithm gives an exact algorithm in quadratic time, and this is essentially best possible under plausible fine-grained complexity assumptions, so a natural problem is to find faster approximation algorithms. When the inputs are two binary strings, there is a simple $\frac{1}{2}$-approximation in linear time: compute the longest common all-0s or all-1s subsequence. It has been open whether a better approximation is possible even in truly subquadratic time. Rubinstein and Song showed that the answer is yes under the assumption that the two input strings have equal lengths. We settle the question, generalizing their result to unequal length strings, proving that, for any $\varepsilon > 0$, there exists $\delta > 0$ and a $(\frac{1}{2} + \delta)$-approximation algorithm for binary LCS that runs in $n^{1+\varepsilon}$ time. As a consequence of our result and a result of Akmal and Vassilevska-Williams, for any $\varepsilon > 0$, there exists a $(\frac{1}{q} + \delta)$-approximation for LCS over q-ary strings in $n^{1+\varepsilon}$ time.

Our techniques build on the recent work of Guruswami, He, and Li who proved new bounds for error-correcting codes tolerating deletion errors. They prove a combinatorial "structure lemma" for strings which classifies them according to their oscillation patterns. We prove and use an algorithmic generalization of this structure lemma, which may be of independent interest.

CCS CONCEPTS

• **Theory of computation** → **Design and analysis of algorithms**; *Error-correcting codes.*

KEYWORDS

longest common subsequence, approximation algorithms, almost-linear time, deletion codes

ACM Reference Format:
Xiaoyu He and Ray Li. 2023. Approximating Binary Longest Common Subsequence in Almost-Linear Time. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23), June 20–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3564246.3585104

*X. He was supported by the NSF Mathematical Sciences Postdoctoral Research Fellowships Program under Grant DMS-2103154. R. Li was supported by the NSF Mathematical Sciences Postdoctoral Research Fellowships Program under Grant DMS-2203067.

1 INTRODUCTION

In this paper, we give improved approximation algorithms for the Longest Common Subsequence (LCS), a fundamental string similarity measure that is of theoretical and practical interest. The LCS of two strings, as the name suggests, is the length of the longest sequence that appears as a (not necessarily contiguous) subsequence in both strings. The LCS is one of the most ubiquitous ways to quantify the similarity of two strings, a task that appears in a variety of contexts from spell checkers to DNA processing.

Computing the LCS is a classic algorithms question. A textbook dynamic programming algorithm gives an exact algorithm in quadratic time $O(n^2)$, while the fastest known algorithm runs in time $O(n^2/\log^2 n)$ [27]. Whether we can improve these algorithms has been a longstanding open question (see, for example, Problem 35 of [21]). Under fine-grained complexity assumptions such as the Strong Exponential Time Hypothesis [2, 5, 17] and even more plausible hypotheses [3], there is no exact algorithm for LCS in time $O(n^{2-\varepsilon})$ with $\varepsilon > 0$. Because of these barriers for exact algorithms, it is natural to wonder whether there are faster approximation algorithms.

When the inputs are two binary strings, the simple algorithm that computes the longest all-0s or all-1s common subsequence gives a $\frac{1}{2}$-approximation in linear time. Despite its simplicity, this has been the best known approximation for binary LCS on arbitrary inputs, even in truly subquadratic time ($n^{2-\varepsilon}$ for an absolute $\varepsilon > 0$). This raises the following natural question.

Question 1.1. Do there exist $\delta, \varepsilon > 0$ and a $(\frac{1}{2} + \delta)$-approximation algorithm of the LCS of two binary strings of length at most n in time $O(n^{2-\varepsilon})$?

Towards Question 1.1, Rubinstein and Song [30] showed that, if we assume the input strings have the same length, for all $\varepsilon > 0$, there is a $(\frac{1}{2} + \delta)$-approximation of the LCS in time $O(n^{1+\varepsilon})$ (δ depends on ε). However, for the general setting of unequal length inputs remained open.

Our main result answers Question 1.1 in full, handling unequal length strings.

Theorem 1.2. *For all $\varepsilon > 0$, there exists an absolute constant $\delta = \delta(\varepsilon) > 0$ and a deterministic algorithm that, given two binary strings x and y of not-necessarily-equal length, outputs a $(\frac{1}{2} + \delta)$-approximation of the longest common subsequence in time $O(n^{1+\varepsilon})$ where $n = \max(|x|, |y|)$.*[1]

[1]Our runtime is actually $O(n \cdot \min(|x|, |y|)^{\varepsilon})$, which is slightly better in the case y is much longer than x, but we state it as is for simplicity.

We note that our algorithm uses the equal-length LCS algorithm of [30] as a black box, so any improvements in the equal-length setting automatically yield improvements in the unequal-length setting. In general, if there is an equal-length LCS algorithm running in time $T(n)$ giving a $(\frac{1}{2}+\delta)$-approximation, our algorithm gives a $O((n+T(n))\log^A n)$ time $(\frac{1}{2}+\delta^A)$-approximation on unequal length strings, for an absolute constant A. Furthermore, while we present our algorithm as outputting the length of the longest common subsequence, we can output the subsequence of the promised length if the black-boxed equal-length LCS algorithm can.

Our work gets around a technical barrier for unequal length strings, which was highlighted in [7]. The algorithms of [30] used the intimate connection between LCS and Edit Distance, the number of insertions, deletions, and substitutions needed to transform one string to another. If we ignore substitutions, Edit Distance and LCS are in fact equivalent to compute exactly. Similar to LCS, there is no exact algorithm for Edit Distance in $n^{2-\varepsilon}$ time with $\varepsilon > 0$, under plausible fine-grained complexity assumptions [3, 13, 17]. Approximation algorithms for Edit Distance are well-studied, and a recent line of work [8–10, 12, 14, 15, 19, 22, 25] culminated in a constant-factor approximation of Edit Distance in almost-linear time [10]. Rubinstein and Song used these approximation algorithms for Edit Distance to obtain their approximation for LCS. However, because they rely on Edit Distance algorithms, they crucially use that the strings are equal length: note that if one string has length n and the other string has length $100n$, their edit distance is always at least $99n$, so even computing a 3-approximation of edit-distance of the two input strings would be unhelpful for approximating LCS.

Our work gets around this problem by using different techniques to handle unequal length strings. Our techniques are adapted from a recent work [23] that proves lower bounds for error-correcting codes correcting deletions [26, 31] via the following combinatorial result about LCS.

Theorem 1.3 ([23], deletion code limitation). *There exists an absolute constants $A, \delta > 0$ such that for any set C of binary strings of length n with $|C| \geq 2^{\log^A n}$, there exist two strings x and y with $\mathsf{LCS}(x,y) \geq (\frac{1}{2}+\delta)n$.*

Intuitively, we may expect the techniques for Theorem 1.3 to be useful here because it shares similarities with our main result, Theorem 1.2. While Theorem 1.3 is a "negative result" for deletion codes, it is a "positive result" in the algorithmic sense, as it shows that among any small set of strings, two of them have a long common subsequence. Furthermore, it has a similar flavor as Question 1.1, as both consider "beating the trivial matching" for LCS in binary strings. Thus, one might suspect then that these two problems are related, and we show indeed they are. On the other hand, adapting the techniques from [23] to our setting is nontrivial as we need to (i) make the combinatorial techniques algorithmic and (ii) handle unequal length strings (note in Theorem 1.3 all strings are of the same length).

Computing LCS is also interesting over larger alphabets. Approximating LCS when there is no restriction on the alphabet has been well studied [11, 16, 24, 28, 29], and currently the best result [11, 28] gives a randomized $\frac{1}{n^{o(1)}}$-approximation in linear time. Over an alphabet of a given size $q > 2$, there is, similar to the binary case, a trivial linear time $\frac{1}{q}$-approximation obtained by taking the longest common constant subsequence. For fixed q, over general q-ary inputs, this was the best known approximation, even in subquadratic time. For q-ary inputs where the two strings have the same length, Akmal and Vassilevska-Williams [7] (see also [6]) generalized the result of Rubinstein and Song, showing for all $\varepsilon > 0$ there is a $(\frac{1}{q}+\delta)$-approximation in $n^{1+\varepsilon}$ time.

By the work of Akmal and Vassilevska-Williams [7], our main result immediately implies improved approximation algorithms over non-binary alphabets, for the general case of not-necessarily-equal length strings. Akmal and Vassilevska-Williams showed that if there is a $(\frac{1}{2}+\delta)$-approximation for binary LCS (which we show), there is a $(\frac{1}{q}+\delta')$-approximation for q-ary LCS in essentially the same runtime. Hence, we have the following corollary.

Corollary 1.4 (Follows from Theorem 1.2 and Theorem 1 of [7]). *For all $\varepsilon > 0$ and integers $q \geq 2$, there exists an absolute constant $\delta = \delta(\varepsilon) > 0$ and a deterministic algorithm that, given two q-ary strings x and y of not-necessarily-equal length, outputs a $(\frac{1}{q}+\delta)$-approximation of the longest common subsequence in time $O(n^{1+\varepsilon})$ where $n = \max(|x|,|y|)$.*

2 PRELIMINARIES

For clarity of presentation, we sometimes drop floors and ceilings where they are not crucial.

Strings. For a string x, a *subsequence* of x is any string obtained by deleting any number of bits of x. A *substring* is a subsequence that appears as consecutive bits of x. Let $0(x)$ and $1(x)$ denote the number of zeros and ones, respectively, in x. A *property P* of binary strings is a set of binary strings. We say a string x *satisfies/has* a property P if x is in the set P.

Intervals. We use interval notation similar to that of [19]. By convention, an interval $I = [a,b]$ denotes the set $\{a+1, a+2, \dots, b\}$, and we write $\operatorname{start} I = a$ and $\operatorname{end} I = b$. Note that I and I' are disjoint if and only if either $\operatorname{end} I' \leq \operatorname{start} I$ or $\operatorname{end} I \leq \operatorname{start} I'$. The *length* of an interval $I = [a,b]$ is $b-a$. For a string x, let x_I denote the contiguous substring $x_{a+1}x_{a+2}\cdots x_b$. By abuse of notation, when the string x is understood, we may use I to refer to the substring x_I. For an integer w, we say interval I is w*-aligned* if $\operatorname{start} I$ and $\operatorname{end} I$ are multiples of w. An interval is a w*-interval* if it has length w and is w-aligned. Let $\operatorname{round}_w(I)$ denote the largest w-aligned subinterval of I. Note we always have $|\operatorname{round}_w(I)| > |I| - 2w$.

For an interval I and integer w, let $\mathcal{I}_w(I)$ be the collection of w-intervals that are subintervals of I. When a string x is understood (as it always will be), we write $\mathcal{I}_w := \mathcal{I}_w(|x|)$. Note that if $|x|$ is a multiple of w, the intervals of $\mathcal{I}_w$ partition $[m]$.

A *rectangle* is a product $I \times J$ where I and J are intervals. A *square* is a rectangle $I \times J$ with $|I| = |J|$. A *certified rectangle* is a pair $(I \times J, \kappa)$ where κ is a positive number.

Define a partial ordering on intervals, where $I < I'$ iff $\operatorname{end} I \leq \operatorname{start} I'$. That is, every element of I is less than every element of I'. Note that if two intervals have nonempty intersection, they are incomparable. We also define a partial ordering on rectangles, where $I \times J < I' \times J'$ iff $I < I'$ and $J < J'$. We say a collection of (certified) rectangles is *ordered* if any two (certified) rectangles are comparable under this partial order.

For any two strings x and y, fix a canonical matching $\tau = \tau(x, y)$ between the bits of x and y that achieves the longest common subsequence (τ is not necessarily unique, but we can fix it to be, say, the lexicographically earliest one). For $I \subset [|x|]$, let J_I^τ denote the (unique) smallest interval such that the bits of x_I are only matched with bits in $y_{J_I^\tau}$ in the matching τ. Note that clearly if I' and I are disjoint, then J_I^τ and $J_{I'}^\tau$ are disjoint.

For any string x, we write $d(x)$ for the density of x, i.e. the ratio between the number of ones in x and the length of x. For $\gamma > 0$, we say an interval I is *γ-balanced in x* if $d(x_I) \in [\frac{1}{2} \pm \gamma]$, and we say I is *γ-imbalanced in x* otherwise. If x is understood (as it always will be), we simply say γ-balanced and γ-imbalanced. A useful property of balanced strings x is that we can find LCS close to $|x|/2$ with any other string of the same length.

Lemma 2.1. *If x and y are strings such that x is γ-balanced and $|x| = |y|$, then $\mathsf{LCS}(x, y) \geq (\frac{1}{2} - \gamma)|x|$.*

Proof. Suppose without loss of generality y has at least $|y|/2$ ones. Then y has at least $|x|/2$ ones. Since x is γ-balanced, then x has at least $(\frac{1}{2} - \gamma)|x|$ ones, so the LCS is at least $(\frac{1}{2} - \gamma)|x|$. □

Algorithms. Let $\mathsf{Trivial}(x, y)$ denote the output of the simple algorithm that outputs the longest all-0s or all-1s subsequence. Clearly $\mathsf{Trivial}(x, y) = \max(\min(0(x), 0(y)), \min(1(x), 1(y)))$.

Rubinstein and Song showed that one can obtain a $(\frac{1}{2} + \delta)$-approximation of equal length LCS. Their result immediately extends to a $(\frac{1}{2} + \delta')$-approximation for near-equal length LCS, which we use.

Theorem 2.2 (Follows immediately from [30]). *For any $\varepsilon > 0$, there exists a $\delta_{eq} = \delta_{eq}(\varepsilon) > 0$ and a $(\frac{1}{2} + \delta_{eq})$-approximation of the LCS of two binary strings x and y with $|x|, |y| \in [(1 - \delta_{eq})n, (1 + \delta_{eq})n]$ in time $O(n^{1+\varepsilon})$.*

Let $\mathsf{EqLCS}(x, y)$ denote the output of the algorithm from Theorem 2.2.

3 PROOF SKETCH

In this section we give a high-level overview of our almost-linear time LCS approximation algorithm, Theorem 1.2. We start by describing the novel ingredient, our algorithmic structure lemma, Lemma 4.1. It states, roughly speaking, that binary strings s of length w can be classified among one of $O(\log w)$ oscillation types or scales, such that for any two strings s, t with the same type, there is a long subinterval s_I in s with $\mathsf{LCS}(s_I, t) > (1/2 + \delta)|s_I|$. Moreover, the lemma is algorithmic in that both the type of s and the long subinterval s_I are computable from s in time nearly linear in w.

To formally define oscillation types, we first introduce the notion of a flag. In a string x, an *ℓ-flag* is an index i such that between the ith one and the $(i + \ell)$-th one, there are strictly more than $10(\ell - 1)$ zeros. In other words, an ℓ-flag is a one-bit in s that is immediately followed by a 0-dense interval of length on the order of ℓ. The existence of many ℓ-flags in x means that x "oscillates at scale ℓ." An *ℓ^+-flag* is an index i that is a t-flag for some $t \geq \ell$, where t must be a power of two. The oscillation types guaranteed by Lemma 4.1 are as follows.

Definition 3.1 (Definition 4.5 below). Let ℓ be a power of two, $\ell \in [1, w]$, and $x \in \{0, 1\}^w$.

(1) We say that x is *ℓ-coarse* if $\ell \geq \varepsilon^2 w$ and there is a ε^2-imbalanced interval I in x of length ℓ. We say x is *coarse* if it is ℓ-coarse for some $\ell \geq \varepsilon^2 w$.
(2) We say that x is *ℓ-fine* if it is not coarse, $\ell < \varepsilon^2 w$, the number of ℓ^+-flags in x is at least εw, and x contains $(0^\ell 1^\ell)^{\varepsilon w/\ell}$ as a subsequence. We say x is *fine* if it is ℓ-fine for some $\ell < \varepsilon^2 w$.

To a first approximation, this means that every string x either has a long imbalanced subinterval or else behaves like the periodic string $(0^\ell 1^\ell)^{w/2\ell}$ for some ℓ.

Now we return to summarizing the proof of Theorem 1.2. By prior results [7, 30] (see also [6]), it suffices to consider the "perfectly balanced case," where the shorter string x has an equal number of zeros and ones.

Theorem 3.2. *For all $\varepsilon > 0$, there exists an absolute constant $\delta = \delta(\varepsilon) > 0$ and a deterministic algorithm that, on input strings x and y with $0(x) = 1(x) \leq \min(0(y), 1(y))$, gives a $(\frac{1}{2} + \delta)$-approximation of the longest common subsequence in time $O(n^{1+\varepsilon})$.*

Theorem 1.2 follows from Theorem 3.2 by prior work [7, 30]; for completeness include the details in Section 6. In the rest of this section, we sketch the proof of Theorem 3.2.

Our algorithm for Theorem 3.2, which is described in pseudocode in Algorithms 1 and 2, is a modification of the standard quadratic time DP algorithm for LCS, which we formulate as follows. The standard DP algorithm computes $\mathsf{LCS}(x, y)$ by computing the full array $\mathsf{DP}[i][j] := \mathsf{LCS}(x_{[i]}, y_{[j]})$, $0 \leq i \leq |x|$, $0 \leq j \leq |y|$ via the recursion

$$\mathsf{DP}[i][j] = \begin{cases} \max(& \mathsf{DP}[i][j-1], & \\ & \mathsf{DP}[i-1][j], & \\ & \mathsf{DP}[i-1][j-1] + 1) & \text{if } x_i = y_i \\ \max(& \mathsf{DP}[i][j-1], & \\ & \mathsf{DP}[i-1][j]) & \text{otherwise.} \end{cases}$$

In total this takes $O(|x| \cdot |y|)$ applications of the recursion. To prove Theorem 3.2, we don't need to compute the exact value of $\mathsf{LCS}(x, y)$, rather, we only need to output a value between $(1/2 + \delta)\,\mathsf{LCS}(x, y)$ and $\mathsf{LCS}(x, y)$. To estimate the LCS efficiently, we modify the DP above by recursing over large subrectangles instead of one bit at a time. We compute a collection of large rectangles $I \times J$ (where I and J are long subintervals of $[|x|]$ and $[|y|]$, respectively) and estimates $\kappa(I \times J)$ for their LCS (we call these *certified rectangles*). We guarantee that $\kappa(I \times J) \leq \mathsf{LCS}(x_I, y_J)$ in every rectangle, and we desire that many of these $\kappa(I \times J)$ are good estimates of $\mathsf{LCS}(x_I, y_J)$. (For readers familiar with [19], finding these rectangles is analogous to their "Covering Phase").

The large rectangles under consideration are θw-aligned subrectangles of $[|x|] \times [|y|]$, where $w \approx |x|/\log|x|$ is a typical length of the x-intervals and θ is a small constant discretization parameter (in the real proof, we use a coarser discretization for x-intervals than for y-intervals, but ignore that here for sake of illustration).

Our modified DP algorithm is then

$$\begin{aligned}\mathsf{DP}[i][j] = \max\big(&\mathsf{DP}[\mathrm{start}\, I - \theta w][\mathrm{start}\, J],\\ &\mathsf{DP}[\mathrm{start}\, I][\mathrm{start}\, J - \theta w],\\ &\mathsf{DP}[\mathrm{start}\, I][\mathrm{start}\, J] + \kappa(I \times J)\\ &\text{over } I \times J \in \mathcal{R} \text{ s.t. end}\, I = i, \text{end}\, J = j\big) \quad (1)\end{aligned}$$

where $\kappa(I\times J)$ denotes the lower bound for $\mathsf{LCS}(x_I, y_J)$ guaranteed by our certification algorithm. Observe that by induction $\mathsf{DP}[i][j]$ is still a lower bound for $\mathsf{LCS}(x_{[i]}, y_{[j]})$. Because of our discretization, we only need to consider i and j a multiple of θw, so the number of dynamic programming states drops from $O(|x|\cdot|y|)$ to $O(\frac{|x|\cdot|y|}{(\theta w)^2}) \le \tilde{O}(\frac{|y|}{|x|})$. Thus it remains to show we can quickly certify a collection of rectangles for which the dynamic program (1) outputs a $(\frac{1}{2}+\delta)$-approximation.

The main step is find a significant fraction of "good" rectangles for which $\kappa(I\times J) > (1/2+\gamma)\,\mathsf{LCS}(x_I, y_J)$. We look for good rectangles in three different ways, as shown in Algorithm 1. (1) First, we check for the "trivial" rectangles where $\mathsf{Trivial}(x_I, y_J) > (1/2+\gamma)|x_I|$ ($\gamma > 0$ chosen very small). (2) Next, we black-box the equal-length LCS algorithm of Rubinstein and Song, and efficiently check for squares $I\times J$ with $|I| = |J|$ and $\mathsf{LCS}(x_I, y_J) > (1/2+\gamma)|x_I|$. (3) Finally — and this is the main technical contribution of our work — we use the algorithmic structure lemma, Lemma 4.1, to efficiently compute "oscillation frequencies" for the intervals I and J. For any given rectangle $I\times J$ where $|J|$ is longer than $|I|$, they can then be certified quickly by checking if J has the same oscillation frequency as I. For technical reasons, for this last type of rectangle we are unable to guarantee that $\mathsf{LCS}(x_I, y_J) > (1/2+\gamma)|x_I|$ as we did for the other two types, but we can instead guarantee the weaker assumption that I has a long subinterval I' for which $\mathsf{LCS}(x_{I'}, y_J) > (1/2+\gamma)|x_{I'}|$. Handling this technicality requires some care, but to get across the main ideas we ignore this detail for the rest of this sketch and imagine that all certified rectangles satisfy $\mathsf{LCS}(x_I, y_J) > (1/2+\gamma)|x_I|$.

We also certify using the trivial algorithm a weaker set of "default" rectangles $I\times J$ where $\mathsf{Trivial}(I, J) \ge (1/2-\gamma^4)|I|$ (the constants are chosen for illustration). These rectangles exist all over the place and are used in the DP to fill in the gaps between the efficient ones above. We may assume $\mathsf{LCS}(x,y) \ge (1-\delta)|x|$ — or else the trivial matching, which is always $|x|/2$ by the setup of Theorem 3.2, gets a $(\frac{1+\delta}{2})$-approximation — and this assumption guarantees we can certify many good rectangles and many default rectangles. We show that while most of the $\kappa(I\times J)$ are $(1/2-\gamma^4)|I|$ coming from the default rectangles, a significant enough fraction of them are $(1/2+\gamma)|I|$ that for the final answer we get $\mathsf{DP}[|x|][|y|] > (1/2+\delta)|x|$.

Since we are going for an almost-linear time algorithm (and not just subquadratic), we need to be careful to certify the rectangles quickly. Note that, naively, there are roughly $(\frac{|y|}{\theta w})^2 \sim \tilde{\Theta}(\frac{|y|}{|x|})^2$ possible J-intervals. If y is much longer than x (say $|y| = |x|^3$), then we cannot simply try to certify every rectangle, or else the runtime is super-linear in the input size, even if we can certify rectangles in constant time. Instead, we restrict ourselves to certifying rectangles $I\times J$ where J is "minimal". That is, for each x-interval I and each $\mathrm{end}(J)$, we look for the minimal J where we can certify $\kappa(I\times J) \ge (1/2+\gamma)|I|$. We can find such J by binary search (the ability to binary search depends on a technical property of the algorithmic structure lemma), so that the number of rectangles we are checking is now only $\tilde{O}(\frac{|y|}{|x|})$, rather than $\tilde{O}(\frac{|y|}{|x|})^2$.

4 ALGORITHMIC STRUCTURE LEMMA

We now state and prove our algorithmic structure lemma. We note that the final algorithm uses this lemma as a black-box, and can be understood without the proof of this lemma. The interested reader can skip to Section 5 after Section 4.1.

4.1 Algorithm Structure Lemma Statement

The following is the key technical lemma. It is inspired by and builds upon the "Structure Lemma" of [23], which was used to prove new deletion code bounds.

Lemma 4.1 (Algorithm Structure Lemma). *There exists an absolute constant $\delta_{code} > 0$ such that for all sufficiently large w, there exists $T \le 2\log w$ and $2T$ string properties $P_1,\dots,P_T, Q_1,\dots,Q_T$ such that:*

1. *If string x has length w, then there exists a $t \in [T]$ such that x has property P_t.*
2. *If $\mathsf{LCS}(x,y) \ge (1-\delta_{code})|x|$ and x has property P_t, then y (not necessarily of length w) has property Q_t.*
3. *Property Q_t is hereditary, meaning that if y has Q_t and y is a subsequence of y', then y' has Q_t.*
4. *For every $t \in [T]$, and strings x and y, we can test if x satisfies P_t in time $O(w\log w)$. We can also preprocess the string y in time $O(|y|\log|y|)$, such that we can answer queries of the form "does y' satisfy Q_t," for substrings y' of y, in $O(w)$ time.*
5. *If string x has length w and property P_t and string y has property Q_t, then there exists an interval $I \subset [w]$ such that $\mathsf{LCS}(x_I, y) \ge \frac{|I|}{2} + \delta_{code} w$. Furthermore, given x and t, the interval I and the promised common subsequence of x_I and y can be chosen independent of y, and both can be found in time $O(w\log w)$.*

Remark 4.2. In item 5, it is easy to see that, if $\gamma \le \delta_{code}/10$, we may additionally assume (by starting with $\delta'_{code} = \delta_{code}/2$) that the interval I is γw-aligned by taking $I' := \mathrm{round}_{\gamma w}(I)$. We do so in the application in Section 5.

We now provide some more intuition for Lemma 4.1. First, we describe the properties P_t and Q_t that we actually use (based on Definition 3.1). For convenience, to define the properties, we index them as $P_{\ell,0}, P_{\ell,1}, P_\ell$ for $\ell \le w$ a power of 2, for a total of roughly $T \sim 3\log w$ properties.

- If $\ell \ge \varepsilon^2 w$ and $b \in \{0,1\}$, then $P_{\ell,b}$ is the property that x is ℓ-coarse, and its imbalanced interval of length ℓ is imbalanced in the direction of b-bits (i.e. has more b's than $\bar{b}$'s).
- If $\ell \ge \varepsilon^2 w$ and $b \in \{0,1\}$, then $Q_{\ell,b}$ is the property that y has at least $\left(\frac{1+\varepsilon^2}{2}\right)\ell$ b-bits.
- If $\ell < \varepsilon^2 w$, then P_ℓ is the property that x is ℓ-fine.
- If $\ell < \varepsilon^2 w$, then Q_ℓ is the property that y contains $y_\ell = (0^\ell 1^\ell)^{\varepsilon w/(5\ell)}$ as a subsequence.

The properties P_t are based on one of the key technical lemmas of the deletion codes bound [23], a combinatorial structure lemma. This structure lemma roughly says that for strings of length n, there are properties $\tilde{P}_1,\dots,\tilde{P}_T$ with $T \le O(\log n)$, such that

(i) any binary string of length n has some property $\tilde{P}_i$, and
(ii) if two strings x and y have property $\tilde{P}_i$, then x and y have (contiguous) substrings x' and y' of length $\Omega(n)$ whose LCS is at least $(\frac{1}{2}+\delta)(\frac{|x'|+|y'|}{2})$ (the real guarantee is stronger but more technical to state).

Theorem 1.3, the deletion codes lower bound, is proved (very roughly) by partitioning each string in C into polylog n substrings, finding by pigeonhole two strings x and y such that the types of the corresponding substrings of x and y agree, and using guarantee (ii) to find a $(\frac{1}{2}+\delta')w$ overall LCS.

Lemma 4.1 is a generalization of this combinatorial structure lemma that is "algorithmic" and "handles unequal length strings." The properties P_t that we choose in Lemma 4.1 are similar to the properties $\tilde{P}_t$ of [23], and it is not hard to check by inspection that the properties $\tilde{P}_t$ of [23] can be tested in linear time. The difficulty lies in finding properties Q_t of strings y that (a) can be "inherited" from properties like $\tilde{P}_t$ if y has a subsequence covering most of x, (b) can be tested efficiently, and (c) still guarantee an LCS advantage between x and y.

Because of Lemma 4.1, we can define the following algorithms, which we use in our final LCS algorithm.

Definition 4.3. For an integer w, let $P_1,\dots,P_T,Q_1,\dots,Q_T$ be the properties from Lemma 4.1. Let $\mathsf{GetPType}_w(x)$ denote the smallest index t such that x has property P_t. Let $\mathsf{IsQType}_w(x,t)$ be true if x has property Q_t and false otherwise. For x satisfying P_t for some t, let $\mathsf{GetI}_w(x,t)$ denote a γw-aligned interval I such that $\mathsf{LCS}(x_I,y) \geq \frac{|I|}{2}+\delta_{code}w$ for all y satisfying Q_t. Such an interval exists by Lemma 4.1 and Remark4.2.

By Lemma 4.1, $\mathsf{GetPType}_w(x)$ can be computed in $O(w\log^2 w)$ time, since one can simply test each of the $O(\log w)$ properties P_t. By Lemma 4.1, any string y can be preprocessed in $O(|y|\log|y|)$ time such that, for any contiguous substring y' of y, $\mathsf{IsQType}_w(y',t)$ can be computed in $O(w)$ time. Note that the input to $\mathsf{GetPType}_w$ must have length w, but the string input to $\mathsf{IsQType}_w$ can have arbitrary length. By Lemma 4.1, $\mathsf{GetI}_w(x,t)$ can be computed in $O(w\log w)$ time.

4.2 Combinatorial Structure Lemma and Types

In a string x, an *ℓ-flag* is an index i such that between the ith one and the $(i+\ell)$-th one, there are strictly more than $10(\ell-1)$ zeros. An *ℓ^+-flag* is an index i that is a t-flag for some $t \geq \ell$, where t must be a power of two. By abuse of notation, if i is a ℓ-flag, we may also call the i-th one of x a ℓ-flag. Note that there are many more zeros than ones between the ith and $(i+\ell)$-th one, so flags tell us where it is more advantageous to use zeros rather than ones in finding long subsequences.

The basis for the algorithmic structure lemma is a combinatorial structure lemma for strings, which we inherit from [23, Lemma 4.1]. We use a weaker form of the lemma, which has a significantly simpler statement and proof, and is also tailored to our algorithmic application. The proof is given in Appendix A.

Lemma 4.4 (Combinatorial Structure Lemma)**.** *For $\varepsilon = 10^{-5}$ and w sufficiently large, at least one of the following two conditions holds for every string $x \in \{0,1\}^w$.*

(1) *There exists $\ell \in [\varepsilon^2 w, w]$ equal to a power of two and an 0.1-imbalanced interval I in x of length ℓ.*
(2) *There exists $\ell \in [1, \varepsilon^2 w)$ equal to a power of two such that the number of ℓ^+-flags in x is at least εw, and x contains $(0^\ell 1^\ell)^{\varepsilon w/\ell}$ as a subsequence.*

For the remainder of Section 4, fix $\varepsilon := 10^{-5}$. Lemma 4.4 shows that every sufficiently long string x is of one of the below types, of which there are $\log w$ total.

Definition 4.5. Let ℓ be a power of two, $\ell \in [1,w]$, and $x \in \{0,1\}^w$.
(1) We say that x is *ℓ-coarse* if $\ell \geq \varepsilon^2 w$ and there is a ε^2-imbalanced interval I in x of length ℓ. We say x is *coarse* if it is ℓ-coarse for some $\ell \geq \varepsilon^2 w$.
(2) We say that x is *ℓ-fine* if it is not coarse, $\ell < \varepsilon^2 w$, the number of ℓ^+-flags in x is at least εw, and x contains $(0^\ell 1^\ell)^{\varepsilon w/\ell}$ as a subsequence. We say x is *fine* if it is ℓ-fine for some $\ell < \varepsilon^2 w$.

Note that for the convenience of our later proofs, we change the imbalanced threshold from 0.1 in Lemma 4.4 to the much smaller ε^2 in the above definition. Since every 0.1-imbalanced interval is also ε^2-imbalanced, Lemma 4.4 implies every sufficiently long string x is of one of the above two types.

4.3 Algorithmic Structure Lemma Ingredients

As in the last section, we fix $\varepsilon = 10^{-5}$. Also, for brevity, for every positive integer ℓ, define the special string

$$y_\ell := (0^\ell 1^\ell)^{\varepsilon w/(5\ell)}.$$

We prove two ingredients to justify our "Q_t" properties in the algorithmic structure lemma. The first is the simple observation that if x is ℓ-fine and $\mathsf{LCS}(x,y) \geq (1-\delta_{code})|x|$, then y inherits the easily testable subsequence y_ℓ from x.

Lemma 4.6. *Let $0 < \delta < \varepsilon/10$, ℓ be a power of two, and $w > \varepsilon^{-2}\ell$. If x is an ℓ-fine string of length w and $\mathsf{LCS}(x,y) \geq (1-\delta)w$, then y_ℓ is a subsequence of y.*

Proof. By definition, if x is ℓ-fine then x contains $(0^\ell 1^\ell)^{\varepsilon w/\ell} = y_\ell^5$ as a subsequence. Since $\mathsf{LCS}(x,y) \geq |x| - \delta w$ and y_ℓ^5 is a subsequence of x, we have $\mathsf{LCS}(y_\ell^5, y) \geq |y_\ell^5| - \delta w$. Thus, there is a subsequence of y obtained by applying δw deletions to y_ℓ^5. By counting, at most $2\delta w/\ell$ of the chunks $0^\ell 1^\ell$ in y_ℓ^5 receive more than $\ell/2$ of these deletions. The remaining $\varepsilon w/\ell - 2\delta w/\ell > 4\varepsilon w/(5\ell)$ chunks each have at least $\ell/2$ zeros and $\ell/2$ ones. Taking ℓ zeros from the first two chunks, ℓ ones from the next two, and so on, we see that y contains y_ℓ as a subsequence, as desired. □

The second ingredient implies that if x is ℓ-fine, then a substring of x can be matched advantageously with y_ℓ.

Lemma 4.7. *Let ℓ be a power of two, and $w > \varepsilon^{-2}\ell$. If a string x of length w is ℓ-fine, then there exists an interval I with $\mathsf{LCS}(x_I, y_\ell) \geq \frac{|I|}{2} + \varepsilon^3|x|$. Furthermore, given x and ℓ, we can determine the interval I and the common subsequence of x_I and y_ℓ in time $O(w\log w)$.*

Proof. The number of ℓ^+-flags in x is at least εw. By pigeonhole, there exists some interval $I' = [a,b]$ of length $4\varepsilon^2 w$ containing at least $2\varepsilon^3 w$ many ℓ^+-flags (the lost factor of two accounts for $2\varepsilon^2 w$ possibly not evenly dividing w). Furthermore, since x is not coarse, we may assume that each such ℓ^+-flag is an ℓ'-flag for some $\ell' \in [\ell, \varepsilon^2 w)$. Thus, the interval $I = [a, b + 11\varepsilon^2 w]$ has length $4\varepsilon^2 w + 11\varepsilon^2 w \le 20\varepsilon^2 w$ and $x' := x_I$ has at least $2\varepsilon^3 w$ many ℓ^+-flags (we cannot simply take $x' = x_{I'}$, as bits at the right end of I may be flags in x but not in $x_{I'}$). Furthermore, x' is ε^2-balanced since it has length at least $\varepsilon^2 w$ and is a substring of x, which is not coarse. We can find interval I', and thus I and x', in time $O(w \log w)$, because (with preprocessing of the string's prefix sums) we can test whether a bit is an ℓ^+-flag in $\log w$ time, so counting the flags in an interval can be done in $O(w \log w)$ time, and there are a constant number of intervals to check.

Now we claim $\mathsf{LCS}(x', y_\ell) \ge \frac{|x'|}{2} + \varepsilon^3 |x|$. Construct a common subsequence x'' of x' and y' as follows: Initialize a counter $i = 1$. While $i \le 1(x')$,

(1) Append a one to x'',
(2) If the ith one of x' is an ℓ'-flag for some $\ell' \ge \ell$, append $1 + \lfloor 10(\ell' - 1) \rfloor$ zeros to x'' and $i \leftarrow i + \ell'$.
(3) Otherwise $i \leftarrow i + 1$.

We claim the subsequence x'' has the following properties.

- x'' is a subsequence of x'.
- x'' is a subsequence of y_ℓ.
- x'' has length at least $\frac{|x'|}{2} + \varepsilon^3 |x|$.

To see the first property, take the subsequence of x' where the one added to x'' when the counter is i is matched to the i-th one of x', and the zeros added when the counter is i are the zeros between the i-th and $(i + \ell')$-th one of x', of which there are at least $1 + \lfloor 10(\ell' - 1) \rfloor$ because i is an ℓ'-flag in x'.

To see the second property, first note that $\lfloor 10(\ell' - 1) \rfloor + 1 \ge \ell'$ for all positive integers ℓ', so all runs of zeros in x'' have length at least ℓ. Write $x'' = 1^{a_1} 0^{a_2} 1^{a_3} 0^{a_4} \cdots 1^{a_{2k+1}}$, where all $a_{2i} \ge \ell$ and $a_{2i-1} \ge 1$ for all positive integers i (except possibly a_{2k+1}, which may be 0). Notice that 1^{a_i} and 0^{a_i} are each subsequences of $(0^\ell 1^\ell)^{r_i}$, where $r_i := \lceil a_i / \ell \rceil \le \frac{a_i}{\ell} + 1$. Thus, x'' is a subsequence of $(0^\ell 1^\ell)^r$ for $r := r_1 + \cdots + r_{2k+1}$. Thus, we have

$$\begin{aligned} r \le r_1 + \cdots + r_{2k+1} &\le \frac{a_1 + \cdots + a_{2k+1}}{\ell} + (2k+1) \\ &< \frac{4(a_1 + \cdots + a_{2k+1})}{\ell} \\ &\le \frac{80\varepsilon^2 w}{\ell} < \frac{\varepsilon w}{5\ell}, \end{aligned}$$

proving that x'' is a subsequence of y_ℓ. In the third inequality above, we used $a_2 + a_4 + \cdots + a_{2k} \ge k\ell$, so $2k + 1 \le 3k < \frac{3(a_1 + \cdots + a_{2k+1})}{\ell}$. In the fourth inequality, we used $a_1 + \cdots + a_{2k+1} = |x''| \le |x'| \le 20\varepsilon^2 w$.

To see the third property, notice that $|x''| - i$ only changes when a run of zeros is added to x''. If this run is added for an ℓ'-flag at i in x', then difference $|x''| - i$ increases by $1 + \lfloor 9(\ell' - 1) \rfloor \ge \ell'$, while the total number of flags skipped over is at most ℓ'. By induction on i, after every while-loop iteration, we have

$$|x''| - i \ge \#\{\ell^+\text{-flags in } [i]\}.$$

so the total length of x'' at the end is at least

$$1(x') + \#\{\ell^+\text{-flags in } x'\} \ge \left(\frac{1}{2} - \varepsilon^2\right)|x'| + 2\varepsilon^3 w \ge \frac{|x'|}{2} + \varepsilon^3 |x|,$$

as desired. The first inequality above follows from the fact that x' is ε^2-balanced, and the second from $|x'| \le 20\varepsilon^2 w$. □

4.4 Proof of the Algorithmic Structure Lemma

Proof of Lemma 4.1. Let $\delta_{code} = \varepsilon^4/2$. We define the properties P_t and Q_t based on the types in Definition 4.5. For convenience, we index them not as $P_1, P_2, \dots, P_T$, but rather as $P_{\ell,0}, P_{\ell,1}, P_\ell$ for $\ell \le w$ a power of 2, for a total of roughly $T \sim 3 \log w$ properties.

- If $\ell \ge \varepsilon^2 w$ and $b \in \{0, 1\}$, then $P_{\ell,b}$ is the property that x is ℓ-coarse, and its imbalanced interval of length ℓ is imbalanced in the direction of b-bits (i.e. has more b's than $\bar{b}$'s).
- If $\ell \ge \varepsilon^2 w$ and $b \in \{0, 1\}$, then $Q_{\ell,b}$ is the property that y has at least $\left(\frac{1+\varepsilon^2}{2}\right)\ell$ b-bits.
- If $\ell < \varepsilon^2 w$, then P_ℓ is the property that x is ℓ-fine.
- If $\ell < \varepsilon^2 w$, then Q_ℓ is the property that y contains $y_\ell = (0^\ell 1^\ell)^{\varepsilon w/(5\ell)}$ as a subsequence.

We now verify the conditions of Lemma 4.1.

(1) *For every length-w string x, there exists a t such that x has property P_t.*
This follows immediately from Lemma 4.4.
(2) *If $\mathsf{LCS}(x, y) \ge (1 - \delta_{code})|x|$ and x has property P_t, then y has property Q_t.*
First suppose $\ell \ge \varepsilon^2 w$, $b \in \{0, 1\}$, and x has property $P_{\ell,b}$. Then x has a substring x_I of length at least $\varepsilon^2 w$ with at least $(\frac{1}{2} + \varepsilon^2)\ell$ b-bits. The longest common subsequence of x_I and y has at least $|I| - \delta_{code} w$ of the bits of x_I, so y has at least $(\frac{1}{2} + \varepsilon^4)\ell - \delta_{code} w \ge (\frac{1+\varepsilon^4}{2})\ell$ b-bits, satisfying property $Q_{\ell,b}$.
If x has property P_ℓ for $\ell < \varepsilon^2 w$, x is ℓ-fine. By Lemma 4.6, y has property Q_ℓ.
(3) *For every t, property Q_t is hereditary, meaning that if y has Q_t and y is a subsequence of y', then y' has Q_t.*
This follows from the definition of Q_t and that being a subsequence is a transitive relation.
(4) *For every t, property P_t can be tested in time $O(w \log w)$, and property Q_t can be tested in time $O(w)$ on substrings of a string y, after $O(|y| \log |y|)$ prepreocessing.*
Testing the P_t's can be done in $O(w \log w)$ because, after $O(w)$ preprocessing by storing all prefix sums, we can check whether any particular index is an ℓ-flag or not in $O(1)$ time, and for any particular property P_t, we need to check at most $O(w \log w)$ flags.
To test Q_t, first note that if we are working with a coarse property $Q_{\ell,b}$, this can be tested in $O(1)$ time after preprocessing prefix sums. To test a fine property Q_ℓ, preprocess the string y as follows: for every index $j \in \{0, 1, \dots, |y|\}$ and bit $b \in \{0, 1\}$, compute $\mathsf{next}_{b,\ell}(j)$, the smallest index j' such that the substring $y_{[j,j']}$ has at least ℓ bits equal to b. For any j and b, $\mathsf{next}_{b,\ell}(j)$ can be computed by a binary search in $\log(|y|)$ time, so the preprocessing takes time $O(|y| \log |y|)$. Now property Q_t can be tested on a substring y_J in $O(w)$ time by evaluating $\mathsf{next}_{1,\ell}(\mathsf{next}_{0,\ell}(\cdots \mathsf{next}_{1,\ell}(\mathsf{next}_{0,\ell}(\mathsf{start}\, J)) \cdots))$, where there are $\varepsilon w/(5\ell)$ calls to each of $\mathsf{next}_{0,\ell}$ and $\mathsf{next}_{1,\ell}$, and checking if the result is at most $\mathsf{end}\, J$.
(5) *If x has property P_t, $|x| = w$, and y has property Q_t, then there exists an interval $I \subset [w]$ such that $\mathsf{LCS}(x_I, y) \ge \frac{|I|}{2} + \delta_{code} w$. Furthermore, given x and t, the interval I and the promised common subsequence of x_I and y can be chosen independent of y, and both can be found in time $O(w \log w)$.*

First suppose x has property $P_{\ell,b}$ and y has property $Q_{\ell,b}$ for $\ell \geq \varepsilon^2 w$ and $b \in \{0,1\}$. Then x has a substring x_I of length ℓ with at least $(\frac{1}{2}+\varepsilon^2)\ell$ b-bits and y has at least $(\frac{1+\varepsilon^2}{2})\ell$ b-bits, so $\mathsf{LCS}(x_I, y) \geq (\frac{1+\varepsilon^2}{2})\ell \geq \frac{|I|}{2} + \frac{\varepsilon^4}{2}w$, as desired. Furthermore, I can be found in linear time by a linear sweep, and the common subsequence is simply $b^{(\frac{1+\varepsilon^2}{2})\ell}$ as desired. Now suppose x has property P_ℓ for $\ell < \varepsilon^2 w$, and y has property Q_ℓ. Thus, x is ℓ-fine and y contains y_ℓ as a subsequence. By Lemma 4.7, there exists an interval I with $\mathsf{LCS}(x_I, y) \geq \frac{|I|}{2} + \varepsilon^3|x|$, as desired. Furthermore, also by Lemma 4.7, I and the common subsequence of x_I and y can be computed from x and t in time $O(w \log w)$, independent of y, as desired.

This proves Lemma 4.1. □

5 ALMOST-LINEAR TIME ALGORITHM

We now give the almost-linear time algorithm for the "equally balanced" case, which implies our main result. Specifically, we prove the following (see Section 6 for how Theorem 1.2 follows from Theorem 3.2).

Theorem (Theorem 3.2, restated). For all $\varepsilon > 0$, there exists an absolute constant $\delta = \delta(\varepsilon) > 0$ and a deterministic algorithm that, on input strings x and y with $0(x) = 1(x) \leq \min(0(y), 1(y))$, gives a $(\frac{1}{2}+\delta)$-approximation of the longest common subsequence in time $O(n^{1+\varepsilon})$.

The algorithm is given in Algorithm 2 with the covering step given in Algorithm 1. The rest of this section proves the correctness.

5.1 Parameters and Notation Conventions

Throughout x and y are the input strings satisfying $0(x) = 1(x) \leq \min(0(y), 1(y))$, and throughout $n = \max(|x|, |y|)$. Let w be the closest power of 2 to $\frac{|x|}{\log|x|}$. We may assume by deleting a negligible number of bits from x and y that $|x|$ and $|y|$ are multiples of w. Let $m_x := \frac{|x|}{w} \sim \log|x|$ and $m_y := \frac{|y|}{w} \sim \frac{|y|\log|x|}{|x|}$. Throughout, we always denote intervals for string x by the letter I, and intervals for string y by the letter J. By abuse of notation, we let intervals I (possibly with decorations) denote the substring x_I, and we let intervals J denote the substring y_J.

Let $\varepsilon > 0$ be such that $O(n^{1+\varepsilon})$ is the desired runtime. Let δ_{code} be the constant from Lemma 4.1. Let $\delta_{eq} = \delta_{eq}(\frac{\varepsilon}{2})$ be the constant from Theorem 2.2. Let $\alpha, \beta, \gamma, \delta, \theta$ be constant powers of $1/2$ that satisfy $\min(\delta_{eq}, \delta_{code}) \geq \alpha \gg \beta \gg \gamma \gg \delta = \theta$. That is, δ is sufficiently small compared to γ, which is sufficiently small compared to β, which is sufficiently small compared to α. For completeness, we note it suffices to take $\theta = \delta = \gamma^8, \gamma = \frac{1}{2}\beta^2, \beta = \alpha^2$. We did not try to optimize our constants. We give the following intuition for the above parameters.

- α lower bounds the LCS advantage gained from both algorithmic structure lemma rectangles and nearly-square rectangles.
- β is the "nearly-square" parameter: in the optimal LCS, intervals I are called *nearly-square* if they get matched to intervals of length $\leq (1+\beta)|I|$. We may assume at most β^2 fraction of intervals are nearly-square or else the nearly-square rectangles (together with "trivial rectangles") give a $(\frac{1}{2} + \text{poly}\,\beta)$-approximation by applying EqLCS to each of them.
- γ is the "imbalanced" parameter and discretization parameter for I-intervals: we may assume most γw-length intervals to be γ-balanced, or else the "Trivial rectangles" give a $(\frac{1}{2} + \text{poly}\,\gamma)$-approximation. We also round all I-intervals so that they are γw-aligned. γ is small enough so that the effect of this rounding is negligible.
- δ is the overall LCS approximation advantage: we obtain a $(\frac{1+\delta}{2})$-approximation. We assume $\mathsf{LCS}(x,y) \geq (1-\delta)|x|$, or else $\mathsf{Trivial}$ gives a $(\frac{1+\delta}{2})$-approximation.
- θ is the discretization parameter for J-intervals: we round all J-intervals so that they are θw aligned. θ is small enough so that the effect of this rounding is negligible. We take θ, the J-interval discretization, to be smaller than γ, the I-interval discretization, so that the gain from matching "Trivial rectangles" is larger than the loss due to discretization.

5.2 Runtime

We now analyze the runtime. We re-emphasize, as we did in the proof sketch, that we need to be careful about factors m_y in our runtime, but not powers of m_x: m_x is only $\log n$, but m_y is roughly $|y|/|x|$, which can be a positive power of $|y|$.

We first run a $O(|y|)$-time preprocessing of prefix-sums that allows us to query zero-counts and one-counts in any interval in either x or y in $O(1)$ time. We also preprocess string y so that we can test every property Q_t efficiently on substrings of y; for each t this takes $O(|y|\log|y|)$ preprocessing time, for a total preprocessing time that is $O(|y|\log^2|y|)$.

The runtime of $\mathsf{CoveringAlgorithm}$ is dominated by calls to $\mathsf{Trivial}$, EqLCS, $\mathsf{GetPType}_w$, $\mathsf{IsQType}_w$, and GetI_w. Note that in Line 4 and Line 7, J can be computed by a binary search over a search space of size m_y/θ, and thus can be found in $\log(m_y/\theta)$ calls to $\mathsf{Trivial}$, which each take $O(1)$ time with preprocessing. Thus, the first nested loop takes $O(m_x^2 m_y \log m_y) \leq \tilde{O}(|y|/|x|)$ time.

The second nested loop has $O(m_x m_y)$ calls to EqLCS, each of which runs in $O(|x|^{1+\frac{\varepsilon}{2}})$ time, and thus takes $O(n^{1+\varepsilon})$ time.

For the third nested loop, the number of calls to $\mathsf{GetPType}_w$ and GetI_w is m_x, and each run in time $O(w\log w)$, so the runtime is at most $\tilde{O}(|x|)$. Because the property Q_t is hereditary, we can compute J in Line 20 by binary search with $\log(m_y/\theta) \leq O(\log|y|)$ calls to $\mathsf{IsQType}_w$, which runs in time $O(w)$ (the binary search crucially saves us a factor of roughly $|y|/|x|$ in the runtime). The number of binary searches is $O(m_x m_y)$, so in total the runtime of this step is $O(m_x m_y \cdot \log(m_y) \cdot w) \leq O(|y|\log^2|y|)$.

There are $O(m_x m_y)$ rectangles, and the dynamic programming has $O(m_x m_y)$ states. The runtime of the dynamic programming is thus $O(m_x m_y) \leq \tilde{O}(|y|/|x|)$, so the total runtime is thus $O(|y|^{1+\varepsilon})$.

Algorithm 1: CoveringAlgorithm

```
Input: $x, y$ such that $1(x) = 0(x) \le \min(1(y), 0(y))$
Output: A set $\mathcal{R}$ of certified rectangles $(I \times J, \kappa)$ where $I$ is
        $\gamma w$-aligned, $J$ is $\theta w$-aligned, and $\mathsf{LCS}(I, J) \ge \kappa$.
// Trivial rectangles
$\mathcal{R} \leftarrow \{([0, |x|] \times [0, |y|], \mathsf{Trivial}([0, |x|], [0, |y|]))\}$
for all $\gamma w$-aligned intervals $I$ do
    for $j = 0, \dots, m_y/\theta$ do
        $J \leftarrow$ the smallest $\theta w$-aligned interval s.t.
          end $J = \theta w j$ and $\mathsf{Trivial}(I, J) \ge (\frac{1}{2} - \sqrt{\delta})|I|$
        if $J$ exists then
            $\mathcal{R} \leftarrow \mathcal{R} \cup (I \times J, \mathsf{Trivial}(I, J))$
        $J \leftarrow$ the smallest $\theta w$-aligned interval s.t.
          end $J = \theta w j$ and $\mathsf{Trivial}(I, J) \ge (\frac{1}{2} + \frac{\gamma}{2})|I|$
        if $J$ exists then
            $\mathcal{R} \leftarrow \mathcal{R} \cup (I \times J, \mathsf{Trivial}(I, J))$
    for $\theta w$-aligned intervals $J$ with $|J| = |I|$ do
        $\mathcal{R} \leftarrow \mathcal{R} \cup (I \times J, \mathsf{Trivial}(I, J))$
// Nearly-square rectangles
for intervals $I \in \mathcal{I}_w$ do
    for $\theta w$-aligned intervals $J$ with
      $|J| \in [(1 - \alpha)w, (1 + \alpha)w]$ do
        $\mathcal{R} \leftarrow \mathcal{R} \cup (I \times J, \mathsf{EqLCS}(I, J))$
// Algorithmic structure lemma rectangles
for $i = 1, \dots, m_x$ do
    $I \leftarrow [(i - 1)w, iw]$
    $t \leftarrow \mathsf{GetPType}_w(x_I)$
    $I' \leftarrow \mathsf{GetI}_w(x_I, t)$
    for $j = 1, \dots, m_y/\theta$ do
        $J \leftarrow$ smallest interval such that end $J = \theta w j$,
          $|J| \ge (1 + 0.9\beta)w$, and $\mathsf{IsQType}_w(y_J, t)$
        if $J$ exists then
            $\mathcal{R} \leftarrow \mathcal{R} \cup (I' \times J, \frac{|I'|}{2} + \alpha w)$
return $\mathcal{R}$.
```

5.3 Correctness Proof, High Level Overview

We need two inequalities about our output, FullLCSAlgorithm(x, y).

$$\mathsf{LCS}(x, y) \ge \mathsf{FullLCSAlgorithm}(x, y) \tag{2}$$

$$\frac{1+\delta}{2}\,\mathsf{LCS}(x, y) \le \mathsf{FullLCSAlgorithm}(x, y) \tag{3}$$

Equation (2) is the easier, which we prove at the end of this section. Equation (3) is the harder direction. We prove it in two cases, based on the following definition.

Definition 5.1. We say a pair of binary strings (x, y) is *good* if

(1) $\mathsf{LCS}(x, y) \ge (1 - \delta)|x|$,
(2) For at least $(1 - \gamma)m_x$ intervals $I \in \mathcal{I}_w$, every $I' \in \mathcal{I}_{\gamma w}(I)$ is γ-balanced, and
(3) At least $1 - \beta^2$ of $I \in \mathcal{I}_w$ satisfy $|J_I^\tau| \ge (1 + \beta)|I|$.

We call a pair *bad* if it is not good.

Algorithm 2: FullLCSAlgorithm

```
Input: $x, y$ such that $1(x) = 0(x) \le \min(1(y), 0(y))$
Output: A $(\frac{1+\delta}{2})$-approximation of LCS
$\mathcal{R} \leftarrow \mathsf{CoveringAlgorithm}(x, y)$
 // DP[$i$][$j$] lower bounds $LCS([0, \gamma w i], [0, \theta w j])$
DP $\leftarrow [0, m_x/\gamma] \times [0, m_y/\theta]$ array, initialized to 0
for $i = 1, \dots, m_x/\gamma$ do
    for $j = 1, \dots, m_y/\theta$ do
        DP[$i$][$j$] $\leftarrow \max(\mathsf{DP}[i - 1][j], \mathsf{DP}[i][j - 1])$
        for $(I \times J, \kappa) \in \mathcal{R}$ with end $I = (\gamma w)i$, end $J = (\theta w)j$,
          do
            DP[$i$][$j$] $\leftarrow$
              $\max\begin{pmatrix} \mathsf{DP}[i][j], \\ \mathsf{DP}[(\text{start } I)/(\gamma w)][(\text{start } J)/(\theta w)] + \kappa \end{pmatrix}$
return DP[$m_x/\gamma$][$m_y/\theta$].
```

Obviously, we cannot determine in almost-linear time if an input is good or bad, since that involves computing LCS(x, y). However, our analysis of the performance of FullLCSAlgorithm(x, y) differs depending on whether the input is good or bad. In Section 5.4, we prove (3) when the input is bad, and in Section 5.5, we prove (3) when the input is good.

In both the easy direction (2) and the hard direction (3), we use the following characterization of the output of the dynamic programming in Algorithm 2. Recall a collection of rectangles is called an *ordered* collection if every pair (I, J) and (I', J') is comparable (i.e. either $I < I'$ and $J < J'$ or $I > I'$ and $J > J'$).

Lemma 5.2. *The output of* FullLCSAlgorithm *is the maximum, over all ordered collections of certified rectangles* $(I_1 \times J_1, \kappa_1), \dots, (I_\ell \times J_\ell, \kappa_\ell)$*, of* $\kappa_1 + \kappa_2 + \cdots + \kappa_\ell$.

Proof. By induction, it follows that DP$[i][j]$ is the maximum, over all ordered collections of certified rectangles $(I_1 \times J_1, \kappa_1), \dots, (I_\ell \times J_\ell, \kappa_\ell)$ contained in $[0, \gamma w i] \times [0, \theta w j]$, of $\kappa_1 + \kappa_2 + \cdots + \kappa_\ell$. Here, we use that, for all rectangles $I \times J$ in $\mathcal{R}$, interval I is γw-aligned and interval J is θw-aligned. □

The next lemma asserts that certified rectangles are indeed "certified."

Lemma 5.3. *Every certified rectangle* $(I \times J, \kappa)$ *in* CoveringAlgorithm *satisfies* LCS$(I, J) \ge \kappa$.

Proof. This is true of all rectangles certified by Trivial and EqLCS by definition. The algorithmic structure lemma rectangles $(I' \times J, \kappa)$ for $\kappa = \frac{|I'|}{2} + \alpha w$ satisfy LCS$(I', J) \ge \kappa$ by Lemma 4.1. □

The easy direction (2) follows from Lemma 5.2 and Lemma 5.3.

Corollary 5.4. FullLCSAlgorithm$(x, y) \le$ LCS(x, y)

Proof. By Lemma 5.2, the output of FullLCSAlgorithm(x, y) equals $\kappa_1 + \cdots + \kappa_\ell$ for some ordered collection of certified rectangles

$(I_1 \times J_1, \kappa_1), \ldots, (I_\ell \times J_\ell, \kappa_\ell)$. Then, by Lemma 5.3, we have

$$\text{FullLCSAlgorithm}(x,y) = \sum_{i=1}^{\ell} \kappa_i \le \sum_{i=1}^{\ell} \text{LCS}(I_i, J_i) \le \text{LCS}(x,y),$$

as desired. □

5.4 Proof of (3) for Bad Inputs

We show that (3) holds in the bad case by conditioning on which case of Definition 5.1 is violated.

Subcase 1: Trivial. In the first subcase, we suppose $\text{LCS}(x,y) \le (1-\delta)|x|$.

Lemma 5.5. *If* $\text{LCS}(x,y) \le (1-\delta)|x|$, *then* (3) *holds.*

Proof. We always have $\text{Trivial}([0,|x|],[0,|y|]) \ge \frac{|x|}{2}$ as $\frac{|x|}{2} = 1(x) = 0(x) \le \min(1(y), 0(y))$. Hence, we have

$$\text{FullLCSAlgorithm}(x,y) \ge \frac{|x|}{2} \ge \frac{1+\delta}{2}\,\text{LCS}(x,y).$$

□

Subcase 2: Locally imbalanced. In the next subcase, we assume $\text{LCS}(x,y) \ge (1-\delta)|x|$ and that a nontrivial fraction of intervals are imbalanced. Since x and y have such a long LCS, we know that most intervals in x appear nearly unmodified in y:

Lemma 5.6. *If w' is a positive integer that divides $|x|$ and* $\text{LCS}(x,y) \ge (1-\delta)|x|$, *then at most $\sqrt{\delta}\frac{|x|}{w'}$ intervals $I_i \in \mathcal{I}_{w'}$ satisfy* $\text{LCS}(I_i, J^\tau_{I_i}) > (1-\sqrt{\delta})|I_i|$.

Proof. To obtain the longest common subsequence of x and y, one applies at most δm deletions. By counting, at most $\sqrt{\delta}\frac{|x|}{w'}$ intervals of $\mathcal{I}_{w'}$ receive more than $\sqrt{\delta}w'$ deletions, and the remaining intervals satisfy the desired inequality. □

We now can establish (3) in this case.

Lemma 5.7. *If at least γm_x many γw-intervals are γ-imbalanced, and* $\text{LCS}(x,y) \ge (1-\delta)|x|$, *then* (3) *holds.*

Proof. Let $I_1 < \cdots < I_{m_x/\gamma}$ be the intervals of $\mathcal{I}_{\gamma w}$. For all $i = 1, \ldots, m_x/\gamma$, let $J_i := \text{round}_{\theta w}(J^\tau_{I_i})$, so that J_i are pairwise disjoint. Let K_{imbal} be the indices i such that I_i is γ-imbalanced. By assumption $|K_{imbal}| \ge \gamma m_x$. Let K_{good} be the indices i such that $\text{LCS}(I_i, J^\tau_{I_i}) \ge (1-\sqrt{\delta})|I_i|$. By Lemma 5.6 with $w' = \gamma w$, $|K_{good}| \ge (1-\sqrt{\delta})\frac{m_x}{\gamma}$.

Observe that under these assumptions, CoveringAlgorithm certifies many rectangles using the trivial algorithm. For $i \in K_{good}$, we have $\text{Trivial}(I_i, J_i) \ge \frac{1}{2}(1-\sqrt{\delta})|I_i| - 2\theta w \ge (\frac{1}{2}-\sqrt{\delta})|I_i|$. Thus, we certify $(I_i \times J'_i, (\frac{1}{2}-\sqrt{\delta})|I_i|)$ for some subinterval $J'_i \subset J_i$, defined as the shortest θw-aligned interval with $\text{end}\, J'_i = \text{end}\, J_i$ and $\text{Trivial}(I_i, J'_i) \ge \frac{1}{2}(1-\sqrt{\delta})|I_i|$.

For $i \in K_{good} \cap K_{imbal}$, we have $\text{Trivial}(I_i, J_i) \ge \text{Trivial}(I_i, J^\tau_{I_i}) - 2\theta w \ge (\frac{1}{2}+\gamma-\sqrt{\delta})|I_i| - 2\theta w > (\frac{1}{2}+\frac{\gamma}{2})\gamma w$. Thus, we certify $(I_i \times J'_i, (\frac{1}{2}+\frac{\gamma}{2})|I_i|)$ for some subinterval $J'_i \subset J_i$, defined as the shortest θw-aligned interval with $\text{end}\, J'_i = \text{end}\, J_i$ and $\text{Trivial}(I_i, J'_i) \ge (\frac{1}{2}+\frac{\gamma}{2})|I_i|$.

Thus CoveringAlgorithm obtains an ordered collection of certified rectangles $(I_i \times J'_i, \kappa_i)$ over $i \in K_{good}$ with $\kappa_i \ge (\frac{1}{2}-\sqrt{\delta})|I_i|$ for all $i \in K_{good}$ and $\kappa_i \ge (\frac{1}{2}-\sqrt{\delta}+\frac{\gamma}{2})|I_i|$ for all $i \in K_{good} \cap K_{imbal}$. Thus,

$$\begin{aligned}
&\text{FullLCSAlgorithm}(x,y)\\
&\quad\ge \sum_{i \in K_{good}} \kappa_i\\
&\quad\ge \left(\frac{1}{2}-\sqrt{\delta}\right)(\gamma w)\cdot|K_{good}| + \frac{\gamma}{2}(\gamma w)\cdot|K_{good}\cap K_{imbal}|\\
&\quad\ge \left(\frac{1+\delta}{2}\right)\text{LCS}(x,y),
\end{aligned}$$

as desired. In the last inequality, we used (i) $|K_{good}| \ge (1-\sqrt{\delta})\frac{m_x}{\gamma}$, (ii) $|K_{good} \cap K_{imbal}| \ge \gamma m_x - \frac{\sqrt{\delta}m_x}{\gamma}$, (iii) $\gamma \gg \delta$, and (iv) $m_x w = |x| \ge \text{LCS}(x,y)$. □

Subcase 3: Many nearly-square intervals. This case applies when the equal-length input algorithm [30] correctly certifies many rectangles. Recall that an interval $I \in \mathcal{I}_w$ is *nearly-square* if $|J^\tau_I| \le (1+\beta)|I|$. For convenience, we call I *oblong* if it is not nearly-square.

Lemma 5.8. *If at least $\beta^2 m_x$ intervals $I \in \mathcal{I}_w$ are nearly-square, then* (3) *holds.*

Proof. Let $I_1 < I_2 < \cdots < I_{m_x}$ be the intervals of $\mathcal{I}_w$. For all $i = 1, \ldots, m_x$, let $J_i := \text{round}_{\theta w}(J^\tau_{I_i})$, so that the J_i are pairwise disjoint. Let K_{short} be the set of indices i such that I_i is nearly-square. By assumption, $|K_{short}| \ge \beta^2 m_x$. Let K_{good} be the set of indices i such that $\text{LCS}(I_i, J^\tau_{I_i}) \ge (1-\sqrt{\delta})|I_i|$. By Lemma 5.6, $|K_{good}| \ge (1-\sqrt{\delta})m_x$.

Just as in the proof of Lemma 5.8, we track the rectangles certified by CoveringAlgorithm. For $i \in K_{good}$, we have $\text{Trivial}(I_i, J_i) \ge (\frac{1}{2}-\sqrt{\delta})|I_i|$, so we certify $(I_i \times J'_i, (\frac{1}{2}-\sqrt{\delta})|I_i|)$ for some subinterval $J'_i \subset J_i$. For $i \in K_{good} \cap K_{short}$, we have $(1+\alpha)w \ge (1+\beta)w \ge |J^\tau_{I_i}| \ge |J_i|$ since I_i is nearly-square, and $|J_i| \ge \text{LCS}(I_i, J_i) \ge \text{LCS}(I_i, J^\tau_{I_i}) - 2\theta w \ge (1-\sqrt{\delta}-2\theta)w \ge (1-\alpha)w$. Hence, $\text{EqLCS}(I_i, J_i)$ is called at Line 14 and has value at least $(\frac{1}{2}+\alpha)\,\text{LCS}(I_i, J_i) \ge (\frac{1}{2}+\alpha)(1-\sqrt{\delta}-2\theta)w > \frac{1+\alpha}{2}w$ by Theorem 2.2.

We thus have an ordered collection of certified rectangles $(I_i \times J'_i, \kappa_i)$ over $i \in K_{good}$ with $\kappa_i \ge (\frac{1}{2}-\sqrt{\delta})w$ for all $i \in K_{good}$ and $\kappa_i \ge (\frac{1}{2}-\sqrt{\delta}+\frac{\alpha}{2})w$ for $i \in K_{good} \cap K_{short}$. Thus,

$$\begin{aligned}
&\text{FullLCSAlgorithm}(x,y)\\
&\quad\ge \sum_{i \in K_{good}} \kappa_i\\
&\quad\ge \left(\frac{1}{2}-\sqrt{\delta}\right)w\cdot|K_{good}| + \frac{\alpha}{2}w\cdot|K_{good}\cap K_{short}|\\
&\quad\ge \left(\frac{1+\delta}{2}\right)\text{LCS}(x,y),
\end{aligned}$$

as desired. In the last inequality, we used (i) $|K_{good}| \ge (1-\sqrt{\delta})m_x$, (ii) $|K_{good} \cap K_{short}| \ge \beta^2 m_x - \sqrt{\delta}m_x$, (iii) $\alpha \gg \beta \gg \delta$, and (iv) $m_x w = |x| \ge \text{LCS}(x,y)$. □

Wrapping up the bad case. We now can prove the following lemma.

Lemma 5.9. *If (x, y) is bad, then (3) holds.*

Proof. If (x, y) is bad, then either item 1, 2, or 3 of Definition 5.1 is violated. If 1 is violated, (3) holds by Lemma 5.5. If 2 is violated, there are at least γm_x intervals I with a γ-imbalanced γw-subinterval, so there are at least γm_x many γ-imbalanced γw-intervals, so by Lemma 5.7, (3) holds. If 3 is violated, (3) holds by Lemma 5.8. □

5.5 Proof of (3) for Good Inputs

Let

$$m'_x := (1 - 2\beta^2) m_x.$$

The following lemma establishes the natural structural property for good inputs.

Lemma 5.10. *If (x, y) is good, then there exist an ordered sequence of rectangles $I_1 \times J_1 < \cdots < I_{m'_x} \times J_{m'_x}$ such that for all i, (i) $I_i \in \mathcal{I}_w$, (ii) every γw-subinterval of I_i is γ-balanced, (iii) $|J_i| \geq (1 + 0.9\beta)w$, and (iv) $\mathsf{IsQType}_w(y_{J_i}, \mathsf{GetPType}_w(x_{I_i}))$ returns true.*

Proof. Among the m_x intervals $I \in \mathcal{I}_w$, all but at most $\sqrt{\delta} m_x$ intervals satisfy $\mathsf{LCS}(I, J_I^\tau) > (1 - \sqrt{\delta})|I|$ by Lemma 5.6, at most γm_x have a γ-imbalanced γw-subinterval since (x, y) is good, and at most $\beta^2 m_x$ are nearly-square since (x, y) is good. Thus, at least $(1 - 2\beta^2) m_x = m'_x$ intervals are (i) satisfying $\mathsf{LCS}(I, J_I^\tau) > (1 - \sqrt{\delta})|I|$, (ii) γ-balanced in all γw-subintervals, and (iii) oblong. Let $I_1 < \cdots < I_{m'_x}$ be m'_x of these intervals. Let $J_i := \mathrm{round}_{\theta w}(J_{I_i}^\tau)$, so these J_i are pairwise disjoint. For all such i, we have

$$|J_i| \geq |J_{I_i}^\tau| - 2\theta w \geq (1 + \beta - 2\theta)w \geq (1 + 0.9\beta)w$$

and

$$\mathsf{LCS}(I_i, J_i) \geq \mathsf{LCS}(I_i, J_{I_i}^\tau) - 2\theta w \geq (1 - \sqrt{\delta} - 2\theta)w \geq (1 - \alpha)w.$$

For all t such that x_{I_i} has property P_t, we have y_{J_i} has property Q_t by Lemma 4.1 (Item 2). Thus, $\mathsf{IsQType}_w(y_{J_i}, t)$ returns true for $t = \mathsf{GetPType}_w(x_{I_i})$. We have found our ordered sequence of rectangles $I_1 \times J_1 < \cdots < I_{m'_x} \times J_{m'_x}$. □

We now can prove the main result for this section.

Lemma 5.11. *If (x, y) is good, then (3) holds.*

Proof. Let $I_1 \times J_1 < I_2 \times J_2 < \cdots < I_{m'_x} \times J_{m'_x}$ be the ordered sequence of rectangles given by Lemma 5.10. By construction, for all $i = 1, \ldots, m'_x$, we have (i) $I_i \in \mathcal{I}_w$, (ii) every γw-subinterval of I_i is γ-balanced, (iii) $|J_i| \geq (1 + 0.9\beta)w$, and (iv) $\mathsf{IsQType}_w(y_{J_i}, \mathsf{GetPType}_w(x_{I_i})) = \text{true}$.

As a result, at loop iteration $i = (\mathrm{end}\, I_i)/w$ and $j = (\mathrm{end}\, J_i)/(\theta w)$ of Line 20, the interval J exists (the interval $J = J_i$ satisfies the requirement, so a minimal J exists). Thus, CoveringAlgorithm certifies a rectangle $(I'_i \times J'_i, \frac{|I'_i|}{2} + \alpha w)$ where I'_i, the output of GetI_w, is a γw-aligned subinterval of I_i and J'_i is a subinterval of J_i with length at least $(1 + 0.9\beta)w$.

We would like to build an ordered collection of certified rectangles containing these $(I'_i \times J'_i, \frac{|I'_i|}{2} + \alpha w)$, which embed more than half of each small interval I'_i into y. However, for each of these rectangles, J'_i is typically much longer than I'_i, so using many of them is extremely wasteful of bits in y. To reduce this issue, we let $t := 3/\beta$ and build an ordered collection using only every t-th rectangle from the preceding family.

Let m''_x be the largest multiple of t less than m'_x. For each i that is a multiple of t, partition I_i into $\tilde{I}_i^L < \tilde{I}_i^M < \tilde{I}_i^R$ where $\tilde{I}_i^M := I'_i$. For i not a multiple of t, let $\tilde{I}_i := I_i$. For i a multiple of t, let $\tilde{J}_i^M := J'_i$.

For i a multiple of t, we claim there exist θw-aligned intervals $\tilde{J}_{i-t}^R < \tilde{J}_{i-t+1} < \tilde{J}_{i-t+2} < \cdots < \tilde{J}_{i-1} < \tilde{J}_i^L$ such that

- $\tilde{J}_i^L < \tilde{J}_i^M$.
- $|\tilde{J}_i^L| = |\tilde{I}_i^L|$.
- $|\tilde{J}_{i-\ell}| = |\tilde{I}_{i-\ell}|$ for $\ell = 1, \ldots, t-1$.
- $|\tilde{J}_{i-t}^R| = |\tilde{I}_{i-t}^R|$ (we take $\tilde{I}_0^R = \emptyset$).
- $\tilde{J}_{i-t}^M < \tilde{J}_{i-t}^R$ (this is vacuously true if $i = t$)

To see that such intervals exist, note that the interval

$$[\mathrm{end}\, \tilde{J}_{i-t}^M, \mathrm{start}\, \tilde{J}_i^M] = [\mathrm{end}\, J'_{i-t}, \mathrm{start}\, J'_i]$$

contains all intervals $J'_{i-\ell}$ for $\ell = 1, \ldots, t-1$. Since each $J'_{i-\ell}$ has length at least $(1 + 0.9\beta)w$, we have

$$\begin{aligned}\mathrm{start}\, J_i^M - \mathrm{end}\, J_{i-t}^M &\geq (t-1)\,(1 + 0.9\beta)w \\ &> (t+1)\,w \\ &\geq |\tilde{I}_i^L| + \sum_{\ell=1}^{t-1} |\tilde{I}_{i-\ell}| + |\tilde{I}_{i-t}^R|.\end{aligned}$$

The last inequality holds as each term on the right is at most w. Thus we can construct the intervals greedily in order

$$\tilde{J}_i^L, \tilde{J}_{i-1}, \tilde{J}_{i-2}, \ldots, \tilde{J}_{i-t+1}, \tilde{J}_{i-t}^R$$

by setting $\mathrm{end}\, \tilde{J}_i^L = \mathrm{start}\, \tilde{J}_i^M$, and then $\mathrm{end}\, \tilde{J}_{i-1} = \mathrm{start}\, \tilde{J}_i^L$, and so on. They will be θw-aligned as all of the $\tilde{I}$ intervals have lengths a multiple of γw, and thus a multiple of θw.

By construction of these intervals, CoveringAlgorithm certifies the following rectangles for $i \leq m''_x$:

$$\begin{array}{lll} (\tilde{I}_i^M \times \tilde{J}_i^M, \kappa_i^M) & \text{where } \kappa_i^M := \dfrac{|\tilde{I}_i^M|}{2} + \alpha w & \text{if } t \mid i \\ (\tilde{I}_i^L \times \tilde{J}_i^L, \kappa_i^L) & \text{where } \kappa_i^L := \mathsf{Trivial}(\tilde{I}_i^L, \tilde{J}_i^L) & \text{if } t \mid i \\ (\tilde{I}_i^R \times \tilde{J}_i^R, \kappa_i^R) & \text{where } \kappa_i^R := \mathsf{Trivial}(\tilde{I}_i^R, \tilde{J}_i^R) & \text{if } t \mid i \\ (\tilde{I}_i \times \tilde{J}_i, \kappa_i) & \text{where } \kappa_i := \mathsf{Trivial}(\tilde{I}_i, \tilde{J}_i) & \text{if } t \nmid i \end{array} \tag{4}$$

The first collection of rectangles comes from the definition of $\tilde{I}_i^M := I'_i$ and $\tilde{J}_i^M := J'_i$. The rest of the rectangles come from the fact that we certify all γw-aligned squares with the trivial algorithm in CoveringAlgorithm Line 11. Furthermore, the rectangles are increasing in i, with additionally $\tilde{I}_i^L \times \tilde{J}_i^L < \tilde{I}_i^M \times \tilde{J}_i^M < \tilde{I}_i^R \times \tilde{J}_i^R$ for i a multiple of t. Hence, the rectangles in (4) form an ordered collection of rectangles. By Lemma 2.1, we also have $\kappa_i^L \geq (\frac{1}{2} - \gamma)|\tilde{I}_i^L|, \kappa_i^R \geq (\frac{1}{2} - \gamma)|\tilde{I}_i^R|$ for i a multiple of t and $\kappa_i \geq (\frac{1}{2} - \gamma)|\tilde{I}_i|$ for all other i, because the intervals $\tilde{I}_i^R, \tilde{I}_i^L, \tilde{I}_i$ are all γw-aligned and thus γ-balanced. Thus, by Lemma 5.2, the output of FullLCSAlgorithm(x, y) is at least

$$\begin{aligned}
&\sum_{\substack{i\le m_x''\\ t\mid i}}\left(\kappa_i^L+\kappa_i^M+\kappa_i^R\right)+\sum_{\substack{i\le m_x''\\ t\nmid i}}\kappa_i\\
&\ge \sum_{\substack{i\le m_x''\\ t\mid i}}\left(\left(\frac{1}{2}-\gamma\right)(|\tilde{I}_i^L|+|\tilde{I}_i^M|+|\tilde{I}_i^R|)+\alpha w\right)+\sum_{\substack{i\le m_x''\\ t\nmid i}}\left(\frac{1}{2}-\gamma\right)|\tilde{I}_i|\\
&\ge \left(\frac{1}{2}-\gamma\right)w\cdot m_x''+\alpha w\cdot\frac{\beta}{3}m_x''\\
&\ge \left(\frac{1+\delta}{2}\right)\mathsf{LCS}(x,y)
\end{aligned}$$

In the third inequality, we used that $m_x''w \ge (m_x'-t)w \ge (1-3\beta^2)m_xw$ and $m_xw=|x|\ge \mathsf{LCS}(x,y)$. □

We can now finish the proof of Theorem 3.2.

Proof of Theorem 3.2. We have now proved that (2) and (3) always hold, and that FullLCSAlgorithm runs in time $O(n^{1+\varepsilon})$, so FullLCSAlgorithm gives a $(\frac{1+\delta}{2})$-approximation of the LCS of two binary strings with $0(x)=1(x)\le\min(0(y),1(y))$ in time $O(n^{1+\varepsilon})$, as desired. □

6 PUTTING IT ALL TOGETHER

In this final section we use standard techniques to finish the proof of Theorem 1.2 given the balanced case Theorem 3.2. This proved in [30] for equal length strings and in [7] for unequal length strings (see also [6]).

Lemma 6.1 (Lemma 13 of [7], see also Lemma 3.5 of [6]). *For every $\rho>0$, there exists $\delta=\delta(\rho)>0$ such that the following holds. There exists an algorithm which, given binary strings x,y with $|x|\le|y|$ and $0(x)=1(y)\le(\frac{1}{2}-\rho)|x|$, computes a $(\frac{1}{2}+\delta)$-approximation of $\mathsf{LCS}(x,y)$ in deterministic linear time.* [2]

Lemma 6.2. *For all $\varepsilon>0$, there exists an absolute constant $\delta=\delta(\varepsilon)>0$ and a deterministic algorithm that, given two strings x and y with $|x|\le|y|$ and $\min(1(x),1(y))=\min(0(x),0(y))$, outputs a $(\frac{1}{2}+\delta)$-approximation of $\mathsf{LCS}(x,y)$ in time $O(|y|^{1+\varepsilon})$.*

Proof. Let $\delta_1=\delta_1(\varepsilon)>0$ be the absolute constant in Theorem 3.2. Let $\delta_2>0$ be the absolute constant in Lemma 6.1 with parameter $\rho=\delta_1/10$. Let $\delta=\min(\delta_1/2,\delta_2)$. As $|x|\le|y|$, we have three cases, and we find a $(\frac{1}{2}+\delta)$-approximation to $\mathsf{LCS}(x,y)$ in each.

Case 1. $0(x)=\min(0(x),0(y)),1(x)=\min(1(x),1(y))$. Then $0(x)=1(x)=|x|/2$ and $\mathsf{LCS}(x,y)\ge|x|/2$. The algorithm in Theorem 3.2, gives a $(\frac{1}{2}+\delta_1)$-approximation of the LCS.

Case 2. $0(y)=\min(0(x),0(y)),1(x)=\min(1(x),1(y))$. We have $1(x)=0(y)\le 0(x)$. There are two subcases.

Subcase 2a. $1(x)\ge(\frac{1}{2}-\rho)|x|$**.** In this case, delete $0(x)-1(x)\le\rho|x|$ zeros from x arbitrarily to get a balanced subsequence x'. Then $\mathsf{LCS}(x',y)\ge\mathsf{LCS}(x,y)-\rho|x|\ge(1-\rho)\,\mathsf{LCS}(x,y)$. Thus, the algorithm in Theorem 3.2 gives a common subsequence of length $(\frac{1}{2}+\delta_1)(1-\rho)\,\mathsf{LCS}(x,y)\ge(\frac{1}{2}+\delta)\,\mathsf{LCS}(x,y)$.

Subcase 2b. $1(x)\le(\frac{1}{2}-\rho)|x|$**.** In this case, Lemma 6.1 with parameter ρ finds a common subsequence of length at least $(\frac{1}{2}+\delta)\,\mathsf{LCS}(x,y)$.

Case 3. $0(x)=\min(0(x),0(y)),1(y)=\min(1(x),1(y))$. Symmetric to case 2. □

Theorem (Theorem 1.2, restated)**.** For all $\varepsilon>0$, there exists an absolute constant $\delta=\delta(\varepsilon)>0$ and a deterministic algorithm that, given two binary strings x and y of not-necessarily-equal length, outputs a $(\frac{1}{2}+\delta)$-approximation of the longest common subsequence in time $O(n^{1+\varepsilon})$ where $n=\max(|x|,|y|)$.

Proof. Let δ_0 be the constant in Lemma 6.2. Let $\delta=\delta_0/5$. Let the input strings be x and y and assume without loss of generality $|x|\le|y|$ and that $\min(0(x),0(y))\ge\min(1(x),1(y))$. We have $\mathsf{Trivial}(x,y)=\min(0(x),0(y))$ and $\mathsf{LCS}(x,y)\le\min(0(x),0(y))+\min(1(x),1(y))$.

If $\min(0(x),0(y))\ge(1+\delta_0)\min(1(x),1(y))$, then, as $\frac{1+\delta_0}{2+\delta_0}>\frac{1}{2}+\delta$, the trivial algorithm gives a $(\frac{1}{2}+\delta)$-approximation of the LCS. Thus we may assume $\min(0(x),0(y))\le(1+\delta_0)\min(1(x),1(y))$. Delete $\min(0(x),0(y))-\min(1(x),1(y))\le\delta_0\min(1(x),1(y))$ zeros from each of x and y arbitrarily to obtain x' and y' with

$$\min(0(x'),0(y'))=\min(1(x'),1(y')).$$

We have

$$\mathsf{LCS}(x',y')\ge\mathsf{LCS}(x,y)-\delta_0\min(1(x),1(y))\ge(1-\delta_0)\,\mathsf{LCS}(x,y).$$

Running the algorithm in Lemma 6.2 gives an approximation to $\mathsf{LCS}(x',y')$ that is at least $(\frac{1}{2}+\delta_0)(1-\delta_0)\,\mathsf{LCS}(x,y)>(\frac{1}{2}+\delta)\,\mathsf{LCS}(x,y)$, as desired. □

7 CONCLUSION AND OPEN QUESTIONS

We close with some related open questions.

- What is the best possible approximation factor of binary LCS in almost-linear or truly subquadratic time? We give a $\frac{1}{2}+\delta$ in almost-linear time. We made no attempt to optimize δ, and currently it depends on the runtime exponent $1+\varepsilon$.
- Related to the above, can we prove fine-grained hardness of approximation results for LCS? It is known that a deterministic $2^{-(\log n)^{1-\delta}}$ approximation in $n^{2-\varepsilon}$ time for LCS over alphabet $n^{o(1)}$ would imply new circuit lower bounds, as would a deterministic $1-\frac{1}{\operatorname{poly}\log n}$-approximation for binary inputs [1, 4, 20].
- We studied the algorithmic question of computing LCS, where, as the previous two questions highlight, the optimal approximation factor is open. We showed this algorithmic question is closed related to an analogous combinatorial question, which is also open: What is the largest constant $\alpha\in(0,1)$

[2]There are several minor differences between this statement and the statement in [7]. First, the statement in [7] says subquadratic time but it actually runs in linear time, similar to the analogous algorithm in [30] who proved Lemma 6.1 for equal-length strings. This was confirmed in private communication with the authors.

Second, [7] prove the statement when $0(x)$ and $1(y)$ are within $\varepsilon|x|$ of each other for some $|x|$, while we only consider when they are equal.

such that in any set $C \subset \{0,1\}^n$ of $|C| \geq 2^{\Omega(n)}$ binary strings, there are always two strings x, y with $\mathsf{LCS}(x,y) \geq \alpha n$? The optimal α is known to be in $[\frac{1}{2} + 10^{-40}, 2 - \sqrt{2}]$ [18, 23], and $1 - \alpha$ quantifies the maximum fraction of adversarial deletions that can be tolerated by a (asymptotically) positive rate code.

It would also be interesting to understand how strong is the connection between the deletion codes question and the algorithmic LCS question. At first blush, it seems that techniques derived solely from analysis of deletion codes should not give an α-approximation for $\alpha > 2 - \sqrt{2} \approx 0.59$ (because of the deletion codes construction [18]), so beating this ratio would show some separation between the two questions.

- How does the optimal subquadratic time or almost-linear time approximation factor grow with the alphabet size? Over alphabet size q, we show that we can beat (barely) the trivial $\frac{1}{q}$-approximation. We know that we can always get a randomized $\frac{1}{n^{o(1)}}$-approximation in linear time [11, 28], which is much better than $\frac{1}{q}$ for large alphabets.
- There is a natural question that arises from another possible approach to proving Theorem 1.2. Define the *directed edit distance* of two strings x, y to be the number of edits needed to get from x to y, where insertions cost 0 and deletions (and substitutions) cost 1. Equivalently, $\vec{\Delta}_{\text{edit}}(x,y) := |x| - \mathsf{LCS}(x,y)$. When the strings are equal length, the directed edit distance is simply half the edit distance. A constant-factor approximation of directed edit distance in almost-linear time would immediately imply Theorem 3.2 and thus Theorem 1.2. This suggest the following question, which may be of independent interest.

Question 7.1. Is there an almost-linear time constant-factor approximation of the directed edit distance?

We note that $\vec{\Delta}_{\text{edit}}(x,y)$ is *not* a metric. Indeed, it is not even symmetric[3], and it does not satisfy the triangle inequality. Thus, the existing edit distance approximation algorithms [10, 15, 19, 25], which rely heavily on the triangle inequality, do not seem to immediately apply to directed edit distance. On the other hand, directed edit distance does satisfy a "directed triangle inequality": for strings x, y, z, we have $\vec{\Delta}_{\text{edit}}(x,z) \leq \vec{\Delta}_{\text{edit}}(x,y) + \vec{\Delta}_{\text{edit}}(y,z)$. This gives some hope that fast approximation algorithms exist.

ACKNOWLEDGMENTS

We thank Saeed Seddighin for introducing us to the question of approximating binary LCS and for suggesting its potential connection to the deletion codes bound [23]. We thank Negev Shekel Nosatzki for helpful discussions about edit-distance algorithms. We thank Shyan Akmal and Virginia Vassilevska-Williams for helpful discussions on their work [7]. We thank Aviad Rubinstein for helpful feedback. We thank Venkatesan Guruswami for helpful feedback and encouragement.

[3] $\vec{\Delta}_{\text{edit}}(0011, 00) = 2$ but $\vec{\Delta}_{\text{edit}}(00, 0011) = 0$

A PROOF OF LEMMA 4.4

Lemma 4.4 is essentially a corollary of the stronger combinatorial structure lemma [23, Lemma 4.1], except that the constant dependences are superior and we make the additional assumption that the lengths involved are all powers of two. For completeness, we include a proof here which is significantly simpler than the proof of [23, Lemma 4.1].

Lemma (Lemma 4.4, restated). *For $\varepsilon = 10^{-5}$ and w sufficiently large, at least one of the following two conditions holds for every string $x \in \{0,1\}^w$.*

(1) *There exists $\ell \in [\varepsilon^2 w, w]$ equal to a power of two and an 0.1-imbalanced interval I in x of length ℓ.*
(2) *There exists $\ell \in [1, \varepsilon^2 w)$ equal to a power of two such that the number of ℓ^+-flags in x is at least εw, and x contains $(0^\ell 1^\ell)^{\varepsilon w/\ell}$ as a subsequence.*

Proof. We first reduce to the case that w is a power of two. Indeed, suppose we show the statement for all lengths w' equal to sufficiently large powers of 2, with a stronger $\varepsilon' = 10^{-4}$ in place of ε. Then, let w' be the largest power of two at most w, and let $x' = x_{[w']}$ be the prefix of x of length $w' > w/2$. Applying our assumption to x', the lemma statement holds for x' with stronger $\varepsilon' = 10^{-4}$. If x' falls into the first case of the lemma, then x' contains a 0.1-imbalanced interval I of length $\ell \in [(\varepsilon')^2 w', w'] \subseteq [\varepsilon^2 w, w]$, so x must fall into the first case as well.

Otherwise, there exists $\ell \in [1, (\varepsilon')^2 w')$ equal to a power of two such that the number of ℓ^+-flags in x' is at least $(\varepsilon')^2 w' \geq \varepsilon w$, and x' contains $(0^\ell 1^\ell)^{(\varepsilon')^2 w'/\ell} \supseteq (0^\ell 1^\ell)^{\varepsilon^2 w/\ell}$. If $\ell \geq \varepsilon^2 w$, then the existence of an ℓ^+-flag implies that there is a 0.1-imbalanced interval of length at least ℓ in x', so x again falls into the first case of the lemma. On the other hand, if $\ell < \varepsilon^2 w$ then x falls into the second case of the lemma, as desired.

Thus, we assume w is a power of two and prove this special case with the stronger constant $\varepsilon = 10^{-4}$. Let $w = 2^K$, and for any $0 \leq k \leq K$ and $1 \leq i \leq 2^{K-k}$, define $I_{k,i} := [(i-1)\cdot 2^k + 1, i \cdot 2^k]$ to be an aligned dyadic interval of length 2^k. Observe that for each k, the intervals $I_{k,i}$ form a partition of $[w]$. If $I_{k,i}$ is 0.1-imbalanced for some k satisfying $2^k \geq \varepsilon^2 w$, case 1 holds and we are done. Thus, we may assume $I_{k,i}$ is 0.1-balanced whenever $2^k \geq \varepsilon^2 w$. We would like to show that case 2 above always holds.

Call an interval I is *sparse* if $d(x_I) < 0.01$, and *dense* otherwise. Let $\mathcal{S}_k$ denote the collection of all maximal sparse dyadic intervals $I_{k,i}$ of length 2^k, i.e. all sparse dyadic intervals $I_{k,i}$ that are not proper subintervals of other sparse $I_{k',i'}$. Let $\mathcal{S} = \bigcup_{k=0}^K \mathcal{S}_k$, so that $\mathcal{S}$ is the collection of all maximal sparse dyadic intervals in x, and the elements of $\mathcal{S}$ are pairwise disjoint.

Observe that sparse intervals are certainly 0.1-imbalanced, so by our previous assumption, $\mathcal{S}_k$ is empty if $2^k \geq \varepsilon w$. On the other hand, we also assumed that $I_{K,1} = [w]$ is 0.1-balanced, so the number of zeros in x is at least $0.4w$. Every zero-bit in x constitutes a sparse dyadic interval $I_{0,i}$ of length 1 by itself, and every sparse dyadic interval lies inside some element of $\mathcal{S}$. Thus, intervals in $\mathcal{S}$ cover all zero-bits in x and have total length at least $0.4w$.

Let if $I = I_{k,i}$ and $i > 1$, define the *predecessor* of I to be $\mathrm{pred}(I) := I_{k,i-1}$.

Claim. *If $k \geq 0$, $1 < i \leq 2^{K-k}$, $I_{k,i} \in \mathcal{S}$, $t = 2^{\max(0,k-5)}$, and $\mathrm{pred}(I_{k,i})$ is dense, then the number of t-flags in $\mathrm{pred}(I_{k,i})$ is at least $0.01 \cdot |\mathrm{pred}(I_{k,i})|$.*

Proof. If $k < 5$ then the assumption that $I_{k,i}$ is sparse implies that it contains only zeros, so the first one-bit in $\mathrm{pred}(I_{k,i})$ is a 1-flag, and this is sufficient. Assume now that $k \geq 5$. Observe that since $I_{k,i}$ is sparse, it contains at least $0.99 \cdot 2^k > 2^{k-1} > 10(t-1)$ zeros and at most $0.01 \cdot 2^k < 2^{k-6} = t/2$ ones. In particular, the last $t/2 = 2^{k-6}$ ones in $x_{\mathrm{pred}(I_{k,i})}$ (or all of them if there are fewer than 2^{k-6}) must all be t-flags. As $\mathrm{pred}(I_{k,i})$ is dense, we are done. □

Thus, dense predecessors of elements of $\mathcal{S}$ contain many flags. In order to make sure these flags are not double-counted, we first pass to a subcollection of $\mathcal{S}$, defined as follows. Write if $I, J \in \mathcal{S}$, write $I \prec J$ if both $\mathrm{pred}(I)$ and $\mathrm{pred}(J)$ exist, and $\mathrm{pred}(I) \subset \mathrm{pred}(J)$. Define $\mathcal{S}'$ to be the subcollection of $\mathcal{S}$ obtained by removing the (at most one) element of the form $I_{k,0}$ without a predecessor, and then removing all elements non-maximal with respect to $\prec$. Observe that if two dyadic intervals satisfy $I \prec J$, then $I \subseteq \mathrm{pred}(J)$, so for any dyadic interval J, the total length of all elements I of $\mathcal{S}$ satisfying $I \prec J$ is at most $|J|$. By passing to $\mathcal{S}'$, we deleted at most half of the total length in $\mathcal{S}$, plus possibly one interval with no predecessor, which has length at most $\varepsilon^2 w$. Thus,

$$\sum_{I \in \mathcal{S}'} |I| \geq \frac{1}{2} \sum_{I \in \mathcal{S}} |I| - \varepsilon^2 w \geq 0.1w.$$

Writing $\mathcal{S}'_{\geq k}$ for the collection of all intervals in $\mathcal{S}'$ with length at least 2^k, we pick k_0 to be the largest $0 \leq k_0 \leq K$ for which

$$\sum_{I \in \mathcal{S}'_{\geq k_0}} |I| \geq 0.01w.$$

Our choice of ℓ is $\ell := 2^{\max(0,k_0-5)}$. Note that $\ell < \varepsilon^2 w$ because $\mathcal{S}_k$ is empty when $2^k \geq \varepsilon^2 w$. We separately prove each of the two required hypotheses.

Claim. *For $\ell = 2^{\max(0,k_0-5)}$, the number of ℓ^+-flags in x is at least εw.*

Proof. For any two dyadic intervals I, J, either $I \prec J$ or $I \cap J = \emptyset$. Thus, $\{\mathrm{pred}(I) | I \in \mathcal{S}'_{\geq k}\}$ is a collection of pairwise-disjoint intervals with total length at least $0.01w$. By the previous claim, the number of ℓ^+-flags in one of these intervals $\mathrm{pred}(I)$ is at least $0.01|\mathrm{pred}(I)| = 0.01|I|$, and so in total the number of ℓ^+-flags in x is at least $10^{-4}w$, as desired. □

It remains to check that x contains $(0^\ell 1^\ell)^{\varepsilon w/\ell}$.

Claim. *For $\ell = 2^{\max(0,k_0-5)}$, x contains $(0^\ell 1^\ell)^{\varepsilon w/\ell}$ as a subsequence.*

Proof. Let $k = k_0+1$. By the maximality of k_0, we have $\sum_{I \in \mathcal{S}'_{\geq k}} |I| < 0.01w$. Let $\mathcal{S}_{\geq k}$ denote the collection of maximal sparse dyadic intervals of length at least 2^k. Reversing the analysis which led to a lower bound on the total length of $\mathcal{S}'$, we obtain

$$\sum_{I \in \mathcal{S}_{\geq k}} |I| \leq 2 \sum_{I \in \mathcal{S}'_{\geq k}} |I| + \varepsilon w \leq 0.1w.$$

Since all sparse dyadic intervals of length 2^k lie inside some element of $\mathcal{S}_{\geq k}$, we see that in total at most $0.1 \cdot 2^{K-k}$ of the dyadic intervals $I_{k,i}$ are sparse.

On the other hand, at most $0.7 \cdot 2^{K-k}$ of them have density greater than 0.99, since otherwise these very dense intervals alone account for at least $0.68w$ ones, making the entire interval $[w]$ 0.1-imbalanced, which is a contradiction. In sum, out of 2^{K-k} total intervals $I_{k,i}$, at most $0.1 \cdot 2^{K-k}$ have density less than 0.01, and at most $0.7 \cdot 2^{K-k}$ have density greater than 0.99, leaving at least $0.2 \cdot 2^{K-k}$ that must each contain $0.01 \cdot 2^k$ zeros and $0.01 \cdot 2^k$ ones. Passing to only these subintervals, we conclude that x contains a subsequence of the form $x' = x_1 x_2 \cdots x_{0.2 \cdot 2^{K-k}}$ where each x_i contains $0.01 \cdot 2^k$ zeros and $0.01 \cdot 2^k$ ones. A string of the form $(1^\ell 0^\ell)^a$ can be found as a subsequence of x' by taking ones from the first $\lceil 100\ell/2^k \rceil$ x_i's, then zeros from the next $\lceil 100\ell/2^k \rceil$, and so on. Since $\ell \geq 2^{k-6}$, we can pick

$$a \geq \frac{0.2 \cdot 2^{K-k}}{2\lceil 100\ell/2^k \rceil} \geq 10^{-4} w/\ell,$$

as desired. □

Combining the above two claims proves that if case 1 of the lemma does not hold, then case 2 does for $\ell = 2^{\max(0,k_0-5)}$. □

REFERENCES

[1] Amir Abboud and Arturs Backurs. 2017. Towards Hardness of Approximation for Polynomial Time Problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA (LIPIcs, Vol. 67)*, Christos H. Papadimitriou (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 11:1–11:26. https://doi.org/10.4230/LIPIcs.ITCS.2017.11

[2] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. 2015. Tight Hardness Results for LCS and Other Sequence Similarity Measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, Venkatesan Guruswami (Ed.). IEEE Computer Society, 59–78. https://doi.org/10.1109/FOCS.2015.14

[3] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. 2016. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, Daniel Wichs and Yishay Mansour (Eds.). ACM, 375–388. https://doi.org/10.1145/2897518.2897653

[4] Amir Abboud and Aviad Rubinstein. 2018. Fast and Deterministic Constant Factor Approximation Algorithms for LCS Imply New Circuit Lower Bounds. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA (LIPIcs, Vol. 94)*, Anna R. Karlin (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 35:1–35:14. https://doi.org/10.4230/LIPIcs.ITCS.2018.35

[5] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. 2014. Consequences of Faster Alignment of Sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 8572)*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.). Springer, 39–51. https://doi.org/10.1007/978-3-662-43948-7_4

[6] Shyan Akmal. 2021. *Longest Common Subsequence Over Constant-Sized Alphabets: Beating the Naive Approximation Ratio.* Master's thesis. Massachusetts Institute of Technology.

[7] Shyan Akmal and Virginia Vassilevska Williams. 2021. Improved Approximation for Longest Common Subsequence over Small Alphabets. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference) (LIPIcs, Vol. 198)*, Nikhil Bansal, Emanuela Merelli, and James Worrell (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:18. https://doi.org/10.4230/LIPIcs.ICALP.2021.13

[8] Alexandr Andoni. 2018. Simpler constant-factor approximation to edit distance problems. (2018). http://www.cs.columbia.edu/~andoni/papers/edit/

[9] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. 2010. Polylogarithmic Approximation for Edit Distance and the Asymmetric Query Complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010,*

October 23-26, 2010, Las Vegas, Nevada, USA. IEEE Computer Society, 377–386. https://doi.org/10.1109/FOCS.2010.43

[10] Alexandr Andoni and Negev Shekel Nosatzki. 2020. Edit Distance in Near-Linear Time: it's a Constant Factor. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, Sandy Irani (Ed.). IEEE, 990–1001. https://doi.org/10.1109/FOCS46700.2020.00096

[11] Alexandr Andoni, Negev Shekel Nosatzki, Sandip Sinha, and Clifford Stein. 2022. Estimating the Longest Increasing Subsequence in Nearly Optimal Time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 708–719. https://doi.org/10.1109/FOCS54457.2022.00073

[12] Alexandr Andoni and Krzysztof Onak. 2012. Approximating Edit Distance in Near-Linear Time. *SIAM J. Comput.* 41, 6 (2012), 1635–1648. https://doi.org/10.1137/090767182

[13] Arturs Backurs and Piotr Indyk. 2015. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM, 51–58. https://doi.org/10.1145/2746539.2746612

[14] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. 2004. Approximating Edit Distance Efficiently. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*. IEEE Computer Society, 550–559. https://doi.org/10.1109/FOCS.2004.14

[15] Joshua Brakensiek and Aviad Rubinstein. 2020. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy (Eds.). ACM, 685–698. https://doi.org/10.1145/3357713.3384282

[16] Karl Bringmann and Debarati Das. 2021. A Linear-Time $n^{0.4}$-Approximation for Longest Common Subsequence. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference) (LIPIcs, Vol. 198)*, Nikhil Bansal, Emanuela Merelli, and James Worrell (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 39:1–39:20. https://doi.org/10.4230/LIPIcs.ICALP.2021.39

[17] Karl Bringmann and Marvin Künnemann. 2015. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, Venkatesan Guruswami (Ed.). IEEE Computer Society, 79–97. https://doi.org/10.1109/FOCS.2015.15

[18] Boris Bukh, Venkatesan Guruswami, and Johan Håstad. 2017. An Improved Bound on the Fraction of Correctable Deletions. *IEEE Trans. Inf. Theory* 63, 1 (2017), 93–103. https://doi.org/10.1109/TIT.2016.2621044

[19] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. 2020. Approximating Edit Distance Within Constant Factor in Truly Sub-quadratic Time. *J. ACM* 67, 6 (2020), 36:1–36:22. https://doi.org/10.1145/3422823

[20] Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N. Rothblum, and Aviad Rubinstein. 2019. Fine-grained Complexity Meets IP = PSPACE. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 1–20. https://doi.org/10.1137/1.9781611975482.1

[21] Vašek Chvátal, David A Klarner, and Donald Ervin Knuth. 1972. *Selected combinatorial research problems*. Computer Science Department, Stanford University.

[22] Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. 2020. Does preprocessing help in fast sequence comparisons?. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy (Eds.). ACM, 657–670. https://doi.org/10.1145/3357713.3384300

[23] Venkatesan Guruswami, Xiaoyu He, and Ray Li. 2021. The zero-rate threshold for adversarial bit-deletions is less than 1/2. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 727–738. https://doi.org/10.1109/FOCS52979.2021.00076

[24] MohammadTaghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. 2019. Approximating LCS in Linear Time: Beating the $\sqrt{n}$ Barrier. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 1181–1200. https://doi.org/10.1137/1.9781611975482.72

[25] Michal Koucký and Michael E. Saks. 2020. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy (Eds.). ACM, 699–712. https://doi.org/10.1145/3357713.3384307

[26] Vladmir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Dokl. (English Translation)* 10, 8 (1966), 707–710.

[27] William J. Masek and Mike Paterson. 1980. A Faster Algorithm Computing String Edit Distances. *J. Comput. Syst. Sci.* 20, 1 (1980), 18–31. https://doi.org/10.1016/0022-0000(80)90002-1

[28] Negev Shekel Nosatzki. 2021. Approximating the Longest Common Subsequence problem within a sub-polynomial factor in linear time. *CoRR* abs/2112.08454 (2021). arXiv:2112.08454 https://arxiv.org/abs/2112.08454

[29] Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. 2019. Approximation Algorithms for LCS and LIS with Truly Improved Running Times. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, David Zuckerman (Ed.). IEEE Computer Society, 1121–1145. https://doi.org/10.1109/FOCS.2019.00071

[30] Aviad Rubinstein and Zhao Song. 2020. Reducing approximate Longest Common Subsequence to approximate Edit Distance. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, Shuchi Chawla (Ed.). SIAM, 1591–1600. https://doi.org/10.1137/1.9781611975994.98

[31] Jeffrey D. Ullman. 1967. On the capabilities of codes to correct synchronization errors. *IEEE Transactions on Information Theory* 13, 1 (1967), 95–105.

Received 2022-11-07; accepted 2023-02-06