Approximating the Longest Common Subsequence problem within a sub-polynomial factor in linear time

Negev Shekel Nosatzki
Columbia University
ns3049@columbia.edu

Abstract

The Longest Common Subsequence (LCS) of two strings is a fundamental string similarity measure with a classical dynamic programming solution taking quadratic time. Despite significant efforts, little progress was made in improving the runtime. Even in the realm of approximation, not much was known for linear time algorithms beyond the trivial $\sqrt{n}$-approximation. Recent breakthrough result provided a $n^{0.497}$-factor approximation algorithm [HSSS19], which was more recently improved to a $n^{0.4}$-factor one[BCD21]. The latter paper also showed a $n^{2-2.5\alpha}$ time algorithm which outputs a $n^\alpha$ approximation to the LCS, but so far no sub-polynomial approximation is known in truly subquadratic time.

In this work, we show an algorithm which runs in $O(n)$ time, and outputs a $n^{o(1)}$-factor approximation to LCS($x, y$), with high probability, for any pair of length $n$ input strings.

Our entire algorithm is merely an efficient black-box reduction to the Block-LIS problem, introduced very recently in [ANSS21], and solving the Block-LIS problem directly.
1 Introduction

The Longest Common Subsequence problem (LCS) is a classical string alignment problem. Formally, given two strings \( x, y \) of length \( n \) over alphabet \( \Sigma \), the longest common subsequence between \( x \) and \( y \) is the largest string which appears as a subsequence in both \( x \) and \( y \). The (length of the) LCS has a classical dynamic programming solution in quadratic time [WF74].

LCS is also a fundamental string similarity measure of key importance both in theory and practice, and therefore has been extensively studied during the course of the past 50 years [UAH76, Hir77, HS77, AG87, EGGI92, SW07, RSSS19, RS20], with the main goal of improving the run-time from polynomial towards close(r) to linear.

Despite significant attention, little progress was made, and the runtime has improved by logarithmic factors only [MP80]. More recently, fine-grained hardness results helped explaining the lack of progress, showing that any truly subquadratic solution for LCS would refute the Strong Exponential Time Hypothesis (SETH) [ABW15, BK15]. Moreover, for the LCS problem in particular, it was later shown that even a high polylogarithmic speedup would be surprising [AHWW16, AB18].

Even before the above hardness results, researchers started considering faster algorithms that approximate the LCS, but there as well the progress has been slow. The trivial \( \sqrt{n} \)-factor approximation was only recently improved to \( \approx n^{0.497} \) [HSSS19] and even more recently to \( n^{0.4} \) [BCD21]. In parallel, a parametrized version was developed which gives a subquadratic solution with improved approximation when the longest common subsequence is promised to be above a certain threshold [RSSS19].

In contrast, the progress for the corresponding Edit Distance problem has been more “rosy”, with a series of advances [BJKK04, BES06, AKO10, CDG+18], recently culminating in a near-linear time, constant factor approximation algorithm [AN20].

1.1 Main result

Our main result is a sub-polynomial factor approximation algorithm for LCS, which runs in linear time.

**Theorem 1.** LCS\(_n\) can be approximated up to \( n^{o(1)} \)-factor, with high probability, and in \( O(n) \) time.

While we do not track the exact approximation factor in our analysis, we note it is \( \exp\left(\frac{\log(n)}{\log\log(n)}\right) \) for some absolute constant \( c \). Interesting enough, we could not find a way to significantly improve the bound above, obtainable in linear time, even if one can afford \( n^{1.99} \) runtime. This relates to the issue of balancing of dense and sparse cases described in [ANSS21], which is the core component of our algorithm.

To improve the bound, one would need to find a more efficient way to solve the Block-LIS algorithm from [ANSS21].

One of the implications of this paper is that an affirmative resolution to the open question posed in [ANSS21] would yield surprising results, and in particular, a near-linear time, \( (1 + \epsilon) \) approximation to LCS, which also provides near-linear \( \epsilon n \) additive approximation to the edit distance.

2 Preliminaries

2.1 Definitions

**Sequences.** Fix alphabet \( \Sigma \). We denote \( x \in \Sigma^n \) as a length \( n \) sequence.

For a sequence \( x \), we denote by \( \psi(x) : \Sigma \rightarrow \mathbb{N} \), the count vector of letters in \( x \), i.e., \( \psi(x)_c \triangleq |x^{-1}(c)| \) for any \( c \in \Sigma \).

We also define integer block sequence \( z \in \mathbb{N}^{n \times k} \), and denote \( |z| \) as the count of all integers in all the blocks (with repetitions).
Monotone sets. We define monotone sets as follows:

**Definition 2.** We say that a set \( P \subseteq \mathbb{N} \times \mathbb{N} \) is monotone iff for all \( ((i, k), (j, l)) \in P \times P \), we have (i) \( i = j \Leftrightarrow l = k \); and (ii) \( i < j \Leftrightarrow k < l \).

One can observe that \( \text{LCS}(x, y) = \max_{P \subseteq \mathbb{N} \times \mathbb{N}} \sum_{(i,j) \in P} 1[x_i = y_j] \)

**The Block-LIS problem.** The Block-LIS problem was defined in [ANSS21]\(^1\). In Block-LIS, the main input consists of \( n \) blocks of (at most) \( k \) elements each, and each block can contribute at most one of its elements to a subsequence. Formally, for an integer block sequence \( z \in \mathbb{N}^{n \times k} \), Block-LIS\((z)\) is the length of a maximal sub-sequence \( \text{OPT} \triangleq \{(w_1, z_w)\}_{w \in [n] \times [k]} \) such that \( \text{OPT} \) is monotone.

**Other notations.** Fix vectors \( a, b \in \mathbb{R}^n \), then \( \min(a, b) \) (respectively \( \max(a, b) \)) is defined, as the vector of minimums (respectively, maximums) for each \((a_i, b_i)\) pair. Also, the notation \( \oplus \) means the direct sum.

### 2.2 Main technique - Block-LIS reduction

Our construction boils down to a formulation of the LCS problem as a (larger) LIS problem, and solve the LIS problem directly. Our reduction at its core is reminiscent to ideas introduced in [HS77, AG87], with two important differences:

1. We provide a simplified approach by direct “black-box” reduction to a LIS problem which is then solved directly. In particular, we reduce to the Block-LIS problem. As a result, the entire construction and proof takes roughly 1.5 pages.

2. We show that the number of “matching pairs” is always bounded by \( 2n \cdot \text{LCS}(x, y) \) using norm inequalities, implying the linear runtime from Lemma 4.3 in [ANSS21].

In particular, we restate the following Definition and Theorem:

**Definition 3.** [ANSS21] For \( \alpha \geq 1 \) and \( \beta > 0 \), an \((\alpha, \beta)\)-approximation \( \hat{q} \) of a quantity \( q \) is an \( \alpha \)-multiplicative and \( \beta \) additive estimation of \( q \), i.e., \( \hat{q} \in [q/\alpha - \beta, q] \).

**Theorem 4.** [ANSS21] Fix \( n, k \in \mathbb{N} \) and \( z \in \mathbb{N}^{n \times k} \). For any \( \lambda = o(1) \), there exists a randomized non-adaptive algorithm \( A_{BL} \) that solves Block-LIS up to \((\alpha, \lambda n)\)-approximation, where \( \alpha = \left( \frac{|z|}{\lambda n} \right)^{o(1)} \) using \( \left( \frac{|z|}{\lambda n} \right) \cdot n^{o(1)} \) time.

**Block-LIS Formulation for LCS.** To prove Theorem 1, we formulate the LCS problem as a single string Block-LIS. Given \( x, y \in \Sigma^n \), we define an integer block sequence \( z \) such that each \( z_i \) is a block of all \( j \in [n] \) indices where \( y_j = x_i \). Formally:

\[
    z \triangleq \oplus_{i \in [n]} \{ y^{-1}(x_i) \}.
\]

We claim the following equivalence:

**Claim 5.** Block-LIS\((z)\) = LCS\((x, y)\).

\(^1\)The original definition of Block-LIS included another extension which is not relevant for our reduction, and hence was omitted for exposition.
Proof. First, observe that any common subsequence of \( x \) and \( y \) has a monotone set \( P \subseteq [n] \times [n] \) where \( x_i = y_j \) for all \((i, j) \in P\). Now, since \( j \in y^{-1}(x_i) \), then the second coordinates of \( P \) are also an increasing subsequence of \( z \).

Similarly, consider the coordinates of an increasing subsequence \( W \subseteq [n] \times [k] \). Then \( \{(w_1, z_w)\}_{w \in W} \) is a monotone set where \( x_{w_1} = y_{z_w} \), and hence \( x \) and \( y \) has a common subsequence of the same length. \( \square \)

Our entire algorithm is merely an efficient implementation of the above reduction, and solving it using Theorem 4. In particular, we show the following:

Lemma 6. Suppose there exists an algorithm \( A_{BL} \) which \( \left(f \left( \frac{|x|}{n} \right), \lambda n \right) \) approximates Block-LIS in time \( t \left( \frac{|x|}{n} \right) \). Then, there exists an algorithm \( A_{LCS} \) approximating LCS\(_n\) up to a factor \( (1+o(1)) \cdot f(n \cdot \log n) \) in time \( O(n \cdot \log(n)) \).

Proof of Theorem 1 using Theorem 4 and lemma 6. The proof follow almost immediately from Theorem 4 and lemma 6. However, a direct invocation of the statements only yields \( n^{1+o(1)} \) time algorithm. To obtain linear time, we first subsample \( U \subseteq [n] \) i.i.d. at rate \( n^{-o(1)} \), and then apply the statements on \( x_U \) and \( y_U \), noting LCS\((x_U, y_U) \in [n^{-o(1)}], 1 \cdot \text{LCS}(x, y) \). The resulting runtime is \( |U|^{1+o(1)} = O(n) \) as needed. \( \square \)

2.3 Useful lower bound for the LCS using H"older’s inequality

H"older’s inequality. A classical norm inequality which says that \( \langle x, y \rangle \leq \|x\|_p \cdot \|y\|_q \) for any \( p+q = 1 \). In particular, \( \langle x, y \rangle \leq \|x\|_\infty \cdot \|y\|_1 \). Such inequality can be extended to show the following bound:

Claim 7. Fix \( x, y \in \mathbb{R}^n \). Then, \( \langle x, y \rangle \leq \|\min\{x, y\}\|_\infty \cdot (\|x\|_1 + \|y\|_1) \).

Proof. Let \( a = \min\{x, y\} \) and \( b = \max\{x, y\} \). Then, by H"older’s inequality,

\[
\langle x, y \rangle = \langle a, b \rangle \leq \|a\|_\infty \cdot \|b\|_1 = \|\min\{x, y\}\|_\infty \cdot \|\max\{x, y\}\|_1 \leq \|\min\{x, y\}\|_\infty \cdot (\|x\|_1 + \|y\|_1).
\]

\( \square \)

Simple lower bound for the LCS. We next state a useful fact about the minimal LCS of two strings:

Fact 8. LCS\((x, y) \geq \|\min\{\psi(x), \psi(y)\}\|_\infty \).

Proof. That fact is immediate, since if each of \( x \) and \( y \) contain \( k \) copies of the same letter \( c \), then \( c^k \) is a common subsequence to both \( x \) and \( y \). \( \square \)

We combine the above to show the following lower bound on the LCS:

Lemma 9. LCS\((x, y) \geq \frac{|z|}{2n} \).

Proof. Observe that by construction \( |z| = \langle \psi(x), \psi(y) \rangle \). Now, invoking Claim 7 and fact 8, we obtain:

\[
|z| \leq \|\min\{\psi(x), \psi(y)\}\|_\infty \cdot (\|\psi(x)\|_1 + \|\psi(y)\|_1) \leq 2n \cdot \text{LCS}(x, y),
\]

which implies the Lemma. \( \square \)

One direct implication of the above is a simple deterministic and exact algorithm for LCS in time \( O(n \ell \log(\ell)) \), where \( \ell = \text{LCS}(x, y) \). While we do not derive it in this paper, one can observe this can be done by simply reducing the LCS problem to Block-LIS, and solving it directly as a standard LIS problem, which takes \( O(|z| \log(\text{LIS}(z))) \) time.
3 Reduction Algorithm

In this section we prove Lemma 6. Namely, we are given an algorithm \( A_{BL} \) for solving Block-LIS\((z)\) with additive error parameter \( \lambda \), and we use it to estimate LCS\((x, y)\).

The algorithm is the straight-forward one:

\[
\begin{align*}
\text{ESTIMATELCS}(x, y): & \\
1. & \text{Store the function } y^{-1} \text{ over } \Sigma \text{ (say, in the sparse matrix representation).} \\
2. & \text{Set } d \leftarrow |z|/2n, \text{ where } z \text{ is defined as in Equation (1).} \\
3. & \text{Run } A_{BL} \text{ with input } z \text{ and } \lambda \leftarrow d/n \log n. \\
4. & \text{Output } \hat{\ell} \leftarrow \text{the maximum between } d \text{ and the output of } A_{BL}.
\end{align*}
\]

Figure 1: Description of the algorithm for estimating the LCS between \( x \) and \( y \)

3.1 Analysis

We show the above algorithm provides the guarantees of Lemma 6.

\textbf{Proof of Lemma 6.} First, let us argue \( \hat{\ell} \in \left[ \frac{1-o(1)}{f(n \log n)}, 1 \right] \cdot LCS(x, y) \).

\textbf{Upper Bound.} The upper-bound is immediate since the output of \( A_{BL} \) is promised to be at most Block-LIS\((z) = LCS(x, y)\), and since \( d \leq LCS(x, y) \) from Lemma 9.

\textbf{Lower Bound.} Let \( \hat{\ell}_A \) be the output of \( A_{BL} \). Then, we are guaranteed

\[
\hat{\ell}_A \geq \frac{\text{Block-LIS}(z)}{f(|z|/\lambda)} - \lambda n = \frac{\text{LCS}(x,y)}{f(n \log n)} - \frac{2d}{\lambda} \log n.
\]

Therefore, \( \hat{\ell} = \max\{\hat{\ell}_A, d\} \geq (1 - o(1)) \cdot \frac{\text{LCS}(x,y)}{f(n \log n)} \) as needed.

Last, we show the runtime is at most \( t \left( O(n \cdot \log(n)), n \right) + O(n) \).

\textbf{Runtime Analysis.} The runtime for the first 2 steps is clearly \( O(n) \). It remains to show the runtime on the Block-LIS algorithm, which is promised to be at most

\[
t \left( \frac{|z|}{\lambda n}, n \right) = t \left( \frac{2d}{\lambda}, n \right) = t \left( O(n \cdot \log(n)), n \right).
\]

This concludes the proof.

4 Acknowledgements

The author would like to thank Alexandr Andoni for useful discussion and comments on earlier drafts of this note.
References

[AB18] Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. *arXiv preprint arXiv:1804.08978*, 2018. 1

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78, 2015. 1

[AG87] Alberto Apostolico and Concettina Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2(1):315–336, 1987. 1, 2

[AHWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 375–388. ACM, 2016. 1

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2010. Full version at http://arxiv.org/abs/1005.4033. 1

[AN20] Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it’s a constant factor. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2020. 1

[ANSS21] Alexandr Andoni, Negev Shekel Nosatzki, Sandip Sinha, and Clifford Stein. Estimating the longest increasing subsequence in nearly optimal time, 2021. 1

[BCD21] Karl Bringmann, Vincent Cohen-Addad, and Debarati Das. A linear-time $n^{0.4}$-approximation for longest common subsequence. *CoRR*, abs/2106.08195, 2021. 1

[BES06] Tuğkan Batu, Funda Ergün, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 792–801, 2006. 1

[BJKK04] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 550–559, 2004. 1

[BK15] Karl Bringmann and Marvin Künemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 79–97. IEEE, 2015. 1

[CDG+18] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucky, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 979–990. IEEE, 2018. 1

[EGGI92] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming i: Linear cost functions. *J. ACM*, 39(3):519–545, jul 1992. 1
[Hir77] Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *J. ACM*, 24(4):664–675, oct 1977.

[HS77] James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest common subsequences. *Commun. ACM*, 20(5):350–353, may 1977.

[HSSS19] Mohammad Taghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. Approximating LCS in Linear Time: Beating the $\sqrt{n}$ Barrier. 2019. (document)

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

[RS20] Aviad Rubinstein and Zhao Song. Reducing approximate longest common subsequence to approximate edit distance. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1591–1600. SIAM, 2020.

[RSSS19] Aviad Rubinstein, Saeed Seddighin, Zhao Song, and Xiaorui Sun. Approximation algorithms for lcs and lis with truly improved running times. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1121–1145. IEEE, 2019.

[SW07] Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA ’07, page 336–345, USA, 2007. Society for Industrial and Applied Mathematics.

[UAH76] JD Ullman, AV Aho, and DS Hirschberg. Bounds on the complexity of the longest common subsequence problem. *Journal of the ACM (JACM)*, 23(1):1–12, 1976.

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