Tight Conditional Lower Bounds for Longest Common Increasing Subsequence*

Lech Duraj[†1], **Marvin Künnemann**[2], **and Adam Polak**[‡3]

1 **Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland**
duraj@tcs.uj.edu.pl

2 **Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany**
marvin@mpi-inf.mpg.de

3 **Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland**
polak@tcs.uj.edu.pl

Abstract

We consider the canonical generalization of the well-studied Longest Increasing Subsequence problem to multiple sequences, called k-LCIS: Given k integer sequences $X_1, \ldots, X_k$ of length at most n, the task is to determine the length of the longest common subsequence of $X_1, \ldots, X_k$ that is also strictly increasing. Especially for the case of $k = 2$ (called LCIS for short), several algorithms have been proposed that require quadratic time in the worst case.

Assuming the Strong Exponential Time Hypothesis (SETH), we prove a tight lower bound, specifically, that no algorithm solves LCIS in (strongly) subquadratic time. Interestingly, the proof makes no use of normalization tricks common to hardness proofs for similar problems such as LCS. We further strengthen this lower bound to rule out $\mathcal{O}\left((nL)^{1-\varepsilon}\right)$ time algorithms for LCIS, where L denotes the solution size, and to rule out $\mathcal{O}\left(n^{k-\varepsilon}\right)$ time algorithms for k-LCIS. We obtain the same conditional lower bounds for the related Longest Common Weakly Increasing Subsequence problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases fine-grained complexity, combinatorial pattern matching, sequence alignments, parameterized complexity, SETH

Digital Object Identifier 10.4230/LIPIcs.IPEC.2017.15

1 Introduction

The longest common subsequence problem (LCS) and its variants are computational primitives with a variety of applications, which includes, e.g., uses as similarity measures for spelling correction [36, 42] or DNA sequence comparison [38, 5], as well as determining the differences of text files as in the UNIX DIFF utility [27]. LCS shares characteristics of both an easy and a hard problem: (*Easy*) A simple and elegant dynamic-programming algorithm computes an LCS of two length-n sequences in time $\mathcal{O}\left(n^2\right)$ [42], and in many practical settings, certain properties of typical input sequences can be exploited to obtain faster, "tailored" solutions

* The full version of this paper is available at: http://arxiv.org/abs/1709.10075.
† Partially supported by Polish National Science Center grant 2016/21/B/ST6/02165.
‡ Partially supported by Polish Ministry of Science and Higher Education program *Diamentowy Grant*.

(e.g., [26, 28, 7, 37]; see also [13] for a survey). (*Hard*) At the same time, no polynomial improvements over the classical solution are known, thus exact computation may become infeasible for very long general input sequences. The research community has sought for a resolution of the question *"Do subquadratic algorithms for LCS exist?"* already shortly after the formalization of the problem [20, 4].

Recently, an answer conditional on the Strong Exponential Time Hypothesis (SETH; see Section 2 for a definition) could be obtained: Based on a line of research relating the satisfiability problem to quadratic-time problems [43, 40, 14, 3] and following a breakthrough result for Edit Distance [9], it has been shown that unless SETH fails, there is no (strongly) subquadratic-time algorithm for LCS [1, 15]. Subsequent work [2] strengthens these lower bounds to hold already under weaker assumptions and even provides surprising consequences of sufficiently strong polylogarithmic improvements.

Due to its popularity and wide range of applications, several variants of LCS have been proposed. This includes the heaviest common subsequence (HCS) [31], which introduces weights to the problem, as well as notions that constrain the structure of the solution, such as the longest common increasing subsequence (LCIS) [45], LCSk [12], constrained LCS [41, 19, 8], restricted LCS [25], and many other variants (see, e.g., [18, 6, 32]). Most of these variants are (at least loosely) motivated by biological sequence comparison tasks. To the best of our knowledge, in the above list, LCIS is the only LCS variant for which (1) the best known algorithms run in quadratic time in the worst case and (2) its definition does not include LCS as a special case (for such generalizations of LCS, the quadratic-time SETH hardness of LCS [1, 15] would transfer immediately). As such, it is open to determine whether there are (strongly) subquadratic algorithms for LCIS or whether such algorithms can be ruled out under SETH. The starting point of our work is to settle this question.

1.1 Longest Common Increasing Subsequence (LCIS)

The Longest Common Increasing Subsequence problem on k sequences (k-LCIS) is defined as follows: Given integer sequences $X_1, \ldots, X_k$ of length at most n, determine the length of the longest sequence Z such that Z is a strictly increasing sequence of integers and Z is a subsequence of each $X_i, i \in \{1, \ldots, k\}$. For $k = 1$, we obtain the well-studied longest increasing subsequence problem (LIS; we refer to [21] for an overview), which has an $\mathcal{O}(n \log n)$ time solution and a matching lower bound in the decision tree model [24]. The extension to $k = 2$, denoted simply as LCIS, has been proposed by Yang, Huang, and Chao [45], partially motivated as a generalization of LIS and by potential applications in bioinformatics. They obtained an $\mathcal{O}(n^2)$ time algorithm, leaving open the natural question whether there exists a way to extend the near-linear time solution for LIS to a near-linear time solution for multiple sequences.

Interestingly, already a classic connection between LCS and LIS combined with a recent conditional lower bound of Abboud, Backurs and Vassilevska Williams [1] yields a partial negative answer assuming SETH.

▶ **Observation 1** (Folklore reduction, implicit in [28], explicit in [31]). *After $\mathcal{O}(kn^2)$ time preprocessing, we can solve k-LCS by a single call to $(k-1)$-LCIS on sequences of length at most n^2.*

Note that by the above reduction, an $\mathcal{O}\left(n^{\frac{3}{2}-\varepsilon}\right)$ time LCIS algorithm would give an $\mathcal{O}\left(n^{3-2\varepsilon}\right)$ time algorithm for 3-LCS, which would refute SETH by a result of Abboud et al. [1].

▶ **Corollary 2.** *Unless SETH fails, there is no $\mathcal{O}\left(n^{\frac{3}{2}-\varepsilon}\right)$ time algorithm for LCIS for any constant $\varepsilon > 0$.*

While this rules out near-linear time algorithms, still an unsatisfying large polynomial gap between best upper and conditional lower bounds persists.

1.2 Our Results

Our first result is a tight SETH-based lower bound for LCIS.

▶ **Theorem 3.** *Unless SETH fails, there is no* $\mathcal{O}\left(n^{2-\varepsilon}\right)$ *time algorithm for LCIS for any constant* $\varepsilon > 0$.

We extend our main result in several directions.

1.2.1 Parameterized Complexity I: Solution Size

Subsequent work [17, 34] improved over Yang et al.'s algorithm when certain input parameters are small. Here, we focus particularly on the solution size, i.e., the length L of the LCIS. Kutz et al. [34] provided an algorithm running in time $\mathcal{O}\left(nL \log\log n + n \log n\right)$. If L is small compared to its worst-case upper bound of n, say $L = n^{\frac{1}{2}\pm o(1)}$, this algorithm runs in strongly subquadratic time. Interestingly, exactly for this case, the reduction from 3-LCS to LCIS of Observation 1 already yields a matching SETH-based lower bound of $(Ln)^{1-o(1)} = n^{\frac{3}{2}-o(1)}$. However, for smaller L, this reduction yields no lower bound at all and only a non-matching lower bound for larger L. We remedy this situation by the following result.[1]

▶ **Theorem 4.** *Unless SETH fails, there is no* $\mathcal{O}\left((nL)^{1-\varepsilon}\right)$ *time algorithm for LCIS for any constant* $\varepsilon > 0$. *This even holds restricted to instances with* $L = n^{\gamma\pm o(1)}$, *for* arbitrarily chosen $0 < \gamma \leqslant 1$.

1.2.2 Parameterized Complexity II: k-LCIS

For constant $k \geqslant 3$, we can solve k-LCIS in $\mathcal{O}\left(n^k \text{polylog}(n)\right)$ time [17, 34], or even $\mathcal{O}\left(n^k\right)$ time (see the appendix in the full version). While it is known that k-LCS cannot be computed in time $\mathcal{O}\left(n^{k-\varepsilon}\right)$ for any constant $\varepsilon > 0, k \geqslant 2$ unless SETH fails [1], this does not directly transfer to k-LCIS, since the reduction in Observation 1 is not tight. However, by extending our main construction, we can prove the analogous result.

▶ **Theorem 5.** *Unless SETH fails, there is no* $\mathcal{O}\left(n^{k-\varepsilon}\right)$ *time algorithm for* k*-LCIS for any constant* $k \geqslant 3$ *and* $\varepsilon > 0$.

1.2.3 Longest Common Weakly Increasing Subsequence (LCWIS)

We consider a closely related variant of LCIS called the Longest Common Weakly Increasing Subsequence (k-LCWIS): Here, given integer sequences $X_1, \ldots, X_k$ of length at most n, the task is to determine the longest *weakly increasing* (i.e. non-decreasing) integer sequence Z that is a common subsequence of $X_1, \ldots, X_k$. Again, we write LCWIS as a shorthand for 2-LCWIS. Note that the seemingly small change in the notion of increasing sequence has a major impact on algorithmic and hardness results: Any instance of LCIS in which the input sequences are defined over a small-sized alphabet $\Sigma \subseteq \mathbb{Z}$, say $|\Sigma| = \mathcal{O}\left(n^{1/2}\right)$, can be solved in strongly subquadratic time $\mathcal{O}\left(nL \log n\right) = \mathcal{O}\left(n^{3/2} \log n\right)$ [34], by using the fact that $L \leqslant |\Sigma|$. In contrast, LCWIS is quadratic-time SETH hard already over slightly

[1] We mention in passing that a systematic study of the complexity of LCS in terms of such input parameters has been performed recently in [16].

superlogarithmic-sized alphabets [39]. We give a substantially different proof for this fact and generalize it to k-LCWIS.

▶ **Theorem 6.** *Unless SETH fails, there is no $\mathcal{O}\left(n^{k-\varepsilon}\right)$ time algorithm for k-LCWIS for any constant $k \geqslant 3$ and $\varepsilon > 0$. This even holds restricted to instances defined over an alphabet of size $|\Sigma| \leqslant f(n) \log n$ for any function $f(n) = \omega(1)$ growing arbitrarily slowly.*

1.3 Discussion, Outline and Technical Contributions

Apart from an interest in LCIS and its close connection to LCS, our work is also motivated by an interest in the *optimality of dynamic programming (DP) algorithms*[2]. Notably, many conditional lower bounds in P target problems with natural DP algorithms that are proven to be near-optimal under some plausible assumption (see, e.g., [14, 3, 9, 10, 1, 15, 11, 22, 33] and [44] for an introduction to the field). Even if we restrict our attention to problems that find optimal sequence alignments under some restrictions, such as LCS, Edit Distance and LCIS, the currently known hardness proofs differ significantly, despite seemingly small differences between the problem definitions. Ideally, we would like to classify the properties of a DP formulation which allow for matching conditional lower bounds.

One step in this direction is given by the *alignment gadget framework* [15]. Exploiting normalization tricks, this framework gives an abstract property of sequence similarity measures to allow for SETH-based quadratic lower bounds. Unfortunately, as it turns out, we cannot directly transfer the alignment gadget hardness proof for LCS to LCIS – some indication for this difficulty is already given by the fact that LCIS can be solved in strongly subquadratic time over sublinear-sized alphabets [34], while the LCS hardness proof already applies to binary alphabets. By collecting gadgetry needed to overcome such difficulties (that we elaborate on below), we hope to provide further tools to generalize more and more quadratic-time lower bounds based on SETH.

1.3.1 Technical Challenges

The known conditional lower bounds for global alignment problems such as LCS and Edit Distance work as follows. The reductions start from the quadratic-time SETH-hard Orthogonal Vectors problem (OV), that asks to determine, given two sets of $(0,1)$-vectors $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}, \mathcal{V} = \{v_0, \ldots, v_{n-1}\} \subseteq \{0,1\}^d$ over $d = n^{o(1)}$ dimensions, whether there is a pair i, j such that u_i and v_j are orthogonal, i.e., whose inner product $(u_i \cdot v_j) := \sum_{k=0}^{d-1} u_i[k] \cdot v_j[k]$ is 0 (over the integers). Each vector u_i and v_j is represented by a (normalized) vector gadget $\mathrm{VG}_{\mathrm{X}}(u_i)$ and $\mathrm{VG}_{\mathrm{Y}}(v_j)$, respectively. Roughly speaking, these gadgets are combined to sequences X and Y such that each candidate for an optimal alignment of X and Y involves locally optimal alignments between n pairs $\mathrm{VG}_{\mathrm{X}}(u_i), \mathrm{VG}_{\mathrm{Y}}(v_j)$ – the optimal alignment exceeds a certain threshold if and only if there is an orthogonal pair u_i, v_j.

An analogous approach does not work for LCIS: Let $\mathrm{VG}_{\mathrm{X}}(u_i)$ be defined over an alphabet Σ and $\mathrm{VG}_{\mathrm{X}}(u_{i'})$ over an alphabet Σ'. If Σ and Σ' overlap, then $\mathrm{VG}_{\mathrm{X}}(u_i)$ and $\mathrm{VG}_{\mathrm{X}}(u_{i'})$ cannot both be aligned in an optimal alignment without interference with each other. On the other hand, if Σ and Σ' are disjoint, then each vector v_j should have its corresponding vector gadget $VG_{\mathrm{Y}}(v_j)$ defined over both Σ and Σ' to enable to align $\mathrm{VG}_{\mathrm{X}}(u_i)$ with $\mathrm{VG}_{\mathrm{Y}}(v_j)$ as well as $\mathrm{VG}_{\mathrm{X}}(u_{i'})$ with $\mathrm{VG}_{\mathrm{Y}}(v_j)$. The latter option drastically increases the size of vector gadgets.

[2] We refer to [46] for a simple quadratic-time DP formulation for LCIS.

Thus, we must define all vector gadgets over a common alphabet Σ and make sure that *only a single pair* $\mathrm{VG}_X(u_i), \mathrm{VG}_Y(v_j)$ is aligned in an optimal alignment (in contrast with n pairs aligned in the previous reductions for LCS and Edit Distance).

1.3.2 Technical Contributions and Proof Outline

Fortunately, a surprisingly simple approach works: As a key tool, we provide *separator sequences* $\alpha_0 \dots \alpha_{n-1}$ and $\beta_0 \dots \beta_{n-1}$ with the following properties: (1) for every $i, j \in \{0, \dots, n-1\}$ the LCIS of $\alpha_0 \dots \alpha_i$ and $\beta_0 \dots \beta_j$ has a length of $f(i+j)$, where f is a linear function, and (2) $\sum_i |\alpha_i|$ and $\sum_j |\beta_j|$ are bounded by $n^{1+o(1)}$. Note that existence of such a gadget is somewhat unintuitive: condition (1) for $i = 0$ and $j = n-1$ requires $|\alpha_0| = \Omega(n)$, yet still the total length $\sum_i |\alpha_i|$ must not exceed the length of $|\alpha_0|$ significantly. Indeed, we achieve this by a careful inductive construction that generates such sequences with heavily varying block sizes $|\alpha_i|$ and $|\beta_j|$.

We apply these separator sequences as follows. We first define simple vector gadgets $\mathrm{VG}_X(u_i), \mathrm{VG}_Y(v_j)$ over an alphabet Σ such that the length of an LCIS of $\mathrm{VG}_X(u_i)$ and $\mathrm{VG}_Y(v_j)$ is $d - (u_i \cdot v_j)$. Then we construct the separator sequences as above over an alphabet $\Sigma_<$ whose elements are strictly smaller than all elements in Σ. Furthermore, we create analogous separator sequences $\alpha'_0 \dots \alpha'_{n-1}$ and $\beta'_0 \dots \beta'_{n-1}$ which satisfy a property like (1) for all suffixes instead of prefixes, using an alphabet $\Sigma_>$ whose elements are strictly larger than all elements in Σ. Now, we define

$$X = \alpha_0 \mathrm{VG}_X(u_0) \alpha'_0 \dots \alpha_{n-1} \mathrm{VG}_X(u_{n-1}) \alpha'_{n-1},$$
$$Y = \beta_0 \mathrm{VG}_Y(v_0) \beta'_0 \dots \beta_{n-1} \mathrm{VG}_Y(v_{n-1}) \beta'_{n-1}.$$

As we will show in Section 3, the length of an LCIS of X and Y is $C - \min_{i,j}(u_i \cdot v_j)$ for some constant C depending only on n and d.

In contrast to previous such OV-based lower bounds, we use heavily varying separators (paddings) between vector gadgets.

1.4 Paper organization

After setting up conventions and introducing our hardness assumptions in Section 2, we give the main construction, i.e., the hardness of LCIS in Section 3. The proofs of Theorems 4, 5 and 6 can be found in the full version. We conclude with some open problems in Section 4.

2 Preliminaries

As a convention, we use capital or Greek letters to denote sequences over integers. Let X, Y be integer sequences. We write $|X|$ for the length of X, $X[k]$ for the k-th element in the sequence X ($k \in \{0, \dots, |X|-1\}$), and $X \circ Y = XY$ for the concatenation of X and Y. We say that Y is a subsequence of X if there exist indices $0 \leqslant i_1 < i_2 < \dots < i_{|Y|} \leqslant |X| - 1$ such that $X[i_k] = Y[k]$ for all $k \in \{0, \dots, |Y|-1\}$. Given any number of sequences $X_1, \dots, X_k$, we say that Y is a common subsequence of $X_1, \dots, X_k$ if Y is a subsequence of each $X_i, i \in \{1, \dots, k\}$. X is called strictly increasing (or weakly increasing) if $X[0] < X[1] < \dots < X[|X|-1]$ (or $X[0] \leqslant X[1] \leqslant \dots \leqslant X[|X|-1]$). For any k sequences $X_1, \dots, X_k$, we denote by $\mathrm{lcis}(X_1, \dots, X_k)$ the length of their longest common subsequence that is strictly increasing.

All of our lower bounds hold assuming the Strong Exponential Time Hypothesis (SETH), introduced by Impagliazzo and Paturi [29, 30]. It essentially states that no exponential speed-up over exhaustive search is possible for the CNF satisfiability problem.

▶ **Hypothesis 7** (Strong Exponential Time Hypothesis (SETH)). *There is no $\varepsilon > 0$ such that for all $d \geqslant 3$ there is an $\mathcal{O}\left(2^{(1-\varepsilon)n}\right)$ time algorithm for d-SAT.*

This hypothesis implies tight hardness of the k-Orthogonal Vectors problem (k-OV), which will be the starting point of our reductions: Given k sets $\mathcal{U}_1, \ldots, \mathcal{U}_k \subseteq \{0,1\}^d$, each with $|\mathcal{U}_i| = n$ vectors over $d = n^{o(1)}$ dimensions, determine whether there is a k-tuple $(u_1, \ldots, u_k) \in \mathcal{U}_1 \times \cdots \times \mathcal{U}_k$ such that $\sum_{\ell=0}^{d-1} \prod_{i=1}^{k} u_i[\ell] = 0$. By exhaustive enumeration, it can be solved in time $\mathcal{O}\left(n^k d\right) = n^{k+o(1)}$. The following conjecture is implied by SETH by the well-known split-and-list technique of Williams [43] (and the sparsification lemma [30]).

▶ **Hypothesis 8** (k-OV conjecture). *Let $k \geqslant 2$. There is no $\mathcal{O}\left(n^{k-\varepsilon}\right)$ time algorithm for k-OV, with $d = \omega(\log n)$, for any constant $\varepsilon > 0$.*

For the special case of $k = 2$, which we simply denote by OV, we obtain the following weaker conjecture.

▶ **Hypothesis 9** (OV conjecture). *There is no $\mathcal{O}\left(n^{2-\varepsilon}\right)$ time algorithm for OV, with $d = \omega(\log n)$, for any constant $\varepsilon > 0$. Equivalently, even restricted to instances with $|\mathcal{U}_1| = n$ and $|\mathcal{U}_2| = n^\gamma$, $0 < \gamma \leqslant 1$, there is no $\mathcal{O}\left(n^{1+\gamma-\varepsilon}\right)$ time algorithm for OV, with $d = \omega(\log n)$, for any constant $\varepsilon > 0$.*

A proof of the folklore equivalence of the statements for equal and unequal set sizes can be found, e.g., in [15].

3 Main Construction: Hardness of LCIS

In this section, we prove quadratic-time SETH hardness of LCIS, i.e., prove Theorem 3. We first introduce an *inflation* operation, which we then use to construct our separator sequences. After defining simple vector gadgets, we show how to embed an Orthogonal Vectors instance using our vector gadgets and separator sequences.

3.1 Inflation

We begin by introducing the inflation operation, which simulates weighing the sequences.

▶ **Definition 10.** For a sequence $A = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ of integers we define:

$$\operatorname{inflate}(A) = \langle 2a_0 - 1, 2a_0, 2a_1 - 1, 2a_1, \ldots, 2a_{n-1} - 1, 2a_{n-1} \rangle .$$

▶ **Lemma 11.** *For any two sequences A and B, $\operatorname{lcis}(\operatorname{inflate}(A), \operatorname{inflate}(B)) = 2 \cdot \operatorname{lcis}(A, B)$.*

Proof. Let C be the longest common increasing subsequence of A and B. Observe that $\operatorname{inflate}(C)$ is a common increasing subsequence of $\operatorname{inflate}(A)$ and $\operatorname{inflate}(B)$ of length $2 \cdot |C|$, thus $\operatorname{lcis}(\operatorname{inflate}(A), \operatorname{inflate}(B)) \geqslant 2 \cdot \operatorname{lcis}(A, B)$.

Conversely, let $\bar{A}$ denote $\operatorname{inflate}(A)$ and $\bar{B}$ denote $\operatorname{inflate}(B)$. Let $\bar{C}$ be the longest common increasing subsequence of $\bar{A}$ and $\bar{B}$. If we divide all elements of $\bar{C}$ by 2 and round up to the closest integer, we end up with a weakly increasing sequence. Now, if we remove duplicate elements to make this sequence strictly increasing, we obtain C, a common increasing subsequence of A and B. At most 2 distinct elements may become equal after division by 2 and rounding, therefore C contains at least $\lceil \operatorname{lcis}(\bar{A}, \bar{B})/2 \rceil$ elements, so $2 \cdot \operatorname{lcis}(A, B) \geqslant \operatorname{lcis}(\bar{A}, \bar{B})$. This completes the proof. ◀

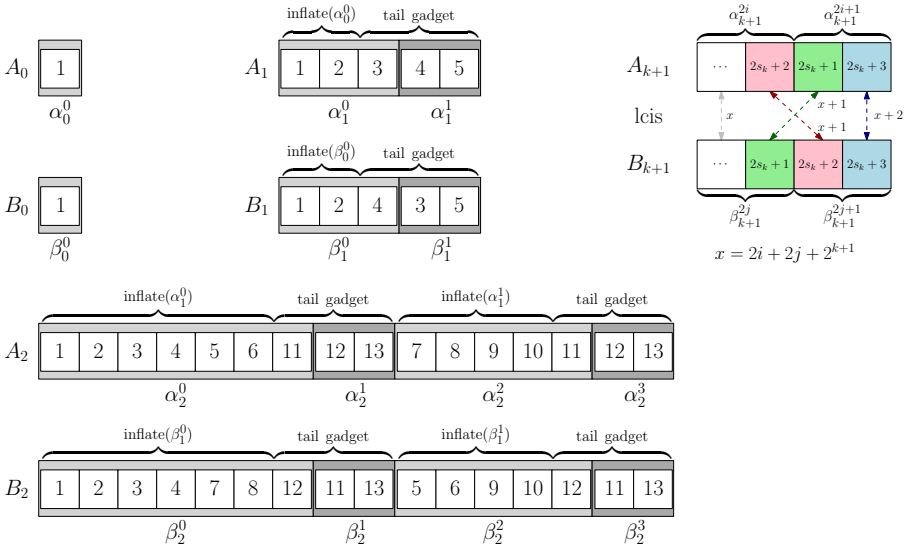

Figure 1 Initial steps of inductive construction of separator sequences (left), and intuition behind tail gadgets (right).

3.2 Separator sequences

Our goal is to construct two sequences A and B which can be split into n blocks, i.e. $A = \alpha_0\alpha_1 \dots \alpha_{n-1}$ and $B = \beta_0\beta_1 \dots \beta_{n-1}$, such that the length of the longest common increasing subsequence of the first i blocks of A and the first j blocks of B equals $i + j$, up to an additive constant. We call A and B *separator sequences*, and use them later to separate vector gadgets in order to make sure that only one pair of gadgets may interact with each other at the same time.

We construct the separator sequences inductively. For every $k \in \mathbb{N}$, the sequences A_k and B_k are concatenations of 2^k blocks (of varying sizes), $A_k = \alpha_k^0\alpha_k^1 \dots \alpha_k^{2^k-1}$ and $B_k = \beta_k^0\beta_k^1 \dots \beta_k^{2^k-1}$. Let s_k denote the largest element of both sequences. As we will soon observe, $s_k = 2^{k+2} - 3$.

The construction works as follows: for $k = 0$, we can simply set A_0 and B_0 as one-element sequences $\langle 1 \rangle$. We then construct A_{k+1} and B_{k+1} inductively from A_k and B_k in two steps. First, we inflate both A_k and B_k, then after each (now inflated) block we insert 3-element sequences, called *tail gadgets*, $\langle 2s_k + 2, 2s_k + 1, 2s_k + 3 \rangle$ for A_{k+1} and $\langle 2s_k + 1, 2s_k + 2, 2s_k + 3 \rangle$ for B_{k+1}. Formally, we describe the construction by defining blocks of the new sequences. For $i \in \{0, 1, \dots, 2^k - 1\}$,

$$\alpha_{k+1}^{2i} = \text{inflate}(\alpha_k^i) \circ \langle 2s_k + 2 \rangle, \qquad \alpha_{k+1}^{2i+1} = \langle 2s_k + 1, 2s_k + 3 \rangle,$$
$$\beta_{k+1}^{2i} = \text{inflate}(\beta_k^i) \circ \langle 2s_k + 1 \rangle, \qquad \beta_{k+1}^{2i+1} = \langle 2s_k + 2, 2s_k + 3 \rangle.$$

Note that the symbols appearing in tail gadgets do not appear in the inflated sequences. The largest element of both new sequences s_{k+1} equals $2s_k + 3$, and solving the recurrence gives indeed $s_k = 2^{k+2} - 3$.

Now, let us prove two useful properties of the separator sequences.

▶ **Lemma 12.** $|A_k| = |B_k| = \left(\frac{3}{2}k + 1\right) \cdot 2^k = \mathcal{O}\left(k2^k\right)$.

Proof. Observe that $|A_{k+1}| = 2|A_k| + 3 \cdot 2^k$. Indeed, to obtain A_{k+1} first we double the size of A_k and then add 3 new elements for each of the 2^k blocks of A_k. Solving the recurrence completes the proof. The same reasoning applies to B_k. ◀

▶ **Lemma 13.** *For every* $i, j \in \{0, 1, \ldots, 2^k - 1\}$, $\operatorname{lcis}(\alpha_k^0 \ldots \alpha_k^i, \beta_k^0 \ldots \beta_k^j) = i + j + 2^k$.

Proof. The proof is by induction on k. Assume the statement is true for k and let us prove it for $k+1$.

The "$\geqslant$" direction. First, consider the case when both i and j are even. Observe that $\operatorname{inflate}(\alpha_k^0 \ldots \alpha_k^{i/2})$ and $\operatorname{inflate}(\beta_k^0 \ldots \beta_k^{j/2})$ are subsequences of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^i$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^j$, respectively. Thus, using the induction hypothesis and inflation properties,

$$\begin{aligned}\operatorname{lcis}(\alpha_{k+1}^0 \ldots \alpha_{k+1}^i, \beta_{k+1}^0 \ldots \beta_{k+1}^j) &\geqslant \operatorname{lcis}(\operatorname{inflate}(\alpha_k^0 \ldots \alpha_k^{i/2}), \operatorname{inflate}(\beta_k^0 \ldots \beta_k^{j/2})) = \\ &= 2 \cdot \operatorname{lcis}(\alpha_k^0 \ldots \alpha_k^{i/2}, \beta_k^0 \ldots \beta_k^{j/2}) = 2 \cdot (i/2 + j/2 + 2^k) = i + j + 2^{k+1}.\end{aligned}$$

If i is odd and j is even, refer to the previous case to get a common increasing subsequence of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^{i-1}$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^j$ of length $i - 1 + j + 2^{k+1}$ consisting only of elements less than or equal to $2s_k$, and append the element $2s_k + 1$ to the end of it. Analogously, for i even and j odd, take such an LCIS of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^i$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^{j-1}$, and append $2s_k + 2$. Finally, for both i and j odd, take an LCIS of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^{i-1}$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^{j-1}$, and append $2s_k + 1$ and $2s_k + 3$.

The "$\leqslant$" direction. We proceed by induction on $i + j$. Fix i and j, and let L be a longest common increasing subsequence of $\alpha_{k+1}^0 \ldots \alpha_{k+1}^i$ and $\beta_{k+1}^0 \ldots \beta_{k+1}^j$.

If the last element of L is less than or equal to $2s_k$, L is in fact a common increasing subsequence of $\operatorname{inflate}(\alpha_k^0 \ldots \alpha_k^{\lfloor i/2 \rfloor})$ and $\operatorname{inflate}(\beta_k^0 \ldots \beta_k^{\lfloor j/2 \rfloor})$, thus, by the induction hypothesis and inflation properties, $|L| \leqslant 2 \cdot (\lfloor i/2 \rfloor + \lfloor j/2 \rfloor + 2^k) \leqslant i + j + 2^{k+1}$.

The remaining case is when the last element of L is greater than $2s_k$. In this case, consider the second-to-last element of L. It must belong to some blocks $\alpha_{k+1}^{i'}$ and $\beta_{k+1}^{j'}$ for $i' \leqslant i$ and $j' \leqslant j$, and we claim that $i = i'$ and $j = j'$ cannot hold simultaneously: by construction of separator sequences, if blocks α_{k+1}^i and β_{k+1}^j have a common element larger than $2s_k$, then it is the only common element of these two blocks. Therefore, it cannot be the case that both $i = i'$ and $j = j'$, because the last two elements of L would then be located in α_{k+1}^i and β_{k+1}^j. As a consequence, $i' + j' < i + j$, which lets us apply the induction hypothesis to reason that the prefix of L omitting its last element is of length at most $i' + j' + 2^{k+1}$. Therefore, $|L| \leqslant 1 + i' + j' + 2^{k+1} \leqslant i + j + 2^{k+1}$, which completes the proof. ◀

Observe that if we reverse the sequences A_k and B_k along with changing all elements to their negations, i.e. x to $-x$, we obtain sequences $\hat{A}_k$ and $\hat{B}_k$ such that $\hat{A}_k$ splits into 2^k blocks $\hat{\alpha}_k^0 \ldots \hat{\alpha}_k^{2^k-1}$, $\hat{B}_k$ splits into 2^k blocks $\hat{\beta}_k^0 \ldots \hat{\beta}_k^{2^k-1}$, and

$$\operatorname{lcis}(\hat{\alpha}_k^i \ldots \hat{\alpha}_k^{2^k-1}, \hat{\beta}_k^j \ldots \hat{\beta}_k^{2^k-1}) = 2 \cdot (2^k - 1) - i - j + 2^k. \tag{1}$$

Finally, observe that we can add any constant to all elements of the sequences A_k and B_k (as well as $\hat{A}_k$ and $\hat{B}_k$) without changing the property stated in Lemma 13 (and its analogue for $\hat{A}_k$ and $\hat{B}_k$, i.e. Equation (1)).

3.3 Vector gadgets

Let $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}$ and $\mathcal{V} = \{v_0, \ldots, v_{n-1}\}$ be two sets of d-dimensional $(0,1)$-vectors.

For $i \in \{0, 1, \ldots, n-1\}$ let us construct the vector gadgets U_i and V_i as $2d$-element sequences, by defining, for every $p \in \{0, 1, \ldots, d-1\}$,

$$(U_i[2p-1], U_i[2p]) = \begin{cases} (2p-1, 2p) & \text{if } u_i[p] = 0, \\ (2p-1, 2p-1) & \text{if } u_i[p] = 1, \end{cases}$$

$$(V_i[2p-1], V_i[2p]) = \begin{cases} (2p, 2p-1) & \text{if } v_i[p] = 0, \\ (2p, 2p) & \text{if } v_i[p] = 1. \end{cases}$$

Observe that at most one of the elements $2p-1$ and $2p$ may appear in the LCIS of U_i and V_j, and it happens if and only if $u_i[p]$ and $v_j[p]$ are not both equal to one. Therefore, $\operatorname{lcis}(U_i, V_j) = d - (u_i \cdot v_j)$, and, in particular, $\operatorname{lcis}(U_i, V_j) = d$ if and only if u_i and v_j are orthogonal.

3.4 Final construction

To put all the pieces together, we plug vector gadgets U_i and V_j into the separator sequences from Section 3.2, obtaining two sequences whose LCIS depends on the minimal inner product of vectors u_i and v_j. We provide a general construction of such sequences, which will be useful for proving further results in the full version of the paper.

▶ **Lemma 14.** *Let $X_0, X_1, \ldots, X_{n-1}$, $Y_0, Y_1, \ldots, Y_{n-1}$ be integer sequences such that none of them has an increasing subsequence longer than δ. Then there exist sequences X and Y of length $\mathcal{O}(\delta \cdot n \log n) + \sum |X_i| + \sum |Y_j|$, constructible in linear time, such that:*

$$\operatorname{lcis}(X, Y) = \max_{i,j} \operatorname{lcis}(X_i, Y_j) + C$$

for a constant C that only depends on n and δ and is $\mathcal{O}(n\delta)$.

Proof. We can assume that $n = 2^k$ for some positive integer k, adding some dummy sequences if necessary. Recall the sequences A_k, B_k, $\hat{A}_k$ and $\hat{B}_k$ constructed in Section 3.2. Let $A, B, \hat{A}, \hat{B}$ be the sequences obtained from $A_k, B_k, \hat{A}_k, \hat{B}_k$ by applying inflation $\lceil \log_2 \delta \rceil$ times (thus increasing their length by a factor of $\ell = 2^{\lceil \log_2 \delta \rceil} \geqslant \delta$). Each of these four sequences splits into (now inflated) blocks, e.g. $A = \alpha_0 \alpha_1 \ldots \alpha_{n-1}$, where $\alpha_i = \operatorname{inflate}^{\lceil \log_2 \delta \rceil}(\alpha_k^i)$.

We subtract from A and B a constant large enough for all their elements to be smaller than all elements of every X_i and Y_j. Similarly, we add to A' and B' a constant large enough for all their elements to be larger than all elements of every X_i and Y_j. Now, we can construct the sequences X and Y as follows:

$$X = \alpha_0 X_0 \hat{\alpha}_0 \alpha_1 X_1 \hat{\alpha}_1 \ldots \alpha_{n-1} X_{n-1} \hat{\alpha}_{n-1},$$
$$Y = \beta_0 Y_0 \hat{\beta}_0 \beta_1 Y_1 \hat{\beta}_1 \ldots \beta_{n-1} Y_{n-1} \hat{\beta}_{n-1}.$$

We claim that

$$\operatorname{lcis}(X, Y) = \ell \cdot (4n-2) + M, \text{ where } M = \max_{i,j} \operatorname{lcis}(X_i, Y_j).$$

Let X_i and Y_j be the pair of sequences achieving $\operatorname{lcis}(X_i, Y_j) = M$. Recall that $\operatorname{lcis}(\alpha_0 \ldots \alpha_i, \beta_0 \ldots \beta_j) = \ell \cdot (i + j + n)$, with all the elements of this common subsequence preceding the elements of X_i and Y_j in X and Y, respectively, and being smaller than them. In the same way $\operatorname{lcis}(\hat{\alpha}_i \ldots \hat{\alpha}_{n-1}, \hat{\beta}_j \ldots \hat{\beta}_{n-1}) = \ell \cdot (2 \cdot (n-1) - (i+j) + n)$ with all the elements of LCIS being greater and appearing later than those of X_i and Y_j. By

concatenating these three sequences we obtain a common increasing subsequence of X and Y of length $\ell \cdot (4n-2) + M$.

We defer the simple remainder of the proof, i.e., proving $\text{lcis}(X,Y) \leqslant \ell \cdot (4n-2) + M$ to the full version of the paper. ◀

Proof of Theorem 3. Let $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}$, $\mathcal{V} = \{v_0, \ldots, v_{n-1}\}$ be two sets of binary vectors in d dimensions. In Section 3.3 we constructed vector gadgets U_i and V_j, for $i, j \in \{0, 1, \ldots, n-1\}$, such that $\text{lcis}(U_i, V_j) = d - (u_i \cdot v_j)$. To these sequences we apply Lemma 14, with $\delta = 2d$, obtaining sequences X and Y of length $\mathcal{O}(n \log n \text{poly}(d))$ such that $\text{lcis}(X,Y) = C + d - \min_{i,j}(u_i \cdot v_j)$ for a constant C. This reduction, combined with an $\mathcal{O}\left(n^{2-\varepsilon}\right)$ time algorithm for LCIS, would yield an $\mathcal{O}\left(n^{2-\varepsilon}\text{polylog}(n)\text{poly}(d)\right)$ algorithm for OV, refuting Hypothesis 9 and, in particular, SETH. ◀

4 Conclusion and Open Problems

We prove a tight quadratic lower bound for LCIS, ruling out strongly subquadratic-time algorithms under SETH. It remains open whether LCIS admits mildly subquadratic algorithms, such as the Masek-Paterson algorithm for LCS [35]. Furthermore, we give tight SETH-based lower bounds for k-LCIS.

For the related variant LCWIS that considers weakly increasing sequences, strongly subquadratic-time algorithms are ruled out under SETH for slightly superlogarithmic alphabet sizes ([39] and Theorem 6). On the other hand, for binary and ternary alphabets, even linear time algorithms exist [34, 23]. Can LCWIS be solved in time $\mathcal{O}\left(n^{2-f(|\Sigma|)}\right)$ for some decreasing function f that yields strongly subquadratic-time algorithms for any constant alphabet size $|\Sigma|$?

Finally, we can compute a $(1+\varepsilon)$-approximation of LCIS in $\mathcal{O}\left(n^{3/2}\varepsilon^{-1/2}\text{polylog}(n)\right)$ time by an easy observation (see the appendix in the full version). Can we improve upon this running time or give a matching conditional lower bound? Note that a positive resolution seems difficult by the reduction in Observation 1: Any n^α, $\alpha > 0$, improvement over this running time would yield a strongly subcubic $(1+\varepsilon)$-approximation for 3-LCS, which seems hard to achieve, given the difficulty to find strongly subquadratic $(1+\varepsilon)$-approximation algorithms for LCS.

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Quadratic-time hardness of LCS and other sequence similarity measures. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 59–78, 2015.
2. 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 *Proc. 48th Annual ACM Symposium on Symposium on Theory of Computing (STOC'16)*, pages 375–388, 2016.
3. Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Proc. of 41st International Colloquium on Automata, Languages, and Programming (ICALP'14)*, pages 39–51, 2014.
4. Alfred V. Aho, Daniel S. Hirschberg, and Jeffrey D. Ullman. Bounds on the complexity of the longest common subsequence problem. *Journal of the ACM*, 23(1):1–12, 1976.
5. Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

6 Hsing-Yen Ann, Chang-Biau Yang, and Chiou-Ting Tseng. Efficient polynomial-time algorithms for the constrained LCS problem with strings exclusion. *Journal of Combinatorial Optimization*, 28(4):800–813, 2014.

7 Alberto Apostolico and Concettina Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2(1):316–336, 1987.

8 Abdullah N. Arslan and Ömer Egecioglu. Algorithms for the constrained longest common subsequence problems. *International Journal of Foundations of Computer Science*, 16(6):1099–1109, 2005.

9 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proc. 47th Annual ACM Symposium on Theory of Computing (STOC'15)*, pages 51–58, 2015.

10 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *Proc. 57th Annual Symposium on Foundations of Computer Science, (FOCS'16)*, pages 457–466, 2016.

11 Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In *Proc. 34th International Conference on Machine Learning (ICML'17)*, 2017. To appear.

12 Gary Benson, Avivit Levy, S. Maimoni, D. Noifeld, and B. Riva Shalom. Lcsk: A refined similarity measure. *Theoretical Computer Science*, 638:11–26, 2016.

13 Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *Proc. 7th International Symposium on String Processing and Information Retrieval (SPIRE'00)*, pages 39–48, 2000.

14 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*, pages 661–670, 2014.

15 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proc. 56th Annual IEEE Symposium on Foundations of Compu ter Science (FOCS'15)*, pages 79–97, 2015.

16 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, 2018. To appear.

17 Wun-Tat Chan, Yong Zhang, Stanley P. Y. Fung, Deshi Ye, and Hong Zhu. Efficient algorithms for finding a longest common increasing subsequence. *Journal of Combinatorial Optimization*, 13(3):277–288, 2007.

18 Yi-Ching Chen and Kun-Mao Chao. On the generalized constrained longest common subsequence problems. *Journal of Combinatorial Optimization*, 21(3):383–392, 2011.

19 Francis Y. L. Chin, Alfredo De Santis, Anna Lisa Ferrara, N. L. Ho, and S. K. Kim. A simple algorithm for the constrained sequence problems. *Inf. Process. Lett.*, 90(4):175–179, 2004. doi:10.1016/j.ipl.2004.02.008.

20 Vaclav Chvatal, David A. Klarner, and Donald E. Knuth. Selected combinatorial research problems. Technical Report CS-TR-72-292, Stanford University, Department of Computer Science, 6 1972.

21 Maxime Crochemore and Ely Porat. Fast computation of a longest increasing subsequence and application. *Information & Computation*, 208(9):1054–1059, 2010.

22 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min,+)-convolution. In *Proc. 44th International Colloquium on Automata, Languages, and Programming (ICALP'17)*, pages 22:1–22:15, 2017.

23 Lech Duraj. A linear algorithm for 3-letter longest common weakly increasing subsequence. *Information Processing Letters*, 113(3):94–99, 2013.

24 Michael L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

25 Zvi Gotthilf, Danny Hermelin, Gad M. Landau, and Moshe Lewenstein. Restricted LCS. In *Proc. 17th International Symposium on String Processing and Information Retrieval (SPIRE'10)*, pages 250–257, 2010.

26 Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675, 1977.

27 J. W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Computing Science Technical Report 41, Bell Laboratories, 1975.

28 James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

29 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

30 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

31 Guy Jacobson and Kiem-Phong Vo. Heaviest increasing/common subsequence problems. In *Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson, Arizona, USA, April 29 - May 1, 1992, Proceedings*, pages 52–66, 1992.

32 Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. The longest common subsequence problem for arc-annotated sequences. *Journal of Discrete Algorithms*, 2(2):257–270, 2004.

33 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-grained Complexity of One-Dimensional Dynamic Programming. In *Proc. 44th International Colloquium on Automata, Languages, and Programming (ICALP'17)*, pages 21:1–21:15, 2017.

34 Martin Kutz, Gerth Stølting Brodal, Kanela Kaligosi, and Irit Katriel. Faster algorithms for computing longest common increasing subsequences. *Journal of Discrete Algorithms*, 9(4):314–325, 2011.

35 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

36 Howard L. Morgan. Spelling correction in systems programs. *Communications of the ACM*, 13(2):90–94, 1970.

37 Eugene W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

38 Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

39 Adam Polak. Why is it hard to beat $O(n^2)$ for longest common weakly increasing subsequence? *CoRR*, abs/1703.01143, 2017.

40 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. 45th Annual ACM Symposium on Symposium on Theory of Computing (STOC'13)*, pages 515–524, 2013.

41 Yin-Te Tsai. The constrained longest common subsequence problem. *Information Processing Letters*, 88(4):173–176, 2003.

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

43 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

44 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *Proc. 10th International Symposium on Parameterized and Exact Computation (IPEC'15)*, pages 17–29, 2015.

45 I-Hsuan Yang, Chien-Pin Huang, and Kun-Mao Chao. A fast algorithm for computing a longest common increasing subsequence. *Information Processing Letters*, 93(5):249–253, 2005.

46 Daxin Zhu, Lei Wang, Tinran Wang, and Xiaodong Wang. A simple linear space algorithm for computing a longest common increasing subsequence. *CoRR*, abs/1608.07002, 2016.