Tight Conditional Lower Bounds for Longest Common Increasing Subsequence

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

[1] Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland, {duraj,polak}@tcs.uj.edu.pl
[2] Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany, marvin@mpi-inf.mpg.de

arXiv:1709.10075v1 [cs.CC] 28 Sep 2017

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 (1) to rule out $\mathcal{O}\left((nL)^{1-\varepsilon}\right)$ time algorithms for LCIS, where L denotes the solution size, (2) to rule out $\mathcal{O}\left(n^{k-\varepsilon}\right)$ time algorithms for k-LCIS, and (3) to follow already from weaker variants of SETH. We obtain the same conditional lower bounds for the related Longest Common Weakly Increasing Subsequence problem.

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 [37, 43] or DNA sequence comparison [39, 5], as well as determining the differences of text files as in the UNIX DIFF utility [28]. 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 $\mathcal{O}\left(n^2\right)$ [43], and in many practical settings, certain properties of typical input sequences can be exploited to obtain faster, "tailored" solutions (e.g., [27, 29, 7, 38]; see also [14] 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 [21, 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 [44, 41, 15, 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, 16]. 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) [32], which introduces weights to the problem, as well as notions that constrain the structure of the solution, such as the longest common increasing subsequence

*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*.

(LCIS) [46], LCSk [13], constrained LCS [42, 20, 8], restricted LCS [26], and many other variants (see, e.g., [19, 6, 33]). 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, 16] 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 [22] for an overview), which has an $\mathcal{O}(n \log n)$ time solution and a matching lower bound in the decision tree model [25]. The extension to $k = 2$, denoted simply as LCIS, has been proposed by Yang, Huang, and Chao [46], 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 [29], explicit in [32])**.** *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.*

Proof. Let $L(\sigma)$ denote the descending sequence of positions k with $X_1[k] = \sigma$. We define sequences $X'_i = L(X_i[0]) \cdots L(X_i[|X_i| - 1])$ for all $i \in \{2, \ldots, k\}$. It is straightforward to see that for any ℓ, the length-ℓ increasing common subsequences of $X'_2, \ldots, X'_k$ are in one-to-one correspondence to length-ℓ common subsequences of $X_1, \ldots, X_k$. Thus, the length of the LCIS of $X'_2, \ldots, X'_k$ is equal to the length of the LCS of $X_1, \ldots, X_k$, and the claim follows since $|L(\sigma)| \leqslant n$ for all σ. □

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

Proof. 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. Such an algorithm would refute SETH by a result of Abboud et al. [1]. □

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 [18, 35] 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. [35] provided an algorithm running in time $\mathcal{O}(nL \log\log n + n \log n)$. 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 [18, 35], or even $\mathcal{O}\left(n^k\right)$ time (see the appendix). 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 $\mathcal{O}\left(nL \log n\right) = \mathcal{O}\left(n^{3/2} \log n\right)$ [35], by using the fact that $L \leqslant |\Sigma|$. In contrast, LCWIS is quadratic-time SETH hard already over slightly superlogarithmic-sized alphabets [40]. 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.*

1.2.4 Strengthening the Hardness

In an attempt to strengthen the conditional lower bounds for Edit Distance and LCS [9, 1, 16], particularly, to obtain barriers even for subpolynomial improvements, Abboud, Hansen, Vassilevska Williams, and Williams [2] gave the first fine-grained reductions from the satisfiability problem on branching programs. Using this approach, the quadratic-time hardness of a problem can be explained by considerably weaker variants of SETH, making the conditional lower bound stronger. We show that our lower bounds also hold under these weaker variants. In particular, we prove the following.

Theorem 7. *There is no strongly subquadratic time algorithm for LCIS, unless there is, for some $\varepsilon > 0$, an $\mathcal{O}\left((2-\varepsilon)^N\right)$ algorithm for the satisfiability problem on branching programs of width W and length T on N variables with $(\log W)(\log T) = o(N)$.*

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., [15, 3, 9, 10, 1, 16, 11, 23, 34] and [45] 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

[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 [17].

[2] We refer to [47] for a simple quadratic-time DP formulation for LCIS.

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* [16]. 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 [35], 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 $\mathrm{VG_X}(u_i)$ and $\mathrm{VG_Y}(v_j)$, 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 $\mathrm{VG_X}(u_i), \mathrm{VG_Y}(v_j)$ – 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 $\mathrm{VG_X}(u_i)$ be defined over an alphabet Σ and $\mathrm{VG_X}(u_{i'})$ over an alphabet Σ'. If Σ and Σ' overlap, then $\mathrm{VG_X}(u_i)$ and $\mathrm{VG_X}(u_{i'})$ 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 $\mathrm{VG_X}(u_i)$ with $\mathrm{VG_Y}(v_j)$ as well as $\mathrm{VG_X}(u_{i'})$ with $\mathrm{VG_Y}(v_j)$. The latter option drastically increases the size of vector gadgets. 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* $\alpha_0 \ldots \alpha_{n-1}$ and $\beta_0 \ldots \beta_{n-1}$ with the following properties: (1) for every $i, j \in \{0, \ldots, n-1\}$ the LCIS of $\alpha_0 \ldots \alpha_i$ and $\beta_0 \ldots \beta_j$ 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 $\alpha'_0 \ldots \alpha'_{n-1}$ and $\beta'_0 \ldots \beta'_{n-1}$ 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

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

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.

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 ($k \in \{0, \ldots, |X|-1\}$), 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 $0 \leqslant i_1 < i_2 < \cdots < i_{|Y|} \leqslant |X|-1$ such that $X[i_k] = Y[k]$ for all $k \in \{0, \ldots, |Y|-1\}$. Given any number of sequences $X_1, \ldots, X_k$, we say that Y is a common subsequence of $X_1, \ldots, X_k$ if Y is a subsequence of each $X_i, i \in \{1, \ldots, k\}$. X is called strictly increasing (or weakly increasing) if $X[0] < X[1] < \cdots < X[|X|-1]$ (or $X[0] \leqslant X[1] \leqslant \cdots \leqslant X[|X|-1]$). For any k sequences $X_1, \ldots, X_k$, we denote by $\mathrm{lcis}(X_1, \ldots, X_k)$ the length of their longest common subsequence that is strictly increasing.

2.1 Hardness Assumptions

All of our lower bounds hold assuming the Strong Exponential Time Hypothesis (SETH), introduced by Impagliazzo and Paturi [30, 31]. It essentially states that no exponential speed-up over exhaustive search is possible for the CNF satisfiability problem.

Hypothesis 8 (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 [44] (and the sparsification lemma [31]).

Hypothesis 9 (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 10 (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 [16].

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 roughly corresponds to weighing the sequences.

Definition 11. *For a sequence $A = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ of integers we define:*

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

Lemma 12. *For any two sequences A and B, $\mathrm{lcis}(\mathrm{inflate}(A), \mathrm{inflate}(B)) = 2 \cdot \mathrm{lcis}(A, B)$.*

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

Conversely, let $\bar{A}$ denote $\mathrm{inflate}(A)$ and $\bar{B}$ denote $\mathrm{inflate}(B)$. 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 $\lceil \mathrm{lcis}(\bar{A}, \bar{B})/2 \rceil$ elements, so $2 \cdot \mathrm{lcis}(A, B) \geqslant \mathrm{lcis}(\bar{A}, \bar{B})$. This completes the proof. □

3.2 Separator sequences

Our goal is to construct two sequences A and B which can be split into n blocks, i.e. $A = \alpha_0\alpha_1 \ldots \alpha_{n-1}$ and $B = \beta_0\beta_1 \ldots \beta_{n-1}$, 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), $A_k = \alpha_k^0\alpha_k^1 \ldots \alpha_k^{2^k-1}$ and $B_k = \beta_k^0\beta_k^1 \ldots \beta_k^{2^k-1}$. 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 $i \in \{0, 1, \ldots, 2^k - 1\}$,

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

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 $s_k = 2^{k+2} - 3$.

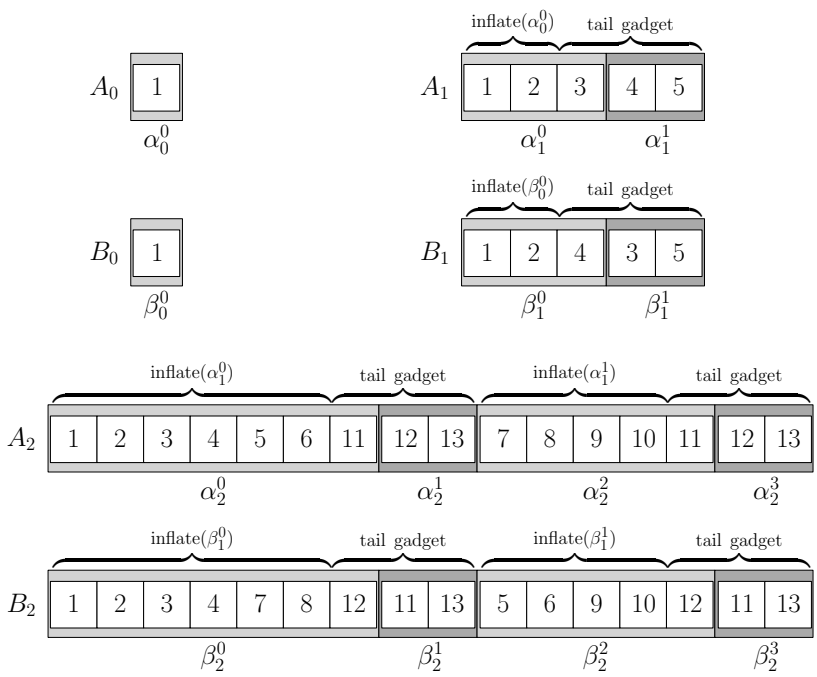

Figure 1: Initial steps of the inductive construction of the separator sequences.

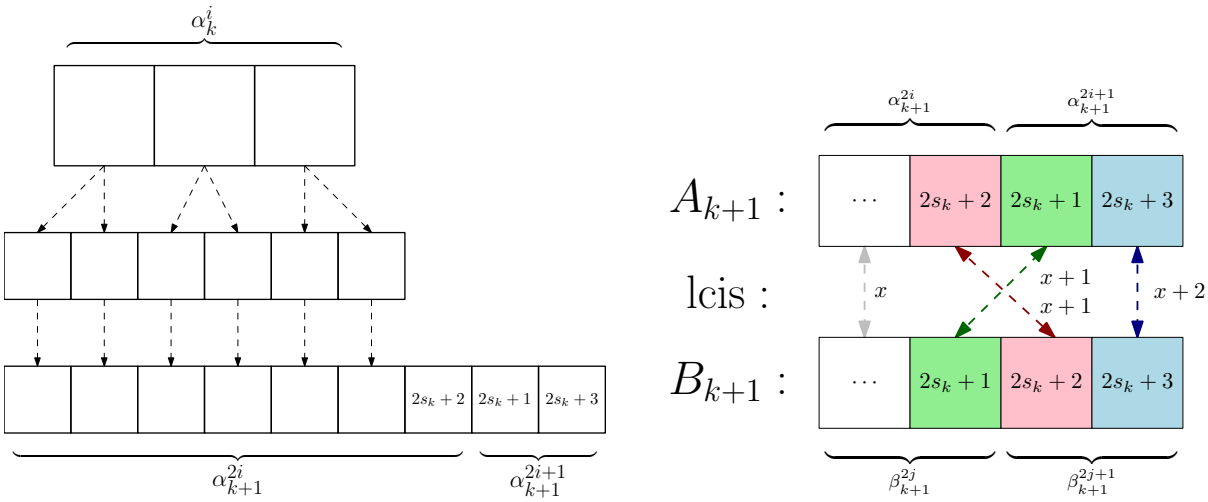

Figure 2: Left: constructing A_{k+1} from A_k. Right: intuition behind tail gadgets, $x = 2i + 2j + 2^{k+1}$.

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

Lemma 13. $|A_k| = |B_k| = \left(\frac{3}{2}k + 1\right) \cdot 2^k = \mathcal{O}\left(k2^k\right)$.

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

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 $\mathrm{inflate}(\alpha_k^0 \ldots \alpha_k^{i/2})$ and $\mathrm{inflate}(\beta_k^0 \ldots \beta_k^{j/2})$ 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,

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

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 $\mathrm{inflate}(\alpha_k^0 \ldots \alpha_k^{\lfloor i/2 \rfloor})$ and $\mathrm{inflate}(\beta_k^0 \ldots \beta_k^{\lfloor j/2 \rfloor})$, 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^i_{k+1} and β^j_{k+1}. 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 $\hat{\alpha}^0_k \dots \hat{\alpha}^{2^k-1}_k$, $\hat{B}_k$ splits into 2^k blocks $\hat{\beta}^0_k \dots \hat{\beta}^{2^k-1}_k$, and

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

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 14 (and its analogue for $\hat{A}_k$ and $\hat{B}_k$, i.e. Equation (1)).

3.3 Vector gadgets

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

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

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

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

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, $\mathrm{lcis}(U_i, V_j) = d - (u_i \cdot v_j)$, and, in particular, $\mathrm{lcis}(U_i, V_j) = d$ 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 in later sections.

Lemma 15. *Let $X_0, X_1, \dots, X_{n-1}$, $Y_0, Y_1, \dots, Y_{n-1}$ 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:*

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

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. $A = \alpha_0 \alpha_1 \dots \alpha_{n-1}$, where $\alpha_i = \mathrm{inflate}^{\lceil \log_2 \delta \rceil}(\alpha^i_k)$.

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:

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

We claim that

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

Let X_i and Y_j be the pair of sequences achieving $\mathrm{lcis}(X_i, Y_j) = M$. Recall that $\mathrm{lcis}(\alpha_0 \dots \alpha_i, \beta_0 \dots \beta_j) = \ell \cdot (i+j+n)$, 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 $\mathrm{lcis}(\hat{\alpha}_i \dots \hat{\alpha}_{n-1}, \hat{\beta}_j \dots \hat{\beta}_{n-1}) = \ell \cdot (2 \cdot (n-1) - (i+j) + n)$ 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 $\ell \cdot (4n-2) + M$.

It remains to prove $\mathrm{lcis}(X,Y) \leqslant \ell \cdot (4n-2) + M$. Let L be any common increasing subsequence of X and Y. Observe that L must split into three (some of them possibly empty) parts $L = SG\hat{S}$ with S consisting only of elements of A and B, G – only elements of X_i and Y_j, and $\hat{S}$ – elements of $\hat{A}$ and $\hat{B}$.

Let x be the last element of S and $\hat{x}$ the first element of $\hat{S}$. We know that x belongs to some blocks α_i of A and β_j of B, and $\hat{x}$ belongs to some blocks $\hat{\alpha}_{\hat{i}}$ of $\hat{A}$ and $\hat{\beta}_{\hat{j}}$ of $\hat{B}$. Obviously $i \leqslant \hat{i}$ and $j \leqslant \hat{j}$. By Lemma 14 and inflation properties we have $|S| \leqslant \ell \cdot (i+j+n)$ and $|\hat{S}| \leqslant \ell \cdot (2 \cdot (n-1) - (\hat{i}+\hat{j}) + n)$. We consider two cases:

Case 1. If $i = \hat{i}$ and $j = \hat{j}$, then G may only contain elements of X_i and Y_j. Therefore

$$|L| \leqslant |S| + \mathrm{lcis}(X_i, Y_j) + |\hat{S}| \leqslant \ell \cdot (4n-2) + M.$$

Case 2. If $i < \hat{i}$ or $j < \hat{j}$, then G must be a strictly increasing subsequence of both $X_i \circ \dots \circ X_{\hat{i}}$ and $Y_j \circ \dots \circ Y_{\hat{j}}$ therefore its length can be bounded by

$$\begin{aligned} |G| \leqslant \min(\delta \cdot (\hat{i} - i + 1), \delta \cdot (\hat{j} - j + 1)) \leqslant \ell \cdot (\min(\hat{i} - i, \hat{j} - j) + 1) \leqslant \\ \leqslant \ell \cdot (\min(\hat{i} - i, \hat{j} - j) + \max(\hat{i} - i, \hat{j} - j)) = \ell \cdot (\hat{i} - i + \hat{j} - j). \end{aligned}$$

On the other hand, $|S| + |\hat{S}| \leqslant \ell \cdot (4n - 2 - (\hat{i} - i) - (\hat{j} - j))$. From that we obtain $|L| \leqslant \ell \cdot (4n-2)$, as desired.

□

We are ready to prove the main result of the paper.

Proof of Theorem 3. Let $\mathcal{U} = \{u_0, \dots, u_{n-1}\}$, $\mathcal{V} = \{v_0, \dots, v_{n-1}\}$ be two sets of d-dimensional binary vectors. In Section 3.3 we constructed vector gadgets U_i and V_j, for $i, j \in \{0, 1, \dots, n-1\}$, such that $\mathrm{lcis}(U_i, V_j) = d - (u_i \cdot v_j)$. To these sequences we apply Lemma 15, with $\delta = 2d$, obtaining sequences X and Y of length $\mathcal{O}(n \log n \mathrm{poly}(d))$ such that $\mathrm{lcis}(X,Y) = C + d - \min_{i,j}(u_i \cdot v_j)$ for a constant C. This reduction, combined with an $\mathcal{O}\left(n^{2-\varepsilon}\right)$ time algorithm for LCIS, would yield an $\mathcal{O}\left(n^{2-\varepsilon}\mathrm{polylog}(n)\mathrm{poly}(d)\right)$ algorithm for OV, refuting Hypothesis 10 and, in particular, SETH. □

With the reduction above, one can not only determine whether there exist a pair of orthogonal vectors or not, but also, in the latter case, calculate the minimum inner product over all pairs of vectors. Formally, by the above construction, we can reduce even the Most Orthogonal Vectors problem, as defined in Abboud et al. [1] to LCIS. This bases hardness of LCIS already on the inability to improve over exhaustive search for the MAX-CNF-SAT problem, which is a slightly weaker conjecture than SETH.

4 Matching Lower Bound for Output-Dependent Algorithms

To prove our bivariate conditional lower bound of $(nL)^{1-o(1)}$, we provide a reduction from an OV instance with unequal vector set sizes.

Proof of Theorem 4. Let $0 < \gamma \leqslant 1$ be arbitrary and consider any OV instance with sets $\mathcal{U}, \mathcal{V} \subseteq \{0,1\}^d$ with $|\mathcal{U}| = n$, $|\mathcal{V}| = m = n^\gamma$ and $d = n^{o(1)}$. We reduce this problem, in linear time in the output size, to an LCIS instance with sequences X and Y satisfying $|X| = |Y| = \mathcal{O}(nd \log n)$ and an LCIS of length $\mathcal{O}(n^\gamma d)$. Theorem 4 is an immediate consequence of the reduction: an $\mathcal{O}\left((nL)^{1-\varepsilon}\right)$ time LCIS algorithm would yield an OV algorithm running in time $\mathcal{O}\left(n^{1+\gamma-\varepsilon'}\right)$, which would refute Hypothesis 10 and, in particular, SETH.

It remains to show the reduction itself. Let $\mathcal{U} = \{u_0, \ldots, u_{n-1}\}$ and $\mathcal{V} = \{v_0, \ldots, v_{m-1}\}$ be two sets of d-dimensional $(0,1)$-vectors. By adding dummy vectors, we can assume without loss of generality that $n = q \cdot m$ for some integer q.

We use the vector gadgets U_i and V_j from Section 3.4. This time, however, we group together every q consecutive gadgets, i.e., $(U_0, \ldots, U_{q-1})$, $(U_q, \ldots, U_{2q-1})$, and so on. Specifically, let $U_i^{[r]}$ be the i-th vector gadget shifted by an integer r (i.e. with r added to all its elements). We define, for each $l \in \{0, 1, \ldots, m-1\}$,

$$\bar{U}_l = U_{lq}^{[2qd]} U_{lq+1}^{[2qd-2d]} \ldots U_{lq+q-1}^{[2d]}.$$

In a similar way, for $j \in \{0, 1, \ldots, m-1\}$, we replicate every V_j gadget q times with appropriate shifts, i.e.,

$$\bar{V}_j = V_j^{[2qd]} V_j^{[2qd-2d]} \ldots V_j^{[2d]}.$$

Let us now determine $\mathrm{lcis}(\bar{U}_l, \bar{V}_j)$. No two gadgets grouped in $\bar{U}_l$ can contribute to an LCIS together, as the later one would have smaller elements. Therefore, only one U_i gadget can be used, paired with the one copy of V_j having the matching shift. This yields $\mathrm{lcis}(\bar{U}_l, \bar{V}_j) = \max_{lq \leqslant i < lq+q} \mathrm{lcis}(U_i, V_j)$, and in turn, also $\max_{l,j} \mathrm{lcis}(\bar{U}_l, \bar{V}_j) = \max_{i,j} \mathrm{lcis}(U_i, V_j) = d - \min_{i,j}(u_i \cdot v_j)$.

Observe that every $\bar{U}_l$ is a concatenation of several U_i gadgets, each one shifted to make its elements smaller than previous ones. Therefore, any increasing subsequence of $\bar{U}_l$ must be contained in a single U_i, and thus cannot be longer than $2d$. The same argument applies to every $\bar{V}_j$. Therefore, we can apply Lemma 15, with $\delta = 2d$, to these sequences, obtaining $\bar{X}$ and $\bar{Y}$ satisfying:

$$\mathrm{lcis}(\bar{X}, \bar{Y}) = C + d - \min_{i,j}(u_i \cdot v_j).$$

Recall that C is some constant dependent only on m and d, and $C = \mathcal{O}(md)$. The length of both $\bar{X}$ and $\bar{Y}$ is $\mathcal{O}(dm \log m + mqd) = \mathcal{O}(nd \log n)$, and the length of the output is $\mathcal{O}(md)$, as desired. □

5 Hardness of k-LCIS

In this section we show that, assuming SETH, there is no $\mathcal{O}\left(n^{k-\varepsilon}\right)$ algorithm for the k-LCIS problem, i.e., we prove Theorem 5. To obtain this lower bound we show a reduction from the k-Orthogonal Vectors problem (for definition, see Section 2). There are two main ingredients of the reduction, i.e. separator sequences and vector gadgets, and both of them can be seen as natural generalizations of those introduced in Section 3.

5.1 Generalizing separator sequences

Please note that in this section we use a notation which is not consistent with the one from Section 3, because it has to accommodate indexing over k sequences.

The aim of this section is to show, for any N that is a power of two, how to construct k sequences $A_1, A_2, \ldots, A_k$ such that each of them can be split into N blocks, i.e. $A_i = \alpha_i^0 \alpha_i^1 \ldots \alpha_i^{N-1}$, and for any choice of $j_1, j_2, \ldots, j_k \in \{0, 1, \ldots, N-1\}$

$$\mathrm{lcis}(\alpha_1^0 \ldots \alpha_1^{j_1}, \alpha_2^0 \ldots \alpha_2^{j_2}, \ldots, \alpha_k^0 \ldots \alpha_k^{j_k}) = j_1 + j_2 + \cdots + j_k + N. \tag{2}$$

As before, we construct separator sequences inductively, doubling the number of blocks in each step. Again, for $N = 1$, we define the sequences by $A_i = \langle 1 \rangle, i \in \{1, \ldots, k\}$.

Suppose we have N-block sequences $A_1, A_2, \ldots, A_k$, $A_i = \alpha_i^0 \alpha_i^1 \ldots \alpha_i^{N-1}$ as above. We show how to construct $2N$-block sequences $B_1, B_2, \ldots, B_k$, $B_i = \beta_i^0 \beta_i^1 \ldots \beta_i^{2N-1}$. Note that inflation properties still hold for k sequences, as the proof of Lemma 12 works in exactly the same way, i.e. inflating all the sequences increases their LCIS by a factor of 2.

To obtain B_i, we first inflate A_i, and then append a tail gadget after each block α_i^j. However, tail gadgets are now more involved.

Let s denote the largest element appearing in $A_1, A_2, \ldots, A_k$. Then the blocks of B_i are

$$\beta_i^{2j} = \mathrm{inflate}(\alpha_i^j) \circ T_i^0, \qquad \beta_i^{2j+1} = T_i^1,$$

where T_i^0 is the sorted sequence of numbers of the form $2s + x$ for $x \in \{1, \ldots, 2^k - 1\}$ such that the i-th bit in the binary representation of x equals 0, while T_i^1 contains those with i-th bit set to 1. Note that for $k = 2$ this exactly leads to the construction from Section 3.

During one construction step, every block doubles its size, and constant number of elements (precisely, $2^k - 1$) is added for every original block. Therefore, the length $L(N)$ of N-block sequences satisfies the recursive equation:

$$L(2N) = 2 \cdot L(N) + (2^k - 1) \cdot N$$

which yields $L(N) = \mathcal{O}(N \log N)$. Note also that the size of the alphabet $S(N)$ used in N-block sequences gives the equation $S(2N) = 2S(N) + 2^k - 1$, as a constant number of elements is added in every step. Therefore $S(N) = \mathcal{O}(N)$.

Lemma 16. *The constructed sequences satisfy, for any $j_1, j_2, \ldots, j_k \in \{0, 1, \ldots, 2N - 1\}$,*

$$\mathrm{lcis}(\beta_1^0 \ldots \beta_1^{j_1}, \beta_2^0 \ldots \beta_2^{j_2}, \ldots, \beta_k^0 \ldots \beta_k^{j_k}) = j_1 + j_2 + \cdots + j_k + 2N,$$

Proof. We prove the claim by induction on $j_1 + j_2 + \cdots + j_k$. In fact, to make the induction work, we need to prove a stronger statement that there always exists a corresponding LCIS that ends on an element less than or equal to $2s + x(j_1, \ldots, j_k)$, where $x(j_1, \ldots, j_k)$ is the integer given by the binary representation $(j_1 \bmod 2, \ldots, j_k \bmod 2)$.

By the inflation properties and the observation that $T_1^0, \ldots, T_k^0$ have no common elements, we obtain the base case $\mathrm{lcis}(\beta_1^0, \ldots, \beta_k^0) = 2 \cdot \mathrm{lcis}(\alpha_1^0, \ldots, \alpha_k^0) = 2N$, with a corresponding LCIS using only elements bounded by $2s$, as desired.

Let $j_1, j_2, \ldots, j_k$ be indices with $j_1 + \cdots + j_k > 0$. Let us first construct a common increasing subsequence of length at least $j_1 + \cdots + j_k + 2N$. If all indices $j_1, \ldots, j_k$ are even, then, for every $i \in \{1, \ldots, k\}$, the prefix $\beta_i^0 \ldots \beta_i^{j_i}$ contains $\mathrm{inflate}(\alpha_i^0 \ldots \alpha_i^{j_i/2})$ as a subsequence. Thus we can find, by inflation properties, a common increasing subsequence of length $2 \cdot (j_1/2 + \cdots + j_k/2 + N) = j_1 + \cdots + j_k + 2N$, as desired. Now, let j_i be any odd index, and let L be the LCIS of the prefixes corresponding to $j_1, \ldots, j_{i-1}, j_i - 1, j_{i+1}, \ldots, j_k$, which ends on an element bounded by $x(j_1, \ldots, j_{i-1}, 0, j_{i+1}, \ldots, j_k)$, of length $j_1 + \cdots + j_k + 2N - 1$ (which exists by the induction hypothesis). Then $L \circ x(j_1, \ldots, j_{i-1}, 1, j_{i+1}, \ldots, j_k)$ is an LCIS for the prefixes corresponding to $j_1, \ldots, j_k$: Indeed, $2s + x(j_1, \ldots, j_k)$ is a common member of $T_1^{j_1 \bmod 2}, \ldots, T_k^{j_k \bmod 2}$, the last parts of these prefixes, and this element is larger and appears later in the sequences than all elements in L (since all T_i^j's are sorted in the increasing order).

For the converse, let L denote the LCIS of $\beta_1^0 \ldots \beta_1^{j_1}$, $\beta_2^0 \ldots \beta_2^{j_2}$, ..., $\beta_k^0 \ldots \beta_k^{j_k}$. Note that if the last symbol of L does not come from the last blocks, i.e. $\beta_1^{j_1}, \beta_2^{j_2}, \ldots, \beta_k^{j_k}$, then L is an LCIS of prefixes corresponding to some $j_1', \ldots, j_k'$ with $j_1' + \cdots + j_k' < j_1 + \cdots + j_k$ and the claim follows from the induction hypotheses. Thus, we may assume that L ends on a common symbol of the last blocks.

If all the indices are even, the last blocks share only elements less than or equal to $2s$ (since $T_1^0, \ldots, T_k^0$ share no elements), thus L is the LCIS of $\mathrm{inflate}(\alpha_i^0, \ldots, \alpha_i^{j_i/2}), i \in \{1, \ldots, k\}$ and the claim follows from the inflation properties. Otherwise, the only element the last blocks have in common is $x(j_1, j_2, \ldots, j_k)$, and thus $L = L' \circ x(j_1, \ldots, j_k)$, where L' is the LCIS of prefixes corresponding to some $j_1', \ldots, j_k'$ with $j_1' + \cdots + j_k' < j_1 + \cdots + j_k$. Thus, $|L| \leqslant j_1' + \cdots + j_k' + 2N + 1 \leqslant j_1 + \cdots + j_k + 2N$, as desired. □

5.2 Generalizing vector gadgets

Each vector gadget is the concatenation of *coordinate gadgets*. Coordinate gadgets for j-th coordinate use elements from the range $\{kj + 1, \ldots, kj + k\}$. If a coordinate is 0, the corresponding gadget contains all k elements sorted in decreasing order, otherwise the gadget for the i-th sequence skips the $kj + i$ element. Formally,

$$\mathrm{VG}_i(u) = \mathrm{CG}_i^0(u[0]) \circ \mathrm{CG}_i^1(u[1]) \circ \cdots \circ \mathrm{CG}_i^{d-1}(u[d-1]),$$

where

$$\mathrm{CG}_i^j(0) = \langle kj + k, kj + (k-1), \ldots, kj + 1 \rangle,$$
$$\mathrm{CG}_i^j(1) = \langle kj + k, kj + (k-1), \ldots, kj + (i+1), kj + (i-1), \ldots, kj + 1 \rangle.$$

Thus, if all k vectors have the j-th coordinate equal 1, there is no common element in the corresponding gadgets. Otherwise, if at least one, say i-th, vector has the j-th coordinate equal 0, the element $kj + i$

appears in all coordinate gadgets. Since the coordinate gadgets are sorted in decreasing order, their LCIS cannot exceed 1. Therefore,

$$\mathrm{lcis}(\mathrm{CG}_1^j(u_1), \mathrm{CG}_2^j(u_2), \ldots, \mathrm{CG}_k^j(u_k)) = 1 - \prod_{i=1}^{k} u_i[j],$$

and ultimately

$$\mathrm{lcis}(\mathrm{VG}_1(u_1), \mathrm{VG}_2(u_2), \ldots, \mathrm{VG}_k(u_k)) = d - \sum_{j=0}^{d-1} \prod_{i=1}^{k} u_i[j].$$

5.3 Putting pieces together

We can finally prove our lower bound for k-LCIS, i.e., Theorem 5.

Proof of Theorem 5. Let $\mathcal{U}_1, \ldots, \mathcal{U}_k \subseteq \{0,1\}^d$ be a k-OV instance with $|\mathcal{U}_i| = n$. By at most doubling the number of vectors in each set, we may assume without loss of generality that n is a power of two.

We construct separator sequences consisting of n blocks. Inflate the sequences $\lceil \log_2 kd \rceil$ times, thus increasing their length by a factor $\ell = 2^{\lceil \log_2 kd \rceil}$, and subtract from all their elements a constant large enough for them to become smaller than all elements of vector gadgets. Let $A_i = \alpha_i^0 \ldots \alpha_i^{n-1}$ denote the thus constructed separator sequence corresponding to set $\mathcal{U}_i$.

Analogously (and as in the proof of Theorem 3), construct, for each $i \in \{1, \ldots, k\}$, the separator sequence $\hat{A}_i = \hat{\alpha}_i^0, \ldots, \hat{\alpha}_i^{n-1}$ by reversing A_i, replacing each element by its additive inverse, and adding a constant large enough to make all the elements larger than vector gadgets (note that each $\hat{\alpha}_i^j$ equals the reverse of α_i^{n-j-1}, with negated elements, shifted by an additive constant). In this way, the analogous property to Equation (2) holds for suffixes instead of prefixes.

Finally, construct sequences $X_1, X_2, \ldots, X_k$ by defining

$$X_i = \alpha_i^0 \mathrm{VG}_i(u_i^0) \hat{\alpha}_i^0 \alpha_i^1 \mathrm{VG}_i(u_i^1) \hat{\alpha}_i^1 \ldots \alpha_i^{n-1} \mathrm{VG}_i(u_i^{n-1}) \hat{\alpha}_i^{n-1},$$

where the VG_i are defined as in Section 5.2. It is straightforward to rework the proof of Theorem 3 to verify that these sequences fulfill

$$\mathrm{lcis}(X_1, X_2, \ldots, X_k) = \ell \cdot (k \cdot (n-1) + 2n) + d - m,$$

where $m = \min_{u_1 \in \mathcal{U}_1, u_2 \in \mathcal{U}_2, \ldots, u_k \in \mathcal{U}_k} \sum_{j=0}^{d-1} \prod_{i=1}^{k} u_i[j]$.

By this reduction, an $\mathcal{O}\left(n^{k-\varepsilon}\right)$ time algorithm for k-LCIS would yield an $\mathcal{O}\left(n^{k-\varepsilon'}\right)$ time k-OV algorithm (for any dimension $d = n^{o(1)}$), thus refuting Hypothesis 9 and, in particular, SETH. □

6 Hardness of k-LCWIS

We shortly discuss the proof of Theorem 6.

Proof sketch of Theorem 6. Note that our lower bound for k-LCIS almost immediately yields a lower bound for k-LCWIS: Trivially, each common increasing subsequence of $X_1, \ldots, X_k$ is also a common *weakly* increasing subsequence. The claim then follows after carefully verifying that, in the constructed sequences, we cannot obtain longer common weakly increasing subsequences by reusing some symbols.

Our claim for k-LCWIS is slightly stronger, however. In particular, we aim to reduce the size of the alphabet over which all the sequences are defined. For this, the key insight is to replace the *inflation* operation $\mathrm{inflate}(\langle a_0, \ldots, a_{n-1} \rangle) = \langle 2a_0 - 1, 2a_0, \ldots, 2a_{n-1} - 1, 2a_{n-1} \rangle$ by

$$\mathrm{inflate}'(\langle a_0, \ldots, a_{n-1} \rangle) = \langle a_0, a_0, \ldots, a_{n-1}, a_{n-1} \rangle,$$

which does not increase the alphabet size, but still satisfies the desired property for k-LCWIS.

Replacing this notion in the proof of Theorem 5, we obtain final sequences $X_1, \ldots, X_k$ by combining separator gadgets over alphabets of size $\mathcal{O}(\log n)$ with vector gadgets over alphabets of size $\mathcal{O}(d)$, where d is the dimension of the vectors in the k-OV instance. Correctness of this construction under k-LCWIS can be verified by reworking the proof of Theorem 5. Thus, we construct hard k-LCWIS instances over an alphabet of size $\mathcal{O}(\log n + d)$, and the claim follows. □

7 Strengthening the Hardness

In this section we show that a natural combination of constructions proposed in the previous sections with the idea of *reachability gadgets* introduced by Abboud et al. [2] lets us strengthen our lower bounds to be derived from considerably weaker assumptions than SETH. Before we do this, we first need to introduce the notion of branching programs.

A *branching program* of width W and length T on N Boolean input variables $x_1, x_2, \ldots, x_N \in \{0,1\}$ is a directed acyclic graph on $W \cdot T$ nodes, arranged into T *layers* of size W each. A node in the k-th layer may have outgoing edges only to the nodes in the $(k+1)$-th layer, and for every layer there is a variable x_i such that every edge leaving this layer is labeled with a constraint of the from $x_i = 0$ or $x_i = 1$. There is a single *start* node in the first layer and a single *accept* node in the last layer. We say that the branching program *accepts* an input $x \in \{0,1\}^N$ if there is a path from the start node to the accept node which uses only edges that are labeled with constraints satisfied by the input x.

The expressive power of branching programs is best illustrated by the theorem of Barrington [12]. It states that any depth-d fan-in-2 Boolean circuit can be expressed as a branching program of width 5 and length 4^d. In particular, NC-circuits can be expressed as constant width quasipolynomial length branching programs.

Given a branching program P on N input variables, the Branching Program Satisfiability problem (BP-SAT) asks if there exists an assignment $x \in \{0,1\}^N$ such that P accepts x. Abboud et al. [2] gave a reduction from BP-SAT to LCS (and some other related problems, such as Edit Distance) on two sequences of length $2^{N/2} \cdot T^{\mathcal{O}(\log W)}$. The reduction proves that a strongly subquadratic algorithm for LCS would imply, among others, exponential improvements over exhaustive search for satisfiability problems not only on CNF formulas (i.e. refuting SETH), but even NC-circuits and circuits representing $o(\sqrt{n})$-space nondeterministic Turing machines. Moreover, even a sufficiently large polylogarithmic improvement would imply nontrivial results in circuit complexity. We refer to the original paper [2] for an in-depth discussion of these consequences.

In this section we prove Theorem 7 and thus show that a subquadratic algorithm for LCIS would have the same consequences. Our reduction from OV to LCIS (presented in Section 3) is built of two ingredients: (1) relatively straightforward vector gadgets, encoding vector inner product in the language of LCIS, and (2) more involved separator sequences, which let us combine many vector gadgets into a single sequence. In order to obtain a reduction from BP-SAT we will need to replace vector gadgets with more complex reachability gadgets. Fortunately, reachability gadgets for LCIS can be constructed in a similar manner as reachability gadgets for LCS proposed in [2].

Proof sketch of Theorem 7. Given a branching program, as in [2], we follow the split-and-list technique of Williams [44]. Assuming for ease of presentation that N is even, we split the input variables into two halves: $x_1, \ldots, x_{N/2}$ and $x_{N/2+1}, \ldots, x_N$. Then, for each possible assignment $a \in \{0,1\}^{N/2}$ of the first half we list a reachability gadget $\mathrm{RG}_{\mathrm{X}}(a)$, and similarly, for each possible assignment $b \in \{0,1\}^{N/2}$ of the second half we list a reachability gadget $\mathrm{RG}_{\mathrm{Y}}(b)$. We shall define the gadgets such that there exists a constant C (depending only on the branching program size) such that $\mathrm{lcis}(\mathrm{RG}_{\mathrm{X}}(a), \mathrm{RG}_{\mathrm{Y}}(b)) = C$ if and only if $a \circ b$ is an assignment accepted by the branching program, and otherwise $\mathrm{lcis}(\mathrm{RG}_{\mathrm{X}}(a), \mathrm{RG}_{\mathrm{Y}}(b)) < C$. The reduction is finished by applying Lemma 15 to the constructed gadgets in order to obtain two sequences such that their LCIS lets us determine whether a satisfying assignment to the branching program exists. The rest of the proof is devoted to constructing suitable reachability gadgets.

We assume without loss of generality that $T = 2^t + 1$ for some integer t. For every $k \in \{0, 1, \ldots, t\}$ and for every two nodes u, v being 2^k layers apart from each other we want to construct two reachability gadgets $\mathrm{RG}_{\mathrm{X}}^{u \to v}$ and $\mathrm{RG}_{\mathrm{Y}}^{u \to v}$ such that, for some constant C_k,

$$\mathrm{lcis}(\mathrm{RG}_{\mathrm{X}}^{u \to v}(a), \mathrm{RG}_{\mathrm{Y}}^{u \to v}(b)) \begin{cases} = C_k & \text{if there is a path from } u \text{ to } v \text{ satisfied by } a \circ b, \\ < C_k & \text{otherwise,} \end{cases}$$

holds for all $a, b \in \{0,1\}^{N/2}$.

Consider $k = 0$, i.e., designing reachability gadgets for nodes in neighboring layers L_j and L_{j+1}. There is a variable x_i such that all edges between L_j and L_{j+1} are labeled with a constraint $x_i = 0$ or $x_i = 1$. We say *the left half is responsible for* x_i if x_i is among the first half $x_1, \ldots, x_{N/2}$ of variables; otherwise, we say the *right half is responsible for* x_i. We set $\mathrm{RG}_{\mathrm{X}}^{u \to v}(a)$ to be an empty sequence if the left half is

responsible for x_i and there is no edge from u to v labeled $x_i = a_i$; otherwise, we set $\mathrm{RG}_{\mathrm{X}}^{u\to v}(a) = \langle 0\rangle$. Similarly, $\mathrm{RG}_{\mathrm{Y}}^{u\to v}(b)$ is an empty sequence if the right half is responsible and there is no edge from u to v labeled with $x_i = b_{i-N/2}$; otherwise $\mathrm{RG}_{\mathrm{Y}}^{u\to v}(b) = \langle 0\rangle$. It is easy to verify that such reachability gadgets satisfy the desired property for $C_0 = 1$.

For $k > 0$, let $w_1, w_2, \dots w_W$ be the nodes in the layer exactly halfway between u and v. Observe that there exists a path from u to v if and only if there exists a path from u to w_i and from w_i to v for some $i \in \{1, 2, \dots, W\}$.

Let $\overline{\mathrm{RG}}_{\mathrm{X}}^{w_i\to v}$ and $\overline{\mathrm{RG}}_{\mathrm{Y}}^{w_i\to v}$ denote the sequences $\mathrm{RG}_{\mathrm{X}}^{w_i\to v}$ and $\mathrm{RG}_{\mathrm{Y}}^{w_i\to v}$ with every element increased by a constant large enough so that all elements are larger than all elements of $\mathrm{RG}_{\mathrm{X}}^{u\to w_i}$ and $\mathrm{RG}_{\mathrm{Y}}^{u\to w_i}$. Observe that $\mathrm{lcis}(\mathrm{RG}_{\mathrm{X}}^{u\to w_i}(a) \circ \overline{\mathrm{RG}}_{\mathrm{X}}^{w_i\to v}(a), \mathrm{RG}_{\mathrm{Y}}^{u\to w_i}(b) \circ \overline{\mathrm{RG}}_{\mathrm{Y}}^{w_i\to v}(b))$ equals $2 \cdot C_{k-1}$ if there is a path $u \rightsquigarrow w_i \rightsquigarrow v$ satisfied by $a \circ b$, and otherwise it is less than $2 \cdot C_{k-1}$. Now, for every i take a different constant q_i and add it to both $\mathrm{RG}_{\mathrm{X}}^{u\to w_i} \circ \overline{\mathrm{RG}}_{\mathrm{X}}^{w_i\to v}$ and $\mathrm{RG}_{\mathrm{Y}}^{u\to w_i} \circ \overline{\mathrm{RG}}_{\mathrm{Y}}^{w_i\to v}$ so that their alphabets are disjoint, and therefore, for $i \neq j$, $\mathrm{lcis}((\mathrm{RG}_{\mathrm{X}}^{u\to w_i}(a) \circ \overline{\mathrm{RG}}_{\mathrm{X}}^{w_i\to v}(a)) + q_i, (\mathrm{RG}_{\mathrm{Y}}^{u\to w_j}(b) \circ \overline{\mathrm{RG}}_{\mathrm{Y}}^{w_j\to v}(b)) + q_j) = 0$ (where $+$ denotes element-wise addition). Finally, apply Lemma 15 to these W pairs of concatenated reachability gadgets (where we choose δ as the maximum length of these gadget) to obtain two reachability gadgets $\mathrm{RG}_{\mathrm{X}}^{u\to v}$ and $\mathrm{RG}_{\mathrm{Y}}^{u\to v}$ such that $\mathrm{lcis}(\mathrm{RG}_{\mathrm{X}}^{u\to v}(a), \mathrm{RG}_{\mathrm{Y}}^{u\to v}(b))$ equals $C + 2 \cdot C_{k-1}$ (for a constant C resulting from the application of Lemma 15) if there exists (for some $i \in \{1, 2, \dots, W\}$) a path $u \rightsquigarrow w_i \rightsquigarrow v$ satisfied by $a \circ b$, and is strictly smaller otherwise, as desired.

Let u_{start} and u_{accept} denote the start node and the accept node of the branching program. Then, $\mathrm{RG}_{\mathrm{X}} = \mathrm{RG}_{\mathrm{X}}^{u_{\mathrm{start}}\to u_{\mathrm{accept}}}$ and $\mathrm{RG}_{\mathrm{Y}} = \mathrm{RG}_{\mathrm{Y}}^{u_{\mathrm{start}}\to u_{\mathrm{accept}}}$ satisfy the property that

$$\mathrm{lcis}(\mathrm{RG}_{\mathrm{X}}(a), \mathrm{RG}_{\mathrm{Y}}(b)) \begin{cases} = C_t & \text{if the branching program accepts } a \circ b, \\ < C_t & \text{otherwise.} \end{cases}$$

Since $\mathrm{RG}_{\mathrm{X}}(a)$ and $\mathrm{RG}_{\mathrm{Y}}(b)$ are constructed in t steps of the inductive construction, and each step increases the length of gadgets by a factor of $\mathcal{O}(W \log W)$, their final length can be bounded by $\mathcal{O}((W \log W)^t)$, which is $T^{\mathcal{O}(\log W)}$. Combining the reachability gadgets $\mathrm{RG}_{\mathrm{X}}(a)$, $a \in \{0,1\}^{N/2}$ and $\mathrm{RG}_{\mathrm{Y}}(b)$, $b \in \{0,1\}^{N/2}$ using Lemma 15 (where we choose δ as the maximum length of the reachability gadgets) yields the desired strings X, Y of length $2^{N/2} \cdot N \cdot T^{\mathcal{O}(\log W)}$ whose LCIS lets us determine satisfiability of the given branching program, thus finishing the proof. □

Similar techniques can be used to analogously strengthen other lower bounds in our paper.

8 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 [36]. Note, however, that our reduction from BP-SAT gives an evidence that shaving many logarithmic factors is immensely difficult. Finally, 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 ([40] and Theorem 6). On the other hand, for binary and ternary alphabets, even linear time algorithms exist [35, 24]. 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 $\mathcal{O}\left(n^{3/2}\varepsilon^{-1/2}\mathrm{polylog}(n)\right)$ time by an easy observation (see the appendix). 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)*, 2016. To appear.

[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] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

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

[14] 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.

[15] 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.

[16] 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.

[17] Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. 2017. Unpublished Manuscript.

[18] 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.

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

[20] 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. *Information Processing Letters*, 90(4):175–179, 2004.

[21] 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.

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

[23] 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)*, 2017. To appear.

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

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

[26] 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.

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

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

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

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

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

[32] 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.

[33] 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.

[34] 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)*, 2017. To appear.

[35] 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.

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

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

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

[39] 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.

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

[41] 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.

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

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

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

[45] 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.

[46] 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.

[47] 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.

Appendix

Theorem 17 (folklore, generalization of [47])**.** *For any $k \geqslant 2$, LCIS of k sequences of length n can be computed in $\mathcal{O}\left(n^k\right)$ time.*

Proof. Let $X_1, X_2, \ldots X_k$ be the input sequences. Let $X[0:i]$ denote the prefix consisting of first i elements of X, with $X[0:0]$ being the empty prefix. Now, for every $i_1, \ldots, i_k \in \{0, 1, \ldots, n\}$ we define $R[i_1, \ldots, i_k]$ to be the length of the LCIS of the prefixes $X_1[0:i_1], X_2[0:i_1], \ldots, X_k[0:i_k]$ with an additional assumption that this common subsequence must end with the element $X_k[i_k]$, i.e. the last element of the last prefix. Observe that it is enough to compute all $R[i_1, \ldots, i_k]$, with the desired answer being simply $\max_{0 \leqslant i_k \leqslant n} R[n, \ldots, n, i_k]$.

The algorithm is based on the fact that $R[i_1, \ldots, i_k]$ satisfies the following two-case recurrence:

- **Case 1.** If $X_k[i_k] \neq X_s[i_s]$ for some $s < k$, the desired common subsequence ends with $X_k[i_k]$ and thus it cannot contain $X_s[i_s]$, so $R[i_1, \ldots, i_s, \ldots, i_k] = R[i_1, \ldots, i_s - 1, \ldots, i_k]$.
- **Case 2.** If $X_1[i_1] = \ldots = X_k[i_k]$, let us call this common symbol σ, and observe that σ is the last element of LCIS. Consider the next-to-last element: it must be certainly smaller than σ, and must appear in the X_k sequence at a position earlier than at $X_k[i_k]$. Therefore $R[i_1, \ldots, i_k] = 1 + \max_{j < i_k, X_k[j] < \sigma} R[i_1 - 1, \ldots, i_{k-1} - 1, j]$.

To obtain the values of R in $\mathcal{O}\left(n^k\right)$ time, the algorithm iterates through all possible $i_1, \ldots, i_k$ with the i_k loop being the innermost one. Obviously, $R[i_1, \ldots, i_k] = 0$ if any of the indices is 0. Before every innermost i_k loop, with fixed $i_1, i_2, \ldots, i_{k-1}$, the algorithm checks whether $X_1[i_1] = \ldots = X_{k-1}[i_{k-1}]$. If so, it sets $\sigma = X_1[i_1] = \ldots = X_{k-1}[i_{k-1}]$, otherwise $\sigma = \text{null}$.

If $\sigma \neq \text{null}$, for every $1 \leqslant i \leqslant n$ let $D[i] = \max_{j < i, X_k[j] < \sigma} R[i_1 - 1, \ldots, i_{k-1} - 1, j]$. Observe that $D[i]$ can be obtained from $D[i-1]$ and $R[i_1 - 1, \ldots, i_{k-1} - 1, i - 1]$ in constant time. Therefore, before the start of the i_k loop, the algorithm can precompute all the $D[i]$ values in $\mathcal{O}(n)$ time, as all the needed $R[i_1 - 1, \ldots, i_{k-1} - 1, i]$ values are already known from earlier iterations.

Throughout the i_k loop the algorithm checks if $X_k[i_k] = \sigma$, which corresponds to Case 2 above. If so, then $R[i_1, \ldots, i_k] = 1 + \max_{j < i_k, X_k[j] < \sigma} R[i_1 - 1, \ldots, i_{k-1} - 1, j] = 1 + D[i_k]$, which is already precomputed. If Case 1 holds, then $R[i_1, \ldots, i_s, \ldots, i_k] = R[i_1, \ldots, i_s - 1, \ldots, i_k]$ for some $s < k$. As the index s is easy to find, and the necessary values in R have been computed earlier, this step also works in constant time (assuming k is fixed).

The above algorithm computes only the length of LCIS. However, it can be easily modified to reconstruct the sequence, using the common dynamic programming techniques (e.g. by storing with every value in R a link to the previous element of LCIS). □

Theorem 18. *A $(1+\varepsilon)$-approximation of LCIS of sequences X, Y of length n can be computed in $\mathcal{O}\left(n^{3/2}\varepsilon^{-1/2}\text{polylog}(n)\right)$ time.*

Proof. First, delete all integers occurring more than $2\sqrt{n/\varepsilon}$ times in total in both of the sequences. Since there are at most $\sqrt{n\varepsilon}$ such integers, this operation decreases the length of the LCIS by at most $\sqrt{n\varepsilon}$. In the resulting instance, there are at most $n^{3/2}\varepsilon^{-1/2}$ matching pairs, i.e., indices i, j with $X[i] = Y[j]$. Thus, the exact LCIS in this instance can be computed in time $\mathcal{O}\left(n^{3/2}\varepsilon^{-1/2}\log n \log\log n\right)$ using an algorithm of Chan et al. [18] running in time $\mathcal{O}(M \log L \log\log n + n \log n)$, where L is the length of the LCIS of X and Y and M is the number of matching pairs. Now, consider two cases. If the algorithm returns a solution Z longer than $\sqrt{n/\varepsilon}$, then Z is a $(1+\varepsilon)$-approximation of the LCIS of the original instance, since the LCIS is bounded by $L \leqslant |Z| + \sqrt{n\varepsilon} \leqslant (1+\varepsilon)|Z|$. In the remaining case, it is guaranteed that $L \leqslant |Z| + \sqrt{n\varepsilon} \leqslant (1+\varepsilon)\sqrt{n/\varepsilon}$. Thus, we may compute the exact LCIS in $\mathcal{O}\left(n^{3/2}\varepsilon^{-1/2}\log n\right)$ time using the algorithm running in $\mathcal{O}(nL \log\log n + n \log n)$ time [35]. □