Multivariate Fine-Grained Complexity of Longest Common Subsequence*

Karl Bringmann† Marvin Künnemann†

Abstract

We revisit the classic combinatorial pattern matching problem of finding a longest common subsequence (LCS). For strings x and y of length n, a textbook algorithm solves LCS in time $O(n^2)$, but although much effort has been spent, no $\mathcal{O}(n^{2-\varepsilon})$-time algorithm is known. Recent work indeed shows that such an algorithm would refute the Strong Exponential Time Hypothesis (SETH) [Abboud, Backurs, Vassilevska Williams FOCS'15; Bringmann, Künnemann FOCS'15].

Despite the quadratic-time barrier, for over 40 years an enduring scientific interest continued to produce fast algorithms for LCS and its variations. Particular attention was put into identifying and exploiting input parameters that yield strongly subquadratic time algorithms for special cases of interest, e.g., differential file comparison. This line of research was successfully pursued until 1990, at which time significant improvements came to a halt. In this paper, using the lens of fine-grained complexity, our goal is to (1) justify the lack of further improvements and (2) determine whether some special cases of LCS admit faster algorithms than currently known.

To this end, we provide a systematic study of the *multivariate complexity* of LCS, taking into account all parameters previously discussed in the literature: the input size $n := \max\{|x|, |y|\}$, the length of the shorter string $m := \min\{|x|, |y|\}$, the length L of an LCS of x and y, the numbers of deletions $\delta := m - L$ and $\Delta := n - L$, the alphabet size, as well as the numbers of matching pairs M and dominant pairs d. For any class of instances defined by fixing each parameter individually to a polynomial in terms of the input size, we prove a SETH-based lower bound matching one of three known algorithms (up to lower order factors of the form $n^{o(1)}$). Specifically, we determine the optimal running time for LCS under SETH as $(n + \min\{d, \delta\Delta, \delta m\})^{1\pm o(1)}$. Polynomial improvements over this running time must necessarily refute SETH or exploit novel input parameters. We establish the same lower bound for any constant alphabet of size at least 3. For binary alphabet, we show a SETH-based lower bound of $(n + \min\{d, \delta\Delta, \delta M/n\})^{1-o(1)}$ and, motivated by difficulties to improve this lower bound, we design an $\mathcal{O}(n+\delta M/n)$-time algorithm, yielding again a matching bound.

We feel that our systematic approach yields a comprehensive perspective on the well-studied multivariate complexity of LCS, and we hope to inspire similar studies of multivariate complexity landscapes for further polynomial-time problems.

*Part of this work was done while visiting the Simons Institute for the Theory of Computing at University of California, Berkeley.

†Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany, (kbringma|marvin)@mpi-inf.mpg.de.

1 Introduction

String comparison is one of the central tasks in combinatorial pattern matching, with various applications such as spelling correction [67, 81], DNA sequence comparison [8], and differential file comparison [45, 65]. Perhaps the best-known measure of string similarity is the length of the longest common subsequence (LCS). A textbook dynamic programming algorithm computes the LCS of given strings x, y of length n in time $\mathcal{O}(n^2)$, and in the worst case only an improvement by logarithmic factors is known [64]. In fact, recent results show that improvements by polynomial factors would refute the Strong Exponential Time Hypothesis (SETH) [1, 28] (see Section 2.2 for a definition).

Despite the quadratic-time barrier, the literature on LCS has been steadily growing, with a changing focus on different aspects of the problem over time (see Section 1.2 for an overview). Spurred by an interest in practical applications, a particular focus has been the design of LCS algorithms for strings that exhibit certain structural properties. This is most prominently witnessed by the UNIX `diff` utility, which quickly compares large, similar files by solving an underlying LCS problem. A practically satisfying solution to this special case was enabled by theoretical advances exploiting the fact that in such instances the LCS differs from the input strings at only few positions (e.g., [65, 69]). In fact, since Wagner and Fischer introduced the LCS problem in 1974 [81], identifying and exploiting structural parameters to obtain faster algorithms has been a decades-long effort [13, 14, 36, 44, 46, 49, 69, 70, 84].

Parameters that are studied in the literature are, besides the input size $n := \max\{|x|, |y|\}$, the length $m := \min\{|x|, |y|\}$ of the shorter string, the size of the alphabet Σ that x and y are defined on, the length L of a longest common subsequence of x and y, the number $\Delta = n - L$ of deleted symbols in the longer string, the number $\delta = m - L$ of deleted symbols in the shorter string, the number of *matching* pairs M, and the number of *dominant* pairs d (see Section 2.1 for definitions). Among the fastest currently known algorithms are an $\tilde{\mathcal{O}}(n + \delta L)$-algorithm due to Hirschberg [44], an $\tilde{\mathcal{O}}(n + \delta\Delta)$-algorithm due to Wu, Manbers, Myers, and Miller [84], and an $\tilde{\mathcal{O}}(n + d)$-algorithm due to Apos-

tolico [13] (with log-factor improvements by Eppstein, Galil, Giancarlo, and Italiano [36]). In the remainder, we refer to such algorithms, whose running time is stated in more parameters than just the problem size n, as *multivariate algorithms*. See Table 1 on the facing page for a non-exhaustive survey containing the asymptotically fastest multivariate LCS algorithms.

The main question we aim to answer in this work is: *Are there significantly faster multivariate LCS algorithms than currently known?* E.g., can ideas underlying the fastest known algorithms be combined to design an algorithm that is much faster than all of them?

1.1 Our Approach and Informal Results We systematically study special cases of LCS that arise from polynomial restrictions of any of the previously studied input parameters. Informally, we define a *parameter setting* (or polynomial restriction of the parameters) as the subset of all LCS instances where each input parameter is individually bound to a polynomial relation with the input size n, i.e., for each parameter p we fix a constant α_p and restrict the instances such that p attains a value $\Theta(n^{\alpha_p})$. An algorithm for a specific parameter setting of LCS receives as input two strings x, y guaranteed to satisfy the parameter setting and outputs (the length of) an LCS of x and y. We call a parameter setting trivial if it is satisfied by at most a finite number of instances; this happens if the restrictions on different parameters are contradictory. For each nontrivial parameter setting, we construct a family of hard instances via a reduction from satisfiability, thus obtaining a conditional lower bound. This greatly extends the construction of hard instances for the $n^{2-o(1)}$ lower bound [1, 28].

Results for large alphabets Since we only consider exact algorithms, any algorithm for LCS takes time $\Omega(n)$. Beyond this trivial bound, for any nontrivial parameter setting we obtain a SETH-based lower bound of

$$\min\{d,\, \delta\Delta,\, \delta m\}^{1-o(1)}.$$

Note that this bound is matched by the known algorithms with running times $\tilde{\mathcal{O}}(n+d)$, $\tilde{\mathcal{O}}(n+\delta L)$[1], and $\tilde{\mathcal{O}}(n+\delta\Delta)$. Thus, our lower bound very well explains the lack of progress since the discovery of these three algorithms (apart from lower-order factors).

[1]Note that $L \leq m$. At first sight it might seem as if the $\tilde{\mathcal{O}}(n+\delta L)$ algorithm could be faster than our lower bound, however, for $L \geq m/2$ we have $\delta L = \Theta(\delta m)$, which appears in our lower bound, and for $L \leq m/2$ we have $\delta = m - L = \Theta(m)$ and thus $\delta L = \Theta(Lm)$ which is $\Omega(d)$ by the relation $d \leq Lm$ (see Table 3), and d appears in our lower bound.

Results for constant alphabet size For the alphabet size $|\Sigma|$, we do not only consider the case of a polynomial relation with n, but also the important special cases of $|\Sigma|$ being any fixed constant. We show that our conditional lower bound for polynomial alphabet size also holds for any constant $|\Sigma| \geq 3$. For $|\Sigma| = 2$, we instead obtain a SETH-based lower bound of

$$\min\{d,\, \delta\Delta,\, \delta M/n\}^{1-o(1)}.$$

This lower bound is weaker than the lower bound for $|\Sigma| \geq 3$ (as the term $\delta M/n$ is at most δm by the trivial bound $M \leq mn$; see Section 2.1 for the definition of M). Surprisingly, a stronger lower bound is impossible (assuming SETH): Motivated by the difficulties to obtain the same lower bound as for $|\Sigma| \geq 3$, we discovered an algorithm with running time $\mathcal{O}(n + \delta M/n)$ for $|\Sigma| = 2$, thus matching our conditional lower bound. To the best of our knowledge, this algorithm provides the first polynomial improvement for a special case of LCS since 1990, so while its practical relevance is unclear, we succeeded in uncovering a tractable special case. Interestingly, our algorithm and lower bounds show that the multivariate fine-grained complexity of LCS differs polynomially between $|\Sigma| = 2$ and $|\Sigma| \geq 3$. So far, the running time of the fastest known algorithms for varying alphabet size differed at most by a logarithmic factor in $|\Sigma|$.

We find it surprising that the hardness assumption SETH is not only sufficient to prove a worst-case quadratic lower bound for LCS, but extends to the *complete spectrum* of multivariate algorithms using the previously used 7 parameters, thus proving an optimal running time bound which was implicitly discovered by the computer science community within the first 25 years of research on LCS (except for the case of $\Sigma = \{0, 1\}$, for which we provide a missing algorithm).

1.2 Related Work on LCS Table 1 on the next page gives a non-comprehensive overview of progress on multivariate LCS, including the asymptotically fastest known algorithms. Note that the most recent polynomial factor improvement for multivariate LCS was found in 1990 [84]. Further progress on multivariate LCS was confined to log-factor improvements (e.g., [36, 49]). Therefore, the majority of later works on LCS focused on transferring the early successes and techniques to more complicated problems, such as longest common increasing subsequence [32, 57, 66, 85], tree LCS [68], and many more generalizations and variants of LCS, see, e.g., [6, 7, 9, 10, 20, 21, 24, 31, 35, 37, 40, 42, 47, 48, 52–54, 56, 59–61, 72, 75, 76, 78, 82]. One branch of generalizations considered the LCS of more than two strings (e.g., [1, 25]), with variations such as string con-

Reference	Running Time
Wagner and Fischer [81]	$\mathcal{O}(mn)$
Hunt and Szymanski [46]	$\mathcal{O}((n+M)\log n)$
Hirschberg [44]	$\mathcal{O}(n\log n + Ln)$
Hirschberg [44]	$\mathcal{O}(n\log n + L\delta\log n)$
Masek and Paterson [64]	$\mathcal{O}(n + nm/\log^2 n)$ assuming $\lvert\Sigma\rvert = \mathcal{O}(1)$
	$\mathcal{O}\big(n + nm\cdot\big(\frac{\log\log n}{\log n}\big)^2\big)$ [2]
Nakatsu, Kambayashi and Yajima [70]	$\mathcal{O}(n\delta)$
Apostolico [13]	$\mathcal{O}(n\log n + d\log(mn/d))$
Myers [69]	$\mathcal{O}(n\log n + \Delta^2)$
Apostolico and Guerra [14]	$\mathcal{O}(n\log n + Lm\min\{\log m, \log(n/m)\})$
Wu, Manbers, Myers and Miller [84]	$\mathcal{O}(n\log n + \delta\Delta)$ [3]
Eppstein, Galil, Giancarlo and Italiano [36]	$\mathcal{O}(n\log n + d\log\log\min\{d, nm/d\})$
Iliopoulos and Rahman [49]	$\mathcal{O}(n + M\log\log n)$

Table 1: Short survey of LCS algorithms. See Section 2.1 for definitions of the parameters. When stating the running times, every factor possibly attaining non-positive values (such as $\delta, \log(n/m)$, etc.) is to be read as $\max\{\cdot, 1\}$. For simplicity, $\log(\Sigma)$-factors have been bounded from above by $\log n$ (see [71] for details on the case of constant alphabet size).

sensus (e.g., [11, 12]) and more (e.g., [9, 19, 34, 40, 41, 60]). Since natural language texts are well compressible, researchers also considered solving LCS directly on compressed strings, using either run-length encoding (e.g., [15, 30, 33, 56]) or straight-line programs and other Lempel-Ziv-like compression schemes (e.g., [39, 43, 62, 77]). Further research directions include approximation algorithms for LCS and its variants (e.g., [41, 42, 58]), as well as the LCS length of random strings [18, 63].

For brevity, here we ignore the equally vast literature on the closely related edit distance. Furthermore, we solely regard the time complexity of computing the length of an LCS and hence omit all results concerning space usage or finding an LCS. See, e.g., [22, 71] for these and other aspects of LCS (including empirical evaluations).

1.3 (Multivariate) Hardness in P After the early success of 3SUM-hardness in computational geometry [38], recent years have brought a wealth of novel conditional lower bounds for polynomial time problems, see, e.g., [1–4, 16, 17, 26–29, 55, 73, 80, 83] and the recent survey [79]. In particular, our work extends the recent successful line of research proving SETH-based lower bounds for a number problems with efficient dynamic programming solutions such as Fréchet distance [26, 29], edit distance [16, 28], LCS and dynamic time warping [1, 28]. Beyond worst-case conditional lower bounds of the form $n^{c-o(1)}$, recently also more detailed lower bounds targeting additional input restrictions have gained interest. Such results come in different flavors, as follows.

Input parameters, polynomial dependence. Consider one or more parameters in addition to the input size n, where the optimal time complexity of the studied problem depends polynomially on n and the parameters. This is the situation in this paper as well as several previous studies, e.g., [4, 26, 55]. To the best of our knowledge, our work is the first of this kind to study combinations of more than two parameters that adhere to a complex set of parameter relations – for previous results, typically the set of non-trivial parameter settings was obvious and simultaneously controlling all parameters was less complex.

Input parameters, superpolynomial dependence. Related to the above setting, parameters have been studied where the time complexity depends polynomially on n and *superpolynomially on the parameters*. If the studied problem is NP-hard then this is known as fixed-parameter tractability (FPT). However, here we focus on problems in P, in which case this situation is known as "FPT in P". Hardness results in this area were initiated by Abboud, Vassilevska Williams, and Wang [4].

A finite/discrete number of special cases. Some input restrictions yield a discrete or even finite set of special cases. For example, Backurs and Indyk [17]

[2] See [23] for how to extend the Masek-Paterson algorithm to non-constant alphabets.

[3] Wu et al. state their running time as $\mathcal{O}(n\delta)$ in the worst case and $\mathcal{O}(n+\delta\Delta)$ in expectation for random strings. However, Myers worst-case variation trick [69, Section 4c] applies and yields the claimed time bound $\mathcal{O}(n\log n + \delta\Delta)$. The additional $\mathcal{O}(n\log n)$ comes from building a suffix tree.

and later Bringmann et al. [27] studied special cases of regular expression pattern matching by restricting the input to certain "types" of regular expressions. The set of types is discrete and infinite, however, there are only finitely many tractable types, and finitely many minimal hardness results. Their approach is similarly systematic to ours, as they classify the complexity of pattern matching for any type of regular expressions. The major difference is that our parameters are "continuous", specifically our parameter exponents α_p are continuous, and thus our algorithms and lower bounds trace a continuous tradeoff.

While in all of the above settings the design of fast multivariate algorithms is well established, tools for proving matching conditional lower bounds have been developed only recently. In particular, the systematic approach to multivariate lower bounds pursued in this paper provides an effective complement to multivariate algorithmic studies in P, since it establishes (near-)optimality and may uncover tractable special cases for which improved algorithms can be found.

Beyond SETH Motivated in part to find barriers even for polylogarithmic improvements on LCS, a surprising result of Abboud et al. [2] strengthens the conditional quadratic-time hardness of LCS substantially. More precisely, they show that a strongly subquadratic-time algorithm for LCS would even refute a natural, weaker variant of SETH on *branching programs*. In the full version of this article, we survey their result and show that the conditional lower bounds we derive in this paper also hold under this weaker assumption.

2 Preliminaries

We write $[n] := \{1, \ldots, n\}$. For a string x, we denote its length by $|x|$, the symbol at its i-th position by $x[i]$, and the substring from position i to position j by $x[i..j]$. If string x is defined over alphabet Σ, we denote the number of occurrences of symbol $\sigma \in \Sigma$ in x by $\#_\sigma(x)$. In running time bounds we write Σ instead of $|\Sigma|$ for readability. For two strings x, y, we denote their concatenation by $x \circ y = xy$ and define, for any $\ell \geq 0$, the ℓ-fold repetition $x^\ell := \bigcirc_{i=1}^{\ell} x$. For any strings x, y we let $\mathrm{LCS}(x, y)$ be any longest common subsequence of x and y, i.e., a string $z = z[1..L]$ of maximum length L such that there are $i_1 < \ldots < i_L$ with $x[i_k] = z[k]$ for all $1 \leq k \leq L$ and there are $j_1 < \ldots < j_L$ with $y[j_k] = z[k]$ for all $1 \leq k \leq L$. For a string x of length n, let $\mathrm{rev}(x) := x[n]\, x[n-1] \ldots x[1]$ denote its reverse.

2.1 Parameter Definitions

We survey parameters that have been used in the analysis of the LCS problem (see also [22, 71]). Let x, y be any strings. By possibly swapping x and y, we can assume that x is the longer of the two strings, so that $n = n(x, y) := |x|$ is the input size (up to a factor of two). Then $m = m(x, y) := |y|$ is the length of the shorter of the two strings. Another natural parameter is the solution size, i.e., the length of any LCS, $L = L(x, y) := |\mathrm{LCS}(x, y)|$.

Since any symbol not contained in x or in y cannot be contained in a LCS, we can ensure the following using a (near-)linear-time preprocessing.

ASSUMPTION 2.1. *Every symbol $\sigma \in \Sigma$ occurs at least once in x and in y, i.e., $\#_\sigma(x), \#_\sigma(y) \geq 1$.*

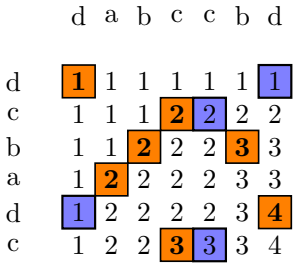
Consider the alphabet induced by x and y after ensuring Assumption 2.1, namely $\Sigma = \{x[i] \mid 1 \leq i \leq |x|\} \cap \{y[j] \mid 1 \leq j \leq |y|\}$. Its size $\Sigma(x, y) := |\Sigma|$ is a natural parameter.

Beyond these standard parameters $n, m, L, |\Sigma|$ (applicable for any optimization problem on strings), popular structural parameters measure the *similarity* and *sparsity* of the strings. These notions are more specific to LCS and are especially relevant in practical applications such as, e.g., the `diff` file comparison utility, where symbols in x and y correspond to lines in the input files.

Notions of similarity To obtain an LCS, we have to delete $\Delta = \Delta(x, y) := n - L$ symbols from x or $\delta = \delta(x, y) := m - L$ symbols from y. Hence for very similar strings, which is the typical kind of input for file comparisons, we expect δ and Δ to be small. This is exploited by algorithms running in time, e.g., $\tilde{\mathcal{O}}(n + \delta\Delta)$ [84] or $\tilde{\mathcal{O}}(n + \delta L)$ [44].

Notions of sparsity Based on the observation that the dynamic programming table typically stores a large amount of redundant information (suggested, e.g., by the fact that an LCS itself can be reconstructed examining only $O(n)$ entries), algorithms have been studied that consider only the most relevant entries in the table. The simplest measure of such entries is the number of *matching pairs* $M = M(x, y) := \#\{(i, j) \mid x[i] = y[j]\}$. Especially for inputs with a large alphabet, this parameter potentially significantly restricts the number of candidate pairs considered by LCS algorithms, e.g., for files where almost all lines occur only once. Moreover, in the special case where x and y are permutations of Σ we have $M = n = m$, and thus algorithms in time $\tilde{\mathcal{O}}(n+M)$ [45, 46, 49] recover the near-linear time solution for LCS of permutations [74].

One can refine this notion to obtain the *dominant pairs*. A pair (i, j) *dominates* a pair (i', j') if we have $i \leq i'$ and $j \leq j'$. A *k-dominant pair* is a pair (i, j) such that $L(x[1..i], y[1..j]) = k$ and no other pair (i', j') with $L(x[1..i'], y[1..j']) = k$ dominates (i, j). By defining $L[i, j] := L(x[1..i], y[1..j])$ and using the well known recursion $L[i, j] = \max\{L[i-1, j], L[i, j-1], L[i-1, j-$



(a)

$$
\begin{aligned}
n(x,y) &:= |x|, \quad m(x,y) := |y| \\
L(x,y) &:= |\mathrm{LCS}(x,y)| \\
\delta(x,y) &:= |y| - L(x,y), \quad \Delta(x,y) := |x| - L(x,y) \\
\Sigma(x,y) &:= \#(\{x[i] \mid 1 \le i \le |x|\} \cap \{y[j] \mid 1 \le j \le |y|\}) \\
M(x,y) &:= \#\{(i,j) \mid x[i] = y[j]\} \\
d(x,y) &:= \#\{(i,j) \mid L[i,j] > L[i-1,j] \text{ and } L[i,j] > L[i,j-1]\}, \\
&\qquad \text{where } L[i,j] := |\mathrm{LCS}(x[1..i], y[1..j])|.
\end{aligned}
$$

(b)

Figure 1: (a) Illustration of the L-table, matching pairs and dominant pairs. Entries marked in orange color and bold letters correspond to dominant pairs (which by definition are also matching pairs), while entries marked in blue are matching pairs only. (b) Summary of all input parameters.

$1]+\mathbf{1}_{x[i]=y[j]}\}$, we observe that (i,j) is a k-dominant pair if and only if $L[i,j] = k$ and $L[i-1,j] = L[i,j-1] = k-1$. Denoting the set of all k-dominant pairs by D_k, the set of *dominant pairs* of x, y is $\bigcup_{k\ge 1} D_k$, and we let $d = d(x,y)$ denote the number of dominant pairs. Algorithms running in time $\tilde{\mathcal{O}}(n+d)$ exploit a small number of dominant pairs [13, 36]. Figure 1a illustrates matching and dominant pairs.

While at first sight the definition of dominant pairs might not seem like the most natural parameter, it plays an important role in analyzing LCS: First, from the set of dominant pairs alone one can reconstruct the L-table that underlies the basic dynamic programming algorithm. Second, the parameter d precisely describes the complexity of one of the fastest known (multivariate) algorithms for LCS. Finally, LCS with parameter d is one of the first instances of the paradigm of *sparse dynamic programming* (see, e.g., [36]).

On practical instances, exploiting similarity notions seems to typically outperform algorithms based on sparsity measures (see [65] for a classical comparison to an algorithm based on the number of matching pairs M [45, 46]). To the best of our knowledge, Figure 1b summarizes all parameters which have been exploited to obtain multivariate algorithms for LCS.

We remark that for some intermediate strings x, y constructed in the proofs, the assumption $|x| \ge |y|$ may be violated; in this case we use the definitions given in Figure 1b (and thus we may have $n(x,y) < m(x,y)$ and $\Delta(x,y) < \delta(x,y)$). Since L, M, d, and Σ are symmetric in the sense $L(x,y) = L(y,x)$, these parameters are independent of the assumption $|x| \ge |y|$.

2.2 Hardness Hypotheses The Strong Exponential Time Hypothesis (SETH) was introduced by Impagliazzo, Paturi, and Zane [50, 51] and essentially asserts that satisfiability has no algorithms that are much faster than exhaustive search. It forms the basis of many conditional lower bounds for NP-hard as well as polynomial-time problems.

HYPOTHESIS 2.1. (SETH) *For any $\varepsilon > 0$ there is a $k \ge 3$ such that k-SAT on n variables cannot be solved in time $\mathcal{O}((2-\varepsilon)^n)$.*

Effectively all known SETH-based lower bounds for polynomial-time problems use reductions via the *Orthogonal Vectors problem* (OV): Given sets $\mathcal{A}, \mathcal{B} \subseteq \{0,1\}^D$ of size $|\mathcal{A}| = n$, $|\mathcal{B}| = m$, determine whether there exist $a \in \mathcal{A}$, $b \in \mathcal{B}$ with $\sum_{i=1}^{D} a[i] \cdot b[i] = 0$ (which we denote by $\langle a, b\rangle = 0$). Simple algorithms solve OV in time $\mathcal{O}(2^D(n+m))$ and $\mathcal{O}(nmD)$. The fastest known algorithm for $D = c(n)\log n$ with $c(n) = n^{o(1)}$ runs in time $n^{2-1/\mathcal{O}(\log c(n))}$ (when $n = m$) [5], which is only slightly subquadratic for $D \gg \log n$. This has led to the Orthogonal Vectors Hypothesis (OVH).

HYPOTHESIS 2.2. (OVH) OV *restricted to $n = |\mathcal{A}| = |\mathcal{B}|$ and $D = n^{o(1)}$ requires time $n^{2-o(1)}$.*

A well-known reduction by Williams [83] shows that SETH implies OVH. Thus, OVH is the weaker assumption and any OVH-based lower bound also implies a SETH-based lower bound. The results in this paper do not only hold assuming SETH, but even assuming the weaker OVH. For simplicity, we will always work with the following equivalent variant of OVH.

HYPOTHESIS 2.3. (UNBALANCED OVH (UOVH)) *For any $\alpha, \beta \in (0,1]$, and computable functions $f(n) = n^{\alpha - o(1)}$, $g(n) = n^{\beta - o(1)}$, the following problem requires time $n^{\alpha+\beta-o(1)}$: Given a number n, solve a*

given OV *instance with* $D = n^{o(1)}$, $|\mathcal{A}| = f(n)$ *and* $|\mathcal{B}| = g(n)$.

LEMMA 2.1. (ESSENTIALLY FOLKLORE) UOVH *is equivalent to* OVH.

UOVH弱於OVH

Proof. Clearly, UOVH implies OVH (using $\alpha = \beta = 1, f(n) = g(n) = n$). For the other direction, assume that UOVH fails and let $\alpha, \beta \in (0, 1]$, $f(n) = n^{\alpha - o(1)}$, and $g(n) = n^{\beta - o(1)}$ be such that OV with $D = n^{o(1)}$ and $|\mathcal{A}| = f(n)$ and $|\mathcal{B}| = g(n)$ can be solved in time $\mathcal{O}(n^{\alpha+\beta-\varepsilon})$ for some constant $\varepsilon > 0$. Consider an arbitrary OV instance $\mathcal{A}, \mathcal{B} \subseteq \{0,1\}^D$ with $D = n^{o(1)}$. We partition $\mathcal{A}$ into $s := \lceil \frac{n}{f(n)} \rceil$ sets $\mathcal{A}_1, \ldots, \mathcal{A}_s$ of size $f(n)$ and $\mathcal{B}$ into $t := \lceil \frac{n}{g(n)} \rceil$ sets $\mathcal{B}_1, \ldots, \mathcal{B}_t$ of size $g(n)$ (note that the last set of such a partition might have strictly less elements, but can safely be filled up using all-ones vectors). By assumption, we can solve each OV instance $\mathcal{A}_i, \mathcal{B}_j$ in time $\mathcal{O}(n^{\alpha+\beta-\varepsilon})$. Since there exist $a \in \mathcal{A}, b \in \mathcal{B}$ with $\langle a, b \rangle = 0$ if and only if there exist $a \in \mathcal{A}_i, b \in \mathcal{B}_j$ with $\langle a_i, b_j \rangle = 0$ for some $i \in [s], j \in [t]$, we can decide the instance $\mathcal{A}, \mathcal{B}$ by sequentially deciding the $s \cdot t = \mathcal{O}(n^{2-(\alpha+\beta)+o(1)})$ OV instances $\mathcal{A}_i, \mathcal{B}_j$. This takes total time $\mathcal{O}(s \cdot t \cdot n^{\alpha+\beta-\varepsilon}) = \mathcal{O}(n^{2-\varepsilon'})$ for any $\varepsilon' < \varepsilon$, which contradicts OVH, proving the claim.

3 Formal Statement of Results

Recall that n is the input size and $\mathcal{P} := \{m, L, \delta, \Delta, |\Sigma|, M, d\}$ is the set of parameters that were previously studied in the literature. We let $\mathcal{P}^* := \mathcal{P} \cup \{n\}$. A parameter setting fixes a polynomial relation between any parameter and n. To formalize this, we call a vector $\boldsymbol{\alpha} = (\alpha_p)_{p \in \mathcal{P}}$ with $\alpha_p \in \mathbb{R}_{\geq 0}$ a *parameter setting*, and an LCS instance x, y *satisfies* the parameter setting $\boldsymbol{\alpha}$ if each parameter p attains a value $p(x, y) = \Theta(n^{\alpha_p})$. This yields a subproblem of LCS consisting of all instances that satisfy the parameter setting. We sometimes use the notation $\alpha_n = 1$.

For our running time bounds, for each parameter $p \in \mathcal{P}$ except for $|\Sigma|$ we can assume $\alpha_p > 0$, since otherwise one of the known algorithms runs in time $\tilde{\mathcal{O}}(n)$ and there is nothing to show. Similarly, for $\alpha_d \leq 1$ there is an $\tilde{\mathcal{O}}(n)$ algorithm and there is nothing to show. For Σ, however, the case $\alpha_\Sigma = 0$, i.e., $|\Sigma| = \Theta(1)$, is an important special case. We study this case more closely by also considering parameter settings that fix $|\Sigma|$ to any specific constant greater than 1.

DEFINITION 3.1. (PARAMETER SETTING) *Fix* $\gamma \geq 1$. *Let* $\boldsymbol{\alpha} = (\alpha_p)_{p \in \mathcal{P}}$ *with* $\alpha_p \in \mathbb{R}_{\geq 0}$. *We define* $\mathrm{LCS}^\gamma(\boldsymbol{\alpha})$ *as the problem of computing the length of an LCS of two given strings* x, y *satisfying* $n^{\alpha_p}/\gamma \leq p(x, y) \leq n^{\alpha_p} \cdot \gamma$ *for every parameter* $p \in \mathcal{P}$, *where* $n = |x|$, *and* $|x| \geq |y|$. *We call* $\boldsymbol{\alpha}$ *and* $\mathrm{LCS}^\gamma(\boldsymbol{\alpha})$ parameter settings. *In some statements we simply write* $\mathrm{LCS}(\boldsymbol{\alpha})$ *to abbreviate that there exists a* $\gamma \geq 1$ *such that the statement holds for* $\mathrm{LCS}^\gamma(\boldsymbol{\alpha})$.

For any fixed alphabet Σ, *constant* $\gamma \geq 1$, *and parameter setting* $\boldsymbol{\alpha}$ *with* $\alpha_\Sigma = 0$, *we also define the problem* $\mathrm{LCS}^\gamma(\boldsymbol{\alpha}, \Sigma)$, *where additionally the alphabet of* x, y *is fixed to be* Σ. *We again call* $(\boldsymbol{\alpha}, \Sigma)$ *and* $\mathrm{LCS}^\gamma(\boldsymbol{\alpha}, \Sigma)$ *parameter settings.*

We call a parameter setting $\boldsymbol{\alpha}$ *or* $(\boldsymbol{\alpha}, \Sigma)$ trivial *if for all* $\gamma \geq 1$ *the problem* $\mathrm{LCS}^\gamma(\boldsymbol{\alpha})$ *or* $\mathrm{LCS}^\gamma(\boldsymbol{\alpha}, \Sigma)$, *respectively, has only finitely many instances.*

As our goal is to prove hardness for any non-trivial parameter setting, for each parameter setting we either need to construct hard instances or verify that it is trivial. That is, in one way or the other we need a complete *classification* of parameter settings into trivial and non-trivial ones. To this end, we need to understand all interactions among our parameters that hold up to constant factors, which is an interesting question on its own, as it yields insight into the structure of strings from the perspective of the LCS problem. For our seven parameters, determining all interactions is a complex task. This is one of the major differences to previous multivariate fine-grained complexity results, where the number of parameters was one, or in rare cases two, limiting the interaction among parameters to a simple level.

THEOREM 3.1. (CLASSIFICATION OF NON-TRIVIALITY) *A parameter setting* $\boldsymbol{\alpha}$ *or* $(\boldsymbol{\alpha}, \Sigma)$ *is non-trivial if and only if it satisfies all restrictions in Table 2.*

Note that the restrictions in Table 2 consist mostly of linear inequalities, and that for small alphabet sizes $|\Sigma| \in \{2, 3\}$ additional parameter relations hold. The proof of this and the following results will be outlined in Section 4. We can now state our main lower bound.

THEOREM 3.2. (HARDNESS FOR LARGE ALPHABET) *For any non-trivial parameter setting* $\boldsymbol{\alpha}$, *there is a constant* $\gamma \geq 1$ *such that* $\mathrm{LCS}^\gamma(\boldsymbol{\alpha})$ *requires time* $\min\{d, \delta\Delta, \delta m\}^{1-o(1)}$, *unless* OVH *fails.*

In the case of constant alphabet size, the (conditional) complexity differs between $|\Sigma| = 2$ and $|\Sigma| \geq 3$. Note that $|\Sigma| = 1$ makes LCS trivial.

THEOREM 3.3. (HARDNESS FOR SMALL ALPHABET) *For any non-trivial parameter setting* $(\boldsymbol{\alpha}, \Sigma)$, *there is a constant* $\gamma \geq 1$ *such that, unless* OVH *fails,* $\mathrm{LCS}^\gamma(\boldsymbol{\alpha}, \Sigma)$ *requires time*

- $\min\{d, \delta\Delta, \delta m\}^{1-o(1)}$ *if* $|\Sigma| \geq 3$,

滿足下列條件的參數為non-trival

Parameter	Restriction
m	$0 \le \alpha_m \le 1$
L	$0 \le \alpha_L \le \alpha_m$
δ	$\begin{cases} 0 \le \alpha_\delta \le \alpha_m & \text{if } \alpha_L = \alpha_m \\ \alpha_\delta = \alpha_m & \text{otherwise} \end{cases}$
Δ	$\begin{cases} \alpha_\delta \le \alpha_\Delta \le 1 & \text{if } \alpha_L = \alpha_m = 1 \\ \alpha_\Delta = 1 & \text{otherwise} \end{cases}$
$\lvert\Sigma\rvert$	$0 \le \alpha_\Sigma \le \alpha_m$
d	$\alpha_d \ge \max\{\alpha_L, \alpha_\Sigma\}$ $\alpha_d \le 2\alpha_L + \alpha_\Sigma$ $\alpha_d \le \min\{\alpha_L + \alpha_m, \alpha_L + \alpha_\Delta\}$
M	$\alpha_M \ge \max\{1, \alpha_d, 2\alpha_L - \alpha_\Sigma\}$ $\alpha_M \le \alpha_L + 1$ if $\lvert\Sigma\rvert = 2$: $\alpha_M \ge \alpha_L + \alpha_m$ if $\lvert\Sigma\rvert = 2$: $\alpha_M \ge 1 + \alpha_d - \alpha_L$ if $\lvert\Sigma\rvert = 3$: $\alpha_M \ge \alpha_m + \alpha_d - \alpha_L$

Table 2: Complete set of restrictions for non-trivial parameter settings.

Relation	Restriction	Reference
$L \le m \le n$		trivial
$L \le d \le M$		trivial
$\Delta \le n$		trivial
$\delta \le m$		trivial
$\delta \le \Delta$		trivial
$\delta = m - L$		by definition
$\Delta = n - L$		by definition
$\lvert\Sigma\rvert \le m$		Assumption 2.1
$n \le M$		Assumption 2.1
$d \le Lm$		see full version
$d \le L^2\lvert\Sigma\rvert$		see full version
$d \le 2L(\Delta + 1)$		see full version
$\lvert\Sigma\rvert \le d$		see full version
$\frac{L^2}{\lvert\Sigma\rvert} \le M \le 2Ln$		see full version
$M \ge Lm/4$	if $\lvert\Sigma\rvert = 2$	see full version
$M \ge nd/(5L)$	if $\lvert\Sigma\rvert = 2$	see full version
$M \ge md/(80L)$	if $\lvert\Sigma\rvert = 3$	see full version

Table 3: Relations between the parameters.

- $\min\left\{d, \delta\Delta, \delta M/n\right\}^{1-o(1)}$ *if* $|\Sigma| = 2$.

Finally, we prove the following algorithmic result, handling binary alphabets faster if M and δ are sufficiently small. This yields matching upper and lower bounds also for $|\Sigma| = 2$.

THEOREM 3.4. (ALGORITHMIC RESULT) *For $|\Sigma| = 2$, LCS can be solved in time $\mathcal{O}(n + \delta M/n)$.*

4 Hardness Proof Overview

In this section we present an overview of the proofs of our main results. We first focus on the large alphabet case, i.e., parameter settings $\boldsymbol{\alpha}$, and discuss small constant alphabets in Section 4.4.

4.1 Classification of Non-trivial Parameter Settings

The only-if-direction of Theorem 3.1 follows from proving ineqalities among the parameters that hold for all strings, and then converting them to inequalities among the α_p's, as follows.

LEMMA 4.1. (PARAMETER RELATIONS) *For any strings x, y the parameter values $\mathcal{P}^*$ satisfy the relations in Table 3. Thus, any non-trivial parameter setting $\boldsymbol{\alpha}$ or $(\boldsymbol{\alpha}, \Sigma)$ satisfies Table 2.*

Proof. (Sketch.) The full proof is deferred to the full version of this article. Some parameter relations follow trivially from the parameter definitions, like $L \le m \le n$. Since by Assumption 2.1 every symbol in Σ appears in x and y, we obtain parameter relations like $|\Sigma| \le m$. Other parameter relations need a non-trivial proof, like $M \ge md/(80L)$ if $|\Sigma| = 3$.

From a relation like $L \le m$ we infer that if $\alpha_L > \alpha_m$ then for sufficiently large n no strings x, y have $L(x, y) = \Theta(n^{\alpha_L})$ and $m(x, y) = \Theta(n^{\alpha_m})$, and thus $\mathrm{LCS}^\gamma(\boldsymbol{\alpha})$ is finite for any $\gamma > 0$. This argument converts Table 3 to Table 2.

For the if-direction of Theorem 3.1, the task is to show that any parameter setting satisfying Table 2 is non-trivial, i.e., to construct infinitely many strings in the parameter setting. We start with a construction that sets a single parameter p as specified by $\boldsymbol{\alpha}$, and all others not too large.

LEMMA 4.2. (PADDINGS) *Let $\boldsymbol{\alpha}$ be a parameter setting satisfying Table 2. For any parameter $p \in \mathcal{P}^*$ and any $n \ge 1$ we can construct strings x_p, y_p such that (1) $p(x_p, y_p) = \Theta(n^{\alpha_p})$, and (2) for all $q \in \mathcal{P}^*$ we have $q(x_p, y_p) = O(n^{\alpha_q})$. Moreover, given n we can compute x_p, y_p, and $L(x_p, y_p)$ in time $\mathcal{O}(n)$.*

Note that although for Theorem 3.1 the *existence* of infinitely many strings would suffice, we even show that they can be *computed* very efficiently. We will use this additional fact in Section 4.2.

Proof. (Sketch.) We defer the full proof to the full version of this article and here only sketch the proof for the parameter $|\Sigma|$. Let $w := 12\ldots t$ be the concatenation of $t := \lceil n^{\alpha_\Sigma} \rceil$ unique symbols. We argue that the strings w, w or the strings $w, \mathrm{rev}(w)$ prove

Lemma 4.2 for parameter $p = |\Sigma|$, depending on the parameter setting $\boldsymbol{\alpha}$. Clearly, both pairs of strings realize an alphabet of size $t = \Theta(n^{\alpha_\Sigma})$, showing (1). By Table 2, we have $\alpha_L = \alpha_m$ or $\alpha_\delta = \alpha_m$. In the first case, we use $L(w, w) = t = O(n^{\alpha_\Sigma})$ together with $\alpha_\Sigma \leq \alpha_m = \alpha_L$, as well as $\delta(w, w) = \Delta(w, w) = 0 \leq n^{\alpha_\delta} \leq n^{\alpha_\Delta}$, to show (2) for the parameters L, δ, Δ. In the second case, we similarly have $L(w, \mathrm{rev}(w)) = 1 \leq n^{\alpha_L}$ and $\delta(w, \mathrm{rev}(w)) = \Delta(w, \mathrm{rev}(w)) = t - 1 = O(n^{\alpha_\Sigma})$ and $\alpha_\Sigma \leq \alpha_m = \alpha_\delta \leq \alpha_\Delta$.

The remaining parameters are straight-forward. Let $(x, y) \in \{(w, w), (w, \mathrm{rev}(w))\}$. We have $n(x, y) = m(x, y) = t = O(n^{\alpha_\Sigma}) = O(n^{\alpha_m}) = O(n)$. Moreover, $d(x, y) \leq M(x, y) = t = O(n^{\alpha_\Sigma}) = O(n^{\alpha_d}) = O(n^{\alpha_M})$. Clearly, the strings and their LCS length can be computed in time $\mathcal{O}(n)$.

To combine the paddings for different parameters, we need the useful property that all studied parameters sum up if we concatenate strings over disjoint alphabets.

LEMMA 4.3. (DISJOINT ALPHABETS) *Let $\Sigma_1, \ldots, \Sigma_k$ be disjoint alphabets and let x_i, y_i be strings over alphabet Σ_i with $|x_i| \geq |y_i|$ for all i. Consider $x := x_1 \ldots x_k$ and $y := y_1 \ldots y_k$. Then for any parameter $p \in \mathcal{P}^*$, we have $p(x, y) = \sum_{i=1}^{k} p(x_i, y_i)$.*

Proof. The statement is trivial for the string lengths n, m, the alphabet size $|\Sigma|$, and the number of matching pairs M. For the LCS length L we observe that any common subsequence z can be decomposed into $z_1 \ldots z_k$ with z_i using only symbols from Σ_i, so that $|z_i| \leq L(x_i, y_i)$ and thus $L(x, y) \leq \sum_{i=1}^{k} L(x_i, y_i)$. Concatenating longest common subsequences of x_i, y_i, we obtain equality. Using $\delta = m - L$ and $\Delta = n - L$, the claim follows also for δ and Δ.

Since every dominant pair is also a matching pair, every dominant pair of x, y stems from prefixes $x_1 \ldots x_j x'$ and $y_1 \ldots y_j y'$, with x' being a prefix of x_{j+1} and y' being a prefix of y_{j+1} for some j. Since $L(x_1 \ldots x_j x', y_1 \ldots y_j y') = \sum_{i=1}^{j} L(x_i, y_i) + L(x', y')$, where the first summand does not depend on x', y', the dominant pairs of x, y of the form $x_1 \ldots x_j x', y_1 \ldots y_j y'$ are in one-to-one correspondence with the dominant pairs of x_{j+1}, y_{j+1}. This yields the claim for parameter d.

With these preparations we can finish our classification.

Proof. (Proof of Theorem 3.1 for large alphabet.) One direction follows from Lemma 4.1. For the other direction, let $\boldsymbol{\alpha}$ be a parameter setting satisfying Table 2. For any $n \geq 1$ consider the instances x_p, y_p constructed in Lemma 4.2, and let them use disjoint alphabets for different $p \in \mathcal{P}^*$. Then the concatenations $x := \bigcirc_{p \in \mathcal{P}^*} x_p$ and $y := \bigcirc_{p \in \mathcal{P}^*} x_p$ form an instance of LCS($\boldsymbol{\alpha}$), since for any parameter $p \in \mathcal{P}^*$ we have $p(x_p, y_p) = \Theta(n^{\alpha_p})$, and for all other instances $x_{p'}, y_{p'}$ the parameter p is $\mathcal{O}(n^{\alpha_p})$, and thus $p(x, y) = \Theta(n^{\alpha_p})$ by the Disjoint Alphabets Lemma. Thus, we constructed instances of LCS($\boldsymbol{\alpha}$) of size $\Theta(n)$ for any $n \geq 1$, so the parameter setting $\boldsymbol{\alpha}$ is non-trivial.

We highlight two major hurdles we had to be overcome to obtain this classification result:

- Some of the parameter relations of Table 3 are scattered through the LCS literature, e.g., the inequality $d \leq Lm$ is mentioned in [14]. In fact, proving any single one of these inequalities is not very hard – the main issue was to find a *complete* set of parameter relations. The authors had to perform many iterations of going back and forth between searching for new parameter relations (i.e., extending Lemma 4.1) and constructing strings satisfying specific parameter relations (i.e., extending Lemma 4.2), until finally coming up with a complete list.
- The dependency of d on the other parameters is quite complicated. Indeed, eight of the parameter relations of Table 3 involve dominant pairs. Apostolico [13] introduced the parameter under the initial impression that "it seems that whenever $[M]$ gets too close to mn, then this forces d to be linear in m". While we show that this intuition is somewhat misleading by constructing instances with high values of both M and d, it is a rather complex task to generate a desired number of dominant pairs while respecting given bounds on all other parameters. Intuitively, handling dominant pairs is hard since they involve restrictions on each pair of prefixes of x and y. For Lemma 4.2, we end up using the strings $(01)^{R+S}$, $0^R(01)^S$ as well as $((1\circ\ldots\circ t)\circ(t'\circ\ldots\circ 1))^R(1\circ\ldots\circ t)^{S-R}$, $(1\circ\ldots\circ t)^S$ for different values of R, S, t, t'.

4.2 Monotonicity of Time Complexity It might be tempting to assume that the optimal running time for solving LCS is monotone in the problem size n and the parameters $\mathcal{P}$ (say up to constant factors, as long as all considered parameters settings are non-trivial). However, since the parameters have complex interactions (see Table 3) it is far from obvious whether this intuition is correct. In fact, the intuition *fails* for $|\Sigma| = 2$, where the running time $\mathcal{O}(n+\delta M/n)$ of our new algorithm is not monotone, and thus also the tight time bound $(n + \min\{d, \delta\Delta, \delta M/n\})^{1\pm o(1)}$ is not monotone.

Nevertheless, we will prove monotonicity for any parameter setting $\boldsymbol{\alpha}$, i.e., when the alphabet size can be

assumed to be at least a sufficiently large constant. To formalize monotonicity, we define the problem $\mathrm{LCS}_{\leq}(\boldsymbol{\alpha})$ consisting of all instances of LCS with all parameters *at most* as in $\mathrm{LCS}(\boldsymbol{\alpha})$.

Definition 4.1. (Downward Closure) *Fix $\gamma \geq 1$ and let $\boldsymbol{\alpha}$ be a parameter setting. We define the* downward closure $\mathrm{LCS}^{\gamma}_{\leq}(\boldsymbol{\alpha})$ *as follows. An instance of this problem is a triple (n, x, y), where $p(x, y) \leq \gamma \cdot n^{\alpha_p}$ for any $p \in \mathcal{P}^*$, and the task is to compute the length of an LCS of x, y. In some statements, we simply write $\mathrm{LCS}_{\leq}(\boldsymbol{\alpha})$ to abbreviate that there exists a $\gamma \geq 1$ such that the statement holds for $\mathrm{LCS}^{\gamma}_{\leq}(\boldsymbol{\alpha})$.*

Similarly, for any fixed alphabet Σ we consider the downward closure $\mathrm{LCS}^{\gamma}_{\leq}(\boldsymbol{\alpha}, \Sigma)$ with instances (n, x, y), where x, y are strings over alphabet Σ and $p(x, y) \leq \gamma \cdot n^{\alpha_p}$ for any $p \in \mathcal{P}^*$.

Lemma 4.4. (Monotonicity) *For any non-trivial parameter setting $\boldsymbol{\alpha}$ and $\beta \geq 1$, $\mathrm{LCS}^{\gamma}(\boldsymbol{\alpha})$ has an $\mathcal{O}(n^{\beta})$-time algorithm for all γ if and only if $\mathrm{LCS}^{\gamma}_{\leq}(\boldsymbol{\alpha})$ has an $\mathcal{O}(n^{\beta})$-time algorithm for all γ.*

Proof. The if-direction follows from the fact that if (x, y) is an instance of $\mathrm{LCS}^{\gamma}(\boldsymbol{\alpha})$ then $(|x|, x, y)$ is an instance of $\mathrm{LCS}^{\gamma}_{\leq}(\boldsymbol{\alpha})$.

For the other direction, let (n, x, y) be an instance of $\mathrm{LCS}_{\leq}(\boldsymbol{\alpha})$. Since $\boldsymbol{\alpha}$ is non-trivial, it satisfies Table 2, by Theorem 3.1. Lemma 4.2 thus allows us to construct paddings x_p, y_p for any $p \in \mathcal{P}^*$ such that (1) $p(x_p, y_p) = \Theta(n^{\alpha_p})$ and (2) (n, x_p, y_p) is an instance of $\mathrm{LCS}_{\leq}(\boldsymbol{\alpha})$. We construct these paddings over disjoint alphabets for different parameters and consider the concatenations $x' := x \circ \bigcirc_{p \in \mathcal{P}} x_p$ and $y' := y \circ \bigcirc_{p \in \mathcal{P}} y_p$. Then (1), (2), and the Disjoint Alphabets Lemma imply that $p(x', y') = \Theta(n^{\alpha_p})$ for any $p \in \mathcal{P}^*$, so that (x', y') is an instance of $\mathrm{LCS}(\boldsymbol{\alpha})$. By assumption, we can thus compute $L(x', y')$ in time $O(n^{\beta})$. By the Disjoint Alphabets Lemma, we have $L(x, y) = L(x', y') - \sum_{p \in \mathcal{P}} L(x_p, y_p)$, and each $L(x_p, y_p)$ can be computed in time $O(n)$ by Lemma 4.2, which yields $L(x, y)$ and thus solves the given instance (n, x, y). We finish the proof by observing that the time to construct x', y' is bounded by $\mathcal{O}(n)$.

Note that this proof was surprisingly simple, considering that monotonicity fails for $|\Sigma| = 2$.

4.3 Hardness for Large Alphabet Since we established monotonicity for parameter settings $\boldsymbol{\alpha}$, it suffices to prove hardness for $\mathrm{LCS}_{\leq}(\boldsymbol{\alpha})$ instead of $\mathrm{LCS}(\boldsymbol{\alpha})$. This makes the task of constructing hard strings considerably easier, since we only have to satisfy upper bounds on the parameters. Note that our main result Theorem 3.2 follows from Lemma 4.4 and Theorem 4.1 below.

Theorem 4.1. (Hardness for Large Alphabet) *For any non-trivial parameter setting $\boldsymbol{\alpha}$, there exists $\gamma \geq 1$ such that $\mathrm{LCS}^{\gamma}_{\leq}(\boldsymbol{\alpha})$ requires time $\min\{d, \delta m, \delta\Delta\}^{1-o(1)}$, unless* OVH *fails.*

Proof. (Sketch.) The full proof is deferred to Section 7. We provide different reductions for the cases $\alpha_\delta = \alpha_m$ and $\alpha_L = \alpha_m$. Intuitively, this case distinction is natural, since after this choice all remaining restrictions from Table 2 are of an easy form: they are linear inequalities.

In the case $\alpha_\delta = \alpha_m$ the complexity $\min\{d, \delta\Delta, \delta m\}^{1\pm o(1)}$ simplifies to $d^{1\pm o(1)}$ (since $\alpha_d \leq \alpha_L + \alpha_m \leq 2\alpha_m = \alpha_\delta + \alpha_m$ and similarly $\alpha_d \leq \alpha_\delta + \alpha_m = 2\alpha_\delta \leq \alpha_\delta + \alpha_\Delta$, see Table 2). This simplification makes this case much easier. For constant alphabet, instantiating the known reduction from OV to LCS [28] such that x chooses one of $\approx L$ vectors and y chooses one of $\approx d/L$ vectors yields the claim. For larger alphabet, the right-hand side of the parameter relation $d \leq L^2|\Sigma|$ increases and allows for potentially more dominant pairs. In this case, the second set of vectors would increase to a size of $\approx d/L = \omega(L)$, and the length of an LCS of this construction becomes too large. We thus adapt the reduction by using the construction for constant alphabet multiple times over disjoint alphabets and concatenating the results (reversing the order in one).

The case $\alpha_L = \alpha_m$ is harder, since all three terms of the complexity $\min\{d, \delta\Delta, \delta m\}^{1\pm o(1)}$ are relevant. The known reduction [28] fails fundamentally in this case, roughly speaking since the resulting δ is always as large as the number of vectors encoded by any of x and y. Hence, we go back to the "normalized vector gadgets" from the known reduction [28], which encode vectors a, b by strings $\mathrm{NVG}(a), \mathrm{NVG}(b)$ whose LCS length only depends on whether a, b are orthogonal. We then carefully embed these gadgets into strings that satisfy any given parameter setting. A crucial trick is to pad each gadget to $\mathrm{NVG}'(a) := 0^{\alpha}1^{\beta}(01)^{\gamma}\mathrm{NVG}(a)1^{\gamma}$ for appropriate lengths α, β, γ. It is easy to see that this constructions ensures the following:

(1vs1) The LCS length of $\mathrm{NVG}'(a), \mathrm{NVG}'(b)$ only depends on whether a, b are orthogonal, and

(2vs1) $\mathrm{NVG}'(b)$ is a subsequence of $\mathrm{NVG}'(a) \circ \mathrm{NVG}'(a')$ for any vectors a, a', b.

In particular, for any vectors $a^{(1)}, \ldots, a^{(2k-1)}$ and $b^{(1)}, \ldots, b^{(k)}$, on the strings $x = \bigcirc_{i=1}^{2k-1} \mathrm{NVG}'(a^{(i)})$ and $y = \bigcirc_{j=1}^{k} \mathrm{NVG}'(b^{(j)})$ we show that any LCS consists of $k-1$ matchings of type 2vs1 and one matching of type 1vs1 (between $\mathrm{NVG}'(b^{(j)})$ and $\mathrm{NVG}'(a^{(2j-1)})$ for some

j). Thus, the LCS length of x and y only depends on whether there exists a j such that $a^{(2j-1)}, b^{(j)}$ are orthogonal. Moreover, since most of y is matched by type 2vs1 and thus completely contained in x, the parameter $\delta(x,y)$ is extremely small compared to the lengths of x and y – which is not achievable with the known reduction [28]. Our proof uses an extension of the above construction, which allows us to have more than one matching of type 1vs1. We think that this 1vs1/2vs1-construction is our main contribution to specific proof techniques and will find more applications.

4.4 Small Alphabet Proving our results for small constant alphabets poses additional challenges. For instance, our proof of Lemma 4.4 fails for parameter settings $(\boldsymbol{\alpha}, \Sigma)$ if $|\Sigma|$ is too small, since the padding over disjoint alphabets produces strings over alphabet size at least $|\mathcal{P}| = 7$. In particular, for $|\Sigma| = 2$ we may not use the Disjoint Alphabets Lemma at all, rendering Lemma 4.2 completely useless. However, the classification Theorem 3.1 still holds for parameter settings $(\boldsymbol{\alpha}, \Sigma)$. A proof is implicit in our constructions, as we construct (infinitely many) hard instances for all parameter settings $(\boldsymbol{\alpha}, \Sigma)$ satisfying Table 2.

As mentioned above, the Monotonicity Lemma (Lemma 4.4) is wrong for $|\Sigma| = 2$, since our new algorithm has a running time $\tilde{\mathcal{O}}(n + \delta M/n)$ which is not monotone. Hence, it is impossible to use general strings from $\mathrm{LCS}_{\leq}(\boldsymbol{\alpha}, \Sigma)$ as a hard core for $\mathrm{LCS}(\boldsymbol{\alpha}, \Sigma)$. Instead, we use strings from a different, appropriately chosen parameter setting $\mathrm{LCS}_{\leq}(\boldsymbol{\alpha}', \Sigma')$ as a hard core. Moreover, instead of padding with new strings x_p, y_p for each parameter, we need an integrated construction where we control all parameters at once. This is a technically demanding task to which we devote a large part of the full version of this article. Since the cases $|\Sigma| = 2$, $|\Sigma| = 3$, and $|\Sigma| \geq 4$ adhere to different relations of Table 2, these three cases have to be treated separately. Furthermore, as for large alphabet we consider cases $\alpha_\delta = \alpha_m$ and $\alpha_L = \alpha_m$. Hence, our reductions are necessarily rather involved and we need to very carefully fine-tune our constructions.

5 Organization

The remainder of the paper gives details to the proofs of Theorems 3.1 and 3.2, following the outline given in Section 4. The proofs of all parameter relations (Lemma 4.1) as well as the proof of the padding lemma (Lemma 4.2) had to be deferred to the full version of this article due to space constraints. Instead, we focus on the proof of Theorem 4.1. To this end, Section 6 starts off with basic facts and technical tools easing the analysis – this includes a simple greedy prefix matching property as well as a surprising technique to reduce the number of dominant pairs of two given strings. Section 7 then constructs hard instances for large alphabet (for the downward closure of any parameter setting, proving Theorem 4.1 and thus Theorem 3.2). Here, we concentrate on the case that necessitates our new 2vs1/1vs1 gadget. The much more intricate case of small constant alphabet sizes such as $|\Sigma| = 2$ had to be omitted, as it takes up a large fraction of the full version of this article. Finally, our new algorithm proving Theorem 3.4 is deferred to the full version as well.

6 Technical Tools and Constructions

To prepare later constructions and ease their analysis, this section collects several technical results. Their proofs had to be deferred to the full version. We start off with the simple fact that equal prefixes can be greedily matched.

LEMMA 6.1. (GREEDY PREFIX MATCHING) *For any strings w, x, y, we have $L(wx, wy) = |w| + L(x, y)$ and $d(wx, wy) = |w| + d(x, y)$.*

The remainder of this section prepares a surprising technique to reduce the number of dominant pairs in an instance: Let x, y be any strings (see Figure 2 for an examplary instance exhibiting a large number of dominant pairs). We can build strings x', y' such that $L(x', y')$ lets us recover $L(x, y)$, but the number of dominant pairs may be reduced significantly, namely to a value $d(x', y') = \mathcal{O}(\delta(x, y) \cdot n(x, y))$, independently of $d(x, y)$.

	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	1	**2**	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
0	1	1	2	2	**3**	3	3	3	3	3	3	3	3	3	3	3	3	3
0	1	1	2	2	3	3	**4**	4	4	4	4	4	4	4	4	4	4	4
0	1	1	2	2	3	3	4	4	**5**	5	5	5	5	5	5	5	5	5
1	1	**2**	2	**3**	3	**4**	4	**5**	5	**6**	6	6	6	6	6	6	6	6
0	1	2	**3**	3	**4**	4	**5**	5	**6**	6	**7**	7	7	7	7	7	7	7
1	1	2	3	**4**	4	**5**	5	**6**	6	**7**	7	**8**	8	8	8	8	8	8
0	1	2	3	4	**5**	5	**6**	6	**7**	7	**8**	8	**9**	9	9	9	9	9
1	1	2	3	4	5	**6**	6	**7**	7	**8**	8	**9**	9	**10**	10	10	10	10
0	1	2	3	4	5	6	**7**	7	**8**	8	**9**	9	**10**	10	**11**	11	11	11
1	1	2	3	4	5	6	7	**8**	8	**9**	9	**10**	10	**11**	11	**12**	12	12
0	1	2	3	4	5	6	7	8	**9**	9	**10**	10	**11**	11	**12**	12	**13**	13
1	1	2	3	4	5	6	7	8	9	**10**	10	**11**	11	**12**	12	**13**	13	**14**

Figure 2: The L-table for the strings $x = (01)^{R+S}$ and $y = 0^R(01)^S$ with $R = 4, S = 5$ (where the entry in row j and column i denotes $L(x[1..i], y[1..j])$). The indicated entries represent dominant pairs.

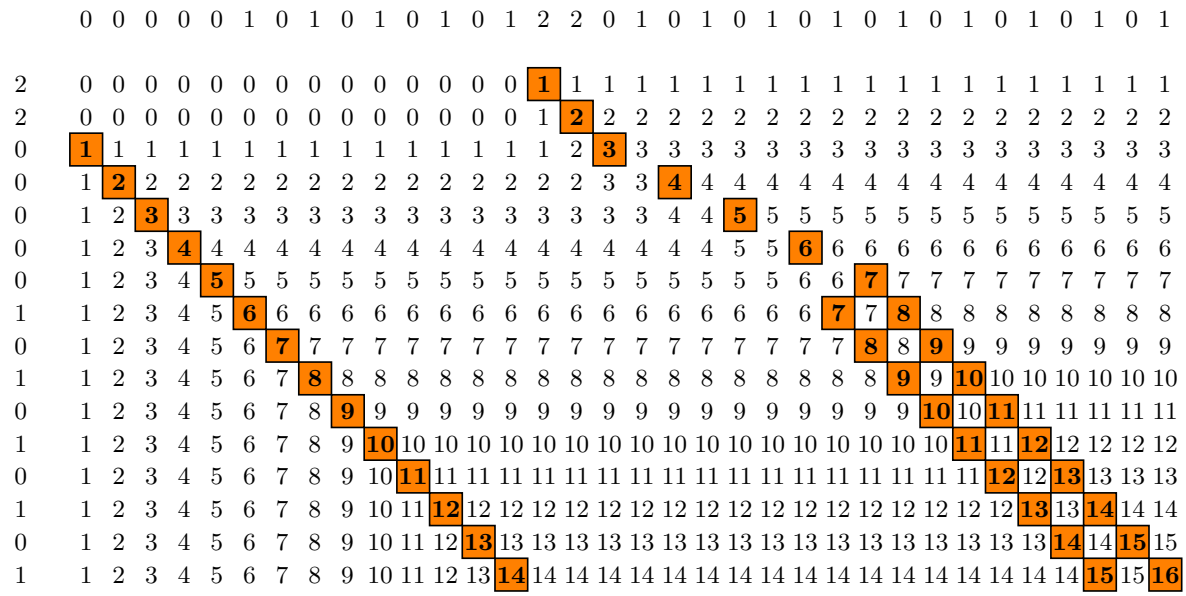


Figure 3: Illustration of Lemma 6.4. The strings $x' = y2^\ell x, y' = 2^\ell y$ are defined using $x = (01)^{R+S}$, $y = 0^R(01)^S$, $R = 4$, $S = 5$, and $\ell = 2$. The number of dominant pairs is significantly reduced compared to Figure 2.

To ease the analysis, the next lemma allows us to "eliminate" 0^ℓ-blocks when computing the LCS of strings of the form $x0^\ell y, 0^\ell z$, provided that ℓ is sufficiently large.

LEMMA 6.2. *For any strings x, y, z and $\ell \geq \#_0(x) + |z|$ we have $L(x0^\ell y, 0^\ell z) = \ell + L(0^{\#_0(x)}y, z)$.*

The following lemma bounds the number of dominant pairs of strings of the form $x' = yx$, $y' = zy$ by $d(x', y') = \mathcal{O}(|z| \cdot |y'|)$. If $|x'| \geq |y'|$, this provides a bound of $\mathcal{O}(\delta(x', y') \cdot m(x', y'))$ instead of the general, weaker bound $\mathcal{O}(\Delta(x', y') \cdot m(x', y'))$ of the parameter relation $d \leq 2L(\Delta + 1)$.

LEMMA 6.3. *For any strings x, y, z, let $x' = yx$, $y' = zy$. Then*

$$d(x', y') \leq |y| \cdot (|z| + 1) + d(x', z) \leq |y| \cdot (|z| + 1) + |z|^2.$$

With the tools collected above, we can finally state our dominant pair reduction technique, whose effect is illustrated in Figure 3.

LEMMA 6.4. (DOMINANT PAIR REDUCTION) *Consider strings x, y and a number $\ell > |y| - L(x, y)$.*

(i) If 2 is a symbol not appearing in x, y, then $x' := y2^\ell x$ and $y' := 2^\ell y$ satisfy $L(x', y') = L(x, y) + \ell$ and $d(x, y) \leq 3\ell \cdot |y|$.

(ii) For any symbols $0, 1$ (that may appear in x, y) set $x'' := 0^k1^k y1^\ell 0^k 1^k x$ and $y'' := 1^\ell 0^k 1^k y$ with $k := 2|y| + |x| + 1$. Then $L(x'', y'') = L(x, y) + \ell + 2k$ and $d(x, y) \leq \mathcal{O}(\ell(|x| + |y| + \ell))$.

7 Hardness for Large Alphabet

In this section, we consider a parameter setting $\boldsymbol{\alpha}$ satisfying the relations of Table 2, and we prove a lower bound for $\text{LCS}_\leq(\boldsymbol{\alpha})$ assuming OVH, thus proving Theorem 4.1. We split our proof into the two cases $\alpha_\delta = \alpha_m$ (where L may be small) and $\alpha_L = \alpha_m$ (where L is large). For readability, but abusing notation, for the target value $\lceil n^{\alpha_p} \rceil$ of parameter p we typically simply write p.

In this section we can assume that

$$\text{(LB)} \qquad \alpha_L, \alpha_m, \alpha_\delta, \alpha_\Delta > 0 \quad \text{and} \quad \alpha_d > 1,$$

since otherwise the known $\tilde{\mathcal{O}}(n + \min\{d, \delta m, \delta\Delta\})$ algorithm runs in (near-)optimal time $\tilde{\mathcal{O}}(n)$ and there is nothing to show (here we used the parameter relations $d \leq Lm$ and $L, m, \delta, \Delta \leq n$).

7.1 Small LCS Assume $\alpha_\delta = \alpha_m$, i.e., $\delta = \Theta(m)$. In this case, the longest common subsequence might be arbitrarily small, i.e., any value $0 < \alpha_L \leq \alpha_m$ is admissible. Due to space constraints, we omit the constructions in this case, which heavily rely on the previous reduction from OV to LCS of [28].

7.2 Large LCS Now assume $\alpha_L = \alpha_m$, i.e., $L = \Theta(m)$. Then the number of deletions in the shorter string might be arbitrary small, i.e., any value $0 < \alpha_\delta \leq \alpha_m$ is admissible. In this case, the construction of [28] is no longer applicable. The new 1vs1/2vs1 gadgets that we design for constructing hard strings for small δ can

be seen as one of our main contributions.

7.2.1 Hard Core The following lemma (which effectively represents an intermediate step in the proof of [28]) yields the basic method to embed sets of vectors into strings x and y.

LEMMA 7.1. *Let two sets $\mathcal{A} = \{a_1, \ldots, a_A\}$ and $\mathcal{B} = \{b_1, \ldots, b_B\}$ of vectors in $\{0,1\}^D$ be given. In time $\mathcal{O}((A+B)D)$ we can construct strings $x_1, \ldots, x_A$ of length ℓ_{X} and $y_1, \ldots, y_B$ of length ℓ_{Y} over alphabet $\{0,1\}$, as well as integers $\rho_1 < \rho_0$, such that for all $i \in [A], j \in [B]$ we have*

(i) $\ell_{\mathrm{Y}} \leq \ell_{\mathrm{X}} \leq \mathcal{O}(D)$,

(ii) $L(x_i, y_j) = \rho_0$ *if* $\langle a_i, b_j \rangle = 0$,

(iii) $L(x_i, y_j) = \rho_1$ *if* $\langle a_i, b_j \rangle \neq 0$, *and*

(iv) $L(x_i, y_j) > \ell_{\mathrm{Y}}/2$.

Proof. We can construct strings $x'_1, \ldots, x'_A$ of length $\ell'_{\mathrm{X}} = \mathcal{O}(D)$ and $y'_1, \ldots, y'_B$ of length $\ell'_{\mathrm{Y}} = \mathcal{O}(D)$ and integers $\rho'_1 < \rho'_0$ as in [28, Claim III.6] (using the so called normalized vector gadget) that satisfy $L(x'_i, y'_j) = \rho'_0$ if $\langle a_i, b_j \rangle = 0$ and $L(x'_i, y'_j) = \rho'_1$ otherwise. To additionally enforce conditions (i) and (iv), we define $x_i := 1^{\ell'_{\mathrm{Y}}} 0^{\ell'_{\mathrm{Y}}+1} x'_i$ and $y_j := 0^{\ell'_{\mathrm{Y}}+1} y'_j$. Since $L(x_i, y_j) = L(x'_i, y'_j) + \ell'_{\mathrm{Y}} + 1$ by Lemmas 6.2 and 6.1, we thus obtain conditions (ii) and (iii) for $\rho_0 := \rho'_0 + \ell'_{\mathrm{Y}} + 1$ and $\rho_1 := \rho'_1 + \ell'_{\mathrm{Y}} + 1$. Since by definition $\ell_{\mathrm{Y}} = 2\ell'_{\mathrm{Y}} + 1$ holds, the first condition follows directly and the trivial bound $L(x_i, y_j) \geq \ell'_{\mathrm{Y}} + 1 > \ell_{\mathrm{Y}}/2$ shows that the last condition is fulfilled.

1vs1/2vs1 gadget The aim of the following construction is to embed given strings $y_1, \ldots, y_Q$ into a string y and strings $x_1, \ldots, x_P$ into x, where $P = \Theta(Q)$, such that in an LCS each y_j is either aligned with a single string x_i or with several strings $x_i, x_{i+1}, \ldots, x_{i'}$. In the first case, $|y_j| - L(x_i, y_j)$ characters of y_j are not contained in an LCS of x and y, while in the second case y_j can be completely aligned. By choosing $P = 2Q - N$ for an arbitrary $1 \leq N \leq Q$, it will turn out that the LCS aligns N strings y_j with a single partner x_i, and the remaining $Q - N$ strings y_j with two strings x_i, x_{i+1} each. Thus, only N strings y_j are not completely aligned.

To formalize this intuition, let $P \geq Q$. We call a set $\Lambda = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ with $0 \leq k \leq Q$ and $1 \leq i_1 < i_2 < \cdots < i_k \leq P$ and $1 \leq j_1 \leq j_2 \leq \cdots \leq j_k \leq Q$ a *(partial) multi-alignment*. Let $\Lambda(j) = \{i \mid (i,j) \in \Lambda\}$. We say that every $j \in [Q]$ with $|\Lambda(j)| = k$ is *k-aligned*. We will also refer to a 1-aligned $j \in [Q]$ as being *uniquely aligned to* i, where $\Lambda(j) = \{i\}$. Every $j \in [Q]$ with $\Lambda(j) = \emptyset$ is called *unaligned*. Note that each $i \in [P]$ occurs in at most one $(i,j) \in \Lambda$. We denote the set of multi-alignments as $\mathbf{\Lambda}^{\mathrm{multi}}_{P,Q}$.

We will also need the following specialization of multi-alignments. We call a multi-alignment $\Lambda \in \mathbf{\Lambda}^{\mathrm{multi}}_{P,Q}$ a *(1,2)-alignment*, if each j is either 1-aligned or 2-aligned. Let $\mathbf{\Lambda}^{1,2}_{P,Q}$ denote the set of all (1,2)-alignments.

Given strings $x_1, \ldots, x_P$ of length ℓ_{X} and $y_1, \ldots, y_Q$ of length ℓ_{Y}, we define the *value* $v(\Lambda)$ of a multi-alignment $\Lambda \in \mathbf{\Lambda}^{\mathrm{multi}}_{P,Q}$ as $v(\Lambda) = \sum_{j=1}^{Q} v_j$ where

$$v_j := \begin{cases} 0 & \text{if } j \text{ is unaligned,} \\ L(x_i, y_j) & \text{if } j \text{ is uniquely aligned to } i, \\ \ell_{\mathrm{Y}} & \text{if } j \text{ is } k\text{-aligned for } k \geq 2. \end{cases}$$

LEMMA 7.2. *Given strings $x_1, \ldots, x_P$ of length ℓ_{X} and $y_1, \ldots, y_Q$ of length ℓ_{Y}, construct*

$$\begin{aligned} x &:= \mathrm{G}(x_1)\,\mathrm{G}(x_2)\,\ldots\,\mathrm{G}(x_P), \\ y &:= \mathrm{G}(y_1)\,\mathrm{G}(y_2)\,\ldots\,\mathrm{G}(y_Q), \end{aligned}$$

where $\mathrm{G}(w) := 0^{\gamma_1}\,1^{\gamma_2}\,(01)^{\gamma_3}\,w\,1^{\gamma_3}$ with $\gamma_3 := \ell_{\mathrm{X}} + \ell_{\mathrm{Y}}$, $\gamma_2 := 8\gamma_3$ and $\gamma_1 := 6\gamma_2$. Then we have

(7.1)
$$\max_{\Lambda \in \mathbf{\Lambda}^{1,2}_{P,Q}} v(\Lambda) \leq L(x,y) - Q(\gamma_1+\gamma_2+3\gamma_3) \leq \max_{\Lambda \in \mathbf{\Lambda}^{\mathrm{multi}}_{P,Q}} v(\Lambda).$$

Proof. For the first inequality of (7.1), let $\Lambda \in \mathbf{\Lambda}^{1,2}_{P,Q}$. For every y_j, we define $z_j = \bigcirc_{i \in \Lambda(j)} \mathrm{G}(x_i)$. Consider a 1-aligned j and let $i \in [P]$ be the index j is uniquely aligned to. We have that $z_j = \mathrm{G}(x_i) = 0^{\gamma_1}1^{\gamma_2}(01)^{\gamma_3}x_i 1^{\gamma_3}$ and hence by Lemma 6.1, we obtain $L(z_j, \mathrm{G}(y_j)) = \gamma_1+\gamma_2+3\gamma_3+L(x_i,y_j) = \gamma_1+\gamma_2+3\gamma_3+v_j$. Likewise, consider a 2-aligned j and let $i, i' \in [P]$ be such that $\Lambda(j) = \{i, i'\}$. Then $z_j = \mathrm{G}(x_i)\mathrm{G}(x_{i'})$. We compute

$$\begin{aligned} &L(z_j, \mathrm{G}(y_j)) \\ &\quad= \gamma_1 + \gamma_2 + 3\gamma_3 + L(x_i 1^{\gamma_3} 0^{\gamma_1} 1^{\gamma_2} (01)^{\gamma_3} x_{i'}, y_j) \\ &\quad\geq \gamma_1 + \gamma_2 + 3\gamma_3 + L((01)^{\gamma_3}, y_j) \\ &\quad= \gamma_1 + \gamma_2 + 3\gamma_3 + \ell_{\mathrm{Y}} = \gamma_1 + \gamma_2 + 3\gamma_3 + v_j, \end{aligned}$$

where the first line follows from Lemma 6.1, the second line from monotonicity and the third line from $\gamma_3 \geq \ell_{\mathrm{Y}} = |y_j|$. Observe that $z_1 z_2 \ldots z_Q$ is a subsequence of x. We conclude that

$$L(x,y) \geq \sum_{j=1}^{Q} L(z_j, \mathrm{G}(y_j)) = Q(\gamma_1+\gamma_2+3\gamma_3) + \sum_{j=1}^{Q} v_j.$$

It remains to prove the second inequality of (7.1). Write $x = z_1 z_2 \ldots z_Q$ such that $L(x,y) = \sum_{j=1}^{Q} L(z_j, \mathrm{G}(y_j))$. We define a multi-alignment Λ by

letting $(i,j) \in \Lambda$ if and only if z_j contains strictly more than half of the 0^{γ_1}-block of $\mathrm{G}(x_i)$. Note that the thus defined set satisfies the definition of a multi-alignment, since no two z_j's can contain more than half of $\mathrm{G}(x_i)$'s 0^{γ_1}-block and if $(i,j),(i',j') \in \Lambda$, then $j < j'$ implies $i < i'$. It remains to show that $L(z_j, \mathrm{G}(y_j)) \leq \gamma_1 + \gamma_2 + 3\gamma_3 + v_j$ for all j to prove the claim.

In what follows, we use the shorthand $H(w) := 1^{\gamma_2}(01)^{\gamma_3} w 1^{\gamma_3}$. Note that $G(w) = 0^{\gamma_1} H(w)$. Consider an unaligned $j \in [Q]$. By definition, z_j is a subsequence of $0^{\gamma_1/2} H(x_i) 0^{\gamma_1/2}$ for some $i \in [P]$. We can thus bound (using Lemma 6.1)

$$\begin{aligned} L(z_j, \mathrm{G}(y_j)) &\leq L(0^{\gamma_1/2} H(x_i) 0^{\gamma_1/2}, 0^{\gamma_1} H(y_j)) \\ &= \frac{\gamma_1}{2} + L(H(x_i) 0^{\gamma_1/2}, 0^{\gamma_1/2} H(y_j)). \end{aligned}$$

By Lemma 6.2 with $\ell := \gamma_1/2 \geq 2\gamma_2 + 6\gamma_3 + \ell_{\mathrm{X}} + \ell_{\mathrm{Y}} = |H(x_i)| + |H(y_j)| \geq \#_0(H(x_i)) + |H(y_j)|$, we have that $L(H(x_i)0^{\gamma_1/2}, 0^{\gamma_1/2}H(y_j))$ equals

$$\begin{aligned} \gamma_1/2 + L(0^{\#_0(H(x_i))}, H(y_j)) &\leq \gamma_1/2 + \#_0(H(y_j)) \\ &\leq \gamma_1/2 + \gamma_3 + \ell_{\mathrm{Y}}. \end{aligned}$$

Hence, in total we have $L(z_j, \mathrm{G}(y_j)) \leq \gamma_1 + \gamma_3 + \ell_{\mathrm{Y}} \leq \gamma_1 + \gamma_2 + 3\gamma_3 = \gamma_1 + \gamma_2 + 3\gamma_3 + v_j$, as desired.

Consider a $j \in [Q]$ that is uniquely aligned (under Λ) to some i. Then z_j is a subsequence of $0^{\gamma_1/2} H(x_{i-1}) 0^{\gamma_1} H(x_i) 0^{\gamma_1/2}$. Analogously to above we compute

$$\begin{aligned} & L(z_j, \mathrm{G}(y_j)) \\ &\leq \frac{\gamma_1}{2} + L(H(x_{i-1}) 0^{\gamma_1} H(x_i) 0^{\gamma_1/2}, 0^{\gamma_1/2} H(y_j)) \\ &= \gamma_1 + L(0^{\#_0(H(x_{i-1})) + \gamma_1} H(x_i) 0^{\gamma_1/2}, H(y_j)) \\ &= \gamma_1 + L(0^{\#_0(H(x_{i-1})) + \gamma_1} 1^{\gamma_2} (01)^{\gamma_3} x_i 1^{\gamma_3} 0^{\gamma_1/2}, \\ &\qquad 1^{\gamma_2} (01)^{\gamma_3} y_j 1^{\gamma_3}). \end{aligned}$$

Using Lemma 6.2 with symbol 0 replaced by 1 yields, since $\ell := \gamma_2 \geq 3\gamma_3 + \ell_{\mathrm{Y}} = |(01)^{\gamma_3} y_j 1^{\gamma_3}|$ and $\#_1(0^{\#_0(H(x_{i-1}))+\gamma_1}) = 0$,

$$\begin{aligned} & L(z_j, \mathrm{G}(y_j)) \\ &\leq \gamma_1 + \gamma_2 + L((01)^{\gamma_3} x_i 1^{\gamma_3} 0^{\gamma_1/2}, (01)^{\gamma_3} y_j 1^{\gamma_3}) \\ &= \gamma_1 + \gamma_2 + 2\gamma_3 + L(x_i 1^{\gamma_3} 0^{\gamma_1/2}, y_j 1^{\gamma_3}). \end{aligned}$$

Similarly, using Lemma 6.2 with symbol 0 replaced by 1 on the reversed strings yields, since $\ell := \gamma_3 \geq \ell_{\mathrm{Y}} = |y_j|$ and $\#_1(0^{\gamma_1/2}) = 0$,

$$L(x_i 1^{\gamma_3} 0^{\gamma_1/2}, y_j 1^{\gamma_3}) = \gamma_3 + L(x_i, y_j).$$

Hence, we obtain the desired $L(z_j, \mathrm{G}(y_j)) \leq \gamma_1 + \gamma_2 + 3\gamma_3 + L(x_i, y_j) = \gamma_1 + \gamma_2 + 3\gamma_3 + v_j$.

It remains to consider $j \in [Q]$ that is k-aligned for $k \geq 2$. In this case, the claim follows from the trivial bound $L(z_j, \mathrm{G}(y_j)) \leq |\mathrm{G}(y_j)| = \gamma_1 + \gamma_2 + 3\gamma_3 + v_j$.

Thus $z_1, \ldots, z_Q$ defines a multi-alignment $\Lambda \in \mathbf{\Lambda}^{\mathrm{multi}}_{P,Q}$ with

$$L(x,y) = \sum_{j=1}^{Q} L(z_j, \mathrm{G}(y_j)) \leq Q(\gamma_1 + \gamma_2 + 3\gamma_3) + v(\Lambda),$$

proving the second inequality of (7.1).

We can now show how to embed an OV instance $\mathcal{A} = \{a_1, \ldots, a_A\}, \mathcal{B} = \{b_1, \ldots, b_B\} \subseteq \{0,1\}^D$ with $A \leq B$ into strings x and y of length $\mathcal{O}(B \cdot D)$ whose LCS can be obtained by deleting at most $\mathcal{O}(A \cdot D)$ symbols from y. For this we will without loss of generality assume that A divides B by possibly duplicating some arbitrary element of $\mathcal{B}$ up to $A - 1$ times without affecting the solution of the instance.

The key idea is that for any P and $Q = 2P - N$ with $N \in \{0, \ldots, P\}$, $\mathbf{\Lambda}^{1,2}_{P,Q}$ is non-empty and each $\Lambda \in \mathbf{\Lambda}^{1,2}_{P,Q}$ has exactly N uniquely aligned $j \in [Q]$ and exactly $P - N$ 2-aligned $j \in [Q]$. At the same time each $\Lambda \in \mathbf{\Lambda}^{\mathrm{multi}}_{P,Q}$ leaves at least N indices $j \in [Q]$ either unaligned or uniquely aligned.

LEMMA 7.3. *Let $a_1, \ldots, a_A, b_1, \ldots b_B \subseteq \{0,1\}^D$ be given with $A \mid B$. Construct the corresponding strings $x_1, \ldots, x_A$ of length ℓ_{X}, $y_1, \ldots, y_B$ of length $\ell_{\mathrm{Y}} \leq \ell_{\mathrm{X}} \leq \mathcal{O}(D)$, and integers ρ_0, ρ_1 as in Lemma 7.1 and define*

$$\begin{aligned} \tilde{x} &:= (\tilde{x}_1, \ldots, \tilde{x}_P) \\ &= (\underbrace{x_1, \ldots, x_A, x_1, \ldots, x_A, \ldots, x_1, \ldots, x_A}_{2\cdot(B/A)+3 \text{ groups of size } A}), \\ \tilde{y} &:= (\tilde{y}_1, \ldots, \tilde{y}_Q) \\ &= (\underbrace{y_1, \ldots, y_1}_{A \text{ copies of } y_1}, y_1, \ldots, y_B, \underbrace{y_1, \ldots, y_1}_{A \text{ copies of } y_1}), \end{aligned}$$

where $P := 2B + 3A$ and $Q := B + 2A$. Then the instance $x := \bigcirc_i \mathrm{G}(\tilde{x}_i)$, $y := \bigcirc_j \mathrm{G}(\tilde{y}_j)$ of Lemma 7.2 (with the corresponding choice of γ_1, γ_2 and γ_3) satisfies the following properties:

(i) For every $i \in [A], j \in [B]$, there is a (1,2)-alignment $\Lambda \in \mathbf{\Lambda}^{1,2}_{P,Q}$ such that some $\ell \in [Q]$ is uniquely aligned to some $k \in [P]$ with $\tilde{x}_k = x_i$ and $\tilde{y}_\ell = y_j$.

(ii) We have $L(x,y) \geq Q(\gamma_1+\gamma_2+3\gamma_3)+(A-1)\rho_1+\rho_0+(Q-A)\ell_{\mathrm{Y}}$ if and only if there are $i \in [A], j \in [B]$ with $\langle a_i, b_j \rangle = 0$.

(iii) We have $|y| \leq |x| \leq \mathcal{O}(B \cdot D)$ and $\delta(x,y) = \mathcal{O}(A \cdot D)$.

Proof. For (i), we let $j \in [B]$ and note that $y_j = \tilde{y}_\ell$ for $\ell := A + j$. We will show that for every $\lambda \in \{0, \dots, A-1\}$, there is a (1,2)-alignment Λ with $(k, \ell) \in \mathbf{\Lambda}^{1,2}_{P,Q}$ where $k := 2(A+j) - 1 - \lambda$. By the cyclic structure of $\tilde{x}$, $(\tilde{x}_k)_{0 \le \lambda < A}$ cycles through all values $x_1, \dots, x_A$. Hence, for some choice of λ the desired $\tilde{x}_k = x_i$ follows, yielding the claim.

To see that for any $\lambda \in \{0, \dots, A-1\}$, some $\Lambda \in \mathbf{\Lambda}^{1,2}_{P,Q}$ with $(k, \ell) \in \Lambda$ exists, observe that there are $\ell - 1 = A + j - 1$ predecessors of $\tilde{y}_\ell$ and $k - 1 = 2(A + j - 1) - \lambda = 2(\ell - 1) - \lambda$ predecessors of $\tilde{x}_k$. Hence there is a (1,2)-alignment $\Lambda_1 \in \mathbf{\Lambda}^{1,2}_{k-1,\ell-1}$ (leaving λ indices $j \in [Q]$ uniquely aligned). Similarly, observe that there are $Q - \ell = B + A - j$ successors of $\tilde{y}_\ell$ and $P - k = 2B + A - 2j + \lambda + 1 = 2(Q - \ell) - (A - \lambda - 1)$ successors of $\tilde{x}_k$, hence there is a (1,2)-alignment $\Lambda_2 \in \mathbf{\Lambda}^{1,2}_{P-k,Q-\ell}$ (which leaves $A - (\lambda + 1)$ indices j uniquely aligned). By canonically composing Λ_1, (k, ℓ) and Λ_2 we can thus obtain $\Lambda \in \mathbf{\Lambda}^{1,2}_{P,Q}$ with $(k, \ell) \in \Lambda$.

For (ii), assume that there are $i \in [A], j \in [B]$ satisfying $\langle a_i, b_j \rangle = 0$. By (i), there is some $\Lambda \in \mathbf{\Lambda}^{1,2}_{P,Q}$ where some $\ell \in [Q]$ is uniquely aligned to some $k \in [P]$ such that $\tilde{x}_k = x_i$ and $\tilde{y}_\ell = y_j$. To apply Lemma 7.2, observe that Λ has $Q - A$ 2-aligned $j \in [Q]$, which contribute value ℓ_{Y} to $v(\Lambda)$, and A uniquely aligned $j \in [Q]$, in particular, ℓ is uniquely aligned to k. Since any $\tilde{x}_i$ corresponds to some $x_{i'}$, every $\tilde{y}_j$ corresponds to some $y_{j'}$ and $L(x_{i'}, y_{j'}) \in \{\rho_0, \rho_1\}$, we conclude that ℓ contributes ρ_0 to $v(\Lambda)$ and the other $A - 1$ uniquely aligned j contribute at least ρ_1. Hence by the lower bound in Lemma 7.2, we obtain $L(x, y) \ge Q(\gamma_1 + \gamma_2 + 3\gamma_3) + v(\Lambda)$, where $v(\Lambda) \ge (A-1)\rho_1 + \rho_0 + (Q - A)\ell_{\mathrm{Y}}$.

Assume now that no $i \in [A], j \in [B]$ satisfy $\langle a_i, b_j \rangle = 0$, and let $\Lambda \in \mathbf{\Lambda}^{\mathrm{multi}}_{P,Q}$. Then any $j \in [Q]$ uniquely aligned to some $i \in [P]$ contributes $L(\tilde{x}_i, \tilde{y}_j) = \rho_1$ to $v(\Lambda)$. Let λ be the number of $j \in [Q]$ that are k-aligned for any $k \ge 2$, each contributing ℓ_{Y} to $v(\Lambda)$. Then there are at most $\min\{P - 2\lambda, Q - \lambda\}$ uniquely aligned $j \in [Q]$ (since every k-aligned j blocks at least two $i \in [P]$ for other alignments), and the remaining $j \in [Q]$ are unaligned, with no contribution to $v(\Lambda)$. Hence $v(\Lambda) \le \lambda\ell_{\mathrm{Y}} + \min\{P - 2\lambda, Q - \lambda\} \cdot \rho_1 = \min\{P\rho_1 + (\ell_{\mathrm{Y}} - 2\rho_1)\lambda, Q\rho_1 + (\ell_{\mathrm{Y}} - \rho_1)\lambda\}$. Note that $\ell_{\mathrm{Y}}/2 < \rho_1 \le \ell_{\mathrm{Y}}$ (by Lemma 7.1(iv)), hence this minimum of linear functions with leading coefficients $\ell_{\mathrm{Y}} - 2\rho_1 < 0$ and $\ell_{\mathrm{Y}} - \rho_1 \ge 0$ is maximized when both have the same value, i.e., when $\lambda = P - Q = Q - A$. Thus, $v(\Lambda) \le (Q - A)\ell_{\mathrm{Y}} + A\rho_1 < (Q - A)\ell_{\mathrm{Y}} + (A - 1)\rho_1 + \rho_0$. Thus by the upper bound of Lemma 7.2 we conclude that $L(x, y) < Q(\gamma_1 + \gamma_2 + 3\gamma_3) + (Q - A)\ell_{\mathrm{Y}} + (A - 1)\rho_1 + \rho_0$.

For (iii), since $P \ge Q$ and $\ell_{\mathrm{X}} \ge \ell_{\mathrm{Y}}$ we have $|x| \ge |y|$, and by $P \le \mathcal{O}(A)$ and $|G(\tilde{x}_i)| \le \mathcal{O}(\ell_{\mathrm{X}}) \le \mathcal{O}(D)$ we obtain $|x| \le \mathcal{O}(AD)$. Note that for any (1,2)-alignment $\Lambda \in \mathbf{\Lambda}^{1,2}_{P,Q}$, we have

$$\begin{aligned} v(\Lambda) &= Q \cdot \ell_{\mathrm{Y}} - \sum_{j \text{ uniquely aligned to } i} (\ell_{\mathrm{Y}} - L(x_i, y_j)) \\ &= Q \cdot \ell_{\mathrm{Y}} - \mathcal{O}(A \cdot D), \end{aligned}$$

since by $P = 2Q - A$ the number of uniquely aligned indices j in Λ equals A, and $\ell_{\mathrm{Y}} = \mathcal{O}(D)$. Hence by Lemma 7.2, $L(x, y) \ge Q(\gamma_1 + \gamma_2 + 3\gamma_3) + Q\ell_{\mathrm{Y}} - \mathcal{O}(A \cdot D) = |y| - \mathcal{O}(A \cdot D)$, implying $\delta(x, y) = |y| - L(x, y) \le \mathcal{O}(A \cdot D)$.

7.2.2 Constant Alphabet First assume $\alpha_\Sigma = 0$ and thus $|\Sigma| = \mathcal{O}(1)$. Consider any $n \ge 1$ and target values $p = n^{\alpha_p}$ for $p \in \mathcal{P}$. We write $\lfloor x \rfloor_2$ for the largest power of 2 less than or equal to x. Let $\mathcal{A} = \{a_1, \dots, a_A\}$, $\mathcal{B} = \{b_1, \dots, b_B\} \subseteq \{0,1\}^D$ be a given OV instance with $D = n^{o(1)}$ and where we set

$$A := \left\lfloor \frac{1}{D} \min\left\{\delta, \frac{d}{\min\{m, \Delta\}}\right\} \right\rfloor_2,$$

$$B := \left\lfloor \frac{1}{D} \min\{m, \Delta\} \right\rfloor_2.$$

By $\alpha_m, \alpha_\Delta \le 1$ and (LB) we obtain $A \ge n^{\min\{\alpha_L, \alpha_d - 1\} - o(1)} = n^{\Omega(1)}$ and $B = n^{\min\{\alpha_m, \alpha_\Delta\} - o(1)} = n^{\Omega(1)}$. Also note that UOVH implies that solving such OV instances takes time $(AB)^{1-o(1)} = \min\{d, \delta m, \delta\Delta\}^{1-o(1)}$, which is the desired bound. We claim that $A \le B$, implying $A \mid B$. Indeed, if $\delta \le d/\min\{m, \Delta\}$ this follows from the simple parameter relations $\delta \le m$ and $\delta \le \Delta$. Otherwise, if $\delta > d/\min\{m, \Delta\}$, then in particular $\delta\Delta > d$, implying $d < \Delta^2$. Together with the parameter relations $d \le Lm \le m^2$ we indeed obtain $d/\min\{m, \Delta\} \le \min\{m, \Delta\}$.

Thus, we may construct strings x, y as in Lemma 7.3. We finish the construction by invoking the Dominant Pair Reduction (Lemma 6.4) to obtain strings $x' := 0^k 1^k y 1^\ell 0^k 1^k x$ and $y' := 1^\ell 0^k 1^k y$ with $k := 2|y| + |x| + 1$ and $\ell := \Theta(A \cdot D)$ with sufficiently large hidden constant, so that $\ell > \delta(x, y)$. Then from the LCS length $L(x', y')$ we can infer whether $\mathcal{A}, \mathcal{B}$ has an orthogonal pair of vectors by $L(x', y') = L(x, y) + \ell + 2k$ and Lemma 7.3(ii). Moreover, this reduction runs in time $\mathcal{O}(|x'| + |y'|) = \mathcal{O}(|x| + |y|) = \mathcal{O}(BD) \le \mathcal{O}(\min\{d, \delta m, \delta\Delta\}^{1-\varepsilon})$ for sufficiently small $\varepsilon > 0$ (since $\alpha_\delta > 0$ and $\alpha_d > 1 \ge \alpha_m, \alpha_\Delta$ by (LB) and Table 2). We claim that (n, x', y') is an instance of $\mathrm{LCS}_{\le}(\boldsymbol{\alpha})$. This shows that any algorithm solving $\mathrm{LCS}_{\le}(\boldsymbol{\alpha})$ in time $\mathcal{O}(\min\{d, \delta m, \delta\Delta\}^{1-\varepsilon})$ implies an algorithm for our OV instances with running time $\mathcal{O}(\min\{d, \delta m, \delta\Delta\}^{1-\varepsilon})$, contradicting UOVH. Hence, in the current case $\alpha_L = \alpha_m$ and $\alpha_\Sigma = 0$, any algorithm for $\mathrm{LCS}_{\le}(\boldsymbol{\alpha})$ takes time $\min\{d, \delta m, \delta\Delta\}^{1-o(1)}$, proving part of Theorem 4.1.

It remains to show the claim that (n, x', y') is an instance of $\mathrm{LCS}_{\leq}(\boldsymbol{\alpha})$. From Lemmas 6.4 and 7.3(iii) we obtain $L(x', y') = \ell + 2k + L(x, y) = \ell + 2k + |y| - \delta(x, y) \geq |y'| - \mathcal{O}(AD)$, and thus $\delta(x', y') \leq \mathcal{O}(AD) \leq \mathcal{O}(\delta)$. Using the parameter relations $L \leq m \leq n$, Lemma 7.3(iii), and the definition of B, we have $L(x', y') \leq m(x', y') \leq n(x', y') = |x'| \leq \mathcal{O}(BD) = \mathcal{O}(\min\{m, \Delta\})$, which together with the relation $m \leq n$ and the assumption $\alpha_L = \alpha_m$ shows that $p(x', y') \leq \mathcal{O}(p) = \mathcal{O}(n^{\alpha_p})$ for $p \in \{L, m, n\}$. Similarly, we obtain $\Delta(x', y') \leq n(x', y') \leq \mathcal{O}(\min\{m, \Delta\}) \leq \mathcal{O}(\Delta)$. Since x', y' use the binary alphabet $\{0, 1\}$, we have $|\Sigma(x', y')| = 2 \leq \mathcal{O}(n^{\alpha_\Sigma})$. For the number of matching pairs we have $M(x', y') \leq n(x', y')^2 = \mathcal{O}((BD)^2) = \mathcal{O}(L^2)$. Since we are in the case $\alpha_\Sigma = 0$, from the parameter relation $M \geq L^2/|\Sigma|$ we obtain $L^2 \leq \mathcal{O}(M)$ and thus also $M(x', y')$ is sufficiently small. Finally, we use Lemma 6.4 to bound $d(x', y') \leq \mathcal{O}(\ell \cdot |y|) \leq \mathcal{O}(AD \cdot BD)$, which by definition of A, B is $\mathcal{O}(d)$. This proves that (n, x', y') belongs to $\mathrm{LCS}_{\leq}(\boldsymbol{\alpha})$.

7.2.3 Superconstant Alphabet The crucial step in extending our construction to larger alphabets is to adapt the 1vs1/2vs1 gadget such that the strings use each symbol in the alphabet Σ roughly evenly, thus reducing the number of matching pairs by a factor $|\Sigma|$.

Recall that given a 2-element alphabet Σ' and a string z over $\{0, 1\}$, we let $z{\uparrow}\Sigma'$ denote the string z lifted to alphabet Σ' by bijectively replacing $\{0, 1\}$ with Σ'.

LEMMA 7.4. *Let $P = 2B + 3A$ and $Q = B + 2A$ for some $A \mid B$. Given strings $x_1, \ldots, x_P$ of length ℓ_{X} and $y_1, \ldots, y_Q$ of length ℓ_{Y}, we define, as in Lemma 7.2, $\mathrm{G}(w) := 0^{\gamma_1}\, 1^{\gamma_2}\, (01)^{\gamma_3}\, w\, 1^{\gamma_3}$ with $\gamma_3 := \ell_{\mathrm{X}} + \ell_{\mathrm{Y}}$, $\gamma_2 := 8\gamma_3$ and $\gamma_1 := 6\gamma_2$. Let $\Sigma_1, \ldots, \Sigma_t$ be disjoint alphabets of size 2 with $Q/t \geq A/2 + 1$. We define*

$$\begin{aligned} x &:= H(x_1)\, H(x_2)\, \ldots\, H(x_P), \\ y &:= \mathrm{G}(y_1){\uparrow}\Sigma_{f(1)}\, \mathrm{G}(y_2){\uparrow}\Sigma_{f(2)}\, \ldots\, \mathrm{G}(y_Q){\uparrow}\Sigma_{f(Q)}, \end{aligned}$$

where $f(j) = \lceil \frac{j}{Q} \cdot t \rceil$ and

$$H(x_i) := \begin{cases} \mathrm{G}(x_i){\uparrow}\Sigma_{k+1} \circ \mathrm{G}(x_i){\uparrow}\Sigma_k, \\ \qquad \text{if } \bigcup_{j=\lceil i/2 \rceil}^{\lfloor (i+A)/2 \rfloor} \{f(j)\} = \{k, k+1\}, \\ \mathrm{G}(x_i){\uparrow}\Sigma_k, \\ \qquad \text{if } \bigcup_{j=\lceil i/2 \rceil}^{\lfloor (i+A)/2 \rfloor} \{f(j)\} = \{k\}. \end{cases}$$

Then we have

(7.2)
$$\max_{\Lambda \in \boldsymbol{\Lambda}^{1,2}_{P,Q}} v(\Lambda) \leq L(x, y) - Q(\gamma_1 + \gamma_2 + 3\gamma_3) \leq \max_{\Lambda \in \boldsymbol{\Lambda}^{\mathrm{multi}}_{P,Q}} v(\Lambda).$$

Proof. Note that $H(\cdot)$ is well-defined, since $f(\cdot)$ maps $\{1, \ldots, Q\}$ to constant-valued intervals of length at least $Q/t - 1 \geq A/2$, as $f(j) = k$ if and only if $j \in \left(\frac{Qk}{t} - \frac{Q}{t}, \frac{Qk}{t}\right]$, containing at least $Q/t - 1$ integers. Hence for every i, the $\leq A/2$ values $f(\lceil i/2 \rceil), \ldots, f(\lfloor (i+A)/2 \rfloor)$ can touch at most 2 different constant-valued intervals.

The proof of (7.2) is based on the proof of Lemma 7.2 (the analogous lemma for alphabet $\Sigma = \{0, 1\}$). For the first inequality of (7.2), let $\Lambda \in \boldsymbol{\Lambda}^{1,2}_{P,Q}$ and define for every j the substring $z'_j = \bigcirc_{i \in \Lambda(j)} H(x_i)$. Note that under Λ, each $\Lambda(j)$ consists of one or two elements from $\{2j - A, \ldots, 2j\}$, since there are at most $2Q - P = A$ uniquely aligned j. In other words, for any $i \in \Lambda(j)$ we have $j \in \{\lceil i/2 \rceil, \ldots, \lfloor (i + A)/2 \rfloor\}$. Thus, by definition each $H(x_i)$ for $i \in \Lambda(j)$ contains $\mathrm{G}(x_i){\uparrow}\Sigma_{f(j)}$ as a substring and hence z'_j contains $\bigcirc_{i \in \Lambda(j)} \mathrm{G}(x_i){\uparrow}\Sigma_{f(j)}$ as a subsequence. This proves

$$\begin{aligned} & L\big(z'_j, \mathrm{G}(y_j){\uparrow}\Sigma_{f(j)}\big) \\ &\quad \geq L\Big(\bigcirc_{i \in \Lambda(j)} \mathrm{G}(x_i){\uparrow}\Sigma_{f(j)}, \mathrm{G}(y_j){\uparrow}\Sigma_{f(j)}\Big) \\ &\quad = L\Big(\bigcirc_{i \in \Lambda(j)} \mathrm{G}(x_i), \mathrm{G}(y_j)\Big), \end{aligned}$$

which reduces the proof to the case of $\Sigma = \{0, 1\}$ – note that the last term is equal to $L(z_j, \mathrm{G}(y_j))$ in the proof of the same inequality of Lemma 7.2 and thus the remainder follows verbatim.

It remains to show the second inequality of (7.2). Essentially as in the proof of Lemma 7.2, we write $x = z'_1 z'_2 \ldots z'_Q$ with $L(x, y) = \sum_{j=1}^{Q} L(z'_j, \mathrm{G}(y_j){\uparrow}\Sigma_{f(j)})$. For every z'_j, we obtain a string z_j by deleting all symbols not contained in $\Sigma_{f(j)}$ and then lifting it to the alphabet $\{0, 1\}$. We conclude that $L(z'_j, \mathrm{G}(y_j){\uparrow}\Sigma_{f(j)}) = L(z_j, \mathrm{G}(y_j))$. We claim that $z := z_1 z_2 \ldots z_Q$ is a subsequence of $x_{\{0,1\}} := \mathrm{G}(x_1) \ldots \mathrm{G}(x_P)$ (which is equal to the string x that we constructed in the case of $\Sigma = \{0, 1\}$). Indeed, if $H(x_i)$ is of the form $w_{k+1} w_k$ for some k with $w_\ell = \mathrm{G}(x_i){\uparrow}\Sigma_\ell$, then symbols of at most one of w_k and w_{k+1} are contained in z. To see this, note that if w_k is not deleted then at least one of its symbols is contained in some z'_j with $f(j) = k$, but then no symbol in w_{k+1} can be contained in $z'_{j'}$ with $f(j') = k + 1$, since this would mean $j' > j$, so w_{k+1} is deleted. Thus,

$$\begin{aligned} L(x, y) &= \sum_{j=1}^{Q} L\big(z'_j, \mathrm{G}(y_j){\uparrow}\Sigma_{f(j)}\big) \\ &= \sum_{j=1}^{Q} L\big(z_j, \mathrm{G}(y_j)\big) \leq L(x_{\{0,1\}}, y_{\{0,1\}}), \end{aligned}$$

where $y_{\{0,1\}} := \mathrm{G}(y_1) \ldots \mathrm{G}(y_Q)$ is the string y that we

constructed in the case of $\Sigma = \{0,1\}$. Hence, the second inequality of (7.2) follows from the proof of Lemma 7.2.

By the same choice of vectors as in Lemma 7.3, we can embed orthogonal vectors instances.

LEMMA 7.5. *Let $a_1, \ldots, a_A, b_1, \ldots b_B \subseteq \{0,1\}^D$ be given with $A \mid B$. Construct $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_P), \tilde{y} = (\tilde{y}_1, \ldots, \tilde{y}_Q)$ with $P := 2B + 3A$ and $Q := B + 2A$ as in Lemma 7.3. For disjoint alphabets $\Sigma_1, \ldots, \Sigma_t$ of size 2 with $Q/t \geq A/2 + 1$, we construct the instance $x := \bigcirc_i H(\tilde{x}_i)$, $y := \bigcirc_j \mathrm{G}(\tilde{y}_j)$ of Lemma 7.4 (with the corresponding choice of γ_1, γ_2 and γ_3). This satisfies the following properties:*

(i) We have that $L(x,y) \geq Q(\gamma_1+\gamma_2+3\gamma_3)+(A-1)\rho_1+\rho_0+(Q-A)\ell_\Upsilon$ if and only if there are $i \in [A], j \in [B]$ with $\langle a_i, b_j \rangle = 0$.

(ii) We have $|y| \leq |x| \leq \mathcal{O}(B \cdot D)$ and $\delta(x,y) = \mathcal{O}(A \cdot D)$.

Proof. The lemma and its proof are a slight adaptation of Lemma 7.3: For (i), since Lemma 7.4 proves (7.2) which is identical to (7.1), we can follow the proof of Lemma 7.3(i) and (ii) verbatim (since we have chosen $\tilde{x}$ and $\tilde{y}$ as in this lemma). For (ii), the bounds $|y| \leq |x| \leq \mathcal{O}(B \cdot D)$ and $\delta(x,y) = \mathcal{O}(A \cdot D)$ follow exactly as in Lemma 7.2 (note that only $|x|$ has increased by at most a factor of 2, so that $|x| \leq \mathcal{O}(B \cdot D)$ still holds by the trivial bound).

We can now finish the proof of Theorem 4.1 for the case of $\alpha_L = \alpha_m$ and $\alpha_\Sigma > 0$. Consider any $n \geq 1$ and target values $p = n^{\alpha_p}$ for $p \in \mathcal{P}$. Let $\mathcal{A} = \{a_1, \ldots, a_A\}$, $\mathcal{B} = \{b_1, \ldots, b_B\} \subseteq \{0,1\}^D$ be a given OV instance with $D = n^{o(1)}$ and where we set, as in the case $\alpha_\Sigma = 0$,

$$A := \left\lfloor \frac{1}{D} \min\left\{\delta, \frac{d}{\min\{m, \Delta\}}\right\} \right\rfloor_2$$
$$B := \left\lfloor \frac{1}{D} \min\{m, \Delta\} \right\rfloor_2.$$

As before, we have $A \mid B$, so we may construct strings x, y as in Lemma 7.5, where we set $t := \min\{\lfloor Q/(A/2+1) \rfloor, |\Sigma|\} = \Theta(\min\{B/A, |\Sigma|\})$. We finish the construction by invoking the Dominant Pair Reduction (Lemma 6.4) to obtain strings $x' := y2^\ell x$ and $y' := 2^\ell y$, where 2 is a symbol not appearing in x, y and we set $\ell := \Theta(A \cdot D)$ with sufficiently large hidden constant, so that $\ell > \delta(x,y)$.

For the remainder of the proof we can follow the case $\alpha_\Sigma = 0$ almost verbatim. The only exception is the bound on the number of matching pairs. Note that symbol 2 appears $\mathcal{O}(AD)$ times in x' and y'. As in x and y every symbol appears roughly equally often and the total alphabet size is $\Theta(t)$, for any symbol $\sigma \neq 2$ we have $\#_\sigma(x) \leq \mathcal{O}(|x|/t)$ and $\#_\sigma(y) \leq \mathcal{O}(|y|/t)$, implying $\#_\sigma(x'), \#_\sigma(y') \leq \mathcal{O}(BD/t)$. Hence, $M(x', y') \leq \mathcal{O}((AD)^2 + t \cdot (BD/t)^2)$. Using $t = \Theta(\min\{B/A, |\Sigma|\})$ and $A \leq B$, we obtain $M(x', y') \leq \mathcal{O}(\max\{AD \cdot BD, (BD)^2/|\Sigma|\}) \leq \mathcal{O}(\max\{d, m^2/|\Sigma|\}$. The assumption $\alpha_L = \alpha_m$ and the parameter relations $M \geq L^2/|\Sigma|$ and $M \geq d$ now imply $M(x', y') \leq \mathcal{O}(M)$. This concludes the proof of Theorem 4.1.

8 Discussion

We present a systematic study of SETH-based lower bounds for special cases of the longest common subsequence problem that are defined by polynomial restrictions of all 7 previously studied input parameters. Our tight conditional lower bounds completely explain the lack of polynomial time improvements since 1990, except for a special regime on $\Sigma = \{0,1\}$, for which we design an improved algorithm matching our lower bound. We conclude that to obtain polynomially faster algorithms one has to either (1) refute the Strong Exponential Time Hypothesis or (2) design new reasonable and algorithmically tractable input parameters.

This work showcases the paradigm of *multivariate fine-grained complexity* on a classic problem with 7 parameters that display a complex set of relations. We believe that this paradigm can be applied to further problems yielding similar systematic insights and helping to find algorithmic improvements for natural restricted-input instances.

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 and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS'14)*, pages 434–443, 2014.

[4] Amir Abboud, Virginia Vassilevska Williams, and

Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*, pages 377–391, 2016.

[5] Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 218–230, 2015.

[6] Muhammad Rashed Alam and M. Sohel Rahman. The substring inclusion constraint longest common subsequence problem can be solved in quadratic time. *Journal of Discrete Algorithms*, 17:67–73, 2012.

[7] Jochen Alber, Jens Gramm, Jiong Guo, and Rolf Niedermeier. Towards optimally solving the longest common subsequence problem for sequences with nested arc annotations in linear time. In *Proc. 13th Annual Symposium on Combinatorial Pattern Matching (CPM'02)*, pages 99–114, 2002.

[8] 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.

[9] Amihood Amir, Zvi Gotthilf, and B. Riva Shalom. Weighted LCS. *Journal of Discrete Algorithms*, 8(3):273–281, 2010.

[10] Amihood Amir, Tzvika Hartman, Oren Kapah, B. Riva Shalom, and Dekel Tsur. Generalized LCS. *Theoretical Computer Science*, 409(3):438–449, 2008.

[11] Amihood Amir, Haim Paryenty, and Liam Roditty. On the hardness of the consensus string problem. *Information Processing Letters*, 113(10-11):371–374, 2013.

[12] Amihood Amir, Haim Paryenty, and Liam Roditty. Configurations and minority in the string consensus problem. *Algorithmica*, 74(4):1267–1292, 2016.

[13] Alberto Apostolico. Improving the worst-case performance of the Hunt-Szymanski strategy for the longest common subsequence of two strings. *Information Processing Letters*, 23(2):63–69, 1986.

[14] Alberto Apostolico and Concettina Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2:316–336, 1987.

[15] Alberto Apostolico, Gad M. Landau, and Steven Skiena. Matching for run-length encoded strings. In *Proc. 1997 International Conference on Compression and Complexity of Sequences (SEQUENCES'97)*, pages 348–356, 1997.

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

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

[18] Ricardo A. Baeza-Yates, Ricard Gavaldá, Gonzalo Navarro, and Rodrigo Scheihing. Bounding the expected length of longest common subsequences and forests. *Theory of Computing Systems*, 32(4):435–452, 1999.

[19] Nikhil Bansal, Moshe Lewenstein, Bin Ma, and Kaizhong Zhang. On the longest common rigid subsequence problem. *Algorithmica*, 56(2):270–280, 2010.

[20] David Becerra, Juan Mendivelso, and Yoan Pinzón. A multiobjective optimization algorithm for the weighted LCS. *Discrete Applied Mathematics*, 212:37–47, 2016.

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

[22] 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.

[23] Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486 – 496, 2008.

[24] Guillaume Blin, Paola Bonizzoni, Riccardo Dondi, and Florian Sikora. On the parameterized complexity of the repetition free longest common subsequence problem. *Information Processing Letters*, 112(7):272–276, 2012.

[25] Guillaume Blin, Laurent Bulteau, Minghui Jiang, Pedro J. Tejada, and Stéphane Vialette. Hardness of longest common subsequence for sequences with bounded run-lengths. In *Proc. 23th Annual Symposium on Combinatorial Pattern Matching (CPM'12)*, pages 138–148, 2012.

[26] 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.

[27] Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *Proc. 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS'17)*, 2017. To appear, arXiv:1611.00918.

[28] 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 Computer Science (FOCS'15)*, pages 79–97, 2015.

[29] Karl Bringmann and Wolfgang Mulzer. Approximability of the Discrete Fréchet Distance. In *Proc. 31st International Symposium on Computational Geometry (SoCG'15)*, pages 739–753, 2015.

[30] Horst Bunke and Janos Csirik. An improved algorithm for computing the edit distance of run-length coded strings. *Information Processing Letters*, 54(2):93–96, 1995.

[31] Mauro Castelli, Riccardo Dondi, Giancarlo Mauri, and Italo Zoppis. The longest filled common subsequence problem. In *Proc. 28th Annual Symposium on Combinatorial Pattern Matching (CPM'17)*, 2017. To appear.

[32] 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.

[33] Maxime Crochemore, Gad M. Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32(6):1654–1673, 2003.

[34] Marek Cygan, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Polynomial-time approximation algorithms for weighted LCS problem. *Discrete Applied Mathematics*, 204:38–48, 2016.

[35] Sebastian Deorowicz. Quadratic-time algorithm for a string constrained LCS problem. *Information Processing Letters*, 112(11):423–426, 2012.

[36] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming I: Linear cost functions. *Journal of the ACM*, 39(3):519–545, July 1992.

[37] Effat Farhana and M. Sohel Rahman. Doubly-constrained LCS and hybrid-constrained LCS problems revisited. *Information Processing Letters*, 112(13):562–565, 2012.

[38] Anka Gajentaan and Mark H. Overmars. On a class of $O(N^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5(3):165–185, October 1995.

[39] Paweł Gawrychowski. Faster algorithm for computing the edit distance between SLP-compressed strings. In *Proc. 19th International Conference on String Processing and Information Retrieval (SPIRE'12)*, pages 229–236, 2012.

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

[41] Zvi Gotthilf, Danny Hermelin, and Moshe Lewenstein. Constrained LCS: Hardness and approximation. In *Proc. 19th Annual Symposium on Combinatorial Pattern Matching (CPM'08)*, pages 255–262, 2008.

[42] Zvi Gotthilf and Moshe Lewenstein. Approximating constrained LCS. In *Proc. 14th International Conference on String Processing and Information Retrieval (SPIRE'07)*, pages 164–172, 2007.

[43] Danny Hermelin, Gad M. Landau, Shir Landau, and Oren Weimann. Unified compression-based acceleration of edit-distance computation. *Algorithmica*, 65(2):339–353, 2013.

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

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

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

[47] Costas S Iliopoulos, Marcin Kubica, M Sohel Rahman, and Tomasz Waleń. Algorithms for computing the longest parameterized common subsequence. In *Proc. 18th Annual Conference on Combinatorial Pattern Matching (CPM'07)*, pages 265–273, 2007.

[48] Costas S. Iliopoulos and M. Sohel Rahman. Algorithms for computing variants of the longest common subsequence problem. *Theoretical Computer Science*, 395(2–3):255–267, 2008.

[49] Costas S. Iliopoulos and M. Sohel Rahman. A new efficient algorithm for computing the longest common subsequence. *Theory of Computing Systems*, 45(2):355–371, 2009.

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

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

[52] Guy Jacobson and Kiem-Phong Vo. Heaviest increasing/common subsequence problems. In *Proc. 3th Annual Symposium on Combinatorial Pattern Matching (CPM'92)*, pages 52–66, 1992.

[53] 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.

[54] Orgad Keller, Tsvi Kopelowitz, and Moshe Lewenstein. On the longest common parameterized subsequence. *Theoretical Computer Science*, 410(51):5347–5353, 2009.

[55] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*, pages 1272–1287, 2016.

[56] Keita Kuboi, Yuta Fujishige, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster STR-IC-LCS computation via RLE. In *Proc. 28th Annual Symposium on Combinatorial Pattern Matching (CPM'17)*, pages 20:1–20:12, 2017.

[57] 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.

[58] Gad M. Landau, Avivit Levy, and Ilan Newman. LCS approximation via embedding into local non-repetitive strings. In *Proc. 20th Annual Symposium on Combinatorial Pattern Matching (CPM'09)*, pages 92–105, 2009.

[59] Gad M. Landau, Eugene Myers, and Michal Ziv-Ukelson. Two algorithms for LCS consecutive suffix alignment. *Journal of Computer and System Sciences*, 73(7):1095–1117, 2007.

[60] Gad M. Landau, Baruch Schieber, and Michal Ziv-Ukelson. Sparse LCS common substring alignment. *Information Processing Letters*, 88(6):259–270, 2003.

[61] Kjell Lemström, Gonzalo Navarro, and Yoan Pinzon. Bit-parallel branch and bound algorithm for transposition invariant LCS. In *Proc. 11th International Conference on String Processing and Information Retrieval (SPIRE'04)*, pages 74–75, 2004.

[62] Yury Lifshits. Processing compressed texts: A tractability border. In *Proc. 18th Annual Symposium on Combinatorial Pattern Matching (CPM'07)*, pages 228–240, 2007.

[63] George S. Lueker. Improved bounds on the average length of longest common subsequences. *Journal of the ACM*, 56(3):17, 2009.

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

[65] Webb Miller and Eugene W. Myers. A file comparison program. *Software: Practice and Experience*, 15(11):1025–1040, 1985.

[66] Johra Muhammad Moosa, M. Sohel Rahman, and Fatema Tuz Zohora. Computing a longest common subsequence that is almost increasing on sequences having no repeated elements. *Journal of Discrete Algorithms*, 20:12–20, 2013.

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

[68] Shay Mozes, Dekel Tsur, Oren Weimann, and Michal Ziv-Ukelson. Fast algorithms for computing tree LCS. *Theoretical Computer Science*, 410(43):4303–4314, 2009.

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

[70] Narao Nakatsu, Yahiko Kambayashi, and Shuzo Yajima. A longest common subsequence algorithm suitable for similar text strings. *Acta Informatica*, 18:171–179, 1982.

[71] Mike Paterson and Vlado Dancík. Longest common subsequences. In *Proc. 19th International Symposium on Mathematical Foundations of Computer Science (MFCS'94)*, pages 127–142, 1994.

[72] Pavel Pevzner and Michael Waterman. Matrix longest common subsequence problem, duality and Hilbert bases. In *Proc. 3th Annual Symposium on Combinatorial Pattern Matching (CPM'92)*, pages 79–89, 1992.

[73] 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.

[74] Thomas G. Szymanski. A special case of the maximal common subsequence problem. Technical report, TR-170, Computer Science Laboratory, Princeton University, 1975.

[75] Alexander Tiskin. Longest common subsequences in permutations and maximum cliques in circle graphs. In *Proc. 17th Annual Symposium on Combinatorial Pattern Matching (CPM'06)*, pages 270–281, 2006.

[76] Alexander Tiskin. Semi-local longest common subsequences in subquadratic time. *Journal of Discrete Algorithms*, 6(4):570–581, 2008.

[77] Alexander Tiskin. Faster subsequence recognition in compressed strings. *Journal of Mathematical Sciences*, 158(5):759–769, 2009.

[78] Alexander Tiskin. Fast distance multiplication of unit-Monge matrices. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 1287–1296, 2010.

[79] 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.

[80] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 645–654, 2010.

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

[82] Biing-Feng Wang, Gen-Huey Chen, and Kunsoo Park. On the set LCS and set-set LCS problems. *Journal of Algorithms*, 14(3):466–477, 1993.

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

[84] Sun Wu, Udi Manber, Gene Myers, and Webb Miller. An $O(NP)$ sequence comparison algorithm. *Information Processing Letters*, 35(6):317–323, 1990.

[85] 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.