Sketching, Streaming, and Fine-Grained Complexity of (Weighted) LCS

Karl Bringmann
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
kbringma@mpi-inf.mpg.de

Bhaskar Ray Chaudhury
Max Planck Institute for Informatics, Saarland Informatics Campus, Graduate School of Computer Science, Saarbrücken, Germany
braycha@mpi-inf.mpg.de

Abstract

We study sketching and streaming algorithms for the Longest Common Subsequence problem (LCS) on strings of small alphabet size $|\Sigma|$. For the problem of deciding whether the LCS of strings x, y has length at least L, we obtain a sketch size and streaming space usage of $\mathcal{O}(L^{|\Sigma|-1} \log L)$. We also prove matching unconditional lower bounds.

As an application, we study a variant of LCS where each alphabet symbol is equipped with a weight that is given as input, and the task is to compute a common subsequence of maximum total weight. Using our sketching algorithm, we obtain an $\mathcal{O}(\min\{nm, n+m^{|\Sigma|}\})$-time algorithm for this problem, on strings x, y of length n, m, with $n \geq m$. We prove optimality of this running time up to lower order factors, assuming the Strong Exponential Time Hypothesis.

2012 ACM Subject Classification Theory of computation → Communication complexity, Theory of computation → Design and analysis of algorithms

Keywords and phrases algorithms, SETH, communication complexity, run-length encoding

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.40

1 Introduction

1.1 Sketching and Streaming LCS

In the Longest Common Subsequence problem (LCS) we are given strings x and y and the task is to compute a longest string z that is a subsequence of both x and y. This problem has been studied extensively, since it has numerous applications in bioinformatics (e.g. comparison of DNA sequences [5]), natural language processing (e.g. spelling correction [40, 49]), file comparison (e.g. the UNIX `diff` utility [23, 38]), etc. Motivated by big data applications, in the first part of this paper we consider space-restricted settings as follows:

- *LCS Sketching:* Alice is given x and Bob is given y. Both also are given a number L. Alice and Bob compute sketches $\mathrm{sk}_L(x)$ and $\mathrm{sk}_L(y)$ and send them to a third person, the referee, who decides whether the LCS of x and y is at least L. The task is to minimize the size of the sketch (i.e., its number of bits) as well as the running time of Alice and Bob (encoding) and of the referee (decoding).
- *LCS Streaming:* We are given L, and we scan the string x from left to right once, and then the string y from left to right once. After that, we need to decide whether the LCS of x and y is at least L. We want to minimize the space usage as well as running time.

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 40; pp. 40:1–40:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Analogous problem settings for the related edit distance have found surprisingly good solutions after a long line of work [11, 29, 46, 16]. For LCS, however, strong unconditional lower bounds are known for sketching and streaming: Even for $L = 4$ the sketch size and streaming memory must be $\Omega(n)$ bits, since the randomized communication complexity of this problem is $\Omega(n)$ [47]. Similarly strong results hold even for *approximating* the LCS length [47], see also [35]. However, these impossibility results construct strings over alphabet size $\Theta(n)$.

In contrast, in this paper we focus on strings x, y defined over a fixed alphabet Σ (of constant size). This is well motivated, e.g., for binary files ($\Sigma = \{0, 1\}$), DNA sequences ($\Sigma = \{A, G, C, T\}$), or English text ($\Sigma = \{a, \ldots, z, A, \ldots, Z\}$ plus punctuation marks). We therefore *suppress factors depending only on* $|\Sigma|$ in $\mathcal{O}$-notation throughout the whole paper. Surprisingly, this setting was ignored in the sketching and streaming literature so far; the only known upper bounds also work in the case of large alphabet and are thus $\Omega(n)$.

Before stating our first main result we define a run in a string as the non extendable repetition of a character. For example the string *baaabc* has a run of character a of length 3. Our first main result is the following deterministic sketch.

▶ **Theorem 1.** *Given a string x of length n over alphabet Σ and an integer L, we can compute a subsequence $C_L(x)$ of x such that (1) $|C_L(x)| = \mathcal{O}(L^{|\Sigma|})$, (2) $C_L(x)$ consists of $\mathcal{O}(L^{|\Sigma|-1})$ runs of length at most L, and (3) any string y of length at most L is a subsequence of x if and only if it is a subsequence of $C_L(x)$. Moreover, $C_L(x)$ is computed by a one-pass streaming algorithm with memory $\mathcal{O}(L^{|\Sigma|-1} \log L)$ and running time $\mathcal{O}(1)$ per symbol of x.*

Note that we can store $C_L(x)$ using $\mathcal{O}(L^{|\Sigma|-1} \log L)$ bits, since each run can be encoded using $\mathcal{O}(\log L)$ bits. This directly yields a solution for *LCS sketching*, where Alice and Bob compute the sketches $\mathrm{sk}_L(x) = C_L(x)$ and $\mathrm{sk}_L(y) = C_L(y)$ and the referee computes an LCS of $C_L(x)$ and $C_L(y)$. If this has length at least L then also x, y have LCS length at least L. Similarly, if x, y have an LCS z of length at least L, then z is also a subsequence of $C_L(x)$ and $C_L(y)$, and thus their LCS length is at least L, showing correctness. The sketch size is $\mathcal{O}(L^{|\Sigma|-1} \log L)$ bits, the encoding time is $\mathcal{O}(n)$, and the decoding time is $\mathcal{O}(L^{2|\Sigma|})$, as LCS can be computed in quadratic time in the string length $\mathcal{O}(L^{|\Sigma|})$.

We similarly obtain an algorithm for *LCS streaming* by computing $C_L(x)$ and then $C_L(y)$ and finally computing an LCS of $C_L(x)$ and $C_L(y)$. The space usage of this streaming algorithm is $\mathcal{O}(L^{|\Sigma|-1} \log L)$, and the running time is $\mathcal{O}(1)$ per symbol of x and y, plus $\mathcal{O}(L^{2|\Sigma|})$ for the last step.

These size, space, and time bounds are surprisingly good for $|\Sigma| = 2$, but quickly deteriorate with larger alphabet size. For very large alphabet size, this deterioration was to be expected due to the $\Omega(n)$ lower bound for $|\Sigma| = \Theta(n)$ from [47]. We further show that this deterioration is necessary by proving optimality of our sketch in several senses:

- We show that for any L, Σ there exists a string x (of length $\mathcal{O}(L^{|\Sigma|})$) such that no string x' of length $o(L^{|\Sigma|})$ has the same set of subsequences of length at most L. Similarly, this string x cannot be replaced by any string consisting of $o(L^{|\Sigma|-1})$ runs without affecting the set of subsequences of length at most L. This shows optimality of Theorem 1 among sketches that replace x by another string x' (not necessarily a subsequence of x) and then compute an LCS of x' and y. See Theorem 4.
- More generally, we study the *Subsequence Sketching* problem: Alice is given a string x and number L and computes $\mathrm{sk}_L(x)$. Bob is then given $\mathrm{sk}_L(x)$ and a string y of length L and decides whether y is a subsequence of x. Observe that any solution for LCS sketching or streaming with size/memory $S = S(L, \Sigma)$ yields a solution for subsequence sketching

with sketch size S.[1] Hence, any lower bound for subsequence sketching yields a lower bound for LCS sketching and streaming. We show that any deterministic subsequence sketch has size $\Omega(L^{|\Sigma|-1}\log L)$ in the worst case over all strings x. This matches the run-length encoding of $C_L(x)$ even up to the $\log L$-factor. If we restrict to strings of length $\Theta(L^{|\Sigma|-1})$, we still obtain a sketch size lower bound of $\Omega(L^{|\Sigma|-1})$. See Theorem 7.
- Finally, randomization does not help either: We show that any randomized subsequence sketch, where Bob may err in deciding whether y is a subsequence of x with small constant probability, has size $\Omega(L^{|\Sigma|-1})$, even restricted to strings x of length $\Theta(L^{|\Sigma|-1})$. See Theorem 10.

We remark that Theorem 1 only makes sense if $L \ll n$. Although this is not the best motivated regime of LCS in practice, it corresponds to testing whether x and y are "very different" or "not very different". This setting naturally occurs, e.g., if one string is much longer than the other, since then $L \leq m \ll n$. We therefore think that studying this regime is justified for the fundamental problem LCS.

1.2 WLCS: In between min-quadratic and rectangular time

As an application of our sketch, we determine the (classic, offline) time complexity of a weighted variant of LCS, which we discuss in the following.

A textbook dynamic programming algorithm computes the LCS of given strings x, y of length n in time $\mathcal{O}(n^2)$. A major result in fine-grained complexity shows that further improvements by polynomial factors would refute the Strong Exponential Time Hypothesis (SETH) [1, 13] (see Section 5 for a definition). In case x and y have different lengths n and m, with $n \geq m$, Hirschberg's algorithm computes their LCS in time $\mathcal{O}((n+m^2)\log n)$ [22], and this is again near-optimal under SETH. This running time could be described as "min-quadratic", as it is quadratic in the minimum of the two string lengths. In contrast, many other dynamic programming type problems have "rectangular" running time[2] $\widetilde{\mathcal{O}}(nm)$, with a matching lower bound of $(nm)^{1-o(1)}$ under SETH, e.g., Fréchet distance [4, 12], dynamic time warping [1, 13], and regular expression pattern matching [43, 10].

Part of this paper is motivated by the intriguing question whether there are problems with *intermediate* running time, between "min-quadratic" and "rectangular". Natural candidates are generalizations of LCS, such as the weighted variant *WLCS* as defined in [1]: Here we have an additional *weight function* $W\colon \Sigma \to \mathbb{N}$, and the task is to compute a common subsequence of x and y with maximum total weight. This problem is a natural variant of LCS that, e.g., came up in a SETH-hardness proof of LCS [1]. It is not to be confused with other weighted variants of LCS that have been studied in the literature, such as a statistical distance measure where given the probability of every symbol's occurrence at every text location the task is to find a long and likely subsequence [6, 18], a variant of LCS that favors consecutive matches [36], or edit distance with given operation costs [13].

Clearly, WLCS inherits the hardness of LCS and thus requires time $(n+m^2)^{1-o(1)}$. However, the matching upper bound $\widetilde{\mathcal{O}}(n+m^2)$ given by Hirschberg's algorithm only works as long as the function W is fixed (then the hidden constant depends on the largest weight). Here, we focus on the variant where the weight function W is part of the input. In this case, the basic $\mathcal{O}(nm)$-time dynamic programming algorithm is the best known.

[1] For LCS sketching this argument only uses that we can check whether y is a subsequence of x by testing whether the LCS length of x and y is $|y|$. For LCS streaming we use the memory state right after reading x as the sketch $\mathrm{sk}_L(x)$ and then use the same argument.

[2] By $\widetilde{\mathcal{O}}$-notation we ignore factors of the form $\mathrm{polylog}(n)$.

Our second main result is to settle the time complexity of WLCS in terms of n and m for any fixed constant alphabet Σ, up to lower order factors $n^{o(1)}$ and assuming SETH.

▶ **Theorem 2.** *WLCS can be solved in time* $\mathcal{O}(\min\{nm, n+m^{|\Sigma|}\})$. *Assuming SETH, WLCS requires time* $\min\{nm, n+m^{|\Sigma|}\}^{1-o(1)}$, *even restricted to* $n = \Theta(m^\alpha)$ *and* $|\Sigma| = \sigma$ *for any constants* $\alpha \in \mathbb{R}, \alpha \geq 1$ *and* $\sigma \in \mathbb{N}, \sigma \geq 2$.

In particular, for $|\Sigma| > 2$ the time complexity of WLCS is indeed "intermediate", in between "min-quadratic" and "rectangular"! To the best of our knowledge, this is the first result of fine-grained complexity establishing such an intermediate running time.

To prove Theorem 2 we first observe that the usual $\mathcal{O}(nm)$ dynamic programming algorithm also works for WLCS. For the other term $n + m^{|\Sigma|}$, we compress x by running the sketching algorithm from Theorem 1 with $L = m$. This yields a string $x' = C_m(x)$ of length $\mathcal{O}(m^{|\Sigma|})$ such that WLCS has the same value on (x, y) and (x', y), since every subsequence of length at most m of x is also a subsequence of x', and vice versa. Running the $\mathcal{O}(nm)$-time algorithm on (x', y) would yield total time $\mathcal{O}(n + m^{|\Sigma|+1})$, which is too slow by a factor m. To obtain an improved running time, we use the fact that x' consists of $\mathcal{O}(m^{|\Sigma|-1})$ runs. We design an algorithm for WLCS on a run-length encoded string x' consisting of r runs and an uncompressed string y of length m running time $\mathcal{O}(rm)$. This generalizes algorithms for LCS with one run-length encoded string [7, 20, 37]. Together, we obtain time $\mathcal{O}(\min\{nm, n + m^{|\Sigma|}\})$. We then show a matching SETH-based lower bound by combining our construction of incompressible strings from our sketching lower bounds (Theorem 4) with the by-now classic SETH-hardness proof of LCS [1, 13].

1.3 Further Related Work

Analyzing the running time in terms of multiple parameters like n, m, L has a long history for LCS [8, 9, 19, 22, 24, 26, 42, 44, 51]. Recently tight SETH-based lower bounds have been shown for all these algorithms [14]. In the second part of this paper, we perform a similar complexity analysis on a weighted variant of LCS. This follows the majority of recent work on LCS, which focused on transferring the early successes and techniques to more complicated problems, such as longest common increasing subsequence [39, 33, 52, 17], tree LCS [41], and many more generalizations and variants of LCS, see, e.g., [32, 15, 48, 28, 3, 34, 30, 21, 45, 25]. For brevity, here we ignore the equally vast literature on the closely related edit distance.

1.4 Notation

For a string x of length n over alphabet Σ, we write $x[i]$ for its i-th symbol, $x[i \dots j]$ for the substring from the i-th to j-th symbol, and $|x|$ for its length. For $c \in \Sigma$ we write $|x|_c := |\{i \mid x_i = c\}|$. For strings x, y we write $x \circ y$ for their concatenation, and for $k \in \mathbb{N}$ we write x^k for the k-fold repetition $x \circ \dots \circ x$. A subsequence of x is any string of the form $y = x[i_1] \circ x[i_2] \circ \dots \circ x[i_\ell]$ with $1 \leq i_1 < i_2 < \dots < i_\ell \leq |x|$; in this case we write $y \preceq x$. A *run* in x is a maximal substring $x[i \dots j] = c^{j-i+1}$, consisting of a single alphabet letter $c \in \Sigma$. Recall that we suppress factors depending only on $|\Sigma|$ in $\mathcal{O}$-notation.

2 Sketching LCS

In this section design a sketch for LCS, proving Theorem 1. Consider any string z defined over alphabet $S \subseteq \Sigma$. We call z a (q, S)-*permutation string* if we can partition $z = z^{(1)} \circ z^{(2)} \circ \dots \circ z^{(q)}$ such that each $z^{(i)}$ contains each symbol in S at least once. Observe that a (q, S) permutation string contains any string y of length at most q over the alphabet S as a subsequence.

Algorithm 1 Outline for computing $C_L(x)$ given a string x and an integer L.

```
initialize C_L(x) as the empty string
for all i from 1 to |x| do
    if for all S ⊆ Σ with x[i] ∈ S, no suffix of C_L(x) is an (L,S)-permutation string then
        set C_L(x) ← C_L(x) ∘ x[i]
return C_L(x)
```

▶ **Claim 3.** *Consider any string $x = x' \circ c \circ x''$, where x', x'' are strings over alphabet Σ and $c \in \Sigma$. Let $S \subseteq \Sigma$. If some suffix of x' is an (L, S)-permutation string and $c \in S$, then for all strings y of length at most L we have $y \preceq x$ if and only if $y \preceq x' \circ x''$.*

Proof. The "if"-direction is immediate. To prove the "only if", consider any subsequence y of x of length $d \leq L$ and let $y = x[i_1] \circ x[i_2] \circ \ldots \circ x[i_d]$. Let ℓ and r be the length of x' and x'', respectively. If $i_k \neq \ell + 1$ for all $1 \leq k \leq d$, then clearly $y \preceq x' \circ x''$. Thus, assume that $i_k = \ell + 1$ for some k. Let a be minimal such that $x[a \ldots \ell]$ only contains symbols in S. By assumption, $x[a \ldots \ell]$ is an (L, S)-permutation string, and $c = x[\ell + 1] \in S$. Let $j \geq 1$ be the minimum index such that $x[i_j] \ldots x[i_k]$ only contains symbols in S. Since j is minimal, $x[i_{j-1}] \notin S$ and thus $i_b < a$ for all $b < j$. Therefore $x[i_1] \circ x[i_2] \circ \ldots \circ x[i_{j-1}] \preceq x[0 \ldots a-1]$. Since $x[a \ldots \ell]$ is an (L, S)-permutation string and $|x[i_j] \circ \ldots \circ x[i_k]| \leq d \leq L$, it follows that $x[i_j] \circ \ldots \circ x[i_k]$ is a subsequence of $x[a \ldots \ell]$. Hence, $x[i_1] \circ \ldots \circ x[i_k] \preceq x'$ and $x[i_{k+1}] \circ \ldots \circ x[i_d] \preceq x''$, and thus $y \preceq x' \circ x''$. ◀

The above claim immediately gives rise to the following one-pass streaming algorithm.

By Claim 3, the string $C_L(x)$ returned by this algorithm satisfies the subsequence property (3) of Theorem 1. Note that any run in $C_L(x)$ has length at most L, since otherwise for $S = \{c\}$ we would obtain an (L, S)-permutation string followed by another symbol c, so that Claim 3 would apply. We now show the upper bounds on the length and the number of runs. Consider a substring $z = C_L(x)[i \ldots j]$ of $C_L(x)$, containing symbols only from $S \subseteq \Sigma$. We claim that z consists of at most $r_L(|S|) := 2(L+1)^{|S|-1} - 1$ runs. We prove our claim by induction on $|S|$. For $|S| = 1$, the claim holds trivially. For $|S| > 1$ and any $k \geq 1$, let i_k be the minimal index such that $z[1 \ldots i_k]$ is a (k, S)-permutation string, or $i_k = \infty$ if no such prefix of z exists. Note that $i_L \geq |z|$, since otherwise a proper prefix of z would be an (L, S)-permutation string, in which case we would have deleted the last symbol of z. The string $z[i_{k-1} + 1 \ldots i_k - 1]$ contains symbols only from $S \setminus \{z[i_k]\}$ and thus by induction hypothesis consists of at most $r_L(|S| - 1)$ runs. Since $i_L \geq |z|$, we conclude that the number of runs in z is at most $L \cdot (r_L(|S| - 1) + 1) \leq L \cdot 2(L+1)^{|S|-2} \leq 2(L+1)^{|S|-1} - 1 = r_L(|S|)$. Thus the number of runs of $C_L(x)$ is at most $r_L(|\Sigma|) \in \mathcal{O}(L^{|\Sigma|-1})$, and since each run has length at most L we obtain $|C_L(x)| \in \mathcal{O}(L^{|\Sigma|})$.

Algorithm 2 shows how to efficiently implement Algorithm 1 in time $\mathcal{O}(1)$ per symbol of x. We maintain a counter t_S (initialized to 0) and a set Q_S (initialized to $\emptyset$) for every $S \subseteq \Sigma$ with the following meaning. After reading $x[1 \ldots i]$, let j be minimal such that $x[j \ldots i]$ consists of symbols in S. Then t_S is the maximum number t such that $x[j \ldots i]$ is a (t, S)-permutation string. Moreover, let k be minimal such that $x[j \ldots k]$ still is a (t_S, S)-permutation string. Then $Q_S \subseteq S$ is the set of symbols that appear in $x[k+1 \ldots i]$. In other words, in the future we only need to read the symbols in $S \setminus Q_S$ to complete a $(t_S + 1, S)$-permutation string. In particular, when reading the next symbol $x[i+1]$, in order to check whether Claim 3 applies we only need to test whether for any $S \subseteq \Sigma$ with $x[i+1] \in S$ we have $t_S \geq L$. Updating t_S and Q_S is straightforward, and shown in Algorithm 2.

Algorithm 2 Computing $C_L(x)$ in time $\mathcal{O}(1)$ per symbol of x.

```
set t_s ← 0, Q_S ← ∅ for all S ⊆ Σ
set C_L(x) to the empty string
for all i from 1 to |x| do
  if t_S < L for all S ⊆ Σ with x[i] ∈ S then
    set C_L(x) ← C_L(x) ∘ x[i]
    for all S such that x[i] ∈ S do
      set Q_S ← Q_S ∪ {x[i]}
      if Q_S = S then
        set Q_S ← ∅
        set t_S ← t_S + 1
    for all S such that x[i] ∉ S do
      set t_S ← 0
      set Q_S ← ∅
```

Since we assume $|\Sigma|$ to be constant, each iteration of the loop runs in time $\mathcal{O}(1)$, and thus the algorithm determines $C_L(x)$ in time $\mathcal{O}(n)$. This finishes the proof of Theorem 1.

3 Optimality of the Sketch

In this section we show that the sketch $C_L(x)$ is optimal in many ways. First, we show that the length and the number of runs are optimal for any sketch that replaces x by any other string z with the same set of subsequences of length at most L.

▶ **Theorem 4.** *For any L and Σ there exists a string x such that for any string z with $\{y \mid y \preceq x,\ |y| \le L\} = \{y \mid y \preceq z,\ |y| \le L\}$ we have $|z| = \Omega(L^{|\Sigma|})$ and z consists of $\Omega(L^{|\Sigma|-1})$ runs.*

Let $\Sigma = \{0, 1, \ldots, \sigma - 1\}$ and $\Sigma_k = \{0, 1, \ldots, k-1\}$. We construct a family of strings $x^{(k)}$ recursively as follows, where $m := L/|\Sigma|$:

$$x^{(0)} = 0^m$$
$$x^{(k)} = (x^{(k-1)} \circ k)^m \circ x^{(k-1)} \quad \text{for } 1 \le k \le \sigma - 1.$$

Theorem 4 now follows from the following inductive claim, for $k = \sigma - 1$.

▶ **Claim 5.** *For any string z with $\{y \mid y \preceq x^{(k)},\ |y| \le m(k+1)\} = \{y \mid y \preceq z,\ |y| \le m(k+1)\}$ we have $|z| \ge m^{k+1}$ and the number of runs in z is at least m^k.*

Proof. We use induction on k. For $k = 0$, since $y = 0^m \preceq x^{(0)}$ we have $z = 0^{m'}$ with $m' \ge m$ and the number of runs in z is exactly 1. For any $k > 0$, if $|x^{(k)}|_k > |z|_k$ then $k^m \preceq x^k$ but $k^m \not\preceq z$, and similarly if $|x^{(k)}|_k < |z|_k$ then $k^{m+1} \preceq z$ but $k^{m+1} \not\preceq x^{(k)}$ (note that $m(k+1) \ge m+1$ since $k \ge 1$, and thus y can be k^{m+1}). This implies $|z|_k = m$ and thus we have $z = z^{(0)} \circ k \circ z^{(1)} \circ k \circ \ldots \circ k \circ z^{(m)}$, where each $z^{(i)}$ is a string on alphabet Σ_{k-1}. Hence, for any $0 \le i \le m$ and string y' of length at most mk, we have $y = k^i y' k^{m-i} \preceq z$ if and only if $y' \preceq z^{(i)}$. Similarly, $y \preceq x^{(k)}$ holds if and only if $y' \preceq x^{(k-1)}$. Since $y \preceq z$ is equivalent to $y \preceq x$ by assumption, we obtain that $y' \preceq z^{(i)}$ is equivalent to $y' \preceq x^{(k-1)}$. By induction hypothesis, $z^{(i)}$ has length at least m^k and consists of at least m^{k-1} runs. Summing over all i, string z has length at least m^{k+1} and consists of at least m^k runs. ◀

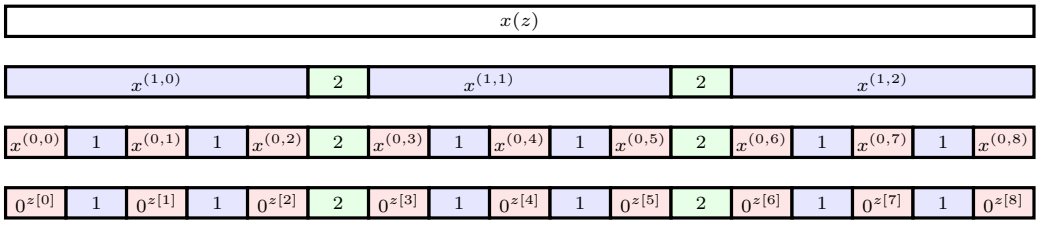

Figure 1 Illustration of constructing $x(z)$ from z. Let $m = \sigma = 3$. Consider a string z of length $m^{\sigma-1} = 9$. The figure shows the construction of $x(z)$ from z.

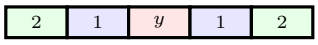

Figure 2 Illustration of the construction of $pat(i, y)$. Let $m = \sigma = 3$. Consider $i = 4 = 1 \cdot 3^1 + 1 \cdot 3^0$. Therefore $\mathrm{pat}(i, y) = 21y12$.

Note that the run-length encoding of $C_L(x)$ has bit length $\mathcal{O}(L^{|\Sigma|-1} \log L)$, since $C_L(x)$ consists of $\mathcal{O}(L^{|\Sigma|-1})$ runs, each of which can be encoded using $\mathcal{O}(\log L)$ bits. We now show that this sketch has optimal size, even in the setting of *Subsequence Sketching*: Alice is given a string x of length n over alphabet Σ and a number L and computes $\mathrm{sk}_L(x)$. Bob is then given $\mathrm{sk}_L(x)$ and a string y of length at most[3] L and decides whether y is a subsequence of x.

We construct the following hard strings for this setting, similarly to the previous construction. Let $\Sigma = \{0, 1, 2, \ldots, \sigma - 1\}$ and $m \in \mathbb{N}$. Consider any vector $z \in \{0, \ldots, m-1\}^k$, where $k := m^{\sigma-1}$. We define the string $x = x(z)$ recursively as follows; see Figure 1 for an illustration:

$$
\begin{aligned}
x(z) &= x^{(\sigma-1,0)} \\
x^{(c,i)} &= \Big(\bigcirc_{j=0}^{m-2} x^{(c-1,m\cdot i+j)} \circ c \Big) \circ x^{(c-1,m\cdot i+m-1)} \quad \text{for } 1 \le c \le \sigma - 1 \\
x^{(0,i)} &= 0^{z[i]}
\end{aligned}
$$

A straightforward induction shows that $|x(z)| \le m^\sigma - 1$. Moreover, for any $0 \le i < m^{\sigma-1}$ with base-m representation $i = \sum_{j=0}^{\sigma-2} i_j \cdot m^j$, where $0 \le i_j < m$, we define the following string; see Figure 2 for an illustration:

$$\mathrm{pat}(i, y) := \big(\bigcirc_{j=1}^{\sigma-1} (\sigma - j)^{i_{\sigma-1-j}} \big) \circ y \circ \big(\bigcirc_{j=1}^{\sigma-1} j^{m-1-i_{j-1}} \big).$$

The following claim shows that testing whether $\mathrm{pat}(i, y)$ is a subsequence of $x(z)$ allows to infer the entries of z.

▶ **Claim 6.** *We have* $\mathrm{pat}(i, y) \preceq x(z)$ *if and only if* $y \preceq 0^{z[i]}$.

Proof. See Figure 3 for illustration. Given i and y, let $z^{(c)} = c^{i_{c-1}} \circ z^{(c-1)} \circ c^{m-1-i_{c-1}}$ for all $1 \le c \le \sigma - 1$, and $z^{(0)} = y$. Note that $z^{(\sigma-1)} = \mathrm{pat}(i, y)$. Set $j_c := \sum_{l=c}^{\sigma-2} i_l \cdot m^{l-c}$, so in particular we have $j_c = m \cdot j_{c-1} + i_c$. Observe that $z^{(c)} \preceq x^{(c,j_c)}$ if and only if

[3] In the introduction, we used a slightly different definition where Bob is given a string of length *exactly* L. This might seem slightly weaker, but in fact the two formulations are equivalent (up to increasing L by 1), as can be seen by replacing x by $x' = 0^L 1x$ and y by $y' = 0^{L-|y|}1y$. Then $y \preceq x$ if and only if $y' \preceq x'$, and y' has fixed length $L + 1$.

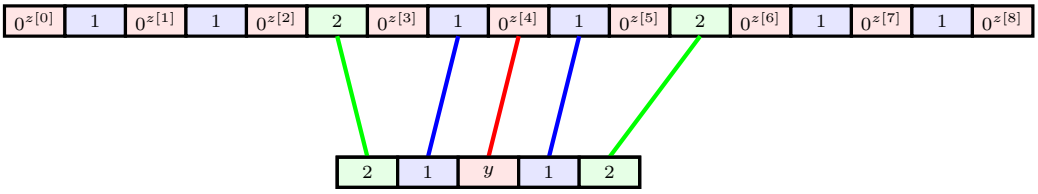

Figure 3 Illustration of Claim 6. Let $m = \sigma = 3$ and $i = 4$. Then $\mathrm{pat}(i, y) = 21y12$. Now observe that $\mathrm{pat}(i, y) \preceq x(z)$ if and only if $y \preceq x^{(0,i)} = 0^{z[i]}$.

$z^{(c-1)} \preceq x^{(c-1,j_{c-1})}$, which follows immediately after matching all c's in $z^{(c)}$ and $x^{(c,j_c)}$. Therefore, $\mathrm{pat}(i, y) = z^{(\sigma-1)} \preceq x^{(\sigma-1,0)} = x(z)$ holds if and only if $z^{(c)} \preceq x^{(c,j_c)}$ for any $c \leq \sigma - 2$. Substituting $c = 0$ we obtain that $\mathrm{pat}(i, y) \preceq x(z)$ holds if and only if $y = z^{(0)} \preceq x^{(0,j_0)} = x^{(0,i)} = 0^{z[i]}$. ◀

▶ **Theorem 7.** *Any deterministic subsequence sketch has size $\Omega(L^{|\Sigma|-1} \log L)$ in the worst case. Restricted to strings of length $\Theta(L^{|\Sigma|-1})$, the sketch size is $\Omega(L^{|\Sigma|-1})$.*

Proof. Let $m := L/|\Sigma|$. Let $z \in \{0, \ldots, m-1\}^k$ with $k = m^{|\Sigma|-1}$ and let $x = x(z)$ as above. Alice is given x, L as input. Notice that there are m^k distinct inputs for Alice. Assume for contradiction that the sketch size is less that $k \cdot \log m$ for every x. Then the total number of distinct possible sketches is strictly less than m^k. Therefore, at least two strings, say $x(z_1)$ and $x(z_2)$, have the same encoding, for some $z_1, z_2 \in \{0, \ldots, m-1\}^k$ with $z_1 \neq z_2$. Let i be such that $z_1[i] \neq z_2[i]$, and without loss of generality $z_1[i] < z_2[i]$. Now set Bob's input to $y = \mathrm{pat}(i, z_2[i])$, which is a valid subsequence of $x(z_2)$, but not of $x(z_1)$. However, since the encoding for both $x(z_2)$ and $x(z_1)$ is the same, Bob's output will be incorrect for at least one of the strings. Finally, note that $|y| \leq m\sigma = L$. Hence, we obtain a sketch size lower bound of $\Omega(k \log m) = \Omega(L^{|\Sigma|-1} \log L)$.

If we instead choose z from $\{0,1\}^k$, then the constructed string $x(z)$ has length $\mathcal{O}(k) = \mathcal{O}(L^{|\Sigma|-1})$, and the same argument as above yields a sketch lower bound of $\Omega(L^{|\Sigma|-1})$. ◀

We now discuss the complexity of randomized subsequence sketching where Bob is allowed to err with probability 1/3. To this end, we will reduce from the *Index* problem.

▶ **Definition 8.** In the *Index* problem, Alice is given an n-bit string $z \in \{0,1\}^n$ and sends a message to Bob. Bob is given Alices's message and an integer $i \in [n]$ and outputs $z[i]$.

Intuitively, since the communication is one-sided, Alice cannot infer i and therefore has to send the whole string z. This intuition also holds for randomized protocols, as follows.

▶ **Fact 9** ([31]). *The randomized one-way communication complexity of Index is $\Omega(n)$.*

Claim 6 shows that subsequence sketching allows us to infer the bits of an arbitrary string z, and thus the hardness of Index carries over to subsequence sketching.

▶ **Theorem 10.** *In a* randomized *subsequence sketch, Bob is allowed to err with probability 1/3. Any randomized subsequence sketch has size $\Omega(L^{|\Sigma|-1})$ in the worst case. This holds even restricted to strings of length $\Theta(L^{|\Sigma|-1})$.*

Proof. We reduce the Index problem to subsequence sketching. Let $z \in \{0,1\}^k$ be the input to Alice in the Index problem, where $k = m^{|\Sigma|-1}$. As above, we construct the corresponding input $x(z)$ to Alice in subsequence sketching. Observe that $|x(z)| = \mathcal{O}(m^{|\Sigma|-1})$. For any input i to Bob in the Index problem, we construct the corresponding input $\mathrm{pat}(i, 0)$ for Bob in subsequence sketching. We have $\mathrm{pat}(i, 0) \preceq x(z)$ if and only if $z[i] = 1$ (by Claim 6). This yields a lower bound of $\Omega(k) = \Omega(m^{|\Sigma|-1}) = \Omega(L^{|\Sigma|-1})$ on the sketch size (by Fact 9). ◀

4 Weighted LCS

▶ **Definition 11.** In the WLCS problem we are given strings x, y of lengths n, m over alphabet Σ and given a function $W\colon \Sigma \to \mathbb{N}$. A weighted longest common subsequence (WLCS) of x and y is any string z with $z \preceq x$ and $z \preceq y$ maximizing $W(z) = \sum_{i=1}^{|z|} W(z[i])$. The task is to compute this maximum weight, which we abbreviate as $\mathrm{WLCS}(x, y)$.

In the remainder of this section we will design an algorithm for computing $\mathrm{WLCS}(x, y)$ in time $\mathcal{O}(\min\{nm, n + m^{|\Sigma|}\})$. This yields the upper bound of Theorem 2. Note that here we focus on computing the maximum weight $\mathrm{WLCS}(x, y)$; standard methods can be applied to reconstruct a subsequence attaining this value. We prove a matching conditional lower bound of $\min\{nm, n + m^{|\Sigma|}\}^{1-o(1)}$ in the next section.

Let x, y, W be given. The standard dynamic programming algorithm for determining $\mathrm{LCS}(x, y)$ in time $\mathcal{O}(nm)$ trivially generalizes to $\mathrm{WLCS}(x, y)$ as well. Alternatively, we can first compress x to $x' := C_m(x)$ in time $\mathcal{O}(n)$ and then compute the $\mathrm{WLCS}(x', y)$, which is equal to $\mathrm{WLCS}(x, y)$ since all subsequences of length at most m of x are also subsequences of $C_m(x)$. We show below in Theorem 12 how to compute WLCS of a run-length encoded string x' with r runs and a string y of length m in time $\mathcal{O}(rm)$. Since $x' = C_m(x)$ consists of $O(m^{|\Sigma|-1})$ runs and the length of y is m, we can compute $\mathrm{WLCS}(x, y) = \mathrm{WLCS}(C_m(x), y)$ in time $\mathcal{O}(m^{|\Sigma|})$. In total, we obtain time $\mathcal{O}(\min\{nm, n + m^{|\Sigma|}\})$.

It remains to solve WLCS on a run-length encoded string x with r runs and a string y of length m in time $\mathcal{O}(rm)$. For (unweighted) LCS a dynamic programming algorithm with this running time was presented by Liu et al. [37]. We first give a brief intuitive explanation as to why their algorithm does not generalize to WLCS. Let $x = c_1^{\ell_1} c_2^{\ell_2} \dots c_r^{\ell_r}$ be the run-length encoded string, where $c_i \in \Sigma$, and let $L_i = \sum_{j=1}^{i} \ell_j$. Let $D(i, j) := \mathrm{WLCS}(x[1 \dots L_i], y[1 \dots j])$. Liu et al.'s algorithm relies on a recurrence for $D(i, j)$ in terms of $D(i, j-1)$. Consider an input like $x = ba_1a_2 \cdots a_kb$ and $y = a_1a_2 \cdots a_kbb$ with $W(b) > \sum_{\ell \in [k]} W(a_\ell)$. Note that $D(k+2, k+1) = \sum_{\ell \in [k]} W(a_\ell) + W(b)$, but $D(k+2, k+2) = 2W(b)$. Thus $D(k+2, k+2) = D(k+2, k+1) - \sum_{\ell \in [k]} W(a_\ell) + W(b)$. Therefore, in the weighted setting $D(i, j)$ and $D(i, j-1)$ can differ by complicated terms that seem hard to figure out locally. Our algorithm that we develop below instead relies on a recurrence for $D(i, j)$ in terms of $D(i-1, j')$.

▶ **Theorem 12.** *Given a run-length encoded string x consisting of r runs, a string y of length m, and a weight function $W\colon \Sigma \to \mathbb{N}$ we can determine $\mathrm{WLCS}(x, y)$ in time $\mathcal{O}(rm)$.*

Proof. We write the run-length encoded string x as $c_1^{\ell_1} c_2^{\ell_2} \dots c_r^{\ell_r}$ with $c_i \in \Sigma$ and $\ell_i \geq 1$. Let $L_i = \sum_{j=1}^{i} \ell_j$. We will build a dynamic programming table D where $D(i, j)$ stores the value $\mathrm{WLCS}(x[1 \dots L_i], y[1 \dots j])$. In particular, $D(0, j) = D(i, 0) = 0$ for all i, j. We will show how to compute this table in $\mathcal{O}(1)$ (amortized) time per entry in the following. Since we can split $\mathrm{WLCS}(x[1 \dots L_i], y[1 \dots j]) = \max_{0 \leq k \leq j} \mathrm{WLCS}(x[1 \dots L_{i-1}], y[1 \dots k]) + \mathrm{WLCS}(c_i^{\ell_i}, y[k+1 \dots j])$, we obtain the recurrence $D(i, j) = \max_{0 \leq k \leq j} D(i-1, k) + W(c_i) \cdot \min\{\ell_i, |y[k+1 \dots j]|_{c_i}\}$. Since $D(i, j)$ is monotonically non-decreasing in i and j, we may rewrite the same recurrence as

$$\begin{aligned} D(i, j) &= \max_{0 \leq k \leq j \,:\, |y[k+1 \dots j]|_{c_i} \leq \ell_i} D(i-1, k) + W(c_i) \cdot |y[k+1 \dots j]|_{c_i}. \\ &= W(c_i) \cdot |y[1 \dots j]|_{c_i} + \max_{0 \leq k \leq j \,:\, |y[k+1 \dots j]|_{c_i} \leq \ell_i} D(i-1, k) - W(c_i) \cdot |y[1 \dots k]|_{c_i} \end{aligned}$$

Algorithm 3 Computing $\mathrm{RTLM}(K_{i,j})$ from $\mathrm{RTLM}(K_{i,j-1})$.

```
Initialize RTLM(K_{i,j}) = RTLM(K_{i,j-1})
while the smallest (=leftmost) element k of RTLM(K_{i,j}) satisfies |y[k+1...j]|_{c_i} > ℓ_i do
    Remove k from RTLM(K_{i,j})
while the largest (=rightmost) element k of RTLM(K_{i,j}) satisfies h_i(k) ≤ h_i(j) do
    Remove k from RTLM(K_{i,j})
Append j to RTLM(K_{i,j})
```

Let $b_{i,j}$ be the minimum value of $0 \leq k \leq j$ such that $|y[k+1\dots j]|_{c_i} \leq \ell_i$. Note that $b_{i,j}$ is well-defined, since for $k = j$ we always have $|y[k+1\dots j]|_{c_i} = 0 \leq \ell_i$, and note that $b_{i,j}$ is monotonically non-decreasing in j. We define the *active k-window* $K_{i,j}$ as the interval $\{b_{i,j}, b_{i,j}+1, \dots, j\}$. Note that $K_{i,j}$ is non-empty and both its left and right boundary are monotonic in j. Let $h_i(k) := D(i-1,k) - W(c_i) \cdot |y[1\dots k]|_{c_i}$ be the *height* of k. We define $\mathit{highest}(K_{i,j})$ as $\max_{k \in K_{i,j}} h_i(k)$. With this notation, we can rewrite the above recurrence as

$$D(i,j) = W(c_i) \cdot |y[1\dots j]|_{c_i} + \mathit{highest}(K_{i,j}).$$

We can precompute all values $|y[1\dots j]|_c$ in $\mathcal{O}(m)$ time. Hence, in order to determine $D(i,j)$ in amortized time $\mathcal{O}(1)$ it remains to compute $\mathit{highest}(K_{i,j})$ in amortized time $\mathcal{O}(1)$. To this end, we maintain the *right to left maximum sequence* of the active window $K_{i,j}$. Specifically, we consider the sequence $\mathrm{RTLM}(K_{i,j}) = \langle k_s, k_{s-1}, \dots, k_1 \rangle$ where $k_1 = j$ and for any $p > 1$, k_p is is the largest number in $K_{i,j}$ with $k_p < k_{p-1}$ and $h_i(k_p) > h_i(k_{p-1})$. In particular, k_s is the largest number in $K_{i,j}$ attaining $h_i(k_s) = \mathit{highest}(K_{i,j})$. Hence, from this sequence $\mathrm{RTLM}(K_{i,j})$ we can determine $\mathit{highest}(K_{i,j})$ and thus $D(i,j)$ in time $\mathcal{O}(1)$. It remains to argue that we can maintain $\mathrm{RTLM}(K_{i,j})$ in amortized time $\mathcal{O}(1)$ per table entry. We sketch an algorithm to obtain $\mathrm{RTLM}(K_{i,j})$ from $\mathrm{RTLM}(K_{i,j-1})$.

It is easy to see correctness, since the first while loop removes right to left maxima that no longer lie in the active window, the second while loop removes right to left maxima that are dominated by the new element j, and the last line adds j. Note that $|y[k+1\dots j]|_c = |y[1\dots j]|_c - |y[1\dots k]|_c$ can be computed in time $\mathcal{O}(1)$ from the precomputed values $|y[1\dots j]|_c$, and thus the while conditions can be checked in time $\mathcal{O}(1)$. A call of Algorithm 3 can necessitate multiple removal operations, but only one insertion. By charging removals to the insertion of the removed element, we see that Algorithm 3 runs in amortized time $\mathcal{O}(1)$. We therefore can compute each table entry $D(i,j)$ in amortized time $\mathcal{O}(1)$ and obtain total time $\mathcal{O}(rm)$. Pseudocode for the complete algorithm is given below. ◀

5 Conditional lower bound for Weighted LCS

In this section, we prove a conditional lower bound for Weighted LCS, based on the standard hypothesis SETH, which was introduced by Impagliazzo, Paturi, and Zane [27] and asserts that satisfiability has no algorithms that are much faster than exhaustive search.

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

Algorithm 4 Computing WLCS(x, y) in time $\mathcal{O}(r \cdot m)$.

```
precompute |y[1...i]|_c for all i ∈ [m] and c ∈ Σ.
set D(i,0) = D(0,j) = 0 for any 0 ≤ i ≤ r and 0 ≤ j ≤ m.
for i = 1,...,r do
    RTLM(K_{i,0}) ← ⟨0⟩.
    for j = 1,...,m do
        Update RTLM(K_{i,j}) as in Algorithm 3
        Let k be the smallest (=leftmost) element of RTLM(K_{i,j})
        Compute highest(K_{i,j}) = h_i(k) = D(i−1,k) − W(c_i)·|y[1...k]|_{c_i}
        D(i,j) ← W(c_i)·|y[1...j]|_{c_i} + highest(K_{i,j}).
return D(r,m)
```

Essentially all known SETH-based lower bounds for polynomial-time problems (e.g. [1, 10, 12, 13, 14]) use reductions via the *Orthogonal Vectors problem* (OV): Given sets A, $B \subseteq \{0,1\}^D$ of size $|A| = N$, $|B| = M$, determine whether there are $a \in A, b \in B$ that are orthogonal, i.e., $\sum_{i=1}^{D} a[i] \cdot b[i] = 0$, where the sum is over the integers. Simple algorithms solve OV in time $\mathcal{O}(2^D(N+M))$ and $\mathcal{O}(NMD)$. The fastest known algorithm for $D = c(N) \log N$ runs in time $N^{2-1/\mathcal{O}(\log c(N))}$ (when $N = M$) [2], which is only slightly subquadratic for $D \gg \log N$. This has led to the following reasonable hypothesis.

(Unbalanced) Orthogonal Vectors Hypothesis (OVH). *For any $\gamma > 0$, OV restricted to $M = \Theta(N^\gamma)$ and $D = N^{o(1)}$ requires time $(NM)^{1-o(1)}$.*

A well-known reduction by Williams [50] shows that SETH implies OVH in case $\gamma = 1$. Moreover, an observation in [14] shows that if OVH holds for some $\gamma > 0$ then it holds for all $\gamma > 0$. Thus, OVH is a weaker assumption than SETH, and any OVH-based lower bound also implies a SETH-based lower bound. The conditional lower bound in this section does not only hold assuming SETH, but even assuming the weaker OVH.

We use the following construction from the OVH-based lower bound for LCS [1, 13]. For binary alphabet, such a construction was given in [13].

▶ **Theorem 13.** *Given $A, B \subseteq \{0,1\}^D$ of size N, in time $\mathcal{O}(DN)$ we can compute strings x_A and y_B on alphabet $\{0,1\}$ of length $\Theta(DN)$ as well as a number τ such that* LCS$(x_A, y_B) \geq \tau$ *holds if and only if there is an orthogonal pair of vectors in A and B. In this construction, x_A and y_B depend only on A and B, respectively, and $|x_A|, |y_B|, \tau$ depend only on N, D.*

We now prove a conditional lower bound for WLCS, i.e., the lower bound of Theorem 2.

▶ **Theorem 14.** *Given strings x, y of lengths n, m with $n \geq m$ over alphabet Σ, computing* WLCS(x, y) *requires time* $\min\{nm, n+m^{|\Sigma|}\}^{1-o(1)}$*, assuming OVH. This holds even restricted to $n = m^{\alpha \pm o(1)}$ and $|\Sigma| = \sigma$ for any fixed constants $\alpha \in \mathbb{R}, \alpha \geq 1$ and $\sigma \in \mathbb{N}, \sigma \geq 2$.*

Proof. Let $\Sigma = \{0, 1, \ldots, \sigma - 1\}$ and $\alpha = \alpha_I + \alpha_F$, where $\alpha_I = \lfloor \alpha \rfloor$ and $\alpha_F = \alpha - \alpha_I$ are the integral and fractional parts. Let $M \in \mathbb{N}$ and set $N = \min\{M^{\alpha_I} \cdot \lceil M^{\alpha_F} \rceil, M^{\sigma-1}\}$. Note that M divides N. Consider any instance $A = \{a_0, a_1, \ldots, a_{N-1}\} \subseteq \{0,1\}^d$ and $B = \{b_0, b_1, \ldots, b_{M-1}\} \subseteq \{0,1\}^D$ of the Orthogonal Vectors problem. Partition A into $A^0, A^1, \ldots, A^{N/M-1}$, where $|A^i| = M$. Then by Theorem 13 we can construct strings $x_A^{(i)}$ and y_B on alphabet $\{0,1\}$ of length $\Theta(DM)$ and $\tau \in \mathbb{N}$ in time $\mathcal{O}(DM)$ such that A^i and B contain an orthogonal pair of vectors if and only if LCS$(x_A^{(i)}, y_B) \geq \tau$. Note that A and B contain an orthogonal pair of vectors if and only if for some $0 \leq i < \frac{N}{M}$, A^i and B

contain an orthogonal pair of vectors. Hence, A and B contain an orthogonal pair if and only if $\max_{0\le i<\frac{N}{M}} \mathrm{LCS}(x_A^{(i)}, y_B) \ge \tau$. In the following, we encode the latter inequality into an instance of WLCS.

For simplicity we only give the proof for integral α and $\alpha < \sigma$ (the remaining cases are omitted and can be found in the appendix). In this case, $N = M^\alpha$ and the running time lower bound that we will prove is $(nm)^{1-o(1)}$.

We set λ to any value such that $\lambda > |y_B|/M$, and note that $\lambda \in \Theta(D)$ suffices. Set $W(k) = \lambda \cdot M^{k-1}$ for $k \ge 2$, and $W(1) = W(0) = 1$. Let $\Sigma_k = \{0, 1, \ldots, k-1\}$. We construct strings x and y as follows:

$$\begin{aligned}
x &= x^{(\alpha,0)} \\
x^{(k,i)} &= \Big(\bigcirc_{j=0}^{M-2} x^{(k-1,M\cdot i+j)} \circ k \Big) \circ x^{(k-1,M\cdot i+(M-1))} \quad \text{for } 2 \le k \le \alpha \\
x^{(1,i)} &= x_A^i \quad \text{for } 0 \le i < N/M \\
y &= y^{(\alpha)} \\
y^{(k)} &= k^{M-1} \circ y^{(k-1)} \circ k^{M-1} \quad \text{for } 2 \le k \le \alpha \\
y^{(1)} &= y_B.
\end{aligned}$$

Observe that for all k, $x^{(k,i)}$ and $y^{(k)}$ are defined on Σ_k. In particular, since $\alpha \le \sigma - 1$ we only use symbols from Σ. Let $\ell(k)$ denote the length of $x^{(k,i)}$ for any i. Observe that $\ell(k) = M \cdot \ell(k-1) + (M-1)$ and $\ell(1) \in \Theta(DM)$. Thus, $\ell(k) \in \Theta(DM^k)$ and $n := |x| \in \Theta(DM^\alpha)$. It is straightforward to see that $m := |y| = \Theta((k+D)M) = \Theta(DM)$, since $k \le |\Sigma| = \mathcal{O}(1)$. Recall that for any string z, $W(z)$ is its total weight.

▶ **Claim 15.** *For any integer $2 \le k \le \alpha$, we have (1) $(M-1) \cdot \sum_{\ell=2}^{k} W(\ell) = \lambda(M^k - M)$ and (2) $W(y^{(k)}) < W(k+1) + \lambda \cdot (M^k - M)$.*

Proof. For (1), we calculate $(M-1) \cdot \sum_{\ell=2}^{k} W(\ell) = (M-1) \cdot \sum_{\ell=1}^{k-1} \lambda M^\ell = \lambda(M^k - M)$. For (2), by definition of $y^{(k)}$ and λ we have

$$W(y^{(k)}) < \lambda M + 2(M-1) \cdot \sum_{\ell=2}^{k-1} W(l) \overset{(1)}{=} \lambda M + 2\lambda(M^k - M) = W(k+1) + \lambda(M^k - M). \blacktriangleleft$$

We now can perform the core step of our correctness argument.

▶ **Lemma 16.** *For any $2 \le k \le \alpha$ and $0 \le i < M^{k-1}$, we have (1) $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) \ge \lambda(M^k - M)$, and (2) $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) = (M-1) \cdot W(k) + \mathrm{WLCS}(x^{(k-1,j)}, y^{(k-1)})$ for some $M \cdot i \le j < M \cdot (i+1)$.*

Proof. For (1), clearly $\bigcirc_{j=2}^{k} j^{M-1}$ is a common subsequence of $x^{(k,i)}$ and $y^{(k)}$. Together with Claim 15.(1), we obtain $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) \ge \sum_{j=2}^{k} (M-1) \cdot W(j) = \lambda(M^k - M)$.

For (2), we claim that k^{M-1} is a subsequence of any WLCS of $x^{(k,i)}$ and $y^{(k)}$. Assuming otherwise, the WLCS can contain at most $M-2$ symbols k and all of $y^{(k-1)}$. Therefore,

$$\begin{aligned}
\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) &\le (M-2) \cdot W(k) + W(y^{(k-1)}) \\
&< (M-2) \cdot W(k) + W(k) + \lambda(M^{k-1} - M) \quad \text{by Claim 15.(2)} \\
&= (M-1) \cdot \lambda M^{k-1} + \lambda(M^{k-1} - M) = \lambda \cdot (M^k - M).
\end{aligned}$$

This contradicts $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) \ge \lambda(M^k - M)$. It follows that k^{M-1} is a subsequence of the WLCS of $x^{(k,i)}$ and $y^{(k)}$. Hence, $\mathrm{WLCS}(x^{(k,i)}, y^{(k)}) = (M-1) \cdot W(k) + \mathrm{WLCS}(x^{(k-1,j)}, y^{(k-1)})$ for some j with $M \cdot i \le j < M \cdot (i+1)$. ◀

Recursively applying the above lemma and substituting $x^{(1,j)}$ by x_A^j, we conclude that $\mathrm{WLCS}(x,y) = \lambda \cdot (M^\alpha - M) + \max_{0 \le j < M^{\alpha-1}} \mathrm{LCS}(x_A^j, y_B)$. Using $M^\alpha = N$ and the construction of x_A^j, y_B, we obtain that $\mathrm{WLCS}(x,y) \ge \lambda(N-M)+\tau$ holds if and only if there is an orthogonal pair of vectors in A and B. Since OVH asserts that solving the OV instance (A,B) in the worst case requires time $(NM)^{1-o(1)}$, even for $D = N^{o(1)}$, we obtain that determining $\mathrm{WLCS}(x,y)$ requires time $(NM)^{1-o(1)} = (nm/D^2)^{1-o(1)} = (nm)^{1-o(1)}$. This completes the proof for all instances where $\alpha < \sigma$ is integral. Note that if $\alpha \ge \sigma$, the claimed lower bound trivially holds as it matches the input size. Now we consider the two remaining cases, where $\sigma - 1 < \alpha < \sigma$ and $\alpha < \sigma - 1$.

Case $\sigma - 1 < \alpha < \sigma$. Then $N = M^{\alpha_I} = M^{\sigma-1}$. We construct strings x and y as follows:

$$x = x^{(\alpha_I,0)} \circ \alpha_I \circ 0^{DM^\alpha}$$
$$y = y^{(\alpha_I)} \circ \alpha_I.$$

Again, since $\alpha_I \le \sigma - 1$ the strings x and y only use symbols in Σ. We now have $n := |x| \in \Theta(DM^\alpha)$ and $m := |y| \in \Theta(DM)$. Clearly, $\mathrm{WLCS}(x,y) \ge \mathrm{WLCS}(x^{(\alpha_I,0)}, y^{(\alpha_I)}) + W(\alpha_I) \ge W(\alpha_I) + \lambda(M^{\alpha_I} - M)$. Similar to the proof for integral α, we claim that α_I^M is a subsequence of the WLCS of x and y. Assuming otherwise, the WLCS of x and y contains at most $M-1$ symbols α_I and all of $y^{(\alpha_I-1)}$. Therefore,

$$\begin{aligned}\mathrm{WLCS}(x,y) &\le (M-1)\cdot W(\alpha_I) + W(y^{(\alpha_I-1)})\\ &< (M-1)\cdot W(\alpha_I) + W(\alpha_I) + \lambda(M^{\alpha_I-1} - M) \quad \text{by Claim 15.(2)}\\ &= W(\alpha_I) + \lambda\cdot(M^{\alpha_I} - M^{\alpha_I-1} + M^{\alpha_I-1} - M) = W(\alpha_I) + \lambda(M^{\alpha_I} - M).\end{aligned}$$

This contradicts $\mathrm{WLCS}(x,y) \ge W(\alpha_I) + \lambda(M^{\alpha_I} - M)$. Hence, α_I^M is a subsequence of the WLCS of x and y, and $\mathrm{WLCS}(x,y) = W(\alpha_I) + \mathrm{WLCS}(x^{(\alpha_I,0)}, y^{(\alpha_I)})$. It follows that $\mathrm{WLCS}(x,y) \ge \lambda M^{\alpha_I-1} + \lambda(M^{\alpha_I} - M) + \tau$ holds if and only if there exists an orthogonal pair of vectors in A and B. OVH asserts that solving the OV instance (A,B) in the worst case requires time $(NM)^{1-o(1)}$, even for $D = N^{o(1)}$. Using $N = \Theta(M^{\alpha_I}) = \Theta(M^{\sigma-1})$, we obtain that determining $\mathrm{WLCS}(x,y)$ requires time $(NM)^{1-o(1)} = (M^\sigma)^{1-o(1)} = ((m/D)^\sigma)^{1-o(1)} = (m^{|\Sigma|})^{1-o(1)}$. This completes the proof in the case $\sigma - 1 < \alpha < \sigma$.

Case $\alpha < \sigma - 1$. In this case $\alpha_I \le \sigma - 2$ and $N = M^{\alpha_I} \cdot \lceil M^{\alpha_F} \rceil$. Let $f = \lceil M^{\alpha_F} \rceil$ as shorthand. We construct x and y as follows:

$$x = \left(\bigcirc_{j=0}^{f-2} x^{(\alpha_I,j)} \circ (\alpha_I + 1) \right) \circ x^{(\alpha_I,f-1)}$$
$$y = (\alpha_I + 1)^f \circ y^{(\alpha_I)} \circ (\alpha_I + 1)^f$$

Once again x and y consist of symbols in Σ, since $\alpha_I \le \sigma - 2$. Since $|x^{(\alpha_I,i)}| \in \Theta(DM^{\alpha_I})$, we have $n := |x| \in \Theta(DM^{\alpha_I+\alpha_F}) = \Theta(DM^\alpha)$, and $m := |y| \in \Theta(DM)$. The same argument as before, now with f instead of M parts, shows that $\mathrm{WLCS}(x,y) = (f-1)W(\alpha_I+1) + \mathrm{WLCS}(x^{(\alpha_I,j)}, y^{(\alpha_I)})$ holds for some $0 \le j < f$. Plugging in $\mathrm{WLCS}(x^{(\alpha_I,j)}, y^{(\alpha_I)})$, we see that

$$\mathrm{WLCS}(x,y) = \lambda(f-1)M^{\alpha_I} + \lambda(M^{\alpha_I} - M) + \max_{0 \le j \le \frac{N}{M}-1} \mathrm{LCS}(x_A^j, y_B).$$

Hence, $\mathrm{WLCS}(x,y) \ge \lambda(f-1)M^{\alpha_I} + \lambda(M^{\alpha_I} - M) + \tau$ holds if and only if there is an orthogonal pair of vectors in A and B. OVH asserts that solving the OV instance (A,B) in the worst

case requires time $(NM)^{1-o(1)}$, even for $D = N^{o(1)}$. Using $N = \Theta(M^{\alpha_I} \cdot f) = \Theta(M^{\alpha})$, we obtain that determining $\mathrm{WLCS}(x, y)$ requires time $(NM)^{1-o(1)} = (M^{\alpha+1})^{1-o(1)} = (nm/D^2)^{1-o(1)} = (nm)^{1-o(1)}$. This completes the proof of the last case $\alpha < \sigma - 1$.

Finally, note that in all cases we constructed strings over alphabet size σ of length $n = M^{\alpha \pm o(1)}$ and $m = M^{1 \pm o(1)}$, and thus $n = m^{\alpha \pm o(1)}$. ◀

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, 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.
3. 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.
4. Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5(1–2):78–99, 1995.
5. Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.
6. Amihood Amir, Zvi Gotthilf, and B. Riva Shalom. Weighted LCS. *Journal of Discrete Algorithms*, 8(3):273–281, 2010.
7. Hsing-Yen Ann, Chang-Biau Yang, Chiou-Ting Tseng, and Chiou-Yi Hor. A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings. *Information Processing Letters*, 108(6):360–364, 2008.
8. Alberto Apostolico. Improving the Worst-Case Performance of the Hunt-Szymanski Strategy for the Longest Common Subsequence of Two Strings. *Inf. Process. Lett.*, 23(2):63–69, 1986. doi:10.1016/0020-0190(86)90044-X.
9. Alberto Apostolico and Concettina Guerra. The Longest Common Subsequence Problem Revisited. *Algorithmica*, 2:316–336, 1987. doi:10.1007/BF01840365.
10. 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.
11. Djamal Belazzougui and Qin Zhang. Edit Distance: Sketching, Streaming, and Document Exchange. In *Proc. 57th Annual Symposium on Foundations of Computer Science*, pages 51–60, 2016.
12. 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. doi:10.1109/FOCS.2014.76.
13. 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.
14. Karl Bringmann and Marvin Künnemann. Multivariate Fine-Grained Complexity of Longest Common Subsequence. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, 2018. To appear.

15 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.

16 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for computing edit distance without exploiting suffix trees. *ArXiv:1607.03718*, 2016.

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

18 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.

19 David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse Dynamic Programming I: Linear Cost Functions. *J. ACM*, 39(3):519–545, July 1992. doi:10.1145/146637.146650.

20 Valerio Freschi and Alessandro Bogliolo. Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism. *Information Processing Letters*, 90(4):167–173, 2004.

21 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.

22 Daniel S. Hirschberg. Algorithms for the Longest Common Subsequence Problem. *J. ACM*, 24(4):664–675, 1977. doi:10.1145/322033.322044.

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

24 James W. Hunt and Thomas G. Szymanski. A Fast Algorithm for Computing Longest Subsequences. *Commun. ACM*, 20(5):350–353, 1977. doi:10.1145/359581.359603.

25 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.

26 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. doi:10.1007/s00224-008-9101-6.

27 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Computer and System Sciences*, 63(4):512–530, 2001.

28 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.

29 Hossein Jowhari. Efficient communication protocols for deciding edit distance. In *Proc. European Symposium on Algorithms*, pages 648–658, 2012.

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

31 Ilan Kremer, Noam Nisan, and Dana Ron. On Randomized One-Round Communication Complexity. *Computational Complexity*, 8(1):21–49, 1999.

32 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)*, 2017. To appear, arXiv:1703.04954.

33 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.

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

35 David Liben-Nowell, Erik Vee, and An Zhu. Finding longest increasing and common subsequences in streaming data. *Journal of Combinatorial Optimization*, 11(2):155–175, 2006.

36 Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

37 Jia Jie Liu, Yue-Li Wang, and Richard CT Lee. Finding a longest common subsequence between a run-length-encoded string and an uncompressed string. *Journal of Complexity*, 24(2):173–184, 2008.

38 Webb Miller and Eugene W. Myers. A File Comparison Program. *Softw., Pract. Exper.*, 15(11):1025–1040, 1985. doi:10.1002/spe.4380151102.

39 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.

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

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

42 Eugene W. Myers. An $O(ND)$ Difference Algorithm and Its Variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446.

43 Gene Myers. A four russians algorithm for regular expression pattern matching. *Journal of the ACM (JACM)*, 39(2):432–448, 1992.

44 Narao Nakatsu, Yahiko Kambayashi, and Shuzo Yajima. A Longest Common Subsequence Algorithm Suitable for Similar Text Strings. *Acta Inf.*, 18:171–179, 1982. doi:10.1007/BF00264437.

45 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.

46 Michael Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1698–1709, 2013.

47 Xiaoming Sun and David P. Woodruff. The communication and streaming complexity of computing the longest common and increasing subsequences. In *Proc. 18th Annual ACM-SIAM Aymposium on Discrete Algorithms*, pages 336–345, 2007.

48 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.

49 Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. *J. ACM*, 21(1):168–173, 1974. doi:10.1145/321796.321811.

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

51 Sun Wu, Udi Manber, Gene Myers, and Webb Miller. An $O(NP)$ Sequence Comparison Algorithm. *Inf. Process. Lett.*, 35(6):317–323, 1990. doi:10.1016/0020-0190(90)90035-V.

52 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.