3
Quadratic Lower Bounds for Sequence Similarity

Version: 172

In this chapter, we show how to derive quadratic-time conditional lower bounds for sequence similarity measures, using the example of the classic Longest Common Subsequence problem.

3.1 Sequence Comparison

Sequences come in very different flavors: strings, geometrical curves, time-series data, etc. On all of these types, a typical task is to *compare* them. Applications include spelling correction and differential file comparison (strings), map-matching GPS trajectories (geometrical curves), clustering audio sequences (time-series), genome assembly (DNA sequences), and many more. For these applications, a number of natural sequence similarity[1] measures exist: for strings, we have measures like the edit distance, longest common subsequence, local alignment; for polygonal curves, we have the Fréchet distance; for time-series data, we have dynamic-time warping; etc.

[1] Or, equivalently, *dis*similarity

Most of these **sequence similarity measures** have in common that they can be computed in time $\mathcal{O}(n^2)$ where n denotes the length of the two input sequences. For many practical applications, this is indeed acceptable: E.g., it is rarely noticeable that computing the `diff` of two files might take quadratic time on worst case instances. For other applications like comparing genome sequences of two biological species, however, input lengths can easily range in the billions, making quadratic-time algorithms infeasibly slow. This often leads practitioners to use heuristics instead, such as the BLAST tool[2] in bioinformatics.

[2] 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

Ideally, we would like to justify why for many sequence similarity measures, quadratic-time algorithms seem to be best possible:

Can we find out what makes a similarity measure hard*?*

Fortunately, we can attack this question using the tools of fine-grained complexity. In particular, after first results for *local alignment*[3], a number of OVH-based lower bounds have been shown for similarity

[3] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *ICALP'14*, pages 39–51, 2014

measures that optimize a global alignment: This line of research was initiated by Bringmann's quadratic conditional lower bound for the *Fréchet distance*[4]. Following this work, a breakthrough result by Backurs and Indyk[5] gave such a lower bound for the *edit distance*. Independent works by Abboud et al.[6] and Bringmann and Künnemann[7] then extended these results to further similarity measures including *Longest Common Subsequence*, *Dynamic-Time Warping*, and a generalization of the *Edit Distance*.

[4] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *FOCS'14*, pages 661–670, 2014

[5] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *STOC'15*, pages 51–58, 2015

[6] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *FOCS'15*, pages 59–78, 2015

[7] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS'15*, pages 79–97, 2015

In this chapter, we show how to derive such lower bounds using the example of the longest common subsequence problem. Here, we give a simplified version of the proof of Bringmann and Künnemann; stronger results can be obtained by more careful constructions.

3.2 *Longest Common Subsequence (LCS)*

We call a string z a **subsequence** of a string x, if z can be obtained by deleting arbitrary characters from x. A **common subsequence** of two strings x and y is a string that is a subsequence of both x and y.

Note that in contrast to a **substring**, a **subsequence** of x does not need to be contiguous in x.

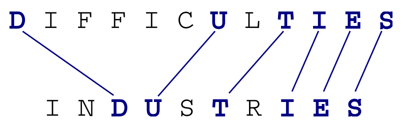


Figure 3.1: Example for an optimal LCS alignment.

Problem 3.1.

Longest Common Subsequence problem (LCS)
Given: *Strings x and y of length at most n*
Determine: *Length of the longest common subsequence of x and y.*

Let us denote the length of a longest common subsequence of x and y by $L(x, y)$. We can compute the value $L(x, y)$ by a simple dynamic-programming algorithm[8] in time $\mathcal{O}(n^2)$: We build a table T defined by $T[i, j] = L(x[1 \dots i], y[1 \dots j])$. It is not difficult to see that

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

Notation: for a string x and and $i \geq 0$, $x[1..i]$ denotes the prefix of the first i characters of x.

$$T[i, j] = \begin{cases} T[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\ \max\{T[i-1, j], T[i, j-1]\} & \text{if } x[i] \neq x[j]. \end{cases}$$

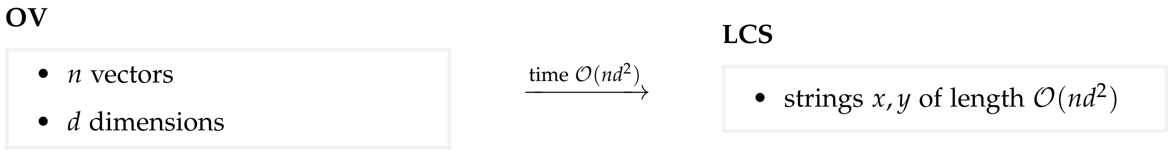
Thus, each entry $T[i, j]$ can be computed in time $\mathcal{O}(1)$, given the preceding entries $T[i-1, j], T[i, j-1], T[i-1, j-1]$. This allows us to compute $L(x, y) = T[|x|, |y|]$ in time $\mathcal{O}(|x| \cdot |y|) = \mathcal{O}(n^2)$. This running time has been slightly improved to $\mathcal{O}(n^2 (\frac{\log \log n}{\log n})^2)$ by Masek and Paterson[9], but for worst-case instances nothing faster is known.

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

The main result of this chapter is that the simple LCS algorithm given above is close to optimal under the OV hypothesis.

Theorem 3.2. *Assuming OVH, there is no $\mathcal{O}(n^{2-\varepsilon})$-time LCS algorithm for any $\varepsilon > 0$.*

We prove the theorem by giving a reduction that transforms an OV instance A, B into strings x, y and an integer τ such that A, B contains an orthogonal pair if and only if $L(x, y) \geq \tau$. The reduction satisfies the following bounds:



We then obtain the conditional lower bound as follows: Assume that there is a $\mathcal{O}(n^{2-\varepsilon})$-time algorithm for LCS. Then for any OV instance, we can run the reduction in time $\mathcal{O}(nd^2)$ to produce the corresponding x, y and τ. Using the LCS algorithm, we can compute $L(x,y)$ in time $\mathcal{O}((nd^2)^{2-\varepsilon}) = \mathcal{O}(n^{2-\varepsilon}\text{poly}(d))$, which allows us to decide the OV instance in time $\mathcal{O}(nd^2 + n^{2-\varepsilon}\text{poly}(d)) = \mathcal{O}(n^{2-\varepsilon}\text{poly}(d))$, which would refute OVH.

It remains to give the reduction, which we develop in the following few subsections. The general idea is to reformulate, piece by piece, the definition of the OV problem using strings and their longest common subsequences.

To prepare this, we introduce the following view of the LCS problem: Given strings x, y, we call a sequence $(i_1, j_1), \ldots, (i_\ell, j_\ell)$ with $x[i_k] = y[i_k]$ for all $k \in [\ell]$ an *alignment* of x and y. An *optimal* alignment is an alignment of longest possible length ℓ. Note that for an optimal alignment $(i_1, j_1), \ldots, (i_\ell, j_\ell)$, the string $x[i_1] \ldots x[i_\ell] = y[j_1] \ldots y[j_\ell]$ is an LCS of x and y. For any k, we say that $x[i_k]$ is *aligned* to $y[j_k]$. In particular, in the example of Figure 3.1, the blue lines indicate the *aligned* characters.

Note: There might be more than one optimal alignment (or even LCS) for x and y. However, we often speak of **the** optimal alignment (or **the** LCS) of x and y to denote an arbitrary, fixed optimal alignment (or LCS).

Coordinate Gadgets

Let us first represent coordinates of vectors by strings, called *coordinate gadgets*. The essential property is that the LCS of two coordinate gadgets expresses the **product** of the corresponding coordinates.

Lemma 3.3. *There are strings $C_A(0), C_A(1), C_B(0), C_B(1) \in \{0,1\}^3$ such that for all $s, t \in \{0,1\}$, we have*

$$L(C_A(s), C_B(t)) = 2(1 - s \cdot t) = \begin{cases} 2 & \text{if } s \cdot t = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. The following strings satisfy the desired condition:

$$C_A(0) := 001, \qquad C_B(0) := 011,$$
$$C_A(1) := 111, \qquad C_B(1) := 000.$$

Indeed, $C_A(1) = 111$ and $C_B(1) = 000$ share no characters, while all other combinations $C_A(s), C_B(t)$ (where $s \cdot t = 0$) have an LCS of length 2: the LCS of $C_A(0) = 001$ and $C_B(1) = 000$ is 00, the LCS of $C_A(1) = 111$ and $C_B(0) = 011$ is 11, and finally, the LCS of $C_A(0) = 001$ and $C_B(0) = 011$ is 01. □

Vector Gadgets

Given the coordinate gadgets, we turn to representing vectors by strings, called *vector gadgets*. The essential property is that the LCS of two vector gadgets expresses the **inner product** of the corresponding vectors.

Lemma 3.4. *There are $V_A, V_B : \{0,1\}^d \to \{0,1,2\}^{3d^2}$, computable in time $\mathcal{O}(d^2)$, such that for all vectors $a, b \in \{0,1\}^d$, we have*

$$L(V_A(a), V_B(b)) = \alpha_d - 2\sum_{k=1}^{d} a[k] \cdot b[k] \begin{cases} = \alpha_d & \text{if } \langle a,b \rangle = 0, \\ \leq \alpha_d - 2, & \text{otherwise,} \end{cases}$$

where $\alpha_d := 3d^2 - d$ is a fixed value.

Proof. We construct the vector gadgets as follows:

$$V_A(a) := C_A(a[1])\ 2^{3d}\ C_A(a[2])\ 2^{3d} \ldots\ 2^{3d}\ C_A(a[d]),$$
$$V_B(b) := C_B(b[1])\ 2^{3d}\ C_B(b[2])\ 2^{3d} \ldots\ 2^{3d}\ C_B(b[d]),$$

Notation: For a string x and a natural number $n \in \mathbb{N}$, we write x^n for the n-fold repetition of x, i.e., the string $\underbrace{x\,x\,\ldots\,x}_{n \text{ times}}$

We only need to show that

$$L(V_A(a), V_B(b)) = 3d(d-1) + \sum_{k=1}^{d} L(C_A(a[k]), C_B(b[k])), \qquad (3.1)$$

since Lemma 3.3 then shows that

$$\begin{aligned} L(V_A(a), V_B(b)) &= 3d(d-1) + \sum_{k=1}^{d} 2(1 - a[k] \cdot b[k]) \\ &= 3d^2 - d - 2\sum_{k=1}^{d} a[k]b[k], \end{aligned}$$

as desired.

We first prove the lower bound of (3.1). We obtain a common subsequence of $V_A(a)$ and $V_B(b)$ by concatenating, for $k = 1, \ldots, d-1$, an LCS of $C_A(a[k])$ and $C_B(b[k])$ and the string 2^{3d}, and finally concatenating an LCS of $C_A(a[d])$ and $C_B(b[d])$. This subsequence consists of $d-1$ blocks of 2s of length $3d$, as well as the sum of $L(C_A(a[k]), C_B(b[k]))$ for $k = 1, \ldots, d$, which yields $L(V_A(a), V_B(b)) \geq 3d(d-1) + \sum_{k=1}^{d} L(C_A(a[k]), C_B(b[k]))$.

It remains to prove the upper bound. Note that if the LCS of $V_A(a)$ and $V_B(b)$ **never aligns** a character of $C_A(a[i])$ with some character of $C_B(b[j])$ with $i \neq j$, then the LCS of $V_A(a)$ and $V_B(b)$ cannot be larger than $3d(d-1) + \sum_{k=1}^{d} L(C_A(a[k]), C_B(b[k]))$: there are only $3d(d-1)$-many 2s and each $C_A(a[k])$ can contribute at most a length of $L(C_A(a[k]), C_B(b[k]))$ to the LCS.

Otherwise, i.e., if there are $i \neq j$ such that some character of $C_A(a[i])$ is aligned to some character of $C_B(b[j])$, the LCS is even smaller: First observe that at least one block of 2s is not aligned to any other block of 2s. Indeed, if $i < j$, then there are only $i-1$ blocks of 2s preceding $C_A(a[i])$ in $V_A(a)$, so at least one of the $j-1 > i-1$ blocks of 2s preceding $C_B(b[j])$ in $V_B(b)$ cannot be aligned to any block of 2s in $V_A(a)$. Thus, even if we align all characters of all $C_A(a[k]), k \in [d]$ perfectly, the LCS still has a length of at most

$$3d(d-2) + \sum_{k=1}^{d} |C_A(a[k])| = 3d(d-2) + 3d = 3d(d-1).$$

This is smaller then the desired upper bound of (3.1), and we are finished. □

Normalized Vector Gadgets

For technical reasons that become apparent later, we need an adaptation of vector gadgets that we call *normalized vector gadgets*. The essential property is that the LCS of two normalized vector gadgets attains one of two values: a large one if the corresponding vectors are **orthogonal**, a small one if they are **not orthogonal**.

Lemma 3.5. *There are $N_A, N_B : \{0,1\}^d \to \{0,1,2,3\}^{6d^2-d-2}$, computable in time $\mathcal{O}(d^2)$, such that for all vectors $a, b \in \{0,1\}^d$, we have*

$$L(N_A(a), N_B(b)) = \begin{cases} \beta_d + 2 & \text{if } \langle a,b\rangle = 0, \\ \beta_d, & \text{otherwise,} \end{cases}$$

where $\beta_d := \alpha_d - 2 = 3d^2 - d - 2$ is a fixed value.

Proof. We construct the normalized vector gadgets as follows

$$\begin{aligned} N_A(a) &:= \; V_A(a) \quad 3^{\alpha_d - 2}, \\ N_B(b) &:= \; 3^{\alpha_d - 2} \quad V_B(b). \end{aligned}$$

We only need to show that

$$L(N_A(a), N_B(b)) = \max\{L(V_A(a), V_B(b)), \alpha_d - 2\}, \tag{3.2}$$

since the claim then follows from Lemma 3.4: If $\langle a,b\rangle = 0$, then $L(V_A(a), V_B(b)) = \alpha_d$ and thus $L(N_A(a), N_B(b)) = \alpha_d = \beta_d + 2$. Otherwise, we have $L(V_A(a), V_B(b)) \leq \alpha_d - 2$, so that $L(N_A(a), N_B(b)) = \alpha_d - 2 = \beta_d$, as desired.

We distinguish two cases: If the LCS of $N_A(a)$ and $N_B(b)$ contains at least one 3, then it must in fact be the string $3^{\alpha_d - 2}$: no other characters can be aligned, as all non-3s *precede* the 3s in $N_A(a)$, while in $N_B(b)$, all non-3s *succeed* the 3s.

Otherwise, i.e., if the LCS of $N_A(a)$ and $N_B(b)$ contains no 3, then it is the LCS of $V_A(a)$ and $V_B(b)$. Thus, we obtain as LCS of $N_A(a), N_B(b)$ the longer string of $3^{\alpha_d - 2}$ and the LCS of $V_A(a), V_B(b)$, concluding the proof of (3.2). □

OR-Gadget

Finally, it remains to represent sets of vectors as strings whose LCS expresses whether or not the sets contain an orthogonal pair. Crucially, we need the possibility that in principle, *any* pair of normalized vector gadgets could be aligned in an optimal alignment.

Lemma 3.6. *Given vector sets $A, B \subseteq \{0,1\}^d$ of size n, in time $\mathcal{O}(nd^2)$, we can construct strings x, y and an integer τ such that*

$L(x,y) \geq \tau$ if and only if A, B contains an orthogonal pair.

Proof. We write $A = \{a_0, \dots, a_{n-1}\}, B = \{b_0, \dots, b_{n-1}\}$ and set $\gamma := 6d^2 - d - 2$. We define

$$\begin{aligned} x &:= N_A(a_0) \quad 4^\gamma \quad N_A(a_1) \quad 4^\gamma \quad \dots \quad N_A(a_{n-1}) \quad 4^\gamma \quad N_A(a_0) \quad 4^\gamma \quad N_A(a_1) \quad 4^\gamma \quad \dots \quad 4^\gamma \quad N_A(a_{n-1}) \\ y &:= \qquad 4^{n\gamma} \quad N_B(b_0) \quad 4^\gamma \quad N_B(b_1) \quad 4^\gamma \quad \dots \quad N_B(b_{n-1}) \quad 4^{n\gamma}. \end{aligned}$$

Intuitively, the LCS of x and y chooses an *offset* Δ such that we align each $N_B(b_j)$ with a partner $N_A(a_{j+\Delta \bmod n})$.

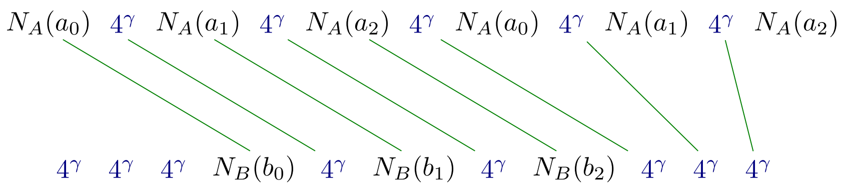
We set $\tau := (2n-1)\gamma + n\beta_d + 2$ and consider first the case that there exists $\bar{i}, \bar{j}$ such that $a_{\bar{i}}, b_{\bar{j}}$ are orthogonal. We need to show that $L(x,y) \geq \tau$.

First, we observe that

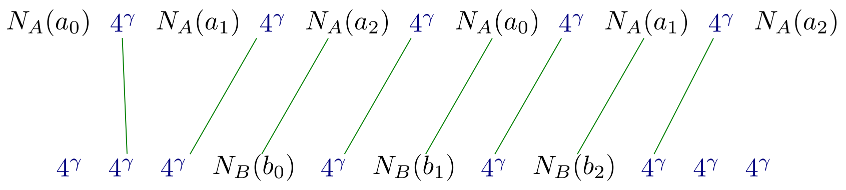
$$L(x,y) \geq (2n-1)\gamma + \max_{\Delta \in \{0,\dots,n-1\}} \sum_{j=0}^{n-1} L(N_A(a_{j+\Delta \bmod n}), N_B(b_j)). \quad (3.3)$$

Indeed, for any $\Delta = 0, \dots, n-1$, we obtain an LCS as follows: we align the first Δ blocks of 4s in x to the first $\Delta\gamma \leq n\gamma$ characters in y. Then, we optimally align, for each $j = 0, \dots, n-1$, each $N_B(b_j)$ with the corresponding $N(a_{j+\Delta \bmod n})$[10], as well as all the blocks of 4s in-between. Finally, we align the remaining $(n-\Delta)$ blocks of 4 in x to the last $(n-\Delta)\gamma \leq n\gamma$ many 4s in y. This yields an LCS that includes $\Delta + (n-1) + n - \Delta = 2n-1$ blocks of 4s, and the LCS's of $N_A(a_{j+\Delta \bmod n})$ and $N_B(b_j)$ for all $j \in \{0, \dots, n-1\}$, which proves (3.3), as Δ was chosen arbitrarily.

[10] Note that by our 0-based indexing and since x has two repetitions of the normalized vector gadgets for $a_0, \dots, a_{n-1}$, the use of $j + \Delta \bmod n$ correctly handles the alignment of those $N_B(b_j)$ for which $\Delta + j \geq n$.


$N_A(a_0)$ 4^γ $N_A(a_1)$ 4^γ $N_A(a_2)$ 4^γ $N_A(a_0)$ 4^γ $N_A(a_1)$ 4^γ $N_A(a_2)$
4^γ 4^γ 4^γ $N_B(b_0)$ 4^γ $N_B(b_1)$ 4^γ $N_B(b_2)$ 4^γ 4^γ 4^γ

(a) $\Delta = 0$


$N_A(a_0)$ 4^γ $N_A(a_1)$ 4^γ $N_A(a_2)$ 4^γ $N_A(a_0)$ 4^γ $N_A(a_1)$ 4^γ $N_A(a_2)$
4^γ 4^γ 4^γ $N_B(b_0)$ 4^γ $N_B(b_1)$ 4^γ $N_B(b_2)$ 4^γ 4^γ 4^γ

(b) $\Delta = 2$

Figure 3.2: Example for optimal alignments of strings x, y.

Let us choose $\Delta := \bar{i} - \bar{j} \bmod n$. Since $\bar{j} + \Delta \bmod n = \bar{i} \bmod n = \bar{i}$, it follows that the sum $\sum_{j=0}^{n-1} L(N_A(a_{j+\Delta \bmod n}), N_B(b_j))$ contains the summand $L(N_A(a_{\bar{i}}), N_B(b_{\bar{j}})) \geq \beta_d + 2$ by Lemma 3.5. Each of the other $n-1$ summands has a value of at least β_d by Lemma 3.5.[11] Thus, for the case that there exists an orthogonal pair, (3.3) yields the lower bound

$$L(x,y) \geq (2n-1)\gamma + (n-1)\beta_d + (\beta_d + 2) = \tau.$$

[11] Note that this precisely the point where we need the normalization of the vector gadgets. If we were working with the simple vector gadgets, a large value for a single pair might be insignificantly small compared to potentially very small values for all other pairs participating in the sum.

It remains to show that $L(x,y) < \tau$ if there is no orthogonal pair. Note that x has $(2n-1)\gamma$ many 4s, so any LCS of x and y can contain at most as many 4s. So, we assume that the 4s have a contribution of $(2n-1)\gamma$ to the LCS. Let us further consider the contribution of the $N_B(b_j)$'s to the longest common subsequence: If no character of $N_B(b_j)$ is aligned to any character of some $N_A(a_i)$, then $N_B(b_j)$ has a contribution of 0. If the characters of $N_B(b_j)$ are aligned only to characters of a single $N_A(a_i)$, then $N_B(b_j)$ has a contribution of $L(N_A(a_i), N_B(b_j)) \leq \beta_b$ by Lemma 3.5 (where we use that a_i, b_j are not orthogonal). Finally, if there are characters of $N_B(b_j)$ that are aligned to characters of at least two $N_A(a_i)$'s, then we can bound its contribution by $|N_B(b_j)| - \gamma = 0$: $N_B(b_j)$ can contribute at most its length $|N_B(b_j)| = \gamma$, but if some character of $N_B(b_j)$ is aligned to some $N_A(a_i)$, and another character of $N_B(b_j)$ is aligned to some other $N_A(a_{i'})$, then we cannot align any 4s between $N_A(a_i)$ and $N_A(a_{i'})$, so we lose at least γ many 4s (reducing the contribution of the 4s correspondingly). In all cases, the contributions of $N_B(b_j)$ is bounded by β_d, and the total length of the LCS, for the case that there is no orthogonal pair, is at most

$$(2n-1)\gamma + n\beta_d = \tau - 2 < \tau. \qquad \square$$

Note that the previous lemma proves the reduction that we claimed for Theorem 3.2, thus concluding our conditional lower bound for LCS.

3.3* Extensions of the result

We have given a proof that solving LCS on strings over an alphabet of size 5 cannot be done in strongly quadratic time, assuming OVH. In fact, using more careful constructions[12], one can show that this lower bound already holds on an **alphabet of size 2** – note that this is optimal, since computing the LCS of strings over alphabet size 1 is trivial.

[12] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS'15*, pages 79–97, 2015

To achieve the lower bound, it is technically useful to generalize the essential tasks in the reduction above (constructing vector gadget, normalized vector gadgets, and the OR gadget) to an abstract task – essentially, optimizing over certain kinds of alignments. Formalizing this task as a construction called *alignment gadget*, one can show that for any similarity measure admitting such an alignment gadget, we immediately obtain an OVH-based quadratic lower bound. Since **Longest Common Subsequence**, **Dynamic Time-Warping**, and a generalization of the **Edit Distance** admit such an alignment gadget, we obtain quadratic lower bounds for all of them.

Let us discuss further extensions of the result.

Lower Order Improvements

Due to the above conditional lower bound, we do not expect $\mathcal{O}(n^{2-\varepsilon})$-time algorithms for LCS. However, a slightly subquadratic algorithm, running in time $\mathcal{O}(n^2 \cdot (\frac{\log\log n}{\log n})^2)$, exists. How much could this algorithm be improved? Could we possibly shave off arbitrarily many logarithmic factors?

Abboud et al.[13] show that this is an immensely difficult task, as an $\mathcal{O}(n^2/\log^{1000} n)$ algorithm for LCS would imply circuit lower bounds that are beyond the current state of the art. In fact, they obtain this result by basing the quadratic hardness of LCS on a weaker assumption than SETH, specifically, a variant of it concerning the satisfiability of branching programs. Subsequent work[14] obtains more detailed lower bounds.

[13] 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 *STOC'16*, pages 375–388, 2016

[14] Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. In *ICALP'18*, pages 8:1–8:18, 2018

Multivariate Analysis

A practical application of LCS is comparing two versions of a file: Viewing each line of a file as a character of a string, the LCS of two files corresponds to their similarity, and all lines not occurring in the LCS are essentially lines that have been edited between the versions.

Given this relevance, it is not surprising that algorithmic work has attempted to find fast LCS algorithms even despite its apparent quadratic time hardness. A particular approach is to exploit input parameters: Let us characterize an LCS input not only by its input size (the total length of the input strings), but also by distinguishing the following *parameters*: (here, we assume that x is the longer input string, i.e., $|x| \geq |y|$):

- the length $n = |x|$ of the longer string,
- the length $m = |y|$ of the shorter string,
- the length $L = L(x, y)$ of the LCS of x, y,
- the size of the alphabet Σ of x, y,
- the number $\delta := m - L$ of characters we need to delete in the shorter string to obtain the LCS,
- the number $\Delta := n - L$ of characters we need to delete in the longer string to obtain the LCS.
- the number of *matching pairs*[15] M,
- the number of *dominant pairs*[16] d.

[15] A **matching pair** is a pair (i, j) such that $x[i] = y[j]$.

[16] A **dominant pair** is a certain type of matching pair. Specifically, it is a matching pair (i, j) such that in the dynamic programming table T, we have $T[i, j] > T[i-1, j]$ and $T[i, j] > T[i, j-1]$.

In practical instances, one would expect certain parameters to be small. This holds in particular for δ and Δ (as we typically compare files with a large LCS L); one can make a similar case for the parameter $d \leq M$. Fortunately, there are fast algorithms making use of these parameters: we have algorithms running in time $\tilde{O}(n + m\delta)$[17], $\tilde{O}(n + \delta\Delta)$[18] and $\tilde{O}(n + d)$ [19]. In the worst case, these algorithms require time $\Omega(n^2)$, i.e., they do not break the OVH barrier. On practical instances, however, these algorithms can be much faster. Surprisingly, we can show that the combination of these three algorithms is essentially best possible under OVH[20].

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

[18] Sun Wu, Udi Manber, Gene Myers, and Webb Miller. An O(NP) sequence comparison algorithm. *Inf. Process. Lett.*, 35(6):317–323, 1990

[19] 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

[20] Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *SODA'18*, pages 1216–1235, 2018

Theorem 3.7. *Assuming OVH, the optimal running time (measured in terms of the parameters $n, m, L, \delta, \Delta, M$, and d) for LCS over constant-sized alphabets is $(n + \min\{\delta\Delta, \delta m, d\})^{1 \pm o(1)}$.*

How to prove such lower bounds (taking into account more than one input parameter) will be the topic of a future chapter.

Further results

Further hardness results include a $n^{k-o(1)}$-time conditional lower bound[21] for computing the LCS of k strings $x_1, \ldots, x_k$. Tight conditional lower bounds can also be achieved for computing the LCS of *grammar compressed* strings[22]. Furthermore, there are fine-grained complexity analyses of variants of LCS such as the Longest Common Increasing Subsequence[23] and a weighted version of LCS[24]. Further work discusses approximation hardness of LCS-related problems, which will be discussed in a future chapter.

[21] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *FOCS'15*, pages 59–78, 2015

[22] Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In *FOCS'17*, pages 192–203, 2017

[23] Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. In *(IPEC'17)*, pages 15:1–15:13, 2017

[24] Karl Bringmann and Bhaskar Ray Chaudhury. Sketching, streaming, and fine-grained complexity of (weighted) LCS. In *FSTTCS'18*, pages 40:1–40:16, 2018

Bibliography

Version: 172

Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. In *ICALP'18*, pages 8:1–8:18, 2018.

Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *ICALP'14*, pages 39–51, 2014.

Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *FOCS'15*, pages 59–78, 2015.

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 *STOC'16*, pages 375–388, 2016.

Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In *FOCS'17*, pages 192–203, 2017.

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.

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.

Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *STOC'15*, pages 51–58, 2015.

Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *FOCS'14*, pages 661–670, 2014.

Karl Bringmann and Bhaskar Ray Chaudhury. Sketching, streaming, and fine-grained complexity of (weighted) LCS. In *FSTTCS'18*, pages 40:1–40:16, 2018.

Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *FOCS'15*, pages 79–97, 2015.

Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *SODA'18*, pages 1216–1235, 2018.

Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. In *(IPEC'17)*, pages 15:1–15:13, 2017.

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

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

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

Sun Wu, Udi Manber, Gene Myers, and Webb Miller. An O(NP) sequence comparison algorithm. *Inf. Process. Lett.*, 35(6):317–323, 1990.