An FPT-Algorithm for Longest Common Subsequence Parameterized by the Maximum Number of Deletions

Laurent Bulteau, Mark Jones, Rolf Niedermeier, Till Tantau

An FPT-Algorithm for Longest Common Subsequence Parameterized by the Maximum Number of Deletions

Laurent Bulteau[1]
LIGM, CNRS, Université Gustave Eiffel, F77454 Marne-la-vallée, France

Mark Jones
Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands

Rolf Niedermeier
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany

Till Tantau
Institute of Theoretical Computer Science, University of Lübeck, Germany

Abstract

In the NP-hard Longest Common Subsequence problem (LCS), given a set of strings, the task is to find a string that can be obtained from every input string using as few deletions as possible. LCS is one of the most fundamental string problems with numerous applications in various areas, having gained a lot of attention in the algorithms and complexity research community. Significantly improving on an algorithm by Irving and Fraser [CPM'92], featured as a research challenge in a 2014 survey paper, we show that LCS is fixed-parameter tractable (FPT) when parameterized by the maximum number of deletions per input string. Given the relatively moderate running time of our algorithm (linear time when the parameter is a constant) and small parameter values to be expected in several applications, we believe that our purely theoretical analysis could finally pave the way to a new, exact and practically useful algorithm for this notoriously hard string problem.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Theory and algorithms for application domains

Keywords and phrases NP-hard string problems, multiple sequence alignment, center string, parameterized complexity, search tree algorithms, enumerative algorithms

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.6

Funding *Mark Jones*: Supported by Netherlands Organization for Scientific Research (NWO) through grants NETWORKS and OCENW.KLEIN.125.
Rolf Niedermeier: Supported by the DFG, NI 369/16, FPTinP.

Acknowledgements This work was initiated during Dagstuhl Seminar 19443, *Algorithms and Complexity in Phylogenetics*, held at Schloss Dagstuhl, Germany, in October 2019 [3].

Dedication to Rolf We dedicate this paper to our dear friend Rolf Niedermeier, who tragically passed away recently. We are deeply affected by this loss of our co-author, colleague, and advisor. Rolf has contributed tremendously to computer science and, in particular, to multivariate algorithms, and should have continued doing so for a long time. The computer science community will build on the foundations he has laid.

[1] Corresponding author

1 Introduction

With its numerous applications including bioinformatics, data compression, and computational linguistics, the NP-hard Longest Common Subsequence (LCS) problem is among the best studied algorithmic string problems. Suiting our subsequent parameterized analysis purposes, we formally define the problem as follows.

Longest Common Subsequence
Input: A set of k strings $\mathcal{S} = \{S_1, \ldots, S_k\}$ on some alphabet Σ, each of length at most n, an integer ℓ.
Parameter: $\Delta = n - \ell$.
Question: Is there a string S of length at least ℓ that is a (not necessarily contiguous) subsequence of each S_i?

For example, for $k = 3$ strings *abcabac*, *acbabc*, *ababcba* (thus, $n = 7$), with $\ell = 5$ we have a yes-instance (with solution string *ababc*), while for $\ell = 6$ this would be a no-instance.

With straightforward dynamic programming, using the number k of strings as a parameter, LCS can be solved in $O(n^k)$ time. The problem has been shown to be W[1]-hard [16], and even W[2]-hard for parameter k, and it has no $O(n^{k-\epsilon})$ algorithm assuming the Strong Exponential Time Hypothesis (SETH) [1]. Indeed, LCS is *the* string problem having received most attention in the early years of parameterized complexity analysis [10]. Unfortunately, so far parameterized complexity analysis beyond trivial algorithmic observations mainly contributed computational hardness results. We refer to some surveys [2, 7, 8] for an overview on research results and open questions for LCS.

We remark that the special case of two input strings (that is, $k = 2$) recently attracted much attention, particularly motivated by the theoretical challenge of breaking the straightforward time bound of $O(n^2)$ [4, 6, 11]. Notably, Bringmann and Künnemann [6] (the corresponding arXiv paper has around 60 pages) also discuss the "maximum number of deletions" parameter we focus on here. We note however that in the context of $k = 2$, the parameter is used to improve over the classical $O(n^2)$ dynamic programming algorithm while maintaining a polynomial running time, while our approach requires an exponential dependency on Δ even for $k = 2$. Indirectly, this parameter already appears in the work of Irving and Fraser [13], who provided two algorithms for LCS with three or more input strings.

Irving and Fraser [13] in their 1992 paper provided an algorithm for LCS running in time $O(kn(n-\ell)^{k-1})$, implying fixed-parameter tractability with respect to the combined parameter k and $n-\ell$, where the latter coincides with our parameter Δ. We are not aware of any improvement since then and this is also reflected by a corresponding challenge featured in a 2014 survey [7, Challenge 9]. Answering positively the research challenge posed there, we improve Irving and Fraser's result to fixed-parameter tractability with respect to only Δ. More specifically, our algorithm runs in time $O((\Delta+1)^{\Delta+1}kn)$, which means linear time when Δ is a constant. In addition, we can *enumerate all* longest common subsequences within this time. Given that it seems natural to assume that in many applications the sought common subsequence is fairly close to every input string (which would imply small values of Δ), this promises to be of also practical relevance. The focus of our work, however, is purely theoretical. Regarding the alphabet size, we focus on the general case where the alphabet is unbounded (in particular, the alphabet size is *not* hidden in the O of the running time). The question of whether our approach can be improved for constant-size alphabets (typically $\{0, 1\}$ or $\{A, T, C, G\}$) is left open.

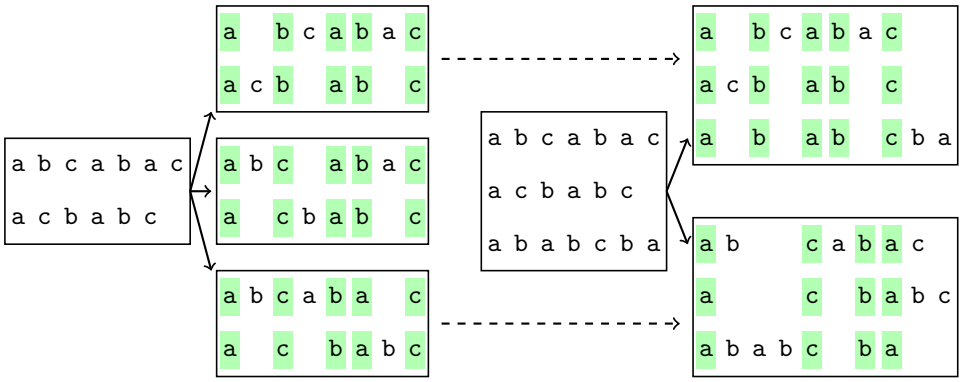

Figure 1 Our approach towards computing the LCS of three strings `abcabac`, `acbabc`, `ababcba`. Left: compute maximal common subsequences of the first two strings (all three subsequences and their alignment with input strings are depicted). Right: compute maximal common subsequences of all three strings by comparing those obtained at the first step with the third input string (only two strings remain after filtering non-maximal common subsequences). The longest result, `ababc` is the LCS of the input strings. Filtering out strings that are shorter than a threshold prevents the number of intermediate strings from growing too fast, yielding our FPT-algorithm.

Figure 1, at a very high level, presents an example for LCS for three input strings together with the main idea behind our recursive approach towards achieving our result, the FPT-algorithm for parameter Δ. More specifically, our algorithm builds and refines an exhaustive list of maximal common subsequences after reading each input string. Maximal common subsequences have been used before in LCS-related questions, see e. g. recent advances by Sakai [17] and Conte et al. [9], but not, to the best of our knowledge, towards exactly solving LCS on multiple input strings.

2 An LCS Algorithm Using Maximal Common Subsequences

In this section, we present a linear-time algorithm for LCS when the number of deletions is a constant. Note that this does not contradict the quadratic lower bound for this problem, since this lower bound only applies to the general case where the number of deletions is unbounded. In particular, the $O(\delta n)$ algorithm by Nakatsu et al. [14] (with $\delta = \min\{|S_i|\} - \ell$) remains better than our algorithm for the two-string case. Furthermore, it is not clear if a smaller (typically constant) alphabet could be exploited in our algorithm or its analysis in order to obtain a better running time.

2.1 Definitions

Strings. The set of strings on an alphabet Σ is denoted Σ^*. The empty string is denoted ϵ, the length of a string $S \in \Sigma^*$ is denoted $|S|$. We write $\cdot$ for the concatenation and $u \cdot T := S$ as a short-hand for "let $u \in \Sigma$ be the first character of S and T be the suffix of S starting from the second character (or $u = T = \epsilon$ if S is empty)". We write $S[i, \ldots, j]$ for the substring of S of all symbols between positions i and j inclusively.

Given two strings S_1, S_2, we write $S_1 \preceq S_2$ (resp. $S_1 \prec S_2$) if S_1 is a (strict) subsequence of S_2. Formally, $\epsilon \preceq S$ for any S and, if $S \preceq S'$, then for any u we have $S \preceq u \cdot S \preceq u \cdot S'$.

Longest and Maximal Common Subsequences. Given a set $\mathcal{S}$ of strings and a nonnegative integer ℓ, let $\mathrm{CS}_\ell(\mathcal{S})$ denote the set of all common subsequences of $\mathcal{S}$ that have length at least ℓ. Let L be the largest integer such that $\mathrm{CS}_L(\mathcal{S})$ is not empty, and let $\mathrm{LCS}(\mathcal{S})$ denote an arbitrary string in $\mathrm{CS}_L(\mathcal{S})$, i. e. a longest common subsequence of $\mathcal{S}$.

Let $\mathrm{MCS}_\ell(\mathcal{S})$ denote the set of all *maximal* common subsequences of $\mathcal{S}$ with length at least ℓ; that is, $S \in \mathrm{MCS}_\ell(\mathcal{S})$ iff $S \in \mathrm{CS}_\ell(\mathcal{S})$ and there is no $S' \in \mathrm{CS}_\ell(\mathcal{S})$ such that $S \prec S'$. Note that, if ℓ is small enough ($\ell \leq L$), then $\mathrm{LCS}(\mathcal{S}) \in \mathrm{MCS}_\ell(\mathcal{S})$, otherwise $\mathrm{MCS}_\ell(\mathcal{S})$ is empty. A set of strings M is an *extended MCS* of $(\mathcal{S}, \ell)$ if $\mathrm{MCS}_\ell(\mathcal{S}) \subseteq M \subseteq \mathrm{CS}_\ell(\mathcal{S})$.

String Parameters. Let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be a set of strings. We write $n(\mathcal{S}) := \max_{S\in\mathcal{S}} |S|$, $m(\mathcal{S}) := \min_{S\in\mathcal{S}} |S|$. Given an integer ℓ, we write $\Delta(\mathcal{S}, \ell) = n(\mathcal{S}) - \ell$ and $\delta(\mathcal{S}, \ell) := m(\mathcal{S}) - \ell$. We omit dependencies on $\mathcal{S}$ and ℓ when the context is clear (e. g., they are given in the lemma statement). Note that $\delta \leq \Delta$.

2.2 Main Results

▶ **Theorem 1.** *Let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be a set of strings and ℓ be an integer. Then an extended MCS of $(\mathcal{S}, \ell)$ with size at most $(\Delta + 1)^\delta$ can be computed in time $O(2^{\delta+\Delta}(\Delta + 1)^\delta kn)$.*

Theorem 1 directly yields an algorithm for LCS, since it suffices to test if an extended MCS of $(\mathcal{S}, \ell)$ is non-empty. Note that the algorithm can be adapted for the optimization formulation of LCS, i. e., when ℓ is not part of the input, with a constant factor in the time complexity (taking δ and Δ with respect to $\ell = |\mathrm{LCS}(\mathcal{S})|$). Indeed, apply Theorem 1 for decreasing values of ℓ starting with $\ell = m$, until a non-empty set is obtained. Then, the resulting set contains the common subsequences of $\mathcal{S}$ of size $\mathrm{LCS}(\mathcal{S})$ (indeed, $\mathrm{MCS}_\ell(\mathcal{S}) = \mathrm{CS}_\ell(\mathcal{S})$ for this value of ℓ), so it contains all longest common subsequences of $\mathcal{S}$. The time complexity of the i-th call, $1 \leq i \leq \delta$, is upper-bounded by $O(2^{i+\Delta}(\Delta + 1)^\delta kn)$. Using $\sum_{i=1}^{\delta} 2^i = O(2^\delta)$, we get the following corollary.

▶ **Corollary 2.** *All longest common subsequences of $\mathcal{S}$ (and a fortiori the value $\mathrm{LCS}(\mathcal{S})$) can be computed in time $O(2^{\delta+\Delta}(\Delta + 1)^\delta kn)$.*

The remainder of the section is dedicated to proving Theorem 1. We first compute the number of strings and their size distribution in the MCS of two strings, then build up on this result to bound the size of the MCS of k strings.

2.3 Extended MCS for Two Strings

Algorithm 1 allows us to compute an extended MCS of two strings. Its correctness is proven using the main recursive relation for MCS given in Lemma 3, while its time complexity is analyzed in Lemmas 6 and 8.

▶ **Lemma 3.** *For any two non-empty strings $S, S' \in \Sigma^*$ and any ℓ, let $u \cdot T := S$ and $u' \cdot T' := S'$ for $u, u' \in \Sigma$.*

If $u = u'$, then $\mathrm{MCS}_\ell(\{S, S'\}) \subseteq \{u \cdot X \mid X \in \mathrm{MCS}_{\ell-1}(\{T, T'\})\}$.

If $u \neq u'$, then $\mathrm{MCS}_\ell(\{S, S'\}) \subseteq \mathrm{MCS}_\ell(\{S, T'\}) \cup \mathrm{MCS}_\ell(\{T, S'\})$.

Proof. Let $R \in \mathrm{MCS}_\ell(\{S, S'\})$, and $r \cdot X := R$.

For the first case ($u = u'$), we show that $r = u$ and X is a maximal common subsequence of $\{T, T'\}$ of length at least $\ell - 1$. Indeed, $r = u$, as otherwise the concatenation $u \cdot r \cdot X$ would also be a common subsequence of $\{S, S'\}$, with $R \prec u \cdot r \cdot X$ (contradicting the

Algorithm 1 Compute a bounded-size extended MCS of two strings.

```
1  algorithm xMCS2(ℓ, S, S′)
2      if ℓ > |S| or ℓ > |S′| then return ∅
3      if S ⪯ S′ then return {S}
4      if S′ ⪯ S then return {S′}
5      u · T := S
6      u′ · T′ := S′
7      if u = u′ then
8          return {u · X | X ∈ xMCS2(ℓ − 1, T, T′)}
9      else
10         return xMCS2(ℓ, S, T′) ∪ xMCS2(ℓ, T, S′)
```

maximality of R). Note that X is a subsequence both of T and T'. Moreover X is maximal, as otherwise, if $X \prec X'$ with X' a common subsequence of T, T', then $u \cdot X'$ would be a common subsequence of S, S' with $R \prec u \cdot X'$ (again, contradicting the maximality of R).

For the second case ($u \neq u'$), we show that R is either in $\mathrm{MCS}_\ell(\{T, S'\})$, or in $\mathrm{MCS}_\ell(\{S, T'\})$ (or both). Indeed, if $r \neq u$, then $R \prec S$ implies $R \prec T$, and R is a common subsequence of $\{T, S'\}$. Otherwise, $r = u \neq u'$, and R is a common subsequence of $\{S, T'\}$. In both cases, R is maximal, since for any R', if R' is a common subsequence of (say) $\{T, S'\}$ with $R \prec R'$, then R' is also a common subsequence of $\{S, S'\}$ which contradicts the maximality of R for $\{S, S'\}$. ◀

Algorithm 1 follows the recursive relation of Lemma 3, along with trivial base cases ($\ell > \min\{|S|, |S'|\}$ or one of S or S' being a subsequence of the other). It also clearly returns only common subsequences of S and S' of length at least ℓ, so it is correct.

▶ **Corollary 4.** *Let $S, S' \in \Sigma^*$ and ℓ be an integer. Then* `xMCS2`(ℓ, S, S') *from Algorithm 1 returns an extended MCS of $(\{S, S'\}, \ell)$.*

▶ Remark 5. The first inclusion in Lemma 3 (case $u = u'$) is actually an equality, but we only need this direction for the algorithm to be correct. The second inclusion, however, may be strict: for example with $S = abcd$ and $S' = dabc$, the string $R = bc$ is a maximal common subsequence of $T = bcd$ and S', but not of S and S' since $R \prec abc$. Such "extra" strings are actually returned by our algorithm, motivating the naming of *extended* MCS (although they could be filtered out, see Remark 7).

We now focus on the time complexity of Algorithm 1.

▶ **Lemma 6.** *Let $S, S' \in \Sigma^*$ have lengths $\ell+\delta$ and $\ell+\Delta = n$, respectively. Then* `xMCS2`(ℓ, S, S') *terminates in time $O(2^{\delta+\Delta}n)$.*

Proof. To achieve the claimed time complexity, we first need to perform the subsequence tests in lines 3 and 4 quickly. For this, we use a precomputed subsequence table: For every pair (i, i') with $1 \in \{1, \ldots, |S|\}$ and $i' \in \{1, \ldots, |S'|\}$ and $|i - i'| \leq \Delta$, let $\texttt{sub}[i, i']$ contain `True` if $S[i, \ldots, |S|]$ is a subsequence of $S'[i', \ldots, |S'|]$. In other words, $\texttt{sub}[i, i']$ is `True` iff the ith suffix of S is a subsequence of the i'th suffix of S'. The entries of this table can be computed in time $O(n\Delta)$ by straightforward dynamic programming using the following relations:

$$\begin{aligned} &\texttt{sub}[i, i'] = \texttt{sub}[i+1, i'+1] && \text{if } S[i] = S[i'], \\ &\texttt{sub}[i, i'] = \texttt{sub}[i+1, i'] \vee \texttt{sub}[i, i'+1] && \text{otherwise.} \end{aligned}$$

Note that during recursive calls, the values of Δ and δ are non-increasing, and $\Delta + \delta$ decreases by 1 in the case where two recursive calls are performed. In particular, if $\ell \leq \min\{|S|, |S'|\}$ in a recursive call, then $\big||S|-|S'|\big| \leq \Delta$, which enables us to use the precomputed table for the subsequence test. So the total number of leaves in the tree of recursive calls is at most $2^{\delta+\Delta}$, each call taking constant time, and the height of this tree is at most $\ell + \delta + \Delta \leq 2n$. Thus the algorithm takes overall time $O(2^{\delta+\Delta}n)$. ◀

▶ Remark 7. Algorithm 1 can be adapted to output only the set of maximal common subsequences, rather than an extended version of it, by simply removing non-maximal strings (which can be done in quadratic time in the size of the output set). However, this does not improve the theoretical size of the returned set since in the worst case it does not filter out any string, but adds a quadratic running time to the complexity. It should be an important step in an implementation of the algorithm, though, since an additive quadratic computation time would probably be quickly compensated by pruning a possibly exponential search tree.

The proof of Lemma 6 gives a first bound on the number of strings returned by `xMCS2` (namely, at most $2^{\delta+\Delta}$). We know that all strings have lengths between ℓ and m. However, we will need an additional ingredient for a more precise analysis of our algorithm for k rather then just two strings: There cannot be many strings of length almost m. Intuitively, a long string in the returned set corresponds to a leaf in the search tree with few branching nodes among its ancestors, which actually helps reducing the size of the search tree. On the other hand, a short string in the returned set will cause less branchings in our next algorithm. Thus, the following lemma describes the repartition of the number of maximal common subsequences of two strings based on their lengths. Note that we would obtain the same bound if we used the filtering step from Remark 7 (i. e., the same formula applies to the set $\mathrm{MCS}_\ell(\{S, S'\})$).

▶ **Lemma 8.** *Let ℓ, d, and d' be integers. Let $S, S' \in \Sigma^*$ be strings of lengths $\ell + d$ and $\ell + d'$, respectively, so $\{\delta, \Delta\} = \{d, d'\}$. Let N_i be the number of strings in $\texttt{xMCS2}(\ell, S, S')$ of length exactly $\ell + d' - i$. Then*

$$\sum_{i=0}^{d'} \frac{N_i}{(d+1)^i} \leq 1.$$

Proof. We prove this property by induction on $|S| + |S'|$.

If $\ell > \min\{|S|, |S'|\}$, then $\texttt{xMCS2}(\ell, \{S, S'\})$ is empty, and the inequality is valid. If S or S' is a subsequence of the other, then $|\texttt{xMCS2}(\ell, S, S')| = 1$, so we have $N_i = 1$ for some i and $N_j = 0$ for $j \neq i$. The above inequality is true in this case as well. Note that this includes the cases where S or S' are empty. In the remaining cases, S and S' are not subsequences of each other, so in particular they are not empty. Let $u \cdot T := S$ and $u' \cdot T' := S'$.

If $u = u'$, then N_i is upper-bounded by the number of strings of length $(\ell - 1) + d' - i$ in $\texttt{xMCS2}(\ell - 1, T, T')$, so we can directly apply the property by induction to get $\sum_{i=0}^{d'} \frac{N_i}{(d+1)^i} \leq 1$.

Otherwise ($u \neq u'$), let N_i^a (resp. N_i^b) be the number of strings of length $\ell + d' - i$ in $\texttt{xMCS2}(\ell, S, T')$ (resp. $\texttt{xMCS2}(\ell, T, S')$). We have $N_i \leq N_i^a + N_i^b \leq 2N_i$ (accounting for the fact that a string counted in N_i must be counted once in one of N_i^a, N_i^b, and at most twice in total). Note that $N_0 = 0$ (otherwise, S and S' have a common subsequence of length $\ell + d' = |S'|$, which implies that S' is a subsequence of S). Thus $N_0^a = N_0^b = 0$.

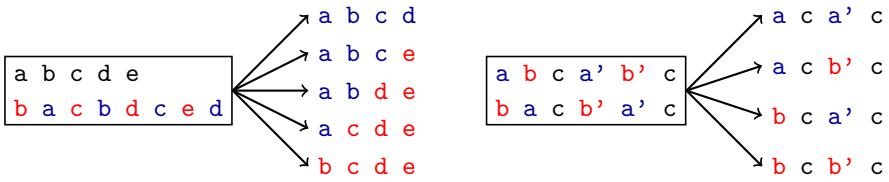

Figure 2 Examples of pairs of strings $\{S, S'\}$ with large $|\mathrm{MCS}_\ell(S, S')|$. Left: A pair with $\delta = 1$, $\Delta = 4$, and $|\mathrm{MCS}_\ell(S, S')| = 5 = 1 + \frac{\Delta}{\delta}$, showing that a dependency on Δ is unavoidable. Right: A pair with 2^δ maximal common subsequences, with $\delta = \Delta = 2$. Proposition 9 is a generalization of both examples that yields strings with $|\mathrm{MCS}_\ell(S, S')| = (\frac{\Delta}{\delta} + 1)^\delta$.

We apply the induction hypothesis first on the pair $\{S, T'\}$. Note that d' decreases by 1 and indices of N_i are shifted by 1, which gives $\sum_{i=0}^{d'-1} N^a_{i+1}/(d+1)^i \leq 1$, so

$$\sum_{i=0}^{d'} \frac{N^a_i}{(d+1)^i} = N^a_0 + \frac{1}{d+1} \sum_{i=1}^{d'} \frac{N^a_i}{(d+1)^{i-1}} \leq \frac{1}{d+1}.$$

Then the induction hypothesis on $\{T, S'\}$ (where d decreases by 1) gives $\sum_{i=0}^{d'} N^b_i/d^i \leq 1$, so

$$\begin{aligned}\sum_{i=0}^{d'} \frac{N^b_i}{(d+1)^i} &= N^b_0 + \sum_{i=1}^{d'} \frac{N^b_i}{(d+1)^i} \\ &= \frac{d}{d+1} \sum_{i=1}^{d'} \frac{d^{i-1}}{(d+1)^{i-1}} \frac{N^b_i}{d^i} \\ &\leq \frac{d}{d+1} \sum_{i=1}^{d'} \frac{N^b_i}{d^i} \leq \frac{d}{d+1}.\end{aligned}$$

Combining both inequalities yields:

$$\begin{aligned}\sum_{i=0}^{d'} \frac{N_i}{(d+1)^i} &\leq \sum_{i=0}^{d'} \frac{N^a_i}{(d+1)^i} + \sum_{i=0}^{d'} \frac{N^b_i}{(d+1)^i} \\ &\leq \frac{d}{d+1} + \frac{1}{d+1} = 1.\end{aligned}$$

◀

Lemma 8 yields an upper bound of $(\Delta + 1)^\delta$ on the size of $\mathrm{MCS}_\ell(S, S')$ (indeed, using $d = \Delta$ and $d' = \delta$, we have $\frac{|\mathrm{MCS}_\ell(S,S')|}{(\Delta+1)^\delta} \leq \sum_{i=0}^{\delta} \frac{N_i}{(\Delta+1)^\delta} \leq 1$). Examples (see Figure 2 and Proposition 9) indicate that this bound is close to being tight, since there exist instances where $|\mathrm{MCS}_\ell(S, S')| = (\frac{\Delta}{\delta} + 1)^\delta$.

▶ **Proposition 9.** *For any integers u and v, there exist an ℓ, an alphabet Σ, and strings $S, S' \in \Sigma^*$ of lengths $\ell + v$ and $\ell + uv$, respectively, such that $|\mathrm{MCS}_\ell(S, S')| \geq (u + 1)^v = (\frac{\Delta}{\delta} + 1)^\delta$.*

Proof. Let $\ell = uv$, and $\Sigma = \{x_{i,j} \mid i \in \{1,\dots,v\}, j \in \{1,\dots,u+1\}\}$ be an alphabet of size $(u+1)v$. Using $\prod$ as the concatenation operator, let $S = \prod_{i=1}^{v} S_i$ and $S' = \prod_{i=1}^{v} S'_i$ with

$$S_i = \prod_{j=1}^{u+1} x_{i,j} \text{ and}$$
$$S'_i = \prod_{j=1}^{u} x_{i,j+1}\ x_{i,j}.$$

Note that the length of S is indeed $|\Sigma| = \ell + v = uv + v = (u+1)v$ and the length of S' is $2uv = \ell + uv$. Since S_i and $S'_{i'}$ only have common characters for $i = i'$, a common subsequence T of S and S' is of the form $T = \prod_{i=1}^{v} T_i$, where T_i is a common subsequence of S_i and S'_i. Each T_i has length at most u (since S_i is not a subsequence of S'_i, any common subsequence has length at most $|S_i| - 1 = u$). If T has length at least $\ell = uv$, then each T_i has length exactly u. There are precisely $u+1$ such common subsequences for each i (all proper subsequences of S_i are also subsequences of S'_i). Counting all combinations of strings T_i, there are a total of $(u+1)^v$ common subsequences of S and S' of length ℓ, and they are all maximal. So $|\mathrm{MCS}_\ell(S, S')| = (u+1)^v$. ◀

2.4 Extended MCS of *k* Strings

We now present our algorithm computing an extended MCS for any number k of strings, using `xMCS2` as a subroutine, see Algorithm 2. We first give the recurrence relation on MCS on which the algorithm is based.

▶ **Lemma 10.** *Let $\mathcal{S} = \{S_1, \dots, S_k\}$ be a set of at least two strings and let ℓ be an integer. Let $M' = \mathrm{MCS}_\ell(\{S_1, \dots, S_{k-1}\})$, then*

$$\mathrm{MCS}_\ell(\mathcal{S}) \subseteq \bigcup_{S' \in M'} \mathrm{MCS}_\ell(\{S', S_k\}).$$

Proof. Consider some string $S \in \mathrm{MCS}_\ell(\mathcal{S})$. Then S is, in particular, a common subsequence of $\{S_1, \dots, S_{k-1}\}$ of length at least ℓ, and so $S \in \mathrm{CS}_\ell(\{S_1, \dots, S_{k-1}\})$. By definition of MCS, there exists a string S' in $\mathrm{MCS}_\ell(\{S_1, \dots, S_{k-1}\})$ such that $S \preceq S'$

Since S is a subsequence of both S' and S_k, we have that $S \in \mathrm{CS}_\ell(\{S', S_k\})$. To see that S is also in $\mathrm{MCS}_\ell(\{S', S_k\})$, assume that $S'' \in \mathrm{CS}_\ell(\{S', S_k\})$ and $S \preceq S''$. Then S'' is in $\mathrm{CS}_\ell(\mathcal{S})$ as $S' \in \mathrm{CS}_\ell(\{S_1, \dots, S_{k-1}\})$; and since S is maximal in $\mathrm{CS}_\ell(\mathcal{S})$, we have $S = S''$.

Thus, S is in $\mathrm{MCS}_\ell(\{S', S_k\})$ for some $S' \in \mathrm{MCS}_\ell(\{S_1, \dots, S_{k-1}\})$, which gives the desired inclusion. ◀

▶ Remark 11. We note that the containment in Lemma 10 may sometimes be strict, as can be seen in the following example with $\ell = 1$. Take $S_1 = abc$ and $S_2 = acb$. Then $\mathrm{MCS}_\ell(\{S_1, S_2\}) = \{ab, ac\}$. Combining strings ab and ac with $S_3 = aab$ yields respectively $\mathrm{MCS}_\ell(\{S_3, ab\}) = \{ab\}$ and $\mathrm{MCS}_\ell(\{S_3, ac\}) = \{a\}$. However, only ab (and not a) is part of $\mathrm{MCS}_\ell(\{S_1, S_2, S_3\})$. As for `xMCS2`, `xMCSk` outputs these extra strings to avoid a costly filtering step without any gain in the worst case.

▶ **Corollary 12.** *Given $\mathcal{S}$ and ℓ, Algorithm 2 correctly computes an extended MCS of $(\mathcal{S}, \ell)$.*

We now upper-bound the number of strings at any point in the set M of the algorithm. The key point here is that this bound does not depend on k or n. This may seem counter-intuitive, compared to the following upper bound: the algorithm starts with a single string,

Algorithm 2 Compute a bounded-size extended MCS of k strings.

```
1  algorithm xMCSk(ℓ, S_1, ..., S_k)
2    assert ∀i: |S_i| ≥ |S_1|
3    if k = 1 then
4      if |S_1| ≥ ℓ then return {S_1} else return ∅
5    else
6      M' ← xMCSk(ℓ, S_1, ..., S_{k−1})
7      M ← ∅
8      for S' in M' do
9        M ← M ∪ xMCS2(ℓ, S_k, S')
10     return M
```

and each recursive call may replace any string by up to $2^{\delta+\Delta}$ strings (cf. the complexity of xMCS2). There are k recursive calls, so this would give a bound of $2^{k(\delta+\Delta)}$ strings in total. The key argument here is that whenever a string is replaced, new strings are strictly shorter than the former. Since we only allow for at most δ deletions (starting from a minimal length input string), this gives a bound depending on δ and Δ only. Our more precise analysis in Lemma 13 allows us to shrink this quantity from $2^{O(\Delta\delta)}$ to $2^{O(\log(\Delta)\delta)}$.

▶ **Lemma 13.** *Let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be a set of strings with S_1 of minimal length (i. e. $|S_1| = m$), and ℓ be an integer. Then*

$$|\texttt{xMCSk}(\ell, \mathcal{S})| \leq (\Delta + 1)^{\delta}.$$

Proof. We prove the following claim by induction on k: *Let $d \geq \Delta$ and let N_i be the number of length-$(\ell + \delta - i)$ strings in $\texttt{xMCSk}(\ell, \mathcal{S})$. Then*

$$\sum_{i=0}^{\delta} \frac{N_i}{(d+1)^i} \leq 1.$$

The lemma's statement follows easily from this claim for $d = \Delta$:

$$\frac{|\texttt{xMCSk}(\ell, \mathcal{S})|}{(\Delta+1)^{\delta}} = \sum_{i=0}^{\delta} \frac{N_i}{(\Delta+1)^{\delta}} \leq \sum_{i=0}^{\delta} \frac{N_i}{(\Delta+1)^i} \leq 1$$

For the inductive proof of the claim, we start with $k = 1$. Then we have a single string in $\texttt{xMCSk}(\ell, \mathcal{S})$, namely, S_1, so $N_i = 1$ for exactly one value of i and 0 otherwise, and the formula is satisfied.

For $k \geq 2$, we have $M' = \texttt{xMCSk}(\ell, \{S_1, \ldots, S_{k-1}\})$. Consider the for-loop in lines 8–9. We assume that when we iterate with $S' \in M'$, the string S' is immediately removed from M'. At any point of the loop, we write σ for the quantity $\sum_{i=0}^{\delta} \frac{N_i}{(d+1)^i}$ where N_i denotes the number of strings of length $\ell + \delta - i$ in $M' \cup M$. Note that by induction, before the first iteration of the loop, $\sigma \leq 1$ as $\delta(\{S_1, \ldots, S_{k-1}\}, \ell) = \delta$ since $|S_1|$ is minimal, and $d \geq \Delta \geq \Delta(\{S_1, \ldots, S_{k-1}\}, \ell)$.

We show that σ may only decrease after each iteration. Consider the iteration for string S', let $d' = |S'| - \ell$ and $j = \delta - d'$ (since S' is a subsequence of S_1, it has length at most $\ell + \delta$, so $d' \leq \delta$ and $j \geq 0$).

First, removing S' from M' makes N_j decrease by one, so σ decreases by $\frac{1}{(d+1)^j}$. Then, we add strings from $\texttt{xMCS2}(\{S_k, S'\})$ to M. Write D_i for the number of such strings of length $\ell + d' - i$. Note that for each pair (i, j) with $j \leq i \leq \delta$, N_i increases by D_{i-j}. By

Lemma 8, $\sum_{i=0}^{d'} \frac{D_i}{(|S_k|-\ell+1)^i} \leq 1$. Since $d \geq \Delta \geq |S_k| - \ell$, $\sum_{i=0}^{d'} \frac{D_i}{(d+1)^i} \leq 1$. Then σ increases by $\sum_{i=0}^{d'} \frac{D_i}{(d+1)^{j+i}} = \frac{1}{(d+1)^j} \sum_{i=j}^{\delta} \frac{D_{i-j}}{(d+1)^i} \leq \frac{1}{(d+1)^j}$. Overall, σ may not increase between two steps, so at the end of the for-loop, $\sum_{i=0}^{\delta} \frac{N_i}{(d+1)^i} \leq 1$. ◀

We can now conlude with the proof of Theorem 1.

Proof of Theorem 1. Given $\mathcal{S}$ and ℓ, Algorithm 2 computes an extended MCS of $(\mathcal{S}, \ell)$ (Corollary 12) of size at most $(\Delta+1)^\delta$ (Lemma 13). Its running time is bounded by k times the complexity of the for-loop, which requires at most $(\Delta+1)^\delta$ calls to `xMCS2`, each taking time $O(2^{\delta+\Delta}n)$ (Lemma 6). This gives the overall complexity of $O(2^{\delta+\Delta}(\Delta+1)^\delta kn)$. ◀

3 Conclusion

Regarding LCS, we have proposed an FPT algorithm for the parameter Δ, i. e., the maximum number of deletions per input string. We leave open whether the complexity can be improved, e. g. using only parameter δ, i. e., the smallest number of deletions per input strings. In other words, the goal is to find an LCS of size ℓ in a set of strings where *one* input string has size at most $\ell + \delta$ (and other strings might be arbitrarily long). Such an algorithm may not compute and store explicitly each MCS, since the number of maximal common subsequences, even with only two input strings, can grow in $(1 + \frac{\Delta}{\delta})^\delta$. Also, it is open whether any improvement can be obtained when the alphabet size is bounded, or when each character has a bounded number of occurrences in each string.

A longest common subsequence can be interpreted as a string that can be obtained with a minimal number of edits (deletions only) from all input strings. Generalizing this notion to other edits (insertions and substitutions) yields the NP-hard Center String problem[2] [15, 12], which is highly related to the problem of Multiple Sequence Alignment in bioinformatics. In future work, one may try to extend our approach in order to design an FPT algorithm for Center String, parameterized by the maximum distance to the input strings. Allowing for a small number of outliers (input strings that are discarded in order to obtain a better solution [5]) would also yield a useful extension of our algorithm.

Finally, a more practical objective towards algorithm engineering would be to design an efficient data structure to store all maximal common subsequences of any number of strings, thus reducing the memory footprint of our algorithm.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 59–78. IEEE Computer Society, 2015.

2 Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *Seventh International Symposium on String Processing and Information Retrieval, SPIRE 2000*, pages 39–48. IEEE Computer Society, 2000.

3 Magnus Bordewich, Britta Dorn, Simone Linz, and Rolf Niedermeier. Algorithms and complexity in phylogenetics (dagstuhl seminar 19443). *Dagstuhl Reports*, 9(10):134–151, 2019.

4 Mahdi Boroujeni, Masoud Seddighin, and Saeed Seddighin. Improved algorithms for edit distance and LCS: beyond worst case. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 1601–1620. SIAM, 2020.

[2] More specifically, given k strings, find a string that minimizes the maximum edit distance to the k strings.

5 Christina Boucher and Bin Ma. Closest string with outliers. *BMC Bioinformatics*, 12(1):1–7, 2011.

6 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1216–1235. SIAM, 2018.

7 Laurent Bulteau, Falk Hüffner, Christian Komusiewicz, and Rolf Niedermeier. Multivariate algorithmics for NP-hard string problems. *Bulletin of the EATCS*, 114, 2014.

8 Laurent Bulteau and Mathias Weller. Parameterized algorithms in bioinformatics: An overview. *Algorithms*, 12(12):256, 2019.

9 Alessio Conte, Roberto Grossi, Giulia Punzi, and Takeaki Uno. Polynomial-delay enumeration of maximal common subsequences. In *26th International Symposium on String Processing and Information Retrieval, SPIRE 2019*, pages 189–202. Springer, 2019.

10 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

11 MohammadTaghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. Approximating LCS in linear time: Beating the $\sqrt{n}$ barrier. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1181–1200. SIAM, 2019.

12 Gary Hoppenworth, Jason W. Bentley, Daniel Gibney, and Sharma V. Thankachan. The fine-grained complexity of median and center string problems under edit distance. In *28th Annual European Symposium on Algorithms, ESA 2020*, volume 173 of *LIPIcs*, pages 61:1–61:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

13 Robert W. Irving and Campbell Fraser. Two algorithms for the longest common subsequence of three (or more) strings. In *Third Annual Symposium on Combinatorial Pattern Matching, CPM 1992*, volume 644 of *Lecture Notes in Computer Science*, pages 214–229. Springer, 1992.

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

15 François Nicolas and Eric Rivals. Hardness results for the center and median string problems under the weighted and unweighted edit distances. *Journal of Discrete Algorithms*, 3(2):390–415, 2005.

16 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003.

17 Yoshifumi Sakai. Maximal Common Subsequence Algorithms. In *29th Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*, volume 105 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:10. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.