



**HAL**  
open science

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

Laurent Bulteau, Mark Jones, Rolf Niedermeier, Till Tantau

► **To cite this version:**

Laurent Bulteau, Mark Jones, Rolf Niedermeier, Till Tantau. An FPT-Algorithm for Longest Common Subsequence Parameterized by the Number of Deletions. 2021. <hal-03322887>

**HAL Id: hal-03322887**

**<https://hal.science/hal-03322887v1>**

Preprint submitted on 19 Aug 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

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

Laurent Bulteau<sup>1</sup> ✉ 

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 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.

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

**2012 ACM Subject Classification** Theory of computation → Fixed parameter tractability

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

**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.

## 1 Introduction

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

LONGEST COMMON SUBSEQUENCE

**Input:** A set of  $k$  strings  $\mathcal{S} = \{S_1, \dots, S_k\}$ , each of length at most  $n$ , an integer  $\ell$ .

**Parameter:**  $\Delta = n - \ell$ .

---

<sup>1</sup> Corresponding author

41 **Question:** Is there a string  $S$  of length at least  $\ell$  that is a (not necessarily contiguous)  
 42 subsequence of each  $S_i$ ?

43 With standard dynamic programming, LCS can be solved in  $O(n^k)$  time; on the contrary,  
 44 it is known to be W[1]-hard [12] respectively W[2]-hard for parameter  $k$  and it has no  $O(n^{k-\epsilon})$   
 45 algorithm under SETH [1]. Indeed, LCS is *the* string problem receiving most attention  
 46 when the field of parameterized complexity [8] started. Unfortunately, so far parameterized  
 47 complexity analysis beyond trivial algorithmic observations mainly contributed computational  
 48 hardness results. We refer to some surveys [2, 6, 7] for an overview on research results (and  
 49 open questions) for LCS.

50 We remark that the special case of two input strings (that is,  $k = 2$ ) recently attrac-  
 51 ted much attention, particularly motivated by the theoretical challenge of breaking the  
 52 straightforward time bound of  $O(n^2)$  [3, 5, 9]. Notably, Bringmann and Künnemann [5]  
 53 (the corresponding arXiv paper has around 60 pages) also discuss the “maximum number  
 54 of deletions” parameter we focus on here. Indirectly, this parameter already appears in the  
 55 work of Irving and Fraser [10], who provided two algorithms for LCS with three or more  
 56 input strings.

57 Irving and Fraser [10] in their 1992 paper provided an algorithm for LCS running in  
 58 time  $O(kn(n - \ell)^{k-1})$ , implying fixed-parameter tractability with respect to the combined  
 59 parameter  $k$  and  $n - \ell$ , where the latter coincides with our parameter  $\Delta$ . We are not aware of  
 60 any improvement since then and this is also reflected by a corresponding challenge featured  
 61 in a 2014 survey [6, Challenge 9]. Answering positively the research challenge posed there,  
 62 we improve Irving and Fraser’s result to fixed-parameter tractability only with respect to  $\Delta$ .  
 63 More specifically, our algorithm runs in time  $O((\Delta + 1)^{\Delta+1}kn)$ , which means linear time  
 64 when  $\Delta$  is a constant. In addition, we can enumerate *all* longest common subsequences  
 65 within this time. Given that it seems natural to assume that in many applications the  
 66 sought common subsequence is fairly close to every input string (which would mean small  $\Delta$ ),  
 67 this promises to be of also practical relevance. However, the focus of this work is purely  
 68 theoretical. We mention in passing that our result holds for arbitrary alphabet size.

69 Figure 1, at a very high level, for three input strings presents an example for LCS and the  
 70 main idea behind our recursive approach towards achieving our result, the FPT-algorithm  
 71 for parameter  $\Delta$ .

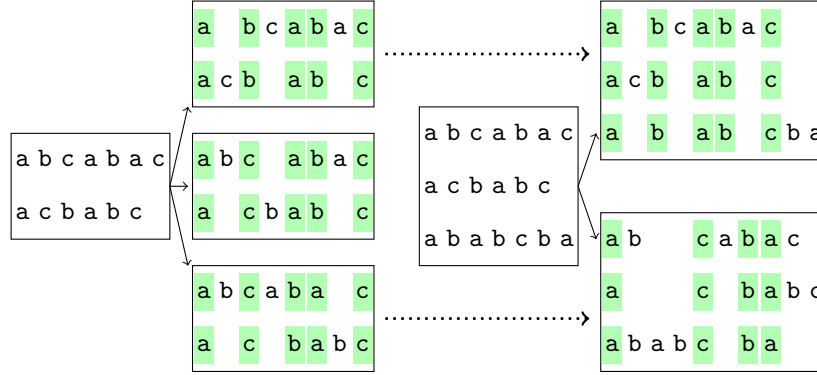
## 72 **2 LCS Algorithm Using Maximal Common Subsequences**

73 In this section, we present a linear-time algorithm for LCS when the number of deletions is a  
 74 constant. Note that it is not incompatible with the quadratic lower bound for this problem,  
 75 since this lower bound only applies to the general case where the number of deletions is  
 76 unbounded. In particular, the  $O(\delta n)$  algorithm by Nakatsu et al. [11] (with  $\delta = \min\{|S_i|\} - \ell$ )  
 77 remains better than our algorithm for the two-string case. Furthermore, it is not clear if a  
 78 smaller (typically constant) alphabet could be exploited in the algorithm or its analysis to  
 79 obtain a better running time.

### 80 **2.1 Definitions**

#### 81 **Strings**

82 The set of strings on an alphabet  $\Sigma$  is denoted  $\Sigma^*$ . The empty string is  $\epsilon$ ,  $|S|$  denotes the  
 83 length of  $S$ . We write  $\cdot$  for the concatenation. We write  $u \cdot T := S$  as a short-hand for “let  $u$



**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.

84 be the first character of  $S$  and  $T$  be the suffix of  $S$  starting from the second character (or  
85  $u = T = \epsilon$  if  $S$  is empty)”).

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

## 88 Longest and Maximal Common Subsequences

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

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

## 96 String parameters

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

## 101 2.2 Main results

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

104 Theorem 1 directly yields an algorithm for LCS, since it suffices to test if an extended  
105 MCS of  $(\mathcal{S}, \ell)$  is non-empty. Note that the algorithm can be adapted for the optimization  
106 formulation of LCS, i.e., when  $\ell$  is not part of the input, with a constant factor in the time  
107 complexity (taking  $\delta$  and  $\Delta$  with respect to  $\ell = |\text{LCS}(\mathcal{S})|$ ). Indeed, apply Theorem 1 for

108 decreasing values of  $\ell$  starting with  $\ell = m$ , until a non-empty set is obtained. Then, the  
 109 resulting set contains the common substrings of  $\mathcal{S}$  of size  $\text{LCS}(\mathcal{S})$  (indeed,  $\text{MCS}_\ell(\mathcal{S}) = \text{CS}_\ell(\mathcal{S})$   
 110 for this value of  $\ell$ ), so it contains all longest common subsequences of  $\mathcal{S}$ . The time complexity  
 111 of the  $i$ -th call,  $1 \leq i \leq \delta$ , is upper-bounded by  $O(2^{i+\Delta}(\Delta + 1)^\delta kn)$ . Using  $\sum_{i=1}^\delta 2^i = O(2^\delta)$ ,  
 112 we get the following corollary.

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

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

### 118 2.3 Extended MCS for two strings

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

122 ► **Lemma 3.** *For any two non-empty strings  $S, S'$  and any  $\ell$ , let  $u \cdot T := S$  and  $u' \cdot T' := S'$ .  
 123 If  $u = u'$ , then  $\text{MCS}_\ell(\{S, S'\}) \subseteq \{u \cdot X \mid X \in \text{MCS}_{\ell-1}(\{T, T'\})\}$ .  
 124 If  $u \neq u'$ , then  $\text{MCS}_\ell(\{S, S'\}) \subseteq \text{MCS}_\ell(\{S, T'\}) \cup \text{MCS}_\ell(\{T, S'\})$ .*

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

126 For the first case ( $u = u'$ ), we show that  $r = u$  and  $X$  is a maximal common subsequence  
 127 of  $\{T, T'\}$  of length at least  $\ell - 1$ . Indeed,  $r = u$ , as otherwise the concatenation  $u \cdot r \cdot X$  would  
 128 also be a common subsequence of  $\{S, S'\}$ , with  $R \prec u \cdot r \cdot X$  (contradicting the maximality of  
 129  $R$ ). Note that  $X$  is a subsequence both of  $T$  and  $T'$ . Moreover  $X$  is maximal, as otherwise, if  
 130  $X \prec X'$  with  $X'$  a common subsequence of  $T, T'$ , then  $u \cdot X'$  would be a common subsequence  
 131 of  $S, S'$  with  $R \prec u \cdot X'$  (again, contradicting the maximality of  $R$ ).

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

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

141 ► **Corollary 4.** *Let  $S, S'$  be two strings and  $\ell$  be an integer. Then  $\text{xMCS2}(\ell, S, S')$  returns an  
 142 extended MCS of  $(\{S, S'\}, \ell)$*

143 ► **Remark 5.** The first inclusion in Lemma 3 (case  $u = u'$ ) is actually an equality, but we  
 144 only need this direction for the algorithm to be correct. The second inclusion however may  
 145 be strict: for example with  $S = ABCD$  and  $S' = DABC$ , string  $R = BC$  is a maximal  
 146 common subsequence of  $T = BCD$  and  $S'$ , but not of  $S, S'$  since  $R \prec ABC$ . Such “extra”  
 147 strings are actually returned by our algorithm, motivating the naming of *extended* MCS  
 148 (although they could be filtered out, see Remark 7).

149 We now focus on the time complexity of Algorithm 1

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

---

```

1  xMCS2( $\ell$ ,  $S$ ,  $S'$ ):
2    if  $\ell > |S|$  or  $\ell > |S'|$ :
3      return {}
4    if  $S$  (resp  $S'$ ) is a substring of  $S'$  (resp  $S$ ):
5      return  $\{S\}$  (resp  $\{S'\}$ )
6     $u \cdot T := S$ 
7     $u' \cdot T' := S'$ 
8    if  $u = u'$ :
9      return  $\{u \cdot X \mid X \text{ in } \text{xMCS2}(\ell-1, T, T')\}$ 
10   else:
11     return  $\text{xMCS2}(\ell, S, T') \cup \text{xMCS2}(\ell, T, S')$ 

```

---

150 ► **Lemma 6.** *Let  $S, S'$  be strings of length respectively  $\ell + \delta$  and  $\ell + \Delta = n$ . Then  $\text{xMCS2}(\ell, S, S')$*   
151 *terminates in time  $O(2^{\delta + \Delta} n)$ .*

152 **Proof.** The time complexity is achieved using a precomputed substring table: for every pair  
153  $i, i'$  with  $1 \leq i \leq |S|$ ,  $1 \leq i' \leq |S'|$ , and  $|i - i'| \leq \Delta$ ,  $\text{sub}[i, i']$  contains **True** if  $S[i \dots |S|]$   
154 is a substring of  $S'[i' \dots |S'|]$ . The entries of this table can be computed in time  $O(n\Delta)$  by  
155 straightforward dynamic programming.

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

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

170 Lemma 6 gives a first bound on the number of strings returned by  $\text{xMCS2}$  (precisely, at  
171 most  $2^{\delta + \Delta}$ ). We know that all strings have lengths between  $\ell$  and  $m$ . However, we will need  
172 an additional information for a more precise analysis of our algorithm on  $k$  instead of two  
173 strings. Namely, the fact that there cannot be many strings of length almost  $m$ . Intuitively,  
174 a long string in the returned set corresponds to a leaf in the search tree with few branching  
175 nodes among its ancestors, which actually helps reducing the size of the search tree. On  
176 the other hand, a short string in the returned set will cause less branchings in our next  
177 algorithm. Thus, the following lemma describes the repartition of the number of maximal  
178 common subsequences of two strings based on their lengths. Note that we would obtain the  
179 same bound if we used the filtering step from Remark 7 (i.e., the same formula applies to  
180 the set  $\text{MCS}_\ell(\{S, S'\})$ ).

181 ► **Lemma 8.** *Let  $\ell, d, d'$  be integers, and  $S, S'$  be two strings of length respectively  $\ell + d$  and*  
182  *$\ell + d'$  (note that  $\{\delta, \Delta\} = \{d, d'\}$ ). Moreover, let  $N_i$  be the number of strings in  $\text{xMCS2}(\ell, S, S')$*   
183 *of length exactly  $\ell + d' - i$ . Then*

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

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

185 If  $\ell > \min\{|S|, |S'|\}$ , then  $\text{xMCS2}(\ell, \{S, S'\})$  is empty, and the inequality is valid. If one  
 186 of  $S, S'$  is a substring of the other, then  $|\text{xMCS2}(\ell, S, S')| = 1$ , so we have  $N_i = 1$  for some  $i$   
 187 and  $N_j = 0$  for  $j \neq i$ . The formula follows in this case as well. Note that this includes the  
 188 cases where  $S$  or  $S'$  are empty.

189 In the remaining cases,  $S$  and  $S'$  are not substrings of each other, so in particular they  
 190 are not empty. Let  $u \cdot T := S, u' \cdot T' := S'$ .

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

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

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

$$\sum_{i=0}^{d'} \frac{N_i^a}{(d+1)^i} = N_0^a + \frac{1}{d+1} \sum_{i=1}^{d'} \frac{N_i^a}{(d+1)^{i-1}} \leq \frac{1}{d+1}.$$

202 Then the induction property on  $\{T, S'\}$  (where  $d$  decreases by 1) gives  $\sum_{i=0}^{d'} \frac{N_i^b}{d^i} \leq 1$ , so

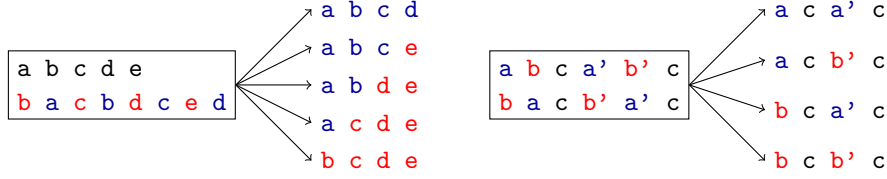
$$\begin{aligned} \sum_{i=0}^{d'} \frac{N_i^b}{(d+1)^i} &= N_0^b + \sum_{i=1}^{d'} \frac{N_i^b}{(d+1)^i} \\ &= \frac{d}{d+1} \sum_{i=1}^{d'} \frac{d^{i-1}}{(d+1)^{i-1}} \frac{N_i^b}{d^i} \\ &\leq \frac{d}{d+1} \sum_{i=1}^{d'} \frac{N_i^b}{d^i} \leq \frac{d}{d+1}. \end{aligned}$$

207 Combining both inequalities yields:

$$\begin{aligned} \sum_{i=0}^{d'} \frac{N_i}{(d+1)^i} &\leq \sum_{i=0}^{d'} \frac{N_i^a}{(d+1)^i} + \sum_{i=0}^{d'} \frac{N_i^b}{(d+1)^i} \\ &\leq \frac{d}{d+1} + \frac{1}{d+1} \\ &= 1 \end{aligned}$$

212

213 Lemma 8 yields an upper bound of  $\Delta^\delta$  on the size of  $\text{MCS}_\ell(S, S')$ . Examples (see Figure 2  
 214 and Proposition 9) indicate that this bound is close to being tight, since there exist instances  
 215 where  $|\text{MCS}_\ell(S, S')| = \left(1 + \frac{\Delta}{\delta}\right)^\delta$ .



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

216 ► **Proposition 9.** *For any integers  $u$  and  $v$ , there exist some  $\ell$  and two strings  $S, S'$  of length*  
 217 *respectively  $\ell + u$  and  $\ell + uv$  such that  $|\text{MCS}_\ell(S, S')| \geq (u + 1)^v = (1 + \frac{\Delta}{\delta})^\delta$ .*

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

$$220 \quad S_i = \prod_{j=1}^{u+1} x_{i,j}$$

$$221 \quad S'_i = \prod_{j=1}^u x_{i,j+1} x_{i,j}.$$

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

## 231 2.4 Extended MCS of $k$ strings

232 We now present our algorithm computing an extended MCS for any number  $k$  of strings,  
 233 using  $\text{xMCS2}$  as a subroutine, see Algorithm 2. We first give the recurrence relation on MCS  
 234 on which the algorithm is based.

► **Lemma 10.** *Let  $S = \{S_1, \dots, S_k\}$  be a set of strings ( $k \geq 2$ ) and  $\ell$  be an integer. Let  $M' = \text{MCS}_\ell(\{S_1, \dots, S_{k-1}\})$ , then*

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

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

238 Since  $S$  is a subsequence of both  $S'$  and  $S_k$ , we have that  $S \in \text{CS}_\ell(\{S', S_k\})$ . To see that  
 239  $S$  is also in  $\text{MCS}_\ell(\{S', S_k\})$ , assume that  $S'' \in \text{CS}_\ell(\{S', S_k\})$  and  $S \preceq S''$ . Then  $S''$  is in  
 240  $\text{CS}_\ell(S)$  (since  $S' \in \text{CS}_\ell(\{S_1, \dots, S_i\})$ ), and since  $S$  is maximal in  $\text{CS}_\ell(S)$ , then  $S = S''$ .

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

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

---

```

1  xMCSk( $\ell, S_1, \dots, S_k$ ):
2    assert( $\forall i, |S_i| \geq |S_1|$ )
3    if  $k = 1$ :
4      if  $|S_1| \geq \ell$ : return  $\{S_1\}$ 
5      else: return  $\{\}$ 
6    else:
7       $M' := \text{xMCSk}(\ell, S_1, \dots, S_{k-1})$ 
8       $M := \{\}$ 
9      for  $S'$  in  $M'$ :
10        $M := M \cup \text{xMCS2}(\ell, S_k, S')$ 
11     return  $M$ 

```

---

243 ▶ **Remark 11.** We note that the containment in Lemma 10 may sometimes be strict, as  
244 can be seen in the following example with  $\ell = 1$ . Take  $S_1 = ABC$  and  $S_2 = ACB$ . Then  
245  $\text{MCS}_\ell(\{S_1, S_2\}) = \{AB, AC\}$ . Combining strings  $AB$  and  $AC$  with  $S_3 = AAB$  yields  
246 respectively  $\text{MCS}_\ell(\{S_3, AB\}) = \{AB\}$  and  $\text{MCS}_\ell(\{S_3, AC\}) = \{A\}$ . However, only  $AB$   
247 (and not  $A$ ) is part of  $\text{MCS}_\ell(\{S_1, S_2, S_3\})$ . As for  $\text{xMCS2}$ ,  $\text{xMCSk}$  outputs these extra strings  
248 to avoid a costly filtering step without any gain in the worst case.

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

250 We now bound the number of strings at any point in the set  $M$  of the algorithm. The key  
251 point here is that this bound does not depend on  $k$  or  $n$ . This may seem counter-intuitive,  
252 compared to the following upper bound: the algorithm starts with a single string, and each  
253 recursive call may replace any string by up to  $2^{\delta+\Delta}$  strings (cf. the complexity of  $\text{xMCS2}$ ).  
254 There are  $k$  recursive calls so this would give a bound of  $2^{k(\delta+\Delta)}$  strings in total. The key  
255 argument here is that whenever a string is replaced, new strings are strictly shorter than  
256 the former. Since we only allow for at most  $\delta$  deletions (starting from a minimal length  
257 input string), this gives a bound depending on  $\delta$  and  $\Delta$  only. Our more precise analysis in  
258 Lemma 13 allows us to shrink this quantity from  $2^{O(\Delta\delta)}$  to  $2^{O(\log(\Delta)\delta)}$ .

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

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

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

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

Note that the lemma statement follows easily from this property for  $d = \Delta$ :

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

261 For  $k = 1$ , we have a single string in  $\text{xMCSk}(\ell, \mathcal{S})$ , namely,  $S_1$ , so  $N_i = 1$  for exactly one  
262 value of  $i$  and 0 otherwise, and the formula is satisfied.

263 For  $k \geq 2$ , we have  $M' = \text{xMCSk}(\ell, \{S_1, \dots, S_{k-1}\})$ . Consider the for-loop in Lines 9–10.  
 264 We assume that when we iterate with  $S' \in M'$ , the string  $S'$  is immediately removed from  $M'$ .  
 265 At any point of the loop, we write  $\sigma$  for the quantity  $\sum_{i=0}^{\delta} \frac{N_i}{(d+1)^i}$  where  $N_i$  denotes the  
 266 number of strings of length  $\ell + \delta - i$  in  $M' \cup M$ . Note that by induction, before the first  
 267 iteration of the loop,  $\sigma \leq 1$  (using the fact that  $\delta(\{S_1, \dots, S_{k-1}\}, \ell) = \delta$  since  $|S_1|$  is minimal,  
 268 and  $d \geq \Delta \geq \Delta(\{S_1, \dots, S_{k-1}\}, \ell)$ ).

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

272 First, removing  $S'$  from  $M'$  makes  $N_j$  decrease by one, so  $\sigma$  decreases by  $\frac{1}{(d+1)^j}$ . Then,  
 273 we insert strings from  $\text{xMCS2}(\{S_k, S'\})$  in  $M$ . Write  $D_i$  for the number of such strings of  
 274 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  
 275 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  $\sigma$  increases  
 276 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,  $\sigma$  may not increase between two  
 277 steps, so at the end of the for loop,  $\sum_{i=0}^{\delta} \frac{N_i}{(d+1)^i} \leq 1$ . ◀

278 We can now conclude with the proof of Theorem 1.

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

### 283 3 Conclusion

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

293 A longest common subsequence can be seen as a string that can be obtained with a  
 294 minimal number of edits (deletions only) from all input strings. Generalizing this notion to  
 295 other edits (insertions and substitutions) yields the Center String problem, which is highly  
 296 related to the problem of Multiple Sequence Alignment in bioinformatics. In future work,  
 297 we aim at extending our approach in order to design an FPT algorithm for Center String,  
 298 parameterized by the maximum distance to input strings. Allowing for a small number of  
 299 outliers (input strings that are discarded in order to obtain a better solution [4]) would also  
 300 be a useful extension of our algorithm.

301 Finally, a more practical objective towards algorithm engineering would be to design an  
 302 efficient data structure to store all maximal common subsequences of any number of strings,  
 303 in order to reduce the memory footprint of our algorithm.

304 ——— **References** ———

- 305 **1** Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for  
306 LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations  
307 of Computer Science, FOCS 2015*, pages 59–78. IEEE Computer Society, 2015.
- 308 **2** Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence  
309 algorithms. In *Seventh International Symposium on String Processing and Information  
310 Retrieval, SPIRE 2000*, pages 39–48. IEEE Computer Society, 2000.
- 311 **3** Mahdi Boroujeni, Masoud Seddighin, and Saeed Seddighin. Improved algorithms for edit  
312 distance and LCS: beyond worst case. In *Proceedings of the 2020 ACM-SIAM Symposium on  
313 Discrete Algorithms, SODA 2020*, pages 1601–1620. SIAM, 2020.
- 314 **4** Christina Boucher and Bin Ma. Closest string with outliers. *BMC Bioinformatics*, 12(1):1–7,  
315 2011.
- 316 **5** Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest  
317 common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on  
318 Discrete Algorithms, SODA 2018*, pages 1216–1235. SIAM, 2018.
- 319 **6** Laurent Bulteau, Falk Hüffner, Christian Komusiewicz, and Rolf Niedermeier. Multivariate  
320 algorithmics for NP-hard string problems. *Bulletin of the EATCS*, 114, 2014.
- 321 **7** Laurent Bulteau and Mathias Weller. Parameterized algorithms in bioinformatics: An overview.  
322 *Algorithms*, 12(12):256, 2019.
- 323 **8** Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.
- 324 **9** MohammadTaghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. Approx-  
325 imating LCS in linear time: Beating the  $\sqrt{n}$  barrier. In *Proceedings of the Thirtieth Annual  
326 ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1181–1200. SIAM, 2019.
- 327 **10** Robert W. Irving and Campbell Fraser. Two algorithms for the longest common subsequence  
328 of three (or more) strings. In *Combinatorial Pattern Matching, Third Annual Symposium,  
329 CPM 1992*, volume 644 of *Lecture Notes in Computer Science*, pages 214–229. Springer, 1992.
- 330 **11** Narao Nakatsu, Yahiko Kambayashi, and Shuzo Yajima. A longest common subsequence  
331 algorithm suitable for similar text strings. *Acta Informatica*, 18:171–179, 1982.
- 332 **12** Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common  
333 supersequence and longest common subsequence problems. *Journal of Computer and System  
334 Sciences*, 67(4):757–771, 2003.