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

L. Bulteau[1], M. Jones[2], R. Niedermeier[3], T. Tantau[4]

[1] LIGM, CNRS, Université Gustave Eiffel, France
[2] Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands
[3] Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
[4] Institute of Theoretical Computer Science, University of Lübeck, Germany

CPM 2022-06-29

This work was initiated during Dagstuhl Seminar 19443, Algorithms and Complexity in Phylogenetics in October 2019.

In Rolf's Memory

In Rolf's Memory

Laurent Bulteau, Mark Jones, Rolf Niedermeier, Till Tantau:
An FPT-Algorithm for Longest Common Subsequence Parameterized by the Maximum Number of Deletions. CPM 2022: 6:1-6:11

Tomohiro Koana, Vincent Froese, Rolf Niedermeier:
Parameterized Algorithms for Matrix Completion with Radius Constraints. CPM 2020: 20:1-20:14

Nathan Schaar, Vincent Froese, Rolf Niedermeier:
Faster Binary Mean Computation Under Dynamic Time Warping. CPM 2020: 28:1-28:13

Sharon Bruckner, Falk Hüffner, Christian Komusiewicz, Rolf Niedermeier, Sven Thiel, Johannes Uhlmann:
Partitioning into Colorful Components by Minimum Edge Deletions. CPM 2012: 56-69

Rudolf Fleischer, Jiong Guo, Rolf Niedermeier, Johannes Uhlmann, Yihui Wang, Mathias Weller, Xi Wu:
Extended Islands of Tractability for Parsimony Haplotyping. CPM 2010: 214-226

Christian Komusiewicz, Rolf Niedermeier, Johannes Uhlmann:
Deconstructing Intractability: A Case Study for Interval Constrained Coloring. CPM 2009: 207-220

Nadja Betzler, Michael R. Fellows, Christian Komusiewicz, Rolf Niedermeier:
Parameterized Algorithms and Hardness Results for Some Graph Motif Problems. CPM 2008: 31-43

Jochen Alber, Jens Gramm, Jiong Guo, Rolf Niedermeier:
Towards Optimally Solving the LONGEST COMMON SUBSEQUENCE Problem for Sequences with Nested Arc Annotations in Linear Time. CPM 2002: 99-114

Jens Gramm, Rolf Niedermeier:
Minimum Quartet Inconsistency Is Fixed Parameter Tractable. CPM 2001: 241-256

Longest Common Subsequence

LCS

- Given strings $S_1, \ldots, S_k$, integer ℓ
- Find S^* of length ℓ, S^* subsequence of each S_i

Longest Common Subsequence

LCS

- Given strings $S_1, \ldots, S_k$, integer ℓ
- Find S^* of length ℓ, S^* subsequence of each S_i

a b c a b a c
a c b a b c
a b a b c b a

$\ell = 5$

Longest Common Subsequence

LCS

- Given strings $S_1, \ldots, S_k$, integer ℓ
- Find S^* of length ℓ, S^* subsequence of each S_i

a	b	c	a	b	a	c
a	c	b	a	b	c	
a	b	a	b	c	b	a

$\ell = 5$

$S^* = \texttt{ababc}$

Longest Common Subsequence

LCS

- Given strings $S_1, \ldots, S_k$, integer ℓ
- Find S^* of length ℓ, S^* subsequence of each S_i

a b c a b a c

a c b a b c

a b a b c b a

$\ell = 5$

$S^* = \texttt{ababc}$

a b c a b a c

a c b a b c

a b a b c b a

Longest Common Subsequence

LCS

- Given strings $S_1, \ldots, S_k$, integer ℓ
- Find S^* of length ℓ, S^* subsequence of each S_i

Previous work (in a *tiny* nutshell)

- For $k = 2$:
- For larger k:

Longest Common Subsequence

LCS

- Given strings $S_1, \ldots, S_k$, integer ℓ
- Find S^* of length ℓ, S^* subsequence of each S_i

Previous work (in a *tiny* nutshell)

- For $k = 2$: very well studied.
  - Solvable in $O(n^2)$ (dynamic programming textbook example),
  - not in $O(n^{2-\epsilon})$ (under SETH, [Abboud et al. '15]),
  - many possible parameterizations (cf [Bringmann et al.' 18])
- For larger k:

Longest Common Subsequence

LCS

- Given strings $S_1, \ldots, S_k$, integer ℓ
- Find S^* of length ℓ, S^* subsequence of each S_i

Previous work (in a *tiny* nutshell)

- For $k = 2$: very well studied.
  - Solvable in $O(n^2)$ (dynamic programming textbook example),
  - not in $O(n^{2-\epsilon})$ (under SETH, [Abboud et al. '15]),
  - many possible parameterizations (cf [Bringmann et al.' 18])
- For larger k:
  - NP-hard [Maier, '78]
  - Aim for FPT algorithms...

Parameters

a b c a b a c
a c b a b c
a b a b c b a

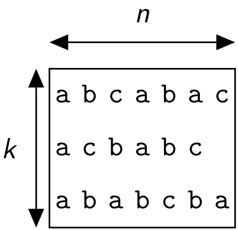
Parameters

n max input length FPT[1]

```
a b c a b a c
a c b a b c
a b a b c b a
```

[1] By exhaustive enumeration

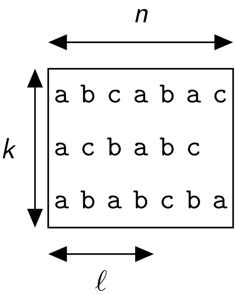
Parameters



max input length	FPT[1]
number of strings	XP[1], W[1]-hard[2]

[1] By exhaustive enumeration +DP
[2] Even for binary alphabets [Pietrzak '03]

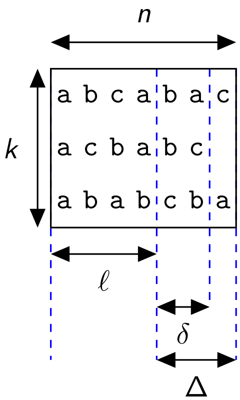
Parameters



max input length	FPT[1]
number of strings	XP[1], W[1]-hard[2]
target length	XP[1], W[2]-hard[3]

[1] By exhaustive enumeration +DP
[2] Even for binary alphabets [Pietrzak '03]
[3] W[1]-hard for $\ell + k$ [Bodlaender et al. '95], FPT for ℓ + alphabet size

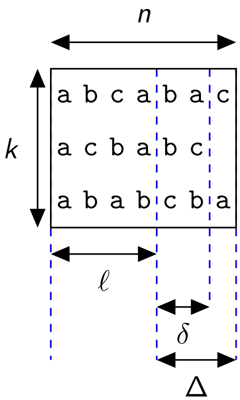
Parameters



max input length	FPT[1]
number of strings	XP[1], W[1]-hard[2]
target length	XP[1], W[2]-hard[3]
min number of deletions	XP[1] (FPT open)
max number of deletions	$O(\Delta^{k-1}nk)$[4]

[1] By exhaustive enumeration +DP
[2] Even for binary alphabets [Pietrzak '03]
[3] W[1]-hard for $\ell + k$ [Bodlaender et al. '95], FPT for ℓ + alphabet size
[4] [Irving and Fraser, CPM '92]

Parameters



max input length	FPT[1]
number of strings	XP[1], W[1]-hard[2]
target length	XP[1], W[2]-hard[3]
min number of deletions	XP[1] (FPT open)
max number of deletions	$O(\Delta^{k-1}nk)$[4] FPT: this talk

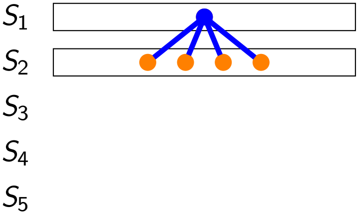
[1] By exhaustive enumeration +DP
[2] Even for binary alphabets [Pietrzak '03]
[3] W[1]-hard for $\ell + k$ [Bodlaender et al. '95], FPT for ℓ + alphabet size
[4] [Irving and Fraser, CPM '92]

Algorithm outline

- Read input string by string
- Maintain a set of candidates
- Pick the longest candidate in the final set

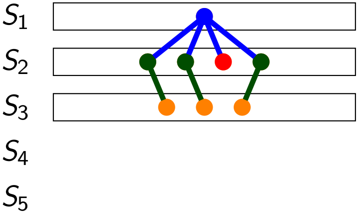
Algorithm outline

- Read input string by string
- Maintain a set of candidates
- Pick the longest candidate in the final set



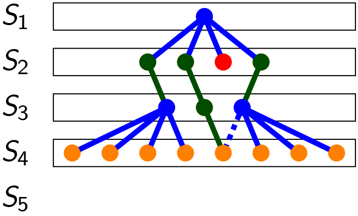
Algorithm outline

- Read input string by string
- Maintain a set of candidates
- Pick the longest candidate in the final set



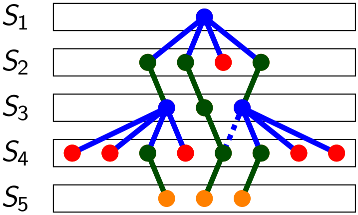
Algorithm outline

- Read input string by string
- Maintain a set of candidates
- Pick the longest candidate in the final set



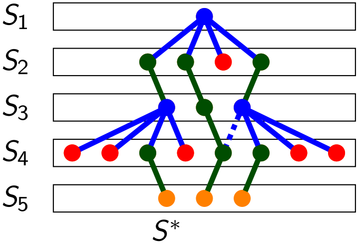
Algorithm outline

- Read input string by string
- Maintain a set of candidates
- Pick the longest candidate in the final set



Algorithm outline

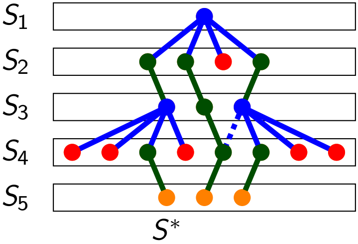
- Read input string by string
- Maintain a set of candidates
- Pick the longest candidate in the final set



Algorithm outline

Rough complexity analysis:

- Branching degree $\leq 4^{\Delta}$
- At most Δ branching candidates along each branch
- Everything else is linear in kn

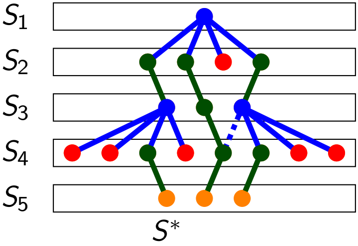


Algorithm outline

Rough complexity analysis:

- Branching degree $\leq 4^{\Delta}$
- At most Δ branching candidates along each branch
- Everything else is linear in kn

$\Rightarrow$ Complexity in $O(4^{\Delta^2} kn)$



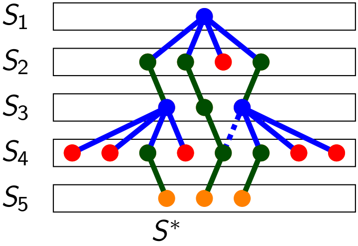
Algorithm outline

Rough complexity analysis:

- Branching degree $\leq 4^{\Delta}$
- At most Δ branching candidates along each branch
- Everything else is linear in kn

$\Rightarrow$ Complexity in $O(4^{\Delta^2} kn)$

(Improved to $O(2^{\delta+\Delta}(\Delta+1)^{\delta} kn)$ with a precise analysis)

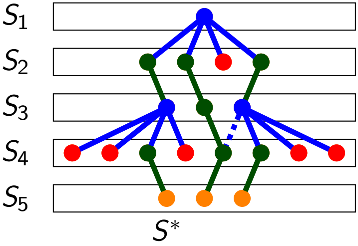


Algorithm outline

Maximal Common Subsequences (MCS)

$T \in \text{MCS}(S_1, \ldots, S_i)$ if T is a subsequence of each S_i and no character can be added to T

- Loop invariant: After reading S_i, candidates contain all strings in $\text{MCS}(S_1, \ldots, S_i)$ of length at least ℓ
- LCS is the longest string in $\text{MCS}(S_1, \ldots, S_k)$



Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we have:
$\mathrm{MCS}(S, S') \subseteq \ldots$

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we have:
$\text{MCS}(S, S') \subseteq \ldots$

- $\{S\}$ if S subsequence of S'

S	S'
`b a b`	`a b c a b`

$\text{MCS}(S, S')$

`b a b`

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we have:
$\mathrm{MCS}(S, S') \subseteq \ldots$

- $\{S\}$ if S subsequence of S'
- $\{S'\}$ if S' subsequence of S

S	S'
`a b c a b`	`b a b`

$\mathrm{MCS}(S, S')$

`b a b`

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we have:
$\mathrm{MCS}(S, S') \subseteq \ldots$

- $\{S\}$ if S subsequence of S'
- $\{S'\}$ if S' subsequence of S
- $u \cdot \mathrm{MCS}(T, T')$ if $u = u'$

S	S'
a b a b c d	a c b a d b

$\mathrm{MCS}(S, S')$

a $\boxed{\text{b a b}}$
a $\boxed{\text{b a d}}$
a $\boxed{\text{c d}}$

$\mathrm{MCS}(\{\mathtt{babcd}, \mathtt{cbadb}\})$

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we have:
$\mathsf{MCS}(S, S') \subseteq \ldots$

- $\{S\}$ if S subsequence of S'
- $\{S'\}$ if S' subsequence of S
- $u \cdot \mathsf{MCS}(T, T')$ if $u = u'$
- $\mathsf{MCS}(S, T') \cup \mathsf{MCS}(S', T)$ if $u \neq u'$

S	S'
c b a b c d	a b c a b

$\mathsf{MCS}(S, S')$

~~b c~~

c a b

b a b

a b c

$\mathsf{MCS}(\{S, \mathtt{bcab}\})$

$\mathsf{MCS}(\{\mathtt{babcd}, S'\})$

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we define:
$\text{xMCS}(S, S') := \ldots$

- $\{S\}$ if S subsequence of S'
- $\{S'\}$ if S' subsequence of S
- $u \cdot \text{xMCS}(T, T')$ if $u = u'$
- $\text{xMCS}(S, T') \cup \text{xMCS}(S', T)$ if $u \neq u'$

S S'

`c b a b c d`   `a b c a b`

$\text{xMCS}(S, S')$

`c a b`

`b b b`

`b a b b c`

`a b c`

$\text{xMCS}(\{S, \texttt{bcab}\})$

$\text{xMCS}(\{\texttt{babcd}, S'\})$

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we define:
$\text{xMCS}_\ell(S, S') := \ldots$

- $\emptyset$ if $|S| < \ell$ or $|S'| < \ell$
- $\{S\}$ if S subsequence of S'
- $\{S'\}$ if S' subsequence of S
- $u \cdot \text{xMCS}_{\ell-1}(T, T')$ if $u = u'$
- $\text{xMCS}_\ell(S, T') \cup \text{xMCS}_\ell(S', T)$ if $u \neq u'$

S	S'
c b a b c d	a b c a b

$\text{xMCS}_3(S, S')$

c a b

b a b

a b c

$\text{xMCS}_3(\{S, \texttt{bcab}\})$

$\text{xMCS}_3(\{\texttt{babcd}, S'\})$

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we define:
$\mathrm{xMCS}_\ell(S, S') := \ldots$

- $\emptyset$ if $|S| < \ell$ or $|S'| < \ell$
- $\{S\}$ if S subsequence of S'
- $\{S'\}$ if S' subsequence of S
- $u \cdot \mathrm{xMCS}_{\ell-1}(T, T')$ if $u = u'$
- $\mathrm{xMCS}_\ell(S, T') \cup \mathrm{xMCS}_\ell(S', T)$ if $u \neq u'$

Correctness

$\mathrm{xMCS}_\ell(S, S')$ contains all MCS of S, S' of length at least ℓ

Complexity

- $|\mathrm{xMCS}_\ell(S, S')| \leq 2^{|S|+|T|-2\ell} \leq 4^{\Delta}$

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we define:

$\mathrm{xMCS}_\ell(S, S') := \ldots$

- $\emptyset$ if $|S| < \ell$ or $|S'| < \ell$
- $\{S\}$ if S subsequence of S'
- $\{S'\}$ if S' subsequence of S
- $u \cdot \mathrm{xMCS}_{\ell-1}(T, T')$ if $u = u'$
- $\mathrm{xMCS}_\ell(S, T') \cup \mathrm{xMCS}_\ell(S', T)$ if $u \neq u'$

Correctness

$\mathrm{xMCS}_\ell(S, S')$ contains all MCS of S, S' of length at least ℓ

Complexity

- $|\mathrm{xMCS}_\ell(S, S')| \leq 2^{|S|+|T|-2\ell} \leq 4^\Delta$
- Can be computed in $O(|\mathrm{xMCS}_\ell(S, S')| \cdot n)$
  using a precomputed table ($O(\Delta n)$ entries):
  Is $S[i, \ldots n]$ a subsequence of $S'[j, \ldots n]$ for $|i - j| \leq \Delta$?

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we define:
$\text{xMCS}_\ell(S, S') := \ldots$

- $\emptyset$ if $|S| < \ell$ or $|S'| < \ell$
- $\{S\}$ if S subsequence of S'
- $\{S'\}$ if S' subsequence of S
- $u \cdot \text{xMCS}_{\ell-1}(T, T')$ if $u = u'$
- $\text{xMCS}_\ell(S, T') \cup \text{xMCS}_\ell(S', T)$ if $u \neq u'$

Tree-Bounding Arguments

For any $X \in \text{xMCS}_\ell(S, S')$:

- $|X| \leq \min(|S|, |S'|)$
- $|X| < \min(|S|, |S'|)$ if $|\text{xMCS}_\ell(S, S')| > 1$

Maximal Common Subsequences of two strings

For two strings $S = u \cdot T$ and $S' = u' \cdot T'$, we define:
$\text{xMCS}_\ell(S, S') := \ldots$

- $\emptyset$ if $|S| < \ell$ or $|S'| < \ell$
- $\{S\}$ if S subsequence of S'
- $\{S'\}$ if S' subsequence of S
- $u \cdot \text{xMCS}_{\ell-1}(T, T')$ if $u = u'$
- $\text{xMCS}_\ell(S, T') \cup \text{xMCS}_\ell(S', T)$ if $u \neq u'$

Tree-Bounding Arguments (precise formulation)

Let $d = |S| - \ell$, $d' = |S'| - \ell$,
and N_i be the number of strings in $\text{xMCS}_\ell(S, S')$ of length $|S'| - i$.

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

(i.e. starting with a single string of length $|S'|$, a string of length m can be replaced by up to $d+1$ strings of length $m-1$)

Maximal Common Subsequence of k strings

Recurrence property

$$\mathrm{MCS}(S_1, \dots S_k) \subseteq \bigcup_{X \in \mathrm{MCS}(S_1, \dots S_{k-1})} \mathrm{MCS}(S_k, X)$$

Maximal Common Subsequence of k strings

Recurrence property

$$\mathrm{MCS}(S_1, \dots S_k) \subseteq \bigcup_{X \in \mathrm{MCS}(S_1,\dots S_{k-1})} \mathrm{MCS}(S_k, X)$$

Algorithm

$$\mathrm{xMCS}_\ell(S_1, \dots S_k) := \bigcup_{X \in \mathrm{xMCS}_\ell(S_1,\dots S_{k-1})} \mathrm{xMCS}_\ell(S_k, X)$$

Maximal Common Subsequence of k strings

Recurrence property

$$\mathrm{MCS}(S_1, \ldots S_k) \subseteq \bigcup_{X \in \mathrm{MCS}(S_1, \ldots S_{k-1})} \mathrm{MCS}(S_k, X)$$

Algorithm

$$\mathrm{xMCS}_\ell(S_1, \ldots S_k) := \bigcup_{X \in \mathrm{xMCS}_\ell(S_1, \ldots S_{k-1})} \mathrm{xMCS}_\ell(S_k, X)$$

Correctness

$\mathrm{xMCS}_\ell(S_1, \ldots S_k)$ contains all MCS of $(S_1, \ldots, S_k)$ of length at least ℓ

Full Example

Input Strings	Length	Current candidates	Weight
$S_1 =$ `atcatac`	$\|S_1\|=7$	`atcatac`	$(\Delta+1)^{-0}$
$S_2 =$ `atcatca`	6		$(\Delta+1)^{-1}$
$S_3 =$ `actatca`	5		$(\Delta+1)^{-2}$
$S_4 =$ `atatcta`	$\ell=4$ $(\Delta=3)$		$(\Delta+1)^{-3}$
$S_5 =$ `cattacc`			
$S_6 =$ `acatcta`			

Total weight: 1.0

Full Example

Input Strings	Length	Current candidates	Weight
$S_1 = \texttt{atcatac}$	$\lvert S_1\rvert = 7$	atcatac	$(\Delta+1)^{-0}$
$S_2 = \texttt{atcatca}$	6	atcata atcatc	$(\Delta+1)^{-1}$
$S_3 = \texttt{actatca}$	5		$(\Delta+1)^{-2}$
$S_4 = \texttt{atatcta}$	$\ell = 4$ ($\Delta = 3$)		$(\Delta+1)^{-3}$
$S_5 = \texttt{cattacc}$			
$S_6 = \texttt{acatcta}$			

Total weight: 0.5

Full Example

Input Strings
$S_1 = \texttt{atcatac}$
$S_2 = \texttt{atcatca}$
$S_3 = \texttt{actatca}$
$S_4 = \texttt{atatcta}$
$S_5 = \texttt{cattacc}$
$S_6 = \texttt{acatcta}$

Length	Current candidates	Weight
$\|S_1\|=7$	atcatac	$(\Delta+1)^{-0}$
6	atcata atcatc	$(\Delta+1)^{-1}$
5	atatc acata atata acatc	$(\Delta+1)^{-2}$
$\ell=4$ $(\Delta=3)$	aata attc aatc atca atta	$(\Delta+1)^{-3}$

Total weight: 0.32812

Full Example

Input Strings
S_1 = atcatac
S_2 = atcatca
S_3 = actatca
S_4 = atatcta
S_5 = cattacc
S_6 = acatcta

Length	Current candidates	Weight
$\|S_1\|=7$	atcatac	$(\Delta+1)^{-0}$
6	atcata atcatc	$(\Delta+1)^{-1}$
5	atatc acata atata acatc	$(\Delta+1)^{-2}$
$\ell=4$ ($\Delta=3$)	aata attc aatc atca atta acta	$(\Delta+1)^{-3}$

Total weight: 0.21875

Full Example

Input Strings	Length	Current candidates	Weight
S_1 = atcatac	$\lvert S_1 \rvert = 7$	atcatac	$(\Delta+1)^{-0}$
S_2 = atcatca	6	atcata atcatc	$(\Delta+1)^{-1}$
S_3 = actatca	5	atatc acata atata acatc	$(\Delta+1)^{-2}$
S_4 = atatcta	$\ell = 4$ ($\Delta = 3$)	aata attc aatc atca atta acta atac	$(\Delta+1)^{-3}$
S_5 = cattacc			
S_6 = acatcta			

Total weight: 0.04687

Full Example

Input Strings

$S_1 = $ `atcatac`

$S_2 = $ `atcatca`

$S_3 = $ `actatca`

$S_4 = $ `atatcta`

$S_5 = $ `cattacc`

$S_6 = $ `acatcta`

Length	Current candidates	Weight
$\lvert S_1 \rvert = 7$	atcatac	$(\Delta+1)^{-0}$
6	atcata atcatc	$(\Delta+1)^{-1}$
5	atatc acata atata acatc	$(\Delta+1)^{-2}$
$\ell = 4$ $(\Delta = 3)$	aata attc aatc atca atta acta atac	$(\Delta+1)^{-3}$

Total weight: 0.01562

Full Example

Input Strings	Length	Current candidates	Weight
$S_1 = \texttt{atcatac}$	$\lvert S_1\rvert=7$	atcatac	$(\Delta+1)^{-0}$
$S_2 = \texttt{atcatca}$	6	atcata atcatc	$(\Delta+1)^{-1}$
$S_3 = \texttt{actatca}$	5	atatc acata atata acatc	$(\Delta+1)^{-2}$
$S_4 = \texttt{atatcta}$	$\ell=4$ ($\Delta=3$)	aata attc aatc atca atta acta atac	$(\Delta+1)^{-3}$
$S_5 = \texttt{cattacc}$			
$S_6 = \texttt{acatcta}$			

Total weight: 0.01562

xMCS size

The total weight is always ≤ 1, so $|\mathrm{xMCS}| \leq (\Delta + 1)^{\Delta}$

Overall Result

Main theorem

All maximal common subsequences (including the LCS) of k strings can be computed in time

$$O\big((4(\Delta+1))^{\Delta} kn\big)$$

Overall Result

Main theorem

All maximal common subsequences (including the LCS) of k strings can be computed in time

$$O\big((4(\Delta+1))^{\Delta}kn\big)$$

With heterogeneous string length

The shortest input string has length $\ell+\delta$: the running time becomes

$$O\big(2^{\delta+\Delta}(\Delta+1)^{\delta}kn\big)$$

Outlooks

#1: Factorize the MCS

- Good memory representation for an exponential number of very similar strings?

Outlooks

#1: Factorize the MCS

- Good memory representation for an exponential number of very similar strings?

$\Rightarrow$ Option: vertex-labelled automaton

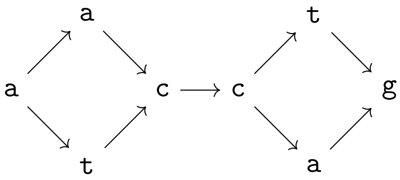
$$
\begin{tikzcd}
 & \mathtt{a} \arrow[dr] & & & \mathtt{t} \arrow[dr] & \\
\mathtt{a} \arrow[ur] \arrow[dr] & & \mathtt{c} \arrow[r] & \mathtt{c} \arrow[ur] \arrow[dr] & & \mathtt{g} \\
 & \mathtt{t} \arrow[ur] & & & \mathtt{a} \arrow[ur] &
\end{tikzcd}
$$

Outlooks

#1: Factorize the MCS

- Good memory representation for an exponential number of very similar strings?

⇒ Option: vertex-labelled automaton



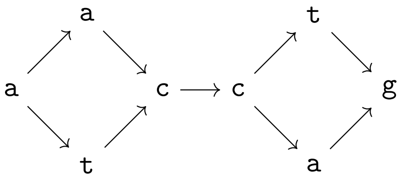
- Use the automaton to factorize MCS computations ?
  - Need to filter out short strings (easy enough)
  - Filter out non-maximal strings (hard)

Outlooks

#1: Factorize the MCS

- Good memory representation for an exponential number of very similar strings?

⇒ Option: vertex-labelled automaton



- Use the automaton to factorize MCS computations ?
  - Need to filter out short strings (easy enough)
  - Filter out non-maximal strings (hard)
- Ideas from MCS enumeration for two strings: [Sakai, CPM'18], [Conte et al, SPIRE'19]

Outlooks

#2: Parameter δ

Is LCS FPT for parameter δ?

- At least one "short" string ($\ell + \delta$), all others may be arbitrarily long
- MCS may not be computed explicitly in this case, as it can be arbitrarily large (examples with $k = 2$ and $|\mathrm{MCS}| \geq (\Delta/\delta)^{\delta}$)

Last-minute update – W[1]-hardness draft for parameter δ from Independent Set

Outlooks

#3: Extend to Edit Distance: Center String

LCS

- Given strings $S_1, \ldots, S_k$ and Δ,
- Find S^* such that S^* is $\leq \Delta$ deletions away from each S_i

Outlooks

#3: Extend to Edit Distance: Center String

Center String – FPT is open

- Given strings $S_1, \ldots, S_k$ and Δ,
- Find S^* such that S^* is $\leq \Delta$ edits away from each S_i

Outlooks

#3: Extend to Edit Distance: Center String

Center String – Tentative approach

Represent all strings at edit distance Δ from each S_i as a union of balls around few "centroids".

$[\mathtt{ABCD}_3 \cap \mathtt{ACBD}_3]$

Outlooks

#3: Extend to Edit Distance: Center String

Center String – Tentative approach

Represent all strings at edit distance Δ from each S_i as a union of balls around few "centroids".

$$[\mathtt{ABCD}_3 \cap \mathtt{ACBD}_3]$$
$$= \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{CBD}_3]$$

Outlooks

#3: Extend to Edit Distance: Center String

Center String – Tentative approach

Represent all strings at edit distance Δ from each S_i as a union of balls around few "centroids".

$$
\begin{aligned}
&[\mathtt{ABCD}_3 \cap \mathtt{ACBD}_3]\\
&\quad = \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{CBD}_3]\\
&\qquad\quad = \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{BD}_2]\\
&\qquad\quad \cup\ \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{BCBD}_2]\\
&\qquad\quad \cup\ \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{BBD}_2]\\
&\qquad\quad \cup\ \mathtt{A\#} \cdot [\mathtt{CD}_2 \cap \mathtt{BD}_2]\\
&\qquad\quad \cup\ \ldots
\end{aligned}
$$

Outlooks

#3: Extend to Edit Distance: Center String

Center String – Tentative approach

Represent all strings at edit distance Δ from each S_i as a union of balls around few "centroids".

$$
\begin{aligned}
&[\mathtt{ABCD}_3 \cap \mathtt{ACBD}_3] \\
&\quad = \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{CBD}_3] \\
&\qquad = \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{BD}_2] \\
&\qquad\quad = \mathtt{ABD}_2 \\
&\qquad \cup\ \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{BCBD}_2] \\
&\qquad \cup\ \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{BBD}_2] \\
&\qquad \cup\ \mathtt{A\#} \cdot [\mathtt{CD}_2 \cap \mathtt{BD}_2] \\
&\qquad \cup\ \ldots
\end{aligned}
$$

Outlooks

#3: Extend to Edit Distance: Center String

Center String – Tentative approach

Represent all strings at edit distance Δ from each S_i as a union of balls around few "centroids".

$$
\begin{aligned}
&[\mathtt{ABCD}_3 \cap \mathtt{ACBD}_3] \\
&\quad = \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{CBD}_3] \\
&\qquad = \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{BD}_2] \\
&\qquad\quad = \mathtt{ABD}_2 \\
&\qquad \cup\ \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{BCBD}_2] \\
&\qquad \cup\ \mathtt{A} \cdot [\mathtt{BCD}_3 \cap \mathtt{BBD}_2] \\
&\qquad \cup\ \mathtt{A\#} \cdot [\mathtt{CD}_2 \cap \mathtt{BD}_2] \\
&\qquad \cup\ \ldots
\end{aligned}
$$

X_i	Δ_i
ABD	2
ABBD	2
ABCBD	2
ACD	2
ACCD	2
ACBCD	2
A##D	1

S at distance ≤ 3 from both ABCD, ACBD

$\Leftrightarrow$

$\exists i$, S at distance $\leq \Delta_i$ from X_i

Outlooks

#3: Extend to Edit Distance: Center String

Center String – Tentative approach

Represent all strings at edit distance Δ from each S_i as a union of balls around few "centroids".

Problem: highly repetitive strings yield too many centroids

Example with $\Delta = 1$:
$S_1 =$BBBBCD
$S_2 =$BBBCD
$S_3 =$BBABCE

Solution: BBABCE

All insertions of A within BBB must be possible after reading S_2

Thank you for your attention !

Thank you for your attention !

Questions, remarks, nice data structures for MCS and algorithms for Center String are welcome