Longest Common Subsequence with Gap Constraints

Duncan Adamson Maria Kosche Tore Koß Florin Manea Stefan Siemer

Göttingen University, Germany

Theorietag 2023
Kaiserslautern

Introduction

Subsequence

w	b	a	c	b	a	a	b	a	d	a

Subsequence

ababa

d

bbb

ba

bc

w | b | a | c | b | a | a | b | a | d | a

baab

a

bb

c

aaaaa

b

Subsequence

ababa

d

abc

bbb

ba

bc

w	b	a	c	b	a	a	b	a	d	a

baab

a

bb

c

aaaaa

b

Subsequence

ababa

d

~~abc~~

bbb

ba

bc

w	b	a	c	b	a	a	b	a	d	a

baab

a

bb

c

aaaaa

b

Subsequences in TCS

Subsequences are a central concept in many different areas of TCS:

- Formal languages and logics (piecewise testable languages, subword order and downward closures).
- Combinatorics on words.
- Chemformatics.
- Modelling concurrency.
- Database theory (event stream processing).
- Algorithms (longest common subsequence, shortest common supersequence).

Subsequence embeddings

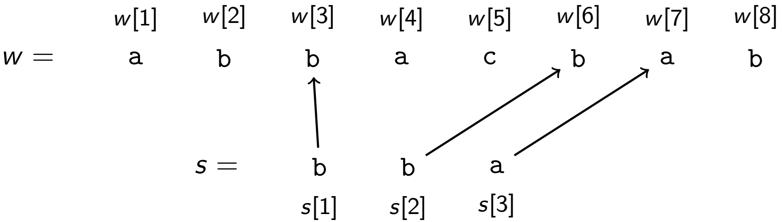
Σ is a finite alphabet, e. g., $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$.

	$w[1]$	$w[2]$	$w[3]$	$w[4]$	$w[5]$	$w[6]$	$w[7]$	$w[8]$
$w =$	a	b	b	a	c	b	a	b

$s =$	b	b	a
	$s[1]$	$s[2]$	$s[3]$

Subsequence embeddings

Σ is a finite alphabet, e. g., $\Sigma = \{a, b, c, d\}$.

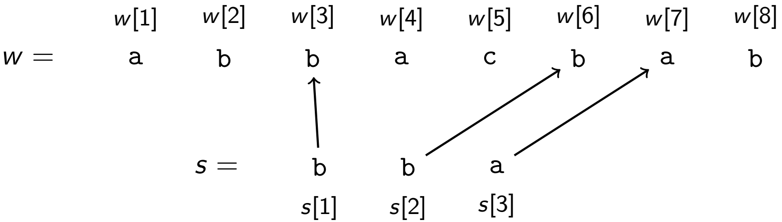


Notation

Embedding: $e : \{1, \ldots, |s|\} \to \{1, \ldots, |w|\}$ with $e(1) < \ldots < e(|s|)$.

Subsequence embeddings

Σ is a finite alphabet, e. g., $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$.

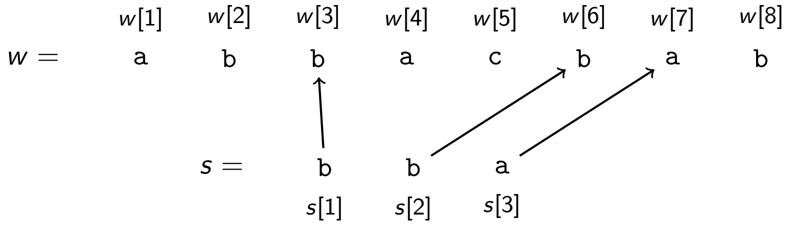


Notation

Embedding: $e : \{1, \ldots, |s|\} \to \{1, \ldots, |w|\}$ with $e(1) < \ldots < e(|s|)$. $s \preceq_e w$: e is an embedding with $s[i] = w[e(i)]\ \forall i \in \{1, 2, \ldots, |s|\}$.

Subsequence embeddings

Σ is a finite alphabet, e. g., $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$.



Notation

Embedding: $e : \{1, \ldots, |s|\} \to \{1, \ldots, |w|\}$ with $e(1) < \ldots < e(|s|)$. $s \preceq_e w$: e is an embedding with $s[i] = w[e(i)]\ \forall i \in \{1, 2, \ldots, |s|\}$.

s is subsequence of w ($s \preceq w$) if there is some embedding e with $s \preceq_e w$.

Subsequence Example

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

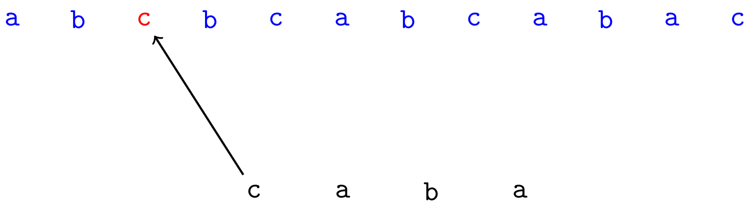
c a b a

Subsequence Example

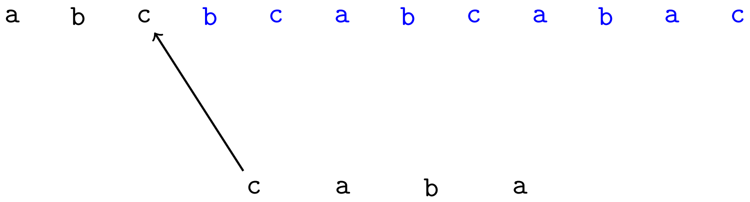
a b c b c a b c a b a c

c a b a

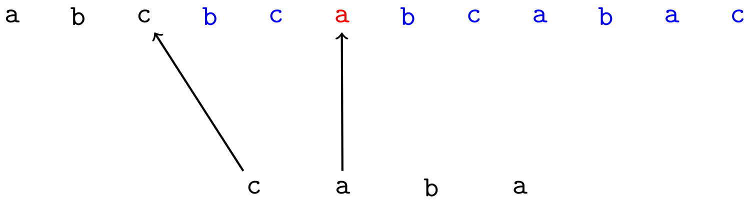
Subsequence Example



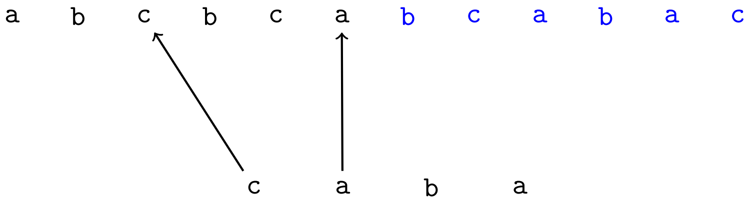
Subsequence Example



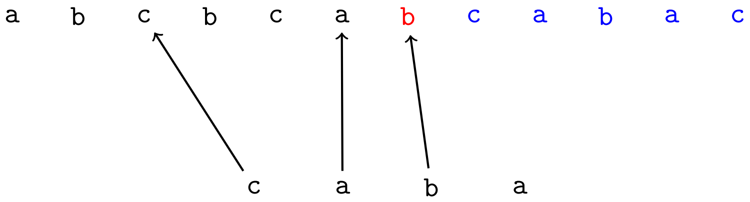
Subsequence Example



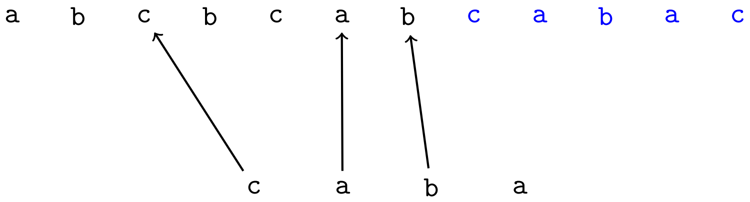
Subsequence Example



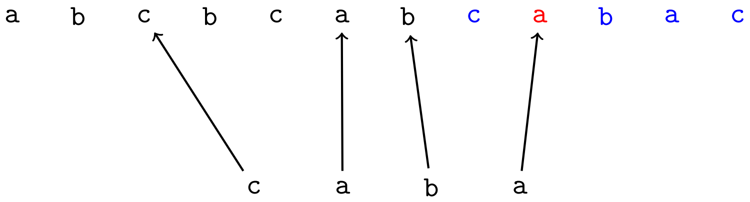
Subsequence Example



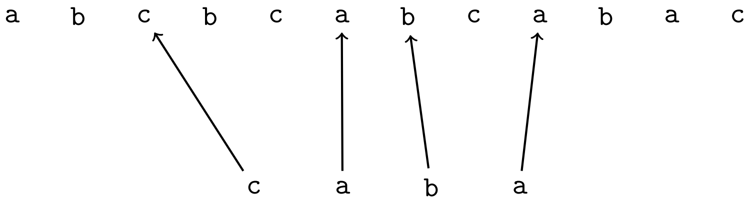
Subsequence Example



Subsequence Example



Subsequence Example



Adding Gap Constraints

Classic setting

Classical subsequences are usually considered with arbitrary embeddings.

Adding Gap Constraints

Classic setting

Classical subsequences are usually considered with arbitrary embeddings.

Constraint setting

Constraints on the embeddings of subsequences (Day, Kosche, Manea, Schmid ISAAC 22).

Adding Gap Constraints

Classic setting

Classical subsequences are usually considered with arbitrary embeddings.

Constraint setting

Constraints on the embeddings of subsequences (Day, Kosche, Manea, Schmid ISAAC 22).

For practical scenarios, it is reasonable to introduce *gap constraints*.
We restrict the length of the gaps by a lower and upper bound.

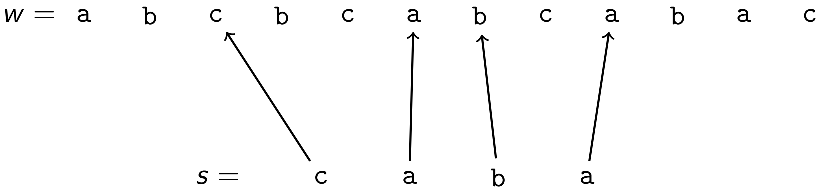
- Alignments of bio-sequences.
- Modelling single processor scheduling with fairness properties.

One could also restrict the letters/languages of the gaps.

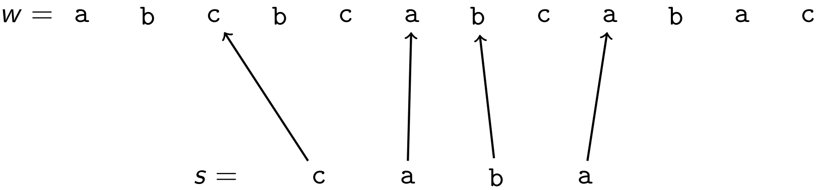
- Complex event processing $\rightarrow$ forbidding events in specific positions of a subsequence (not in this paper).

$w =$ a b c b c a b c a b a c

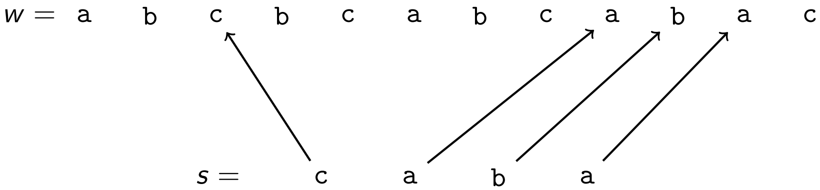
$s =$ c a b a



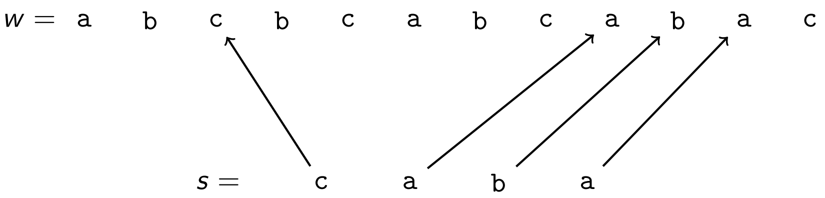
$$\mathrm{gap}_e(w,i) = w[e(i)+1..e(i+1)-1] \qquad i \in \{1,\ldots,|w|-1\}$$



$$\text{gap}_e(w,1) = \texttt{b}\,\texttt{c}$$
$$\text{gap}_e(w,2) = \varepsilon$$
$$\text{gap}_e(w,3) = \texttt{c}$$



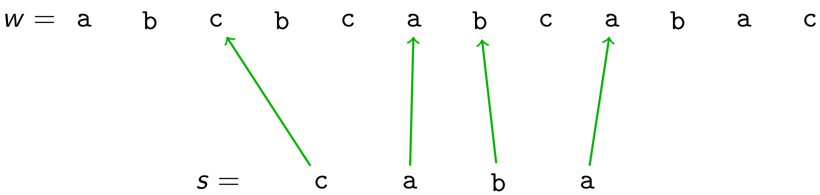
$$\mathrm{gap}_{\bar{e}}(w, 1) = \mathtt{b\,c\,a\,b\,c}$$
$$\mathrm{gap}_{\bar{e}}(w, 2) = \varepsilon$$
$$\mathrm{gap}_{\bar{e}}(w, 3) = \varepsilon$$



$$\mathrm{gap}_{\bar{e}}(w, 1) = \mathtt{b\,c\,a\,b\,c}$$
$$\mathrm{gap}_{\bar{e}}(w, 2) = \varepsilon$$
$$\mathrm{gap}_{\bar{e}}(w, 3) = \varepsilon$$

Gap constraints

$gc = (C_1, \ldots, C_{|s|-1})$, where $C_i = (\ell_i, u_i)$ for every $i \in \{1, \ldots, |s| - 1\}$.
(tuple of gap constraints)
The embedding e *satisfies* gc *w.r.t.* s, if, for every $i \in [|s| - 1]$, $\ell_i \leq |\mathrm{gap}_e(w, i)| \leq u_i$.



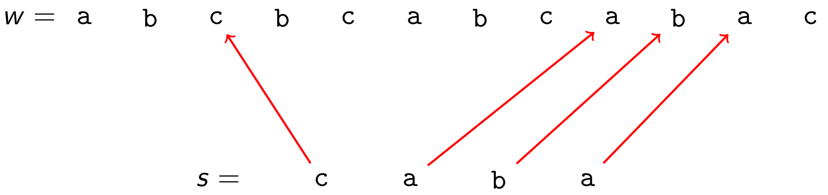
$\text{gap}_e(w, 1) = \text{b c}$ $\quad C_1 = (1, 2)$

$\text{gap}_e(w, 2) = \varepsilon$ $\quad C_2 = (0, 3)$

$\text{gap}_e(w, 3) = \text{c}$ $\quad C_3 = (1, 3)$

Gap constraints

$gc = (C_1, \ldots, C_{|s|-1})$, where $C_i = (\ell_i, u_i)$ for every $i \in \{1, \ldots, |s| - 1\}$.
(tuple of gap constraints)
The embedding e *satisfies gc w.r.t. s*, if, for every $i \in [|s| - 1]$, $\ell_i \leq |\text{gap}_e(w, i)| \leq u_i$.



$\text{gap}_{\bar{e}}(w, 1) = \texttt{b c a b c}$ $\quad C_1 = (1, 2)$

$\text{gap}_{\bar{e}}(w, 2) = \varepsilon$ $\quad C_2 = (0, 3)$

$\text{gap}_{\bar{e}}(w, 3) = \varepsilon$ $\quad C_3 = (1, 3)$

Gap constraints

$gc = (C_1, \ldots, C_{|s|-1})$, where $C_i = (\ell_i, u_i)$ for every $i \in \{1, \ldots, |s| - 1\}$.
(tuple of gap constraints)
The embedding e *satisfies* gc *w.r.t.* s, if, for every $i \in [|s| - 1]$, $\ell_i \leq |\text{gap}_e(w, i)| \leq u_i$.

LCS Problem

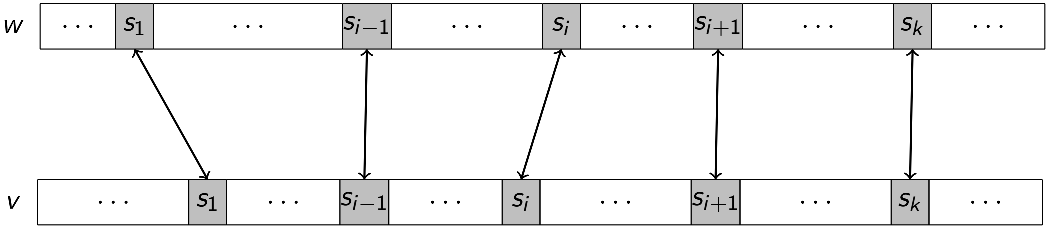
Problem

Given v, w of size m and n, compute the largest $k \in [m]$ such that there exists a common subsequence s of both v and w with $|s| = k$.

LCS Problem

Problem

Given v, w of size m and n, compute the largest $k \in [m]$ such that there exists a common subsequence s of both v and w with $|s| = k$.



LCS Result

Idea

Classical dynamic programming approach.

LCS Result

Idea

Classical dynamic programming approach.

Algorithm and lower bound (Abboud, Backurs, Williams)

Folklore algorithm solving LCS in $O(N)$ time ($N = mn$). Conditional lower bound as a $O(N^{1-\epsilon})$ would refute SETH.

LCS with gaps

Problem: We want to add gap constraints, but we dont know how long the LCS is.

LCS with gaps

Problem: We want to add gap constraints, but we dont know how long the LCS is.

Consider various scenarios:

- Given a large enough tuple of constraints in the input.

LCS with gaps

Problem: We want to add gap constraints, but we dont know how long the LCS is.

Consider various scenarios:

- Given a large enough tuple of constraints in the input.
- All gaps have the same constraint.

LCS with gaps

Problem: We want to add gap constraints, but we dont know how long the LCS is.

Consider various scenarios:

- Given a large enough tuple of constraints in the input.
- All gaps have the same constraint.
- Draw from a constant sized pool of constraints.

LCS with gaps

Problem: We want to add gap constraints, but we dont know how long the LCS is.

Consider various scenarios:

- Given a large enough tuple of constraints in the input.
- All gaps have the same constraint.
- Draw from a constant sized pool of constraints.
- Gap constraints are determined by surrounding letters.

Problems

LCS-MC

Problem

Given $v, w \in \Sigma^*$ and an $(m-1)$-tuple of gap-length constraints gc, compute the largest $k \in \mathbb{N}$ such that there exists a common $gc[1:k-1]$-subsequence s of v and w, with $|s| = k$.

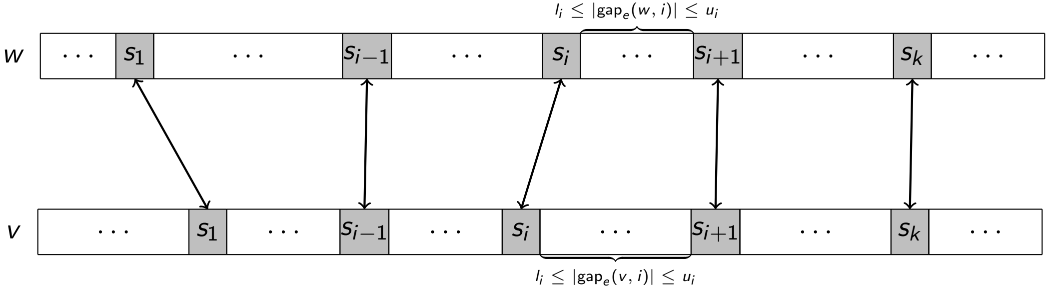
LCS is a particular case of $\mathrm{LCS\text{-}MC}$, where $gc = ((0, n), \ldots, (0, n))$.

LCS-MC

Problem

Given $v, w \in \Sigma^*$ and an $(m-1)$-tuple of gap-length constraints gc, compute the largest $k \in \mathbb{N}$ such that there exists a common $gc[1:k-1]$-subsequence s of v and w, with $|s| = k$.

LCS is a particular case of LCS-MC, where $gc = ((0, n), \ldots, (0, n))$.



LCS-MC Results

Idea

- for each $p \in [m]$, a matrix $M_p \in \{0,1\}^{m \times n}$
- $M_p[i,j] = 1$ if and only if there exists a string s with $|s| = p$ ending in $v[i]$ and $w[j]$ satisfying $gc[1 : p-1]$.
- compute M_1 by setting $M_1[i,j] = 1$ if and only if $v[i] = w[j]$
- $M_p[i][j] = 1$, iff $M_{p-1}[I][J]$ contains a 1 and $v[i] = w[j]$.
- $M_{k+1}[\cdot][\cdot] = 0$, while $M_k[i][j] = 1$ for some i, j.

LCS-MC Results

Idea

- for each $p \in [m]$, a matrix $M_p \in \{0, 1\}^{m \times n}$
- $M_p[i, j] = 1$ if and only if there exists a string s with $|s| = p$ ending in $v[i]$ and $w[j]$ satisfying $gc[1 : p - 1]$.
- compute M_1 by setting $M_1[i, j] = 1$ if and only if $v[i] = w[j]$
- $M_p[i][j] = 1$, iff $M_{p-1}[I][J]$ contains a 1 and $v[i] = w[j]$.
- $M_{k+1}[\cdot][\cdot] = 0$, while $M_k[i][j] = 1$ for some i, j.

Result

$\mathrm{LCS\text{-}MC}$ can be solved in $O(Nk)$ time, where k is the largest number for which there exists a common $gc[1 : k - 1]$-subsequence s of v and w.

LCS-MC-INC Problem

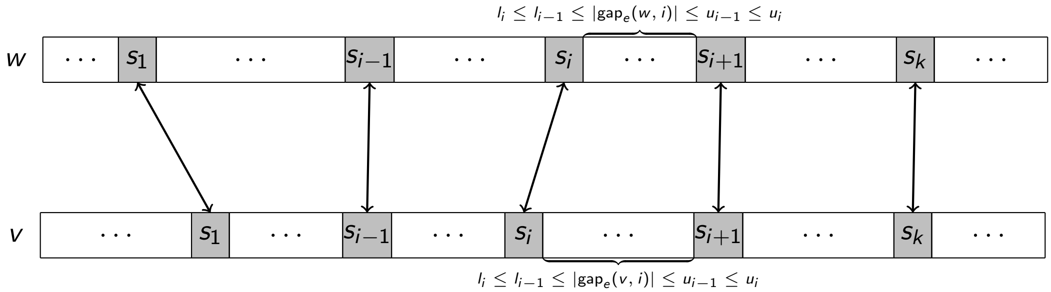
Problem

Consider the variation of $\mathrm{LCS\text{-}MC}$ for increasing tuples of gap-length constraints gc; this variant is called $\mathrm{LCS\text{-}MC\text{-}INC}$.

LCS-MC-INC Problem

Problem

Consider the variation of LCS-MC for increasing tuples of gap-length constraints gc; this variant is called LCS-MC-INC.



LCS-MC-INC Results

Idea

- Matrix $M[i][j] = p$ if there is subsequence s with $|s| = p$ ending in $v[i]$, $w[j]$ satisfying $gc[1 : p-1]$.
- Using a 2D-Segment tree data structure for efficient maximum queries..

LCS-MC-INC Results

Idea

- Matrix $M[i][j] = p$ if there is subsequence s with $|s| = p$ ending in $v[i]$, $w[j]$ satisfying $gc[1 : p-1]$.
- Using a 2D-Segment tree data structure for efficient maximum queries..

Result

LCS-MC-INC can be solved in $O(N \log^2 N)$.

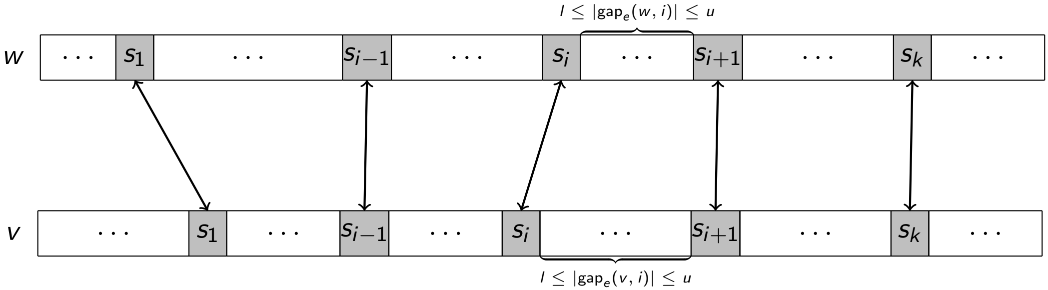
LCS-1C Problem

Problem

Given $v, w \in \Sigma^*$ and an $(m-1)$-tuple of identical gap-length constraints $gc = ((\ell, u), \ldots, (\ell, u))$, compute the largest $k \in \mathbb{N}$ such that there exists a common $gc[1 : k-1]$-subsequence s of v and w, with $|s| = k$.

Problem

Given $v, w \in \Sigma^*$ and an $(m-1)$-tuple of identical gap-length constraints $gc = ((\ell, u), \ldots, (\ell, u))$, compute the largest $k \in \mathbb{N}$ such that there exists a common $gc[1 : k-1]$-subsequence s of v and w, with $|s| = k$.



LCS-1C Results

Idea

- Matrix $M[i][j] = p$ if there is subsequence s with $|s| = p$ ending in $v[i]$, $w[j]$ satisfying $gc[1 : p-1]$ where $gc[i] = gc[i+1]$.
- Use maximum queues on columns to report maximum of relevant part within each column.
- Use maximum queue on each row that gets maxima of each column as input.

LCS-1C Results

Idea

- Matrix $M[i][j] = p$ if there is subsequence s with $|s| = p$ ending in $v[i]$, $w[j]$ satisfying $gc[1 : p-1]$ where $gc[i] = gc[i+1]$.
- Use maximum queues on columns to report maximum of relevant part within each column.
- Use maximum queue on each row that gets maxima of each column as input.

Result

LCS-1C can be solved in $O(N)$ time.

LCS-Σ Problem

Problem

Given two words $v, w \in \Sigma^*$ and two functions $left : \Sigma \to [n] \times [n]$ and $right : \Sigma \to [n] \times [n]$, compute the largest number $k \in \mathbb{N}$ such that there exists a common $(left, right)$-subsequence s of v and w, with $|s| = k$.

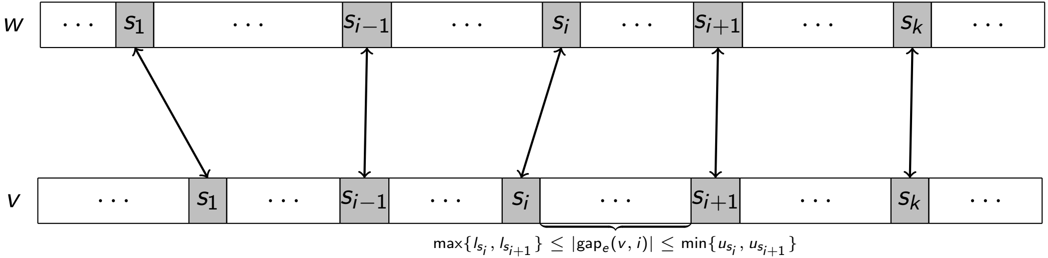
When $left(a) = (0, n)$ for all $a \in \Sigma$ (respectively, $right(a) = (0, n)$ for all $a \in \Sigma$), the gap constraints are defined only by the function *right* (respectively, *left*), and the problem $\mathrm{LCS} - \Sigma$ is denoted $\mathrm{LCS} - \Sigma\mathrm{R}$ (respectively, $\mathrm{LCS} - \Sigma\mathrm{L}$).

LCS-Σ Problem

Problem

Given two words $v, w \in \Sigma^*$ and two functions $left : \Sigma \to [n] \times [n]$ and $right : \Sigma \to [n] \times [n]$, compute the largest number $k \in \mathbb{N}$ such that there exists a common $(left, right)$-subsequence s of v and w, with $|s| = k$.

When $left(a) = (0, n)$ for all $a \in \Sigma$ (respectively, $right(a) = (0, n)$ for all $a \in \Sigma$), the gap constraints are defined only by the function *right* (respectively, *left*), and the problem $\mathrm{LCS} - \Sigma$ is denoted $\mathrm{LCS} - \Sigma\mathrm{R}$ (respectively, $\mathrm{LCS} - \Sigma\mathrm{L}$).



LCS-Σ Results

$|\Sigma| = \sigma$

Idea LCS-ΣR

- maintain σ many submatrices.
- Two dimensional Range Maximum Query data structure.

LCS-Σ Results

$|\Sigma| = \sigma$

Idea LCS-ΣR

- maintain σ many submatrices.
- Two dimensional Range Maximum Query data structure.

LCS-Σ

- maintain σ^2 many submatrices.
- maintain σ many 2D-RMQ data structures.

LCS-Σ Results

$|\Sigma| = \sigma$

Idea LCS-ΣR

- maintain σ many submatrices.
- Two dimensional Range Maximum Query data structure.

LCS-Σ

- maintain σ^2 many submatrices.
- maintain σ many 2D-RMQ data structures.

Result

$\mathrm{LCS}\text{-}\Sigma\mathrm{R}$, $\mathrm{LCS}\text{-}\Sigma\mathrm{L}$ can be solved in $O(\min\{N\sigma, N \log m\})$.
$\mathrm{LCS}\text{-}\Sigma$ can be solved in $O(\min\{N\sigma^2, N\sigma \log m\})$ time.

Conclusion

Summary

- LCS-MC can be solved in $O(Nk)$ time.
- LCS-MC-INC can be solved in $O(N \log^2 N)$.
- LCS-1C can be solved in $\mathrm{O}(N)$ time.
- LCS-O(1)C-SYNC can be solved in $\mathrm{O}(N)$ time.
- LCS-Σ can be solved in $O(\min\{N\sigma^2, N\sigma \log m\})$ time.
- LCS-ΣR, LCS-ΣL can be solved in $O(\min\{N\sigma, N \log m\})$.
- LCS-BR can be solved in $O(NB^{o(1)})$time.

Outlook

- Can the results of $\mathrm{LCS\text{-}MC}$ be improved? (or lower bounds)
- Improving Σ dependancy in $\mathrm{LCS} - \Sigma$?
- Other constraints? (e.g. Regular language constraints).
- Efficiently computing the actual longest common constrained subsequence.
- Different constraints on embeddings in v and w.
- Consider any subsequence problem in the context of gap constraints.

Outlook

- Can the results of $\mathrm{LCS\text{-}MC}$ be improved? (or lower bounds)
- Improving Σ dependancy in $\mathrm{LCS} - \Sigma$?
- Other constraints? (e.g. Regular language constraints).
- Efficiently computing the actual longest common constrained subsequence.
- Different constraints on embeddings in v and w.
- Consider any subsequence problem in the context of gap constraints.

Thank you!