Maximal Common Subsequence Enumeration[1]

How Graph Structure Helped Solve a String Problem

Giulia Punzi
PhD Student in Computer Science

Università di Pisa
Department of Computer Science

Mauriana Pesaresi PhD Seminars – April 20th 2020

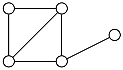
[1] A. Conte, R. Grossi, G. Punzi, T. Uno; "Maximal Common Subsequence Enumeration", SPIRE 2019.

Introduction
Outline

1. Strings and Graphs
2. Our String Problem: Enumerating Maximal Common Subsequences
3. Why is it hard?
4. A Change of Perspective: Graphs
5. Conclusions and Future Work

Introduction
Strings and Graphs

a	b	a	c	b



Strings and Graphs are both ubiquitous in Computer Science.

Strings: most information is textual.

Graphs: essential to represent relationships and network structure.

Oftentimes, the two structures are combined:

- Bioinformatics: DNA sequences are represented with deBruijn graphs;
- Search Engines: textual information naturally linked with a graph structure;
- DFAs: graphs which correspond to regular languages.

Introduction

Combining Strings and Graphs

Oftentimes, the two structures are combined:

- Bioinformatics: DNA sequences are represented with deBruijn graphs;
- Search Engines: textual information naturally linked with a graph structure;
- DFAs: graphs which correspond to regular languages.

$\downarrow$

We will study one instance where a difficult string problem was solved using the underlying graph structure: **Maximal Common Subsequence Enumeration**

Introduction

Maximal Common Subsequences

Given an alphabet Σ, a **string** is a concatenation of any number of its characters. A **subsequence** of a string X, denoted $S \subset X$, is a string obtained from X by removing any number of not necessarily contiguous characters.

Given an alphabet Σ, a **string** is a concatenation of any number of its characters. A **subsequence** of a string X, denoted $S \subset X$, is a string obtained from X by removing any number of not necessarily contiguous characters.

Definition

Given X, Y over Σ, a **Longest Common Subsequence** (LCS) between them is a common subsequence of maximum length.

Introduction

Maximal Common Subsequences

Given an alphabet Σ, a **string** is a concatenation of any number of its characters. A **subsequence** of a string X, denoted $S \subset X$, is a string obtained from X by removing any number of not necessarily contiguous characters.

Definition

Given X, Y over Σ, a **Longest Common Subsequence** (LCS) between them is a common subsequence of maximum length.

Definition (Sakai 2018)

Given X, Y over Σ, a string S is a **Maximal Common Subsequence** of X and Y, denoted $S \in MCS(X, Y)$, if

1. $S \subset X$ and $S \subset Y$;
2. $S \subset W$ with $W \subset X$, $W \subset Y \Rightarrow S = W$.

Example

Let $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$ and consider

$$X = \boxed{\texttt{A}}\texttt{T}\boxed{\texttt{C}}\texttt{AGG}\boxed{\texttt{T}}$$
$$Y = \texttt{G}\boxed{\texttt{AC}}\texttt{TA}\boxed{\texttt{T}}$$

then:

1. $S = \texttt{ACT}$ is a common subsequence of X and Y.

Introduction

Maximal Common Subsequences

Example

Let $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$ and consider

$$X = \texttt{ATCAGGT}$$
$$Y = \texttt{GACTAT}$$

then:

1. $S = \texttt{ACT}$ is a common subsequence of X and Y;
2. $MCS(X, Y) = \{\texttt{ACAT}, \texttt{ATAT}, \texttt{GT}\}$.

LCS: one of the main string comparison tools

LCS: one of the main string comparison tools

$\downarrow$

Limitation: LCS has a quadratic conditional lower bound (Abboud et al, 2015)

LCS: one of the main string comparison tools

$\downarrow$

Limitation: LCS has a quadratic conditional lower bound (Abboud et al, 2015)

MCS are a natural generalization of LCS.

- One MCS can be found in $O(n \log \log(n))$ time (Sakai 2018)
- Might reveal alternative smaller alignments

Our Aim: Efficient MCS Enumeration

Enumeration algorithm: it lists every element of a given set exactly once.

Our Aim: Efficient MCS Enumeration

Enumeration algorithm: it lists every element of a given set exactly once.

Polynomial-delay: delay between output of consecutive solutions is polynomial.

Our Aim: Efficient MCS Enumeration

Enumeration algorithm: it lists every element of a given set exactly once.

Polynomial-delay: delay between output of consecutive solutions is polynomial.

Problem (MCS Enumeration)

List all **distinct** maximal common subsequences $S \in MCS(X, Y)$, for X, Y of length $O(n)$ over Σ of size σ, with polynomial delay.

Our Aim: Efficient MCS Enumeration

Enumeration algorithm: it lists every element of a given set exactly once.

Polynomial-delay: delay between output of consecutive solutions is polynomial.

Problem (MCS Enumeration)

List all **distinct** maximal common subsequences $S \in MCS(X, Y)$, for X, Y of length $O(n)$ over Σ of size σ, with polynomial delay.

Note that by distinct we mean as elements of the set $MCS(X, Y)$:
strings with multiple occurrences need to be output once.

Our Aim: MCS Enumeration

Example (Enumeration)

$$X = \texttt{TAAGCC}$$
$$Y = \texttt{TAGACT}$$

Output:

Our Aim: MCS Enumeration

Example (Enumeration)

$$X = \boxed{\texttt{TA}}\texttt{A}\boxed{\texttt{GC}}\texttt{C}$$
$$Y = \boxed{\texttt{TAG}}\texttt{A}\boxed{\texttt{C}}\texttt{T}$$

Output:

Our Aim: MCS Enumeration

Example (Enumeration)

$$X = \boxed{\texttt{T}}\texttt{A}\boxed{\texttt{AGC}}\texttt{C}$$
$$Y = \boxed{\texttt{TAG}}\texttt{A}\boxed{\texttt{C}}\texttt{T}$$

Output:

Our Aim: MCS Enumeration

Example (Enumeration)

$$X = \fbox{T}\texttt{A}\fbox{AG}\texttt{C}\fbox{C}$$
$$Y = \fbox{TAG}\texttt{A}\fbox{C}\texttt{T}$$

Output:

Our Aim: MCS Enumeration

Example (Enumeration)

$$X = \boxed{\texttt{TA}}\texttt{A}\boxed{\texttt{G}}\texttt{C}\boxed{\texttt{C}}$$
$$Y = \boxed{\texttt{TAG}}\texttt{A}\boxed{\texttt{C}}\texttt{T}$$

Output:

Our Aim: MCS Enumeration

Example (Enumeration)

$$X = \texttt{TAAGCC}$$
$$Y = \texttt{TAGACT}$$

Output:

- TAGC

Our Aim: MCS Enumeration

Example (Enumeration)

$$X = \boxed{\texttt{TAA}}\texttt{G}\boxed{\texttt{C}}\texttt{C}$$
$$Y = \boxed{\texttt{TA}}\texttt{G}\boxed{\texttt{AC}}\texttt{T}$$

Output:

- TAGC

Our Aim: MCS Enumeration

Example (Enumeration)

$$X = \boxed{\texttt{TAA}}\texttt{GC}\boxed{\texttt{C}}$$
$$Y = \boxed{\texttt{TA}}\texttt{G}\boxed{\texttt{AC}}\texttt{T}$$

Output:

- `TAGC`

Our Aim: MCS Enumeration

Example (Enumeration)

$$X = \texttt{TAAGCC}$$
$$Y = \texttt{TAGACT}$$

Output:

- TAGC
- TAAC

Pitfalls of MCS Enumeration

Pitfalls of MCS Enumeration

1. Using a divide and conquer approach

Pitfalls of MCS Enumeration

1. ~~Using a divide and conquer approach~~
   MCS do not naturally combine.

Example

$$X = \texttt{AGA}|\texttt{TGA}$$
$$Y = \texttt{TAG}|\texttt{GAT}$$

$MCS(X, Y) = \{\texttt{AGGA}, \texttt{AGAT}, \texttt{TGA}\}$: the combination AGT of the two blue submaximals is not maximal.

Pitfalls of MCS Enumeration

1. ~~Using a divide and conquer approach~~
   MCS do not naturally combine.

2. Thinking that MCS are a small number

Pitfalls of MCS Enumeration

1. ~~Using a divide and conquer approach~~
   MCS do not naturally combine.

2. ~~Thinking that MCS are a small number~~
   MCS can be exponential even for $|\Sigma| = 2$.

Example

The two strings

$$X = \mathtt{A} \circ (\mathtt{CCA})^n; \quad Y = \mathtt{A} \circ (\mathtt{CA})^{\lfloor \frac{3n}{2} \rfloor}.$$

have an exponential number of MCS.

Pitfalls of MCS Enumeration

1. ~~Using a divide and conquer approach~~
   MCS do not naturally combine.

2. ~~Thinking that MCS are a small number~~
   MCS can be exponential even for $|\Sigma| = 2$.

3. Using an incremental approach?
   Let X and Y be any two strings; is it true that

$$MCS(X, Y) \circ c \leftrightarrow MCS(X, Y \circ c)?$$

Pitfalls of MCS Enumeration

Incremental Approach is Inefficient

Some incremental properties can be derived, but they are intrinsically inefficient.

Pitfalls of MCS Enumeration

Incremental Approach is Inefficient

Some incremental properties can be derived, but they are intrinsically inefficient.

Example

$$X = \texttt{ACCACCACCA}$$
$$Z = \texttt{ACACACACA}$$

Consider X and $Y = Z \circ Z$, and we proceed incrementally over Y. Since $X \subset Y$, $MCS(X, Y) = \{X\}$ but when we are at half length, $|MCS(X, Z)|$ is exponential.

Pitfalls of MCS Enumeration

Incremental Approach is Inefficient

Some incremental properties can be derived, but they are intrinsically inefficient.

Example

$$X = \texttt{ACCACCACCA}$$
$$Z = \texttt{ACACACACA}$$

Consider X and $Y = Z \circ Z$, and we proceed incrementally over Y. Since $X \subset Y$, $MCS(X,Y) = \{X\}$ but when we are at half length, $|MCS(X,Z)|$ is exponential.

$\longrightarrow$ it leads to an **exponential delay** algorithm!

Challenge: Polynomial Delay?

Goal: Design of a polynomial delay enumeration algorithm for MCS.

Challenge: Polynomial Delay?

Goal: Design of a polynomial delay enumeration algorithm for MCS.

Idea: Instead of finding maximals of the prefixes, we find prefixes of maximals.

Challenge: Polynomial Delay?

Goal: Design of a polynomial delay enumeration algorithm for MCS.

Idea: Instead of finding maximals of the prefixes, we find prefixes of maximals.

Definition

$P \subset X, Y$ is called a **valid prefix** if $\exists W$ such that $P \circ W \in MCS(X, Y)$.

Challenge: Polynomial Delay?

Goal: Design of a polynomial delay enumeration algorithm for MCS.

Idea: Instead of finding maximals of the prefixes, we find prefixes of maximals.

Definition

$P \subset X, Y$ is called a **valid prefix** if $\exists W$ such that $P \circ W \in MCS(X, Y)$.

If we have a characterization for valid prefixes, we can build increasingly long prefixes of maximals by appending valid characters, until we generate all MCS.

Unshiftable Edges

Bipartite String Graph

Definition (String Graphs and Mappings)

Given two strings X, Y, the corresponding **Bipartite String Graph** (BSG) is the bipartite graph $G(X, Y)$ that has one vertex for each position of X and of Y, and edge set $E = \{(i, j) \mid X[i] = Y[j]\}$. A **mapping** of a string graph is a subset of the edges $\mathcal{P} \subseteq E$ such that $\forall (i, j)$, $(h, k) \in \mathcal{P}$ we have $i \leq h \iff j \leq k$.

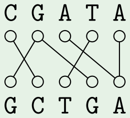
Unshiftable Edges
Bipartite String Graph

Definition (String Graphs and Mappings)

Given two strings X, Y, the corresponding **Bipartite String Graph** (BSG) is the bipartite graph $G(X, Y)$ that has one vertex for each position of X and of Y, and edge set $E = \{(i, j) \mid X[i] = Y[j]\}$. A **mapping** of a string graph is a subset of the edges $\mathcal{P} \subseteq E$ such that $\forall (i, j)$, $(h, k) \in \mathcal{P}$ we have $i \leq h \iff j \leq k$.

Example

The BSG for the two strings $X =$ `CGATA` and $Y =$ `GCTGA` is given by



Unshiftable Edges

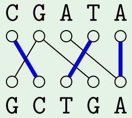
Bipartite String Graph

Definition (String Graphs and Mappings)

Given two strings X, Y, the corresponding **Bipartite String Graph** (BSG) is the bipartite graph $G(X, Y)$ that has one vertex for each position of X and of Y, and edge set $E = \{(i, j) \mid X[i] = Y[j]\}$. A **mapping** of a string graph is a subset of the edges $\mathcal{P} \subseteq E$ such that $\forall (i, j)$, $(h, k) \in \mathcal{P}$ we have $i \leq h \iff j \leq k$.

Example

The BSG for the two strings $X =$ `CGATA` and $Y =$ `GCTGA` is given by



A mapping of the graph is shown in blue.

Unshiftable Edges

Maximal Mappings and MCS

Definition

A mapping $\mathcal{P}$ of a BSG is said to be **maximal** if adding any edge (i, j) to $\mathcal{P}$ no longer yields a mapping.

Unshiftable Edges

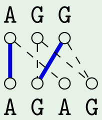
Maximal Mappings and MCS

Definition

A mapping $\mathcal{P}$ of a BSG is said to be **maximal** if adding any edge (i, j) to $\mathcal{P}$ no longer yields a mapping.

Example (MCS $\neq$ maximal mappings)

Every MCS corresponds to a maximal mapping, but the opposite does not hold. Consider $X = \texttt{AGG}$ and $Y = \texttt{AGAG}$:



The blue mapping is maximal, but it does not correspond to any MCS.

Unshiftable Edges

Definition

Definition

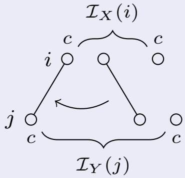
Let $\mathcal{I}_X(i)$ be the substring $X[i+1, ..., next_X(i)]$ (analogously for Y).

Unshiftable Edges
Definition

Definition

Let $\mathcal{I}_X(i)$ be the substring $X[i+1, ..., next_X(i)]$ (analogously for Y).
An edge (i, j) is **unshiftable** ($(i, j) \in \mathcal{U}$) if and only if either

- (Base case) It corresponds to the last pairwise occurrence in the strings of character $X[i] = Y[j]$.
- (Otherwise) There is at least one unshiftable edge in $G(\mathcal{I}_X(i), \mathcal{I}_Y(j))$.



Unshiftable Edges
Example

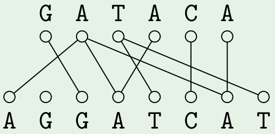
Intuition: every unshiftable belongs to a maximal mapping where it cannot be "pushed further right" while spelling the same word.

Unshiftable Edges

Example

Intuition: every unshiftable belongs to a maximal mapping where it cannot be "pushed further right" while spelling the same word.

Example

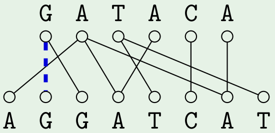


Unshiftable Edges

Example

Intuition: every unshiftable belongs to a maximal mapping where it cannot be "pushed further right" while spelling the same word.

Example



Unshiftable Edges

Still not enough

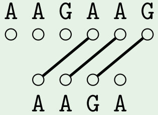
Example (MCS $\neq$ maximal unshiftable mappings)

Unshiftable Edges

Still not enough

Example (MCS $\neq$ maximal unshiftable mappings)

Consider $X =$ AAGAAG, $Y =$ AAGA. In the corresponding graph, we have a maximal rightmost unshiftable mapping for the string AAG:



even though this word is not maximal: the only MCS is the whole AAGA.

Extending the Prefix
Candidate Extensions

P valid prefix $\rightarrow$ formally define Ext_P set of **candidate extensions**: being a candidate is necessary for having a valid extension.

Intuition: "first unshiftable edges after P".

Extending the Prefix

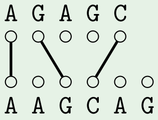
Candidate Extensions

P valid prefix $\rightarrow$ formally define Ext_P set of **candidate extensions**: being a candidate is necessary for having a valid extension.

Intuition: "first unshiftable edges after P".

Example (The condition is not sufficient)

Consider $X =$ AGAGC, $Y =$ AAGCAG. We have $MCS(X, Y) = \{$AGAG, AAGC$\}$.
Clearly, $P =$ AG is a valid prefix.



The edge for C is in Ext_P, but AGC is not a valid prefix.

Extending the Prefix

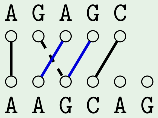
Candidate Extensions

P valid prefix $\rightarrow$ formally define Ext_P set of **candidate extensions**: being a candidate is necessary for having a valid extension.

Intuition: "first unshiftable edges after P".

Example (The condition is not sufficient)

Consider $X =$ AGAGC, $Y =$ AAGCAG. We have $MCS(X, Y) = \{$AGAG, AAGC$\}$. Clearly, $P =$ AG is a valid prefix.



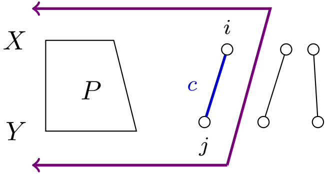
The edge for C is in Ext_P, but AGC is not a valid prefix.

Correctness of Extension

Theorem (Correctness)

Let P be a valid prefix of some $M \in MCS(X, Y)$. Then $P \circ c$ is still a valid prefix if and only if the following two conditions hold:

1. $\exists(i, j) \in Ext_P$ *corresponding to character* c*;*
2. $P \in MCS(X_{<i}, Y_{<j})$.



The Algorithm
Binary Partition Paradigm

The Algorithm

Binary Partition Paradigm

Binary partition: enumerative scheme based on iterative partitions of solutions.

Partition solutions into smaller sets characterized by disjoint properties, until we get to singletons $\rightarrow$ obtain tree with every and only feasible solution as leaves

The Algorithm

Binary Partition Paradigm

Binary partition: enumerative scheme based on iterative partitions of solutions.

Partition solutions into smaller sets characterized by disjoint properties, until we get to singletons $\rightarrow$ obtain tree with every and only feasible solution as leaves

$$\mathcal{P} \equiv \text{having string } P \text{ as a prefix.}$$
$$\downarrow$$
$$\text{branching into possibly } |\Sigma| \text{ partitions}$$

The Algorithm
Binary Partition Paradigm

Binary partition: enumerative scheme based on iterative partitions of solutions.

Partition solutions into smaller sets characterized by disjoint properties, until we get to singletons $\rightarrow$ obtain tree with every and only feasible solution as leaves

$$\mathcal{P} \equiv \text{having string } P \text{ as a prefix.}$$
$$\downarrow$$
$$\text{branching into possibly } |\Sigma| \text{ partitions}$$

Complexity: If the partition oracle takes polynomial time and the height of the tree is polynomial, then the algorithm is polynomial delay.

The EnumerateMCS Algorithm

```
procedure EnumerateMCS(X, Y, Σ)
    𝒰 = FindUnshiftables((|X|, |Y|))
    BinaryPartition(#, {(−1, −1)})
end procedure

procedure BinaryPartition(P, L_P)
    compute the set of extensions Ext_P using 𝒰
    if Ext_P = ∅ then Output P
    else
        for (i, j) ∈ Ext_P corresponding to some c ∈ Σ do
            if P ∈ MCS(X_{<i}, Y_{<j}) then
                let (l, m) be the last edge of L_P
                find leftmost mapping edge (l_c, m_c) for c in G(X_{>l}, Y_{>m})
                BinaryPartition(P c, L_P ∪ (l_c, m_c))
            end if
        end for
    end if
end procedure
```

The EnumerateMCS Algorithm

procedure EnumerateMCS(X, Y, Σ)
    $\mathcal{U}$ **= FindUnshiftables(**$(|X|, |Y|)$**)** $\longrightarrow$
    BinaryPartition($\#$, $\{(-1, -1)\}$)
end procedure

For each $e \in \mathcal{U}$ find previous pairwise occurrences of every $c \in \Sigma$, then add edge to $\mathcal{U}$ if not already present: $O(\sigma n^2 \log(n))$ time

procedure BinaryPartition(P, L_P)
    compute the set of extensions Ext_P using $\mathcal{U}$
    if $Ext_P = \emptyset$ **then Output** P
    else
        for $(i, j) \in Ext_P$ corresponding to some $c \in \Sigma$ **do**
            if $P \in MCS(X_{<i}, Y_{<j})$ **then**
                let (l, m) be the last edge of L_P
                find leftmost mapping edge (l_c, m_c) for c in $G(X_{>l}, Y_{>m})$
                BinaryPartition($P\,c$, $L_P \cup (l_c, m_c)$)
            end if
        end for
    end if
end procedure

The EnumerateMCS Algorithm

procedure EnumerateMCS(X, Y, Σ)
  $\mathcal{U}$ = FindUnshiftables($(|X|, |Y|)$)
  BinaryPartition($\#$, $\{(-1,-1)\}$)
end procedure

procedure BinaryPartition(P, L_P)
  compute the set of extensions Ext_P **using** $\mathcal{U}$ $\longrightarrow$
  if $Ext_P = \emptyset$ **then Output** P
  else
    for $(i, j) \in Ext_P$ corresponding to some $c \in \Sigma$ **do**
      if $P \in MCS(X_{<i}, Y_{<j})$ **then**
        let (l, m) be the last edge of L_P
        find leftmost mapping edge (l_c, m_c) for c in $G(X_{>l}, Y_{>m})$
        BinaryPartition($P\,c$, $L_P \cup (l_c, m_c)$)
      end if
    end for
  end if
end procedure

Parse unshiftable edges: $O(n^2)$ time

The EnumerateMCS Algorithm

procedure EnumerateMCS(X, Y, Σ)
    $\mathcal{U} = \text{FindUnshiftables}((|X|, |Y|))$
    BinaryPartition($\#$, $\{(-1, -1)\}$)
end procedure

procedure BinaryPartition(P, L_P)
    compute the set of extensions Ext_P using $\mathcal{U}$
    if $Ext_P = \emptyset$ **then Output** P
    else
        for $(i, j) \in Ext_P$ corresponding to some $c \in$
            if $P \in MCS(X_{<i}, Y_{<j})$ **then** $\longrightarrow$
                let (l, m) be the last edge of L_P
                find leftmost mapping edge (l_c, m_c) for c in $G(X_{>l}, Y_{>m})$
                BinaryPartition($P\,c$, $L_P \cup (l_c, m_c)$)
            end if
        end for
    end if
end procedure

This can be done in $O(|P|) = O(n)$ time (Sakai 2018)

The EnumerateMCS Algorithm

procedure EnumerateMCS(X, Y, Σ)
    $\mathcal{U}$ = FindUnshiftables$((|X|, |Y|))$
    BinaryPartition($\#$, $\{(-1, -1)\}$)
end procedure

procedure BinaryPartition(P, L_P)
    compute the set of extensions Ext_P using $\mathcal{U}$
    if $Ext_P = \emptyset$ **then Output** P
    else
        for $(i, j) \in Ext_P$ corresponding to some $c \in \Sigma$ **do**
            if $P \in MCS(X_{<i}, Y_{<j})$ **then**
                let (l, m) be the last edge of L_P
                find leftmost mapping edge (l_c, m_c) **for** c **in** $G(X_{>l}, Y_{>m})$
                BinaryPartition($P\,c$, $L_P \cup (l_c, m_c)$)
            end if
        end for
    end if
end procedure

Logarithmic in length of strings: $O(\log(n))$ time

The EnumerateMCS Algorithm

procedure EnumerateMCS(X, Y, Σ)
  $\mathcal{U}$ = FindUnshiftables$((|X|, |Y|))$
  BinaryPartition($\#$, $\{(-1,-1)\}$)
end procedure

procedure BinaryPartition(P, L_P)
  compute the set of extensions Ext_P using $\mathcal{U}$
  if $Ext_P = \emptyset$ **then Output** P
  else
    for $(i,j) \in Ext_P$ corresponding to some $c \in \Sigma$ **do**
      if $P \in MCS(X_{<i}, Y_{<j})$ **then**
        let (l, m) be the last edge of L_P
        find leftmost mapping edge (l_c, m_c) for c in $G(X_{>l}, Y_{>m})$
        BinaryPartition($P\,c$, $L_P \cup (l_c, m_c)$)
      end if
    end for
  end if
end procedure

Partition oracle: $O(n^2 + |Ext_P|(n + \log(n))) = O(n^2)$ time

The EnumerateMCS Algorithm
Final Complexity

Height of the partition tree: $O(n)$ (length of longest MCS)

$\Rightarrow$ $O(n^3)$ delay
$O(\sigma n^2 \log(n))$ preprocessing time
$O(n^2)$ space.

Height of the partition tree: $O(n)$ (length of longest MCS)

$\Rightarrow O(n^3)$ delay
$O(\sigma n^2 \log(n))$ preprocessing time
$O(n^2)$ space.

Theorem

There is a $O(n\sigma(\sigma + \log n))$ polynomial-delay enumeration algorithm for MCS enumeration, with $O(n^2(\sigma + \log n))$ preprocessing time and $O(n^2)$ space.

Conclusions and Future Work

Conclusions and Future Work

- We investigated the string problem of enumerating MCS for the first time; it turned out to be hard to approach with standard techniques.
- Changing our perspective by looking at the strings as a graph was crucial to derive fundamental properties, and eventually solve the problem.
- MCS are just one of many string problems with interesting applications: similar shift in perspective might help solve other difficult problems.

Conclusions and Future Work

- We investigated the string problem of enumerating MCS for the first time; it turned out to be hard to approach with standard techniques.
- Changing our perspective by looking at the strings as a graph was crucial to derive fundamental properties, and eventually solve the problem.
- MCS are just one of many string problems with interesting applications: similar shift in perspective might help solve other difficult problems.

Future Work:

- Explore further connections between LCS and MCS.

Conclusions and Future Work

- We investigated the string problem of enumerating MCS for the first time; it turned out to be hard to approach with standard techniques.
- Changing our perspective by looking at the strings as a graph was crucial to derive fundamental properties, and eventually solve the problem.
- MCS are just one of many string problems with interesting applications: similar shift in perspective might help solve other difficult problems.

Future Work:

- Explore further connections between LCS and MCS.
- Find other applications of graph-theoretic tools to string problems.

Thank you for your attention!

Any Questions?

Feel free to email me at giulia.punzi@phd.unipi.it

References

Y. Sakai, "Maximal Common Subsequence Algorithms"; in 29th Annual Symposium on Combinatorial Pattern Matching, 1-10, 2018.

A. Conte, R. Grossi, G. Punzi, T. Uno, (2019) "Polynomial-Delay Enumeration of Maximal Common Subsequences"; in: Brisaboa N., Puglisi S. (eds) String Processing and Information Retrieval (SPIRE 2019), Lecture Notes in Computer Science, vol 11811, 2019.

Pitfalls of MCS

Incremental Approach

Let X and Y' be any two strings. Consider $Y = Y' \circ c$;

$$MCS(X, Y') \circ c \leftrightarrow MCS(X, Y)?$$

Example

- (Some MCS are not found) Let

$$X = \texttt{AGCG}$$
$$Y = \underbrace{\texttt{ACG}}_{Y'}|\texttt{C}$$

  $MCS(X, Y') = \{\texttt{ACG}\}$: $\texttt{AGC} \in MCS(X, Y)$ was not found.

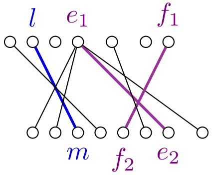
- (Some strings found are not MCS) Instead in

$$X = \texttt{AAGACT}$$
$$Y = \underbrace{\texttt{AGCAG}}_{Y'}|\texttt{C}$$

  we have $\texttt{AGC} \in MCS(X, Y')$, but $\texttt{AGC} \notin MCS(X, Y)$.

Extending the Prefix
The Cross

Let P be a prefix of some $W \in MCS(X,Y)$. Given a character $c \in \Sigma$, we would like to find a necessary and sufficient condition for $P \circ c$ to still be a valid prefix.



Definition (Cross)

Given an edge (l,m), its following **cross** $\chi_{(l,m)} = \{e,f\}$ is given by (at most) two edges such that:

- $e = (e_1, e_2) \in \mathcal{U}$ is such that $e_1 = \min\{h_1 > l \mid \exists h_2 > m : (h_1,h_2) \in \mathcal{U}\}$.
- $f = (f_1, f_2) \in \mathcal{U}$ is such that $f_2 = \min\{h_2 > m \mid \exists h_1 > l : (h_1,h_2) \in \mathcal{U}\}$.

Extending the Prefix

Candidate Extensions

Definition

Let P be a prefix of some MCS, with its leftmost mapping L_P ending at edge $l = (l, m)$, and let $\chi_{(l,m)} = (e, f)$ be its cross. We define the set of the **"Mikado" edges** after P as

$$Mk_P = \{(i, j) \in \mathcal{U} \mid e_1 \leq i \leq f_1 \text{ and } f_2 \leq j \leq e_2\}.$$

From these we extract the **candidate extensions** for P as follows

$$Ext_P = \{(i, j) \in Mk_P \mid \nexists (h, k) \in Mk_P \setminus (i, j) \text{ such that } h \leq i \text{ and } k \leq j\}.$$

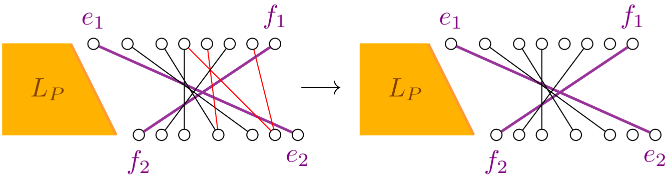


Figure: Mk_P set transformed into Ext_P

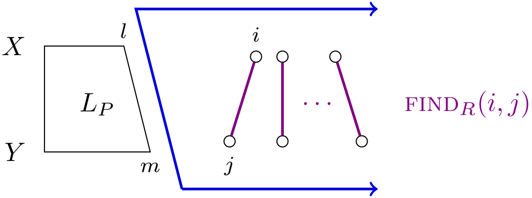
Correctness

FIND_R procedure

Given $(i, j) \in \mathcal{U}$, $\text{FIND}_R(i, j)$ returns a maximal mapping in $G(X_{>i}, Y_{>j})$.

Lemma 1

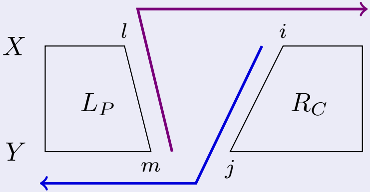
Let P be a valid prefix with leftmost mapping ending with edge (l, m), and let $(i, j) \in Ext_P$. Then, $\text{FIND}_R(i, j)$ returns a mapping whose corresponding subsequence is $M \in MCS(X_{>l}, Y_{>m})$.



Theorem (MCS Combination)

Let P and C be common subsequences of X, Y. Let (l, m) be the last edge of the leftmost mapping of P, and (i, j) be the first edge of the rightmost mapping of C. Then:

$$P \circ C \in MCS(X, Y) \iff P \in MCS(X_{<i}, Y_{<j}) \text{ and } C \in MCS(X_{>l}, Y_{>m}).$$



Theorem (Correctness)

Let P be a valid prefix of some $M \in MCS(X,Y)$, with leftmost mapping L_P ending with edge (l,m). Then $P \circ c$ is still a valid prefix if and only if the following two conditions hold:

1. *$\exists(i,j) \in Ext_P$ corresponding to character c;*
2. *$P \in MCS(X_{<i}, Y_{<j})$.*

Proof.

We have said that the conditions are necessary in the previous sections. We know that $\textsc{Find}_R(i,j) = C \in MCS(X_{>l}, Y_{>m})$. By hypothesis $P \in MCS(X_{<i}, Y_{<j})$, therefore by the MCS combination theorem we have $P \circ C \in MCS(X,Y)$. This latter string starts with $P \circ c$, which is therefore a good prefix. □