Linear Space is Impractical: Constructing Antidictionaries in Output-Sensitive Space

Lorraine A.K. Ayad Golnaz Badkobeh Gabriele Fici
Alice Héliou Solon P. Pissis

CWI meeting

Amsterdam, The Netherlands, 22 Feb. 2019

Combinatorial Pattern Matching

- **Pattern Matching**

$x = \texttt{ababb} \qquad y = \texttt{abb}\underline{\texttt{ababb}}\texttt{bab}\underline{\texttt{ababb}}\texttt{a}$

Combinatorial Pattern Matching

- **Pattern Matching**

  $x =$ ababb $\quad y =$ abbababbbabababba

- **Indexing**

  $y =$ abbababbbabababba $\quad x =$ ababb

Combinatorial Pattern Matching

- **Pattern Matching**

$x =$ ababb $\qquad y =$ abbababbbabababba

- **Indexing**

$y =$ abbababbbabababba $\qquad x =$ ababb

- **Comparison**

$x =$ abbababbbabababba $\qquad$ abbababbbabababba

$y =$ abbabbbabbbabba $\qquad$ abba--bbbabbbabba

Combinatorial Pattern Matching

- **Pattern Matching**

  $x = \mathtt{ababb}$ $y = \mathtt{abb\underline{ababb}bab\underline{ababb}a}$

- **Indexing**

  $y = \mathtt{abbababbbababbba}$ $x = \mathtt{ababb}$

- **Comparison**

  $x = \mathtt{abbababbbababbba}$ abbababbbababbba

  $y = \mathtt{abbabbbabbbabba}$ abba--bbbabbbabba

- **Regularities**

  $x = \mathtt{abbabaabaabababba}$ abbabaabaabababba

Applications of Combinatorial Pattern Matching

Applications of Combinatorial Pattern Matching

- **Pattern Matching**

$x = \texttt{ababb}$ $y = \texttt{abb}\underline{\texttt{ababb}}\texttt{bab}\underline{\texttt{ababb}}\texttt{a}$

Applications of Combinatorial Pattern Matching

- **Pattern Matching**

$x = \texttt{ababb}$ $\quad$ $y = \texttt{abb}\underline{\texttt{ababb}}\texttt{bab}\underline{\texttt{ababb}}\texttt{a}$

text editors; *grep* command-line utility; etc.

Applications of Combinatorial Pattern Matching

- **Pattern Matching**

  $x = \texttt{ababb} \qquad y = \texttt{abb}\underline{\texttt{ababb}}\texttt{bab}\underline{\texttt{ababb}}\texttt{a}$

  text editors; *grep* command-line utility; etc.

- **Indexing**

  $y = \texttt{abbababbbababababba} \qquad x = \texttt{ababb}$

Applications of Combinatorial Pattern Matching

- **Pattern Matching**

  $x = \texttt{ababb} \qquad y = \texttt{abb}\underline{\texttt{ababb}}\texttt{bab}\underline{\texttt{ababb}}\texttt{a}$

  text editors; *grep* command-line utility; etc.

- **Indexing**

  $y = \texttt{abbababbbabababba} \qquad x = \texttt{ababb}$

  indexing genomes; indexing a collection of documents; etc.

Applications of Combinatorial Pattern Matching

- **Pattern Matching**

  $x =$ ababb $y =$ abbababbbabababba

  text editors; *grep* command-line utility; etc.

- **Indexing**

  $y =$ abbababbbabababba $x =$ ababb

  indexing genomes; indexing a collection of documents; etc.

- **Comparison**

  $x =$ abbababbbabababba abbababbbabababba

  $y =$ abbabbbabbbabba abba--bbbabbbabba

Applications of Combinatorial Pattern Matching

- **Pattern Matching**

  $x = \texttt{ababb}$ $y = \texttt{abbababbbabababba}$

  text editors; *grep* command-line utility; etc.

- **Indexing**

  $y = \texttt{abbababbbabababba}$ $x = \texttt{ababb}$

  indexing genomes; indexing a collection of documents; etc.

- **Comparison**

  $x = \texttt{abbababbbabababba}$ abbababbbabababba

  $y = \texttt{abbabbbabbbabba}$ abba--bbbabbbabba

  diff command-line utility; aligning genomic sequences; etc.

Applications of Combinatorial Pattern Matching

- **Pattern Matching**

  $x = \texttt{ababb}$ $\quad y = \texttt{abbababbbabababba}$

  text editors; *grep* command-line utility; etc.

- **Indexing**

  $y = \texttt{abbababbbabababba}$ $\quad x = \texttt{ababb}$

  indexing genomes; indexing a collection of documents; etc.

- **Comparison**

  $x = \texttt{abbababbbabababba}$ $\quad$ abbababbbabababba

  $y = \texttt{abbabbbabbbabba}$ $\quad$ abba--bbbabbbabba

  diff command-line utility; aligning genomic sequences; etc.

- **Regularities**

  $x = \texttt{abbabaabaabababba}$ $\quad$ abbabaabaabababba

Applications of Combinatorial Pattern Matching

- **Pattern Matching**

  $x = \texttt{ababb}$ $y = \texttt{abbababbbabababba}$

  text editors; *grep* command-line utility; etc.

- **Indexing**

  $y = \texttt{abbababbbabababba}$ $x = \texttt{ababb}$

  indexing genomes; indexing a collection of documents; etc.

- **Comparison**

  $x = \texttt{abbababbbabababba}$ abbababbbabababba

  $y = \texttt{abbabbbabbbabba}$ abba--bbbabbbabba

  diff command-line utility; aligning genomic sequences; etc.

- **Regularities**

  $x = \texttt{abbabaabaabababba}$ abbabaabaabababba

  data compression; repetitive DNA patterns; etc.

What's "Big"?

What's "Big"?

"*Big data refers to datasets that are too large or complex for traditional data-processing application software to adequately deal with*"—Wikipedia

What's "Big"?

"*Big data refers to datasets that are too large or complex for traditional data-processing application software to adequately deal with*"—Wikipedia

What's "Big Data" for an algorithmicist?

What's "Big"?

"*Big data refers to datasets that are too large or complex for traditional data-processing application software to adequately deal with*"—Wikipedia

What's "Big Data" for an algorithmicist?

Let's think of a well-studied computational problem.

What's "Big"?

"*Big data refers to datasets that are too large or complex for traditional data-processing application software to adequately deal with*"—Wikipedia

What's "Big Data" for an algorithmicist?

Let's think of a well-studied computational problem.

$$13, 5, 17, 11, 3 \xrightarrow{\text{Sorting Algorithm}} 3, 5, 11, 13, 17$$

What's "Big"?

"*Big data refers to datasets that are too large or complex for traditional data-processing application software to adequately deal with*"—Wikipedia

What's "Big Data" for an algorithmicist?

Let's think of a well-studied computational problem.

$$13, 5, 17, 11, 3 \xrightarrow{\text{Sorting Algorithm}} 3, 5, 11, 13, 17$$

When is the input dataset "Big"?

What's "Big"?

"*Big data refers to datasets that are too large or complex for traditional data-processing application software to adequately deal with*"—Wikipedia

What's "Big Data" for an algorithmicist?

Let's think of a well-studied computational problem.

$$13, 5, 17, 11, 3 \xrightarrow{\text{Sorting Algorithm}} 3, 5, 11, 13, 17$$

When is the input dataset "Big"?

When a traditional algorithm (e.g. word-RAM mergesort) fails.

What's "Big"?

"*Big data refers to datasets that are too large or complex for traditional data-processing application software to adequately deal with*"—Wikipedia

What's "Big Data" for an algorithmicist?

Let's think of a well-studied computational problem.

$$13, 5, 17, 11, 3 \xrightarrow{\text{Sorting Algorithm}} 3, 5, 11, 13, 17$$

When is the input dataset "Big"?

When a traditional algorithm (e.g. word-RAM mergesort) fails.

For instance, when the dataset (or data structure) do not fit in RAM.

What's "Big"?

"*Big data refers to datasets that are too large or complex for traditional data-processing application software to adequately deal with*"—Wikipedia

What's "Big Data" for an algorithmicist?

Let's think of a well-studied computational problem.

$$13, 5, 17, 11, 3 \xrightarrow{\text{Sorting Algorithm}} 3, 5, 11, 13, 17$$

When is the input dataset "Big"?

When a traditional algorithm (e.g. word-RAM mergesort) fails.

For instance, when the dataset (or data structure) do not fit in RAM.

So, we define "Big" relative to the available internal memory (RAM).

Minimal Absent Words

Minimal Absent Words

Definition

A word v is absent from word w if v does not occur as a subword of w.

Minimal Absent Words

Definition

A word v is absent from word w if v does not occur as a subword of w.

An absent word is minimal if all its proper subwords occur in w.

Minimal Absent Words

Definition

A word v is absent from word w if v does not occur as a subword of w.

An absent word is minimal if all its proper subwords occur in w.

Example

Let $w = \texttt{abaab}$. The minimal absent words (MAWs) for w are:

$$\mathcal{M}_w = \{\texttt{aaa}, \texttt{aaba}, \texttt{bab}, \texttt{bb}\}$$

Minimal Absent Words

Definition

A word v is absent from word w if v does not occur as a subword of w.

An absent word is minimal if all its proper subwords occur in w.

Example

Let $w = \mathtt{abaab}$. The minimal absent words (MAWs) for w are:

$$\mathcal{M}_w = \{\mathtt{aaa}, \mathtt{aaba}, \mathtt{bab}, \mathtt{bb}\}$$

Theorem

1. *A word of length n has $\Theta(n)$ different MAWs.*
2. *All MAWs of a word of length n can be computed in $\mathcal{O}(n)$ time.*
3. *Any word w of length n is reconstructible in $\mathcal{O}(n)$ time from $\mathcal{M}_w$.*

Applications of Minimal Absent Words

Applications of Minimal Absent Words

Definition

The set $\mathcal{M}_w$ of MAWs of w is called the antidictionary of w.

Applications of Minimal Absent Words

Definition

The set $\mathcal{M}_w$ of MAWs of w is called the antidictionary of w.

Antidictionaries are used in many real-world applications:

Applications of Minimal Absent Words

Definition

The set $\mathcal{M}_w$ of MAWs of w is called the antidictionary of w.

Antidictionaries are used in many real-world applications:

- Data compression (e.g., on-line lossless compression)
- Sequence comparison (e.g., alignment-free sequence comparison)
- Pattern matching (e.g., on-line string matching)
- Bioinformatics (e.g., pathogen-specific signature)

Applications of Minimal Absent Words

Definition

The set $\mathcal{M}_w$ of MAWs of w is called the antidictionary of w.

Antidictionaries are used in many real-world applications:

- Data compression (e.g., on-line lossless compression)
- Sequence comparison (e.g., alignment-free sequence comparison)
- Pattern matching (e.g., on-line string matching)
- Bioinformatics (e.g., pathogen-specific signature)

Most of the times, a reduced antidictionary $\mathcal{M}^\ell$ is considered:

Applications of Minimal Absent Words

Definition

The set $\mathcal{M}_w$ of MAWs of w is called the antidictionary of w.

Antidictionaries are used in many real-world applications:

- Data compression (e.g., on-line lossless compression)
- Sequence comparison (e.g., alignment-free sequence comparison)
- Pattern matching (e.g., on-line string matching)
- Bioinformatics (e.g., pathogen-specific signature)

Most of the times, a reduced antidictionary $\mathcal{M}^{\ell}$ is considered:

- Consists of MAWs whose length is bounded by some threshold ℓ

Applications of Minimal Absent Words

Definition

The set $\mathcal{M}_w$ of MAWs of w is called the antidictionary of w.

Antidictionaries are used in many real-world applications:

- Data compression (e.g., on-line lossless compression)
- Sequence comparison (e.g., alignment-free sequence comparison)
- Pattern matching (e.g., on-line string matching)
- Bioinformatics (e.g., pathogen-specific signature)

Most of the times, a reduced antidictionary $\mathcal{M}^\ell$ is considered:

- Consists of MAWs whose length is bounded by some threshold ℓ
- Max len of a MAW is $2 +$ max len r of a repeated subword

Applications of Minimal Absent Words

Definition

The set $\mathcal{M}_w$ of MAWs of w is called the antidictionary of w.

Antidictionaries are used in many real-world applications:

- Data compression (e.g., on-line lossless compression)
- Sequence comparison (e.g., alignment-free sequence comparison)
- Pattern matching (e.g., on-line string matching)
- Bioinformatics (e.g., pathogen-specific signature)

Most of the times, a reduced antidictionary $\mathcal{M}^\ell$ is considered:

- Consists of MAWs whose length is bounded by some threshold ℓ
- Max len of a MAW is $2 +$ max len r of a repeated subword
- For a random word of length n this is $r = \Theta(\log n)$

Our Motivation

Our Motivation

The most efficient algorithm for computing MAWs of a word of length n:

Our Motivation

The most efficient algorithm for computing MAWs of a word of length n:

- $\mathcal{O}(n)$ time and space using *suffix array*, a global data structure [Barton, Héliou, Mouchard, P, 2014]

Our Motivation

The most efficient algorithm for computing MAWs of a word of length n:

- $\mathcal{O}(n)$ time and space using *suffix array*, a global data structure [Barton, Héliou, Mouchard, P, 2014]
- Uses $20n$ words of space

Our Motivation

The most efficient algorithm for computing MAWs of a word of length n:

- $\mathcal{O}(n)$ time and space using *suffix array*, a global data structure [Barton, Héliou, Mouchard, P, 2014]
- Uses $20n$ words of space
- For the human genome ($n \approx 3 \times 10^9$), we need 60 GB of RAM

Our Motivation

The most efficient algorithm for computing MAWs of a word of length n:

- $\mathcal{O}(n)$ time and space using *suffix array*, a global data structure [Barton, Héliou, Mouchard, P, 2014]
- Uses $20n$ words of space
- For the human genome ($n \approx 3 \times 10^9$), we need 60 GB of RAM

Example

In the human genome, for $\ell = 12$, $||\mathcal{M}^{12}|| \approx 10^6 \ll n$.

Our Motivation

The most efficient algorithm for computing MAWs of a word of length n:

- $\mathcal{O}(n)$ time and space using *suffix array*, a global data structure [Barton, Héliou, Mouchard, P, 2014]
- Uses $20n$ words of space
- For the human genome ($n \approx 3 \times 10^9$), we need 60 GB of RAM

Example

In the human genome, for $\ell = 12$, $||\mathcal{M}^{12}|| \approx 10^6 \ll n$.

Problem

Can we compute $\mathcal{M}^\ell$ in output-sensitive space?

Our Problem

Our Problem

Intuition:

Our Problem

Intuition:

- Divide input into k words, each of which, alone, fits in RAM:

Our Problem

Intuition:

- Divide input into k words, each of which, alone, fits in RAM:

$$y = y_1 \# y_2 \# \cdots \# y_k$$

Our Problem

Intuition:

- Divide input into k words, each of which, alone, fits in RAM:

$$y = y_1 \# y_2 \# \cdots \# y_k$$

- Compute $\mathcal{M}_y^\ell$ incrementally from the MAWs of this concatenation

Our Problem

Intuition:

- Divide input into k words, each of which, alone, fits in RAM:

$$y = y_1 \# y_2 \# \cdots \# y_k$$

- Compute $\mathcal{M}_y^\ell$ incrementally from the MAWs of this concatenation

Formally, we state the following:

Our Problem

Intuition:

- Divide input into k words, each of which, alone, fits in RAM:

$$y = y_1 \# y_2 \# \cdots \# y_k$$

- Compute $\mathcal{M}_y^\ell$ incrementally from the MAWs of this concatenation

Formally, we state the following:

Problem

Given k words $y_1, y_2, \ldots, y_k$ and $\ell > 0$, compute the set $\mathcal{M}^\ell_{y_1\#\ldots\#y_k}$ of minimal absent words of length $\leq \ell$ of $y_1\#y_2\#\ldots\#y_k$.

Our Problem

Intuition:

- Divide input into k words, each of which, alone, fits in RAM:

$$y = y_1 \# y_2 \# \cdots \# y_k$$

- Compute $\mathcal{M}_y^\ell$ incrementally from the MAWs of this concatenation

Formally, we state the following:

Problem

Given k words $y_1, y_2, \ldots, y_k$ and $\ell > 0$, compute the set $\mathcal{M}^\ell_{y_1\#\ldots\#y_k}$ of minimal absent words of length $\leq \ell$ of $y_1\#y_2\#\ldots\#y_k$.

e.g. k chromosomes of a genome or a collection of k documents

The Sketch

The Sketch

We have k iterations computing: $\mathcal{M}^{\ell}_{y_1}, \mathcal{M}^{\ell}_{y_1\#y_2}, \ldots, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$.

The Sketch

We have k iterations computing: $\mathcal{M}^{\ell}_{y_1}, \mathcal{M}^{\ell}_{y_1\#y_2}, \ldots, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$.

At the Nth iteration we consider: $y_1\#y_N, y_2\#y_N, \ldots, y_{N-1}\#y_N$.

The Sketch

We have k iterations computing: $\mathcal{M}^{\ell}_{y_1}, \mathcal{M}^{\ell}_{y_1\#y_2}, \ldots, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$.

At the Nth iteration we consider: $y_1\#y_N, y_2\#y_N, \ldots, y_{N-1}\#y_N$.

We are allowed to store $y_i\#y_N$

The Sketch

We have k iterations computing: $\mathcal{M}^{\ell}_{y_1}, \mathcal{M}^{\ell}_{y_1\#y_2}, \ldots, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$.

At the Nth iteration we consider: $y_1\#y_N, y_2\#y_N, \ldots, y_{N-1}\#y_N$.

We are allowed to store $y_i\#y_N$ but not the whole $y_1\#\ldots\#y_N$!

The Sketch

We have k iterations computing: $\mathcal{M}^{\ell}_{y_1}, \mathcal{M}^{\ell}_{y_1\#y_2}, \ldots, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$.

At the Nth iteration we consider: $y_1\#y_N, y_2\#y_N, \ldots, y_{N-1}\#y_N$.

We are allowed to store $y_i\#y_N$ but not the whole $y_1\#\ldots\#y_N$!

For ease of comprehension let us assume: $y = y_1\#y_2$.

The Sketch

We have k iterations computing: $\mathcal{M}^{\ell}_{y_1}, \mathcal{M}^{\ell}_{y_1\#y_2}, \ldots, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$.

At the Nth iteration we consider: $y_1\#y_N, y_2\#y_N, \ldots, y_{N-1}\#y_N$.

We are allowed to store $y_i\#y_N$ but not the whole $y_1\#\ldots\#y_N$!

For ease of comprehension let us assume: $y = y_1\#y_2$.

Let $x \in \mathcal{M}^{\ell}_{y}$. We separate two cases:

The Sketch

We have k iterations computing: $\mathcal{M}^{\ell}_{y_1}, \mathcal{M}^{\ell}_{y_1\#y_2}, \ldots, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$.

At the Nth iteration we consider: $y_1\#y_N, y_2\#y_N, \ldots, y_{N-1}\#y_N$.

We are allowed to store $y_i\#y_N$ but not the whole $y_1\#\ldots\#y_N$!

For ease of comprehension let us assume: $y = y_1\#y_2$.

Let $x \in \mathcal{M}^{\ell}_{y}$. We separate two cases:

1. x belongs to $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 1)

The Sketch

We have k iterations computing: $\mathcal{M}^{\ell}_{y_1}, \mathcal{M}^{\ell}_{y_1 \# y_2}, \ldots, \mathcal{M}^{\ell}_{y_1 \# \ldots \# y_k}$.

At the Nth iteration we consider: $y_1 \# y_N, y_2 \# y_N, \ldots, y_{N-1} \# y_N$.

We are allowed to store $y_i \# y_N$ but not the whole $y_1 \# \ldots \# y_N$!

For ease of comprehension let us assume: $y = y_1 \# y_2$.

Let $x \in \mathcal{M}^{\ell}_{y}$. We separate two cases:

1. x belongs to $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 1)
2. x does not belong to $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 2)

Combinatorial Results: x is in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 1)

Combinatorial Results: x is in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 1)

Lemma (Case 1)

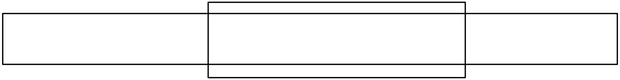
$x \in \mathcal{M}^{\ell}_{y_1}$ belongs to $\mathcal{M}^{\ell}_{y}$ iff x is a superword of a word in $\mathcal{M}^{\ell}_{y_2}$.

Combinatorial Results: x is in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 1)

Lemma (Case 1)

$x \in \mathcal{M}^{\ell}_{y_1}$ belongs to $\mathcal{M}^{\ell}_{y}$ iff x is a superword of a word in $\mathcal{M}^{\ell}_{y_2}$.

x :

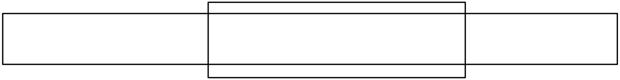


Combinatorial Results: x is in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 1)

Lemma (Case 1)

$x \in \mathcal{M}^{\ell}_{y_1}$ *belongs to* $\mathcal{M}^{\ell}_{y}$ *iff* x *is a superword of a word in* $\mathcal{M}^{\ell}_{y_2}$.

x :



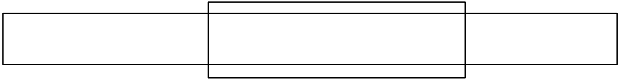
Example

Let $y_1 =$ abaab, $y_2 =$ bbaaab and $\ell = 5$. $y =$ abaab#bbaaab.
$\mathcal{M}^{\ell}_{y_1} = \{$bb,aaa,bab,aaba$\}$ and
$\mathcal{M}^{\ell}_{y_2} = \{$bbb,aaaa,baab,aba,bab,abb$\}$.

Lemma (Case 1)

$x \in \mathcal{M}^{\ell}_{y_1}$ belongs to $\mathcal{M}^{\ell}_{y}$ iff x is a superword of a word in $\mathcal{M}^{\ell}_{y_2}$.

x :



Example

Let $y_1 =$ abaab, $y_2 =$ bbaaab and $\ell = 5$. $y =$ abaab#bbaaab.
$\mathcal{M}^{\ell}_{y_1} = \{$bb,aaa,bab,aaba$\}$ and
$\mathcal{M}^{\ell}_{y_2} = \{$bbb,aaaa,baab,aba,bab,abb$\}$.

$$\mathcal{M}^{\ell}_{y} \cap (\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}) = \{\texttt{aaaa,bab,aaba,abb,bbb}\}.$$

Combinatorial Results: x is not in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 2)

Let $\mathcal{R}^{\ell}_{y_1}$ be the set obtained from $\mathcal{M}^{\ell}_{y_1}$ after removing Case 1 MAWs.

Combinatorial Results: x is not in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 2)

Let $\mathcal{R}^{\ell}_{y_1}$ be the set obtained from $\mathcal{M}^{\ell}_{y_1}$ after removing Case 1 MAWs.

Let $\mathcal{R}^{\ell}_{y_2}$ be the set obtained from $\mathcal{M}^{\ell}_{y_2}$ after removing Case 1 MAWs.

Combinatorial Results: x is not in $\mathcal{M}_{y_1}^{\ell} \cup \mathcal{M}_{y_2}^{\ell}$ (Case 2)

Let $\mathcal{R}_{y_1}^{\ell}$ be the set obtained from $\mathcal{M}_{y_1}^{\ell}$ after removing Case 1 MAWs.

Let $\mathcal{R}_{y_2}^{\ell}$ be the set obtained from $\mathcal{M}_{y_2}^{\ell}$ after removing Case 1 MAWs.

Lemma (Case 2)

x has a prefix x_2 in $\mathcal{R}_{y_2}^{\ell}$ and a suffix x_1 in $\mathcal{R}_{y_1}^{\ell}$.

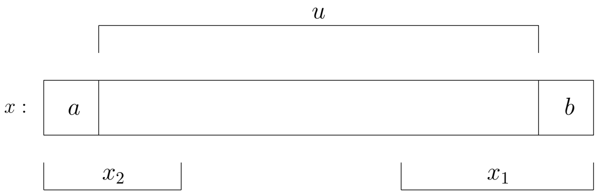
Combinatorial Results: x is not in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 2)

Let $\mathcal{R}^{\ell}_{y_1}$ be the set obtained from $\mathcal{M}^{\ell}_{y_1}$ after removing Case 1 MAWs.

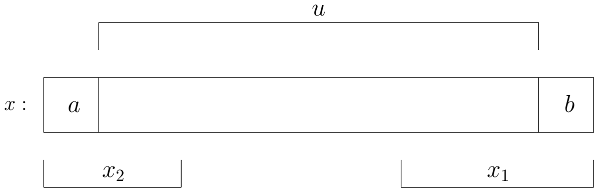
Let $\mathcal{R}^{\ell}_{y_2}$ be the set obtained from $\mathcal{M}^{\ell}_{y_2}$ after removing Case 1 MAWs.

Lemma (Case 2)

x has a prefix x_2 in $\mathcal{R}^{\ell}_{y_2}$ and a suffix x_1 in $\mathcal{R}^{\ell}_{y_1}$.



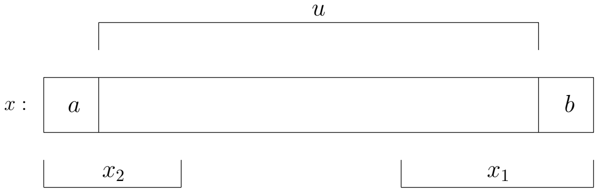
Combinatorial Results: x is not in $\mathcal{M}_{y_1}^{\ell} \cup \mathcal{M}_{y_2}^{\ell}$ (Case 2)



Example

Let $y_1 =$ abaab, $y_2 =$ bbaaab and $\ell = 5$. $y =$ abaab#bbaaab.
$\mathcal{R}_{y_1}^{\ell} = \{bb,aaa\}$ and $\mathcal{R}_{y_2}^{\ell} = \{$baab,aba$\}$.

Combinatorial Results: x is not in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 2)

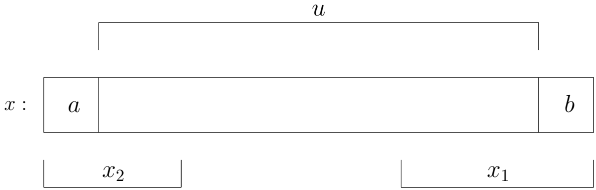


Example

Let $y_1 =$ abaab, $y_2 =$ bbaaab and $\ell = 5$. $y =$ abaab#bbaaab.
$\mathcal{R}^{\ell}_{y_1} = \{bb,aaa\}$ and $\mathcal{R}^{\ell}_{y_2} = \{$baab,aba$\}$.

Consider $x =$ abaaa $\in \mathcal{M}^{\ell}_{y} \setminus (\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2})$ (Case 2 MAW).

Combinatorial Results: x is not in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 2)



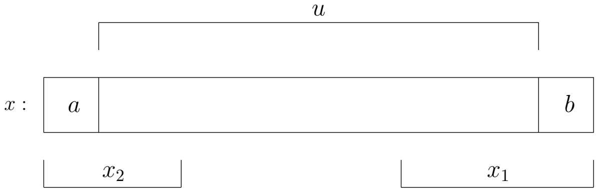
Example

Let $y_1 = \texttt{abaab}$, $y_2 = \texttt{bbaaab}$ and $\ell = 5$. $y = \texttt{abaab\#bbaaab}$.
$\mathcal{R}^{\ell}_{y_1} = \{\texttt{bb},\texttt{aaa}\}$ and $\mathcal{R}^{\ell}_{y_2} = \{\texttt{baab},\texttt{aba}\}$.

Consider $x = \texttt{abaaa} \in \mathcal{M}^{\ell}_{y} \setminus (\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2})$ (Case 2 MAW).

There is an $x_2 \in \mathcal{R}^{\ell}_{y_2}$ that is a prefix of abaa and this is aba.

Combinatorial Results: x is not in $\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2}$ (Case 2)



Example

Let $y_1 = $ abaab, $y_2 = $ bbaaab and $\ell = 5$. $y = $ abaab#bbaaab.
$\mathcal{R}^{\ell}_{y_1} = \{bb,aaa\}$ and $\mathcal{R}^{\ell}_{y_2} = \{$baab,aba$\}$.

Consider $x = $ abaaa $\in \mathcal{M}^{\ell}_{y} \setminus (\mathcal{M}^{\ell}_{y_1} \cup \mathcal{M}^{\ell}_{y_2})$ (Case 2 MAW).

There is an $x_2 \in \mathcal{R}^{\ell}_{y_2}$ that is a prefix of abaa and this is aba.
There is an $x_1 \in \mathcal{R}^{\ell}_{y_1}$ that is a suffix of abaaa and this is aaa.

Our Result

Our Result

Let $n = |y_1\# \dots \# y_k|$.

Our Result

Let $n = |y_1 \# \ldots \# y_k|$.

Let $\textsc{MaxIn}$ be the length of the longest word in $\{y_1, \ldots, y_k\}$.

Our Result

Let $n = |y_1\# \ldots \# y_k|$.

Let $\textsc{MaxIn}$ be the length of the longest word in $\{y_1, \ldots, y_k\}$.

Let $\textsc{MaxOut} = \max\{|| \mathcal{M}^{\ell}_{y_1\#\ldots\#y_N} || : N \in [1, k]\}$.

Our Result

Let $n = |y_1\# \ldots \# y_k|$.

Let $\textsc{MaxIn}$ be the length of the longest word in $\{y_1, \ldots, y_k\}$.

Let $\textsc{MaxOut} = \max\{|| \mathcal{M}^{\ell}_{y_1\#\ldots\#y_N} || : N \in [1, k]\}$.

We use standard string processing data structures to arrive at:

Our Result

Let $n = |y_1\#\ldots\#y_k|$.

Let $\textsc{MaxIn}$ be the length of the longest word in $\{y_1, \ldots, y_k\}$.

Let $\textsc{MaxOut} = \max\{||\, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_N} \,|| : N \in [1, k]\}$.

We use standard string processing data structures to arrive at:

Theorem

We compute $\mathcal{M}^{\ell}_{y_1}, \ldots, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_k}$ *in* $\mathcal{O}(kn + \sum_{N=1}^{k} ||\, \mathcal{M}^{\ell}_{y_1\#\ldots\#y_N} \,||)$ *total time using* $\mathcal{O}(\textsc{MaxIn} + \textsc{MaxOut})$ *space.*

Experiments on Human Genome

Experiments on Human Genome

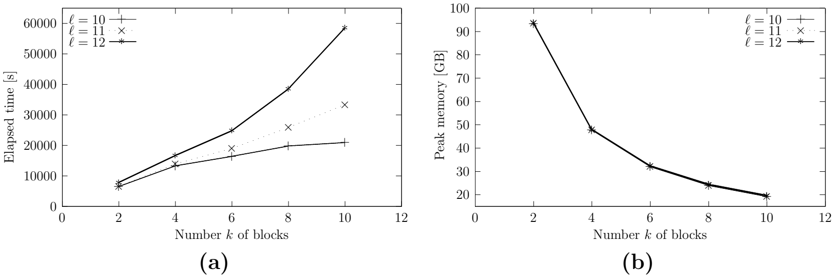


Figure: Time-space tradeoff

Final Remarks

Final Remarks

1. Space-efficient algorithms designed for global data structures can be directly applied to the k blocks in our technique

Final Remarks

1. Space-efficient algorithms designed for global data structures can be directly applied to the k blocks in our technique
2. Our technique could serve as a basis for parallelising the construction: several blocks are processed concurrently

Final Remarks

1. Space-efficient algorithms designed for global data structures can be directly applied to the k blocks in our technique
2. Our technique could serve as a basis for parallelising the construction: several blocks are processed concurrently
3. There is a connection between MAWs and other word regularities. Our technique could potentially be applied to computing these

Final Remarks

1. Space-efficient algorithms designed for global data structures can be directly applied to the k blocks in our technique
2. Our technique could serve as a basis for parallelising the construction: several blocks are processed concurrently
3. There is a connection between MAWs and other word regularities. Our technique could potentially be applied to computing these

Accepted to DCC 2019: `arxiv.org/abs/1902.04785`

Final Remarks

1. Space-efficient algorithms designed for global data structures can be directly applied to the k blocks in our technique
2. Our technique could serve as a basis for parallelising the construction: several blocks are processed concurrently
3. There is a connection between MAWs and other word regularities. Our technique could potentially be applied to computing these

Accepted to DCC 2019: `arxiv.org/abs/1902.04785`

Thanks!