Minimal absent words in a sliding window & applications to on-line pattern matching

Maxime Crochemore\textsuperscript{1,2}, Alice Héliou\textsuperscript{3}, Gregory Kucherov\textsuperscript{2}, Laurent Mouchard\textsuperscript{4}, Solon Pissis\textsuperscript{1}, Yann Ramusat\textsuperscript{5}

\textsuperscript{1} Department of Informatics, King’s College London, London, UK
\textsuperscript{2} CNRS & Université Paris-Est
\textsuperscript{3} LIX, Ecole Polytechnique, CNRS, INRIA, Université Paris-Saclay
\textsuperscript{4} University of Rouen, LITIS EA 4108, TIBS, Rouen
\textsuperscript{5} DI ENS, CNRS, PSL Research University & INRIA Paris

11 septembre 2017 – FCT Bordeaux
1. Minimal absent words
   - Definition
   - Applications
   - Computation

2. Minimal absent words over a sliding window
Definition: Minimal Absent Word

A minimal absent word of a sequence is an absent word whose proper factors (longest prefix, and longest suffix) all occur in the sequence.

An upper bound on the number of minimal absent words is $O(\sigma n)$.

Crochemore et al. 1998, Mignosi et al. 2002

\[ S = \text{ACACAAAGC} \]
Definition: Minimal Absent Word

A minimal absent word of a sequence is an absent word whose proper factors (longest prefix, and longest suffix) all occur in the sequence.

An upper bound on the number of minimal absent words is $O(\sigma n)$.

Crochemore et al. 1998, Mignosi et al. 2002

$S=\text{ACACACAGC}$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Definition: Minimal Absent Word

A minimal absent word of a sequence is an absent word whose proper factors (longest prefix, and longest suffix) all occur in the sequence.

An upper bound on the number of minimal absent words is $O(\sigma n)$.

Crochemore et al. 1998, Mignosi et al. 2002

$S=A C A C A A G C$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Definition: Minimal Absent Word

A minimal absent word of a sequence is an absent word whose proper factors (longest prefix, and longest suffix) all occur in the sequence.

An upper bound on the number of minimal absent words is $O(\sigma n)$.

Crochemore et al. 1998, Mignosi et al. 2002

$S=A C A C A A G C$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Definition: Minimal Absent Word

A minimal absent word of a sequence is an absent word whose proper factors (longest prefix, and longest suffix) all occur in the sequence.

An upper bound on the number of minimal absent words is $O(\sigma n)$.

Crochemore et al. 1998, Mignosi et al. 2002

$S = \text{A C A C A A G C}$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Definition : Minimal Absent Word

A minimal absent word of a sequence is an absent word whose proper factors (longest prefix, and longest suffix) all occur in the sequence.

An upper bound on the number of minimal absent words is \( O(\sigma n) \).

Crochemore et al. 1998, Mignosi et al. 2002

\[ S = A C A C A G C \]

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Definition: Minimal Absent Word

A minimal absent word of a sequence is an absent word whose proper factors (longest prefix, and longest suffix) all occur in the sequence.

An upper bound on the number of minimal absent words is $O(\sigma n)$.

Crochemore et al. 1998, Mignosi et al. 2002

$S = A C A C A A G C$
AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Definition: Minimal Absent Word

A minimal absent word of a sequence is an absent word whose proper factors (longest prefix, and longest suffix) all occur in the sequence.

An upper bound on the number of minimal absent words is $O(\sigma n)$.

Crochemore et al. 1998, Mignosi et al. 2002

$S=\text{ACACAGC}$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Applications

Biology

- 3 sequences (TTTCGCCCGACT, TACGCCCTATCG, CCTACGCGCAAA), found in Ebola genomes as coding for proteins are absent from the Human genome.
Applications

**Biology**
- 3 sequences (TTTCGCCCGACT, TACGCCCTATCG, CCTACGCGCAAA), found in Ebola genomes as coding for proteins are absent from the Human genome.

**BioInformatics**
- Metric based on minimal absent words → Phylogeny (Chairungsee et al., 2012, Crochemore et al, 2016).
Applications

**Biology**
- 3 sequences (TTTCGCCCGACT, TACGCCCTATCG, CCTACGCGCAAA), found in Ebola genomes as coding for proteins are absent from the Human genome.

**BioInformatics**
- Metric based on minimal absent words → Phylogeny (Chairungsee et al., 2012, Crochemore et al, 2016).

**Computer Science**
Definition: Maximal repeated pair

A maximal repeated pair in a $S$ is a triple $(i, j, w)$ such that:

- $w$ occurs in $S$ at positions $i$ and $j$
- $S[i - 1] \neq S[j - 1]$
- $S[i + |w|] \neq S[j + |w|]$
Definition : Maximal repeated pair

A maximal repeated pair in a $S$ is a triple $(i, j, w)$ such that :

- $w$ occurs in $S$ at positions $i$ and $j$
- $S[i - 1] \neq S[j - 1]$
- $S[i + |w|] \neq S[j + |w|]$

Lemma

If $awb$ is a minimal absent word of $S$, then there exist positions $i$ and $j$ such that $(i, j, w)$ is a maximal repeated pair of $S$. 
**Definition : Maximal repeated pair**

A maximal repeated pair in a $S$ is a triple $(i, j, w)$ such that:

- $w$ occurs in $S$ at positions $i$ and $j$
- $S[i - 1] \neq S[j - 1]$
- $S[i + |w|] \neq S[j + |w|]$

**Lemma**

If $awb$ is a minimal absent word of $S$, then there exist positions $i$ and $j$ such that $(i, j, w)$ is a maximal repeated pair of $S$.

**Sequence $S$**

A a minimal absent word of $S
Definition : Maximal repeated pair

A maximal repeated pair in a $S$ is a triple $(i, j, w)$ such that:
- $w$ occurs in $S$ at positions $i$ and $j$
- $S[i - 1] \neq S[j - 1]$
- $S[i + |w|] \neq S[j + |w|]$

Lemma

If $awb$ is a minimal absent word of $S$, then there exist positions $i$ and $j$ such that $(i, j, w)$ is a maximal repeated pair of $S$. 

Sequence $S$

$\text{A} \text{ a minimal absent word of } S$

longest prefix of $A$
Definition: Maximal repeated pair

A maximal repeated pair in a sequence $S$ is a triple $(i, j, w)$ such that:

- $w$ occurs in $S$ at positions $i$ and $j$
- $S[i - 1] \neq S[j - 1]$
- $S[i + |w|] \neq S[j + |w|]$

Lemma

If $awb$ is a minimal absent word of $S$, then there exist positions $i$ and $j$ such that $(i, j, w)$ is a maximal repeated pair of $S$.

Sequence $S$

```
A a minimal absent word of $S$
longest suffix of $A$
```
Definition: Maximal repeated pair

A maximal repeated pair in a $S$ is a triple $(i, j, w)$ such that:
- $w$ occurs in $S$ at positions $i$ and $j$
- $S[i - 1] \neq S[j - 1]$
- $S[i + |w|] \neq S[j + |w|]$

Lemma

If $awb$ is a minimal absent word of $S$, then there exist positions $i$ and $j$ such that $(i, j, w)$ is a maximal repeated pair of $S$. 
Suffix tree of $S = \text{ACACAAAGC\#}$
Suffix tree of $S = \text{ACAACAAGC}\#$

0 1 2 3 4 5 6 7 8

A(0,0) -> C(1,1) -> GC# (6,8)

AGC# (5,8) -> CA(1,2) -> CA# (6,8)

4

AGC# (5,8) -> CAAGC# (3,8)

5

AGC# (5,8) -> 0

1

3

A(2,2) -> # (8,8)

7

GC# (6,8)
Suffix tree of $S = \textcolor{blue}{A}C\textcolor{blue}{A}\textcolor{blue}{C}A\textcolor{blue}{A}\textcolor{blue}{G}\textcolor{blue}{C}\#$

Minimal absent words

Computation
Suffix tree of $S = \text{ACACAGC}\#$

Minimal absent words

Computation
Suffix tree of $S = \text{ACA} \text{CAAGC}\#$

$$\begin{array}{c}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\text{A}(0,0) & \text{C}(1,1) & \text{A}(2,2) & \text{G} & \text{C} & 6
\end{array}$$
Suffix tree of $S = \text{ACACAGC}$

```
0 1 2 3 4 5 6 7 8
```

Minimal absent words

Computation
Suffix tree of $S = A C A C A A G C \#$
Suffix tree of $S = A C A C A A G C \#$

$\begin{array}{ccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}$

$A(0,0)$

$C(1,1)$

$A(2,2)$

$GC\# (6,8)$
Suffix tree of $S = A C A C A A G C \#$
Suffix tree of $S = \text{A C A C A A G C} \#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Suffix tree of $S = \text{ACAACAAGC}\#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Suffix tree of \( S = A C A C A A G C \# \)
AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

\[
\begin{array}{cccccccccc}
\text{AGC}\#(5,8) & \text{CA}(1,2) & \text{GC}\#(6,8) & & & & & & \text{GC}\#(6,8) & 6 \\
\text{4} & \text{5} & \text{7} & \text{2} & \text{0} & \text{3} & \text{1} & & \\
\end{array}
\]

\[\{A,C\} \varnothing A\]
Minimal absent words

Computation

Suffix tree of $S = \text{A C A C A A G C}\#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

$0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8$

$A(0,0)$

$C(1,1)$

$\text{AGC}\# (6,8)$

$\text{GC}\# (6,8)$

$\text{A}(2,2)$

$\{\text{A,C}\} \varnothing \text{ A}$

$\{\text{A,G}\} \varnothing \text{ C}$
Suffix tree of $S = \texttt{A}\texttt{C}\texttt{A}\texttt{C}\texttt{A}\texttt{A}\texttt{G}\texttt{C}\#$

$\text{AAA,AAC,CACAC,CAG,CC,CG,GA,GCA,GG}$
Suffix tree of $S = A C A C A A G C \#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

{A, C} ⊥ A
{A, G} ⊥ C
A ⊥ G
Suffix tree of $S = ACA C A A G C \#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Minimal absent words

Computation

Suffix tree of $S = \text{AACGCAACAAGC#}$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

A(1,2)

C(2,2)

A(1,1)

C(1,1)

GC# (6,8)

A(0,0)

AGC# (5,8)

CA# (6,8)

GA# (8,8)

C(2,2)

C(1,1)

A(0,0)

AGC# (5,8)

CA# (6,8)

GA# (8,8)

C(2,2)

C(1,1)

A(0,0)

AGC# (5,8)

CA# (6,8)

GA# (8,8)
Minimal absent words

Computation

Suffix tree of \( S = \text{A C A C A A G C \#} \)

\[ \text{AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG} \]
Suffix tree of $S = A C A C A A G C \#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Minimal absent words

Computation

Suffix tree of $S = A C A C A A G C \#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

$0 1 2 3 4 5 6 7 8$

A(0,0) → C(1,1) → GC# (6,8) → 6

A(2,2) → CG# (6,8) → 7

AGC# (5,8) → CA(1,2) → 5

AGC# (5,8) → CA(1,2) → 2

AGC# (5,8) → CA(1,2) → 0

AGC# (5,8) → CA(1,2) → 3

AGC# (5,8) → CA(1,2) → 1

C ACA A
Minimally absent words

\[ S = A C A C A A G C \#
\]

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

Suffix tree of \( S = A C A C A A G C \#
\)

\[ AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG \]

C ACA A
∅ ACA C
Suffix tree of $S = \text{ACACAAGC}$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

$A(0,0)$
$C(1,1)$
$A(2,2)$
$\text{GC} \# (6,8)$
$\emptyset \text{ ACA A}$
$C \text{ ACA C}$
Suffix tree of $S = \text{A C A C A A G C} #$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Suffix tree of $S = \text{ACAACAAGC}\#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG
Suffix tree of $S = A C A C A A G C \#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

0 1 2 3 4 5 6 7 8

A(0,0) GC# (6,8)
C(1,1) # (8,8) A(2,2)

AGC# (5,8) CA(1,2) GC# (6,8)
CAAGC# (3,8) A(0,0)

4 5 7

AGC# (5,8) CAAGC# (3,8)
AGC# (8,8) CAAGC# (3,8)

2 0 3 1
Minimal absent words

Computation

Suffix tree of $S = \text{AACACAAGC}\#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

A CA A
Minimal absent words

Suffix tree of $S = \text{AACACAAGC}\#$

AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]
The suffix tree of $S = A C A C A A G C \#$

$AAA, AAC, CACAC, CAG, CC, CG, GA, GCA, GG$
1 Minimal absent words

2 Minimal absent words over a sliding window
   • Problem definition
   • Ukkonen construction algorithm of the suffix tree
   • The suffix tree for a sliding window
   • Algorithm for maws in a sliding window
Minimal absent words on a sliding window

- Sequence $S$ of size $n$, over a constant size alphabet,
- Sliding window of size $m$ on $S: S[i \ldots i + m - 1]$.

For all word $x$ we denote by $M(x)$ its set of minimal absent words.
Minimal absent words on a sliding window

- Sequence $S$ of size $n$, over a constant size alphabet,
- Sliding window of size $m$ on $S : S[i \ldots i + m - 1]$.

For all word $x$ we denote by $M(x)$ its set of minimal absent words.

$$M(S[i \ldots i + m - 1])$$
Minimal absent words on a sliding window

Lemma

\[ \sum_{i=0}^{n-m} |M(y[i \ldots i + m - 1])| \] is upper bounded by \( \mathcal{O}(nm) \).
Minimal absent words on a sliding window

Lemma

\[
\sum_{i=0}^{n-m} |M(y[i \ldots i + m - 1])| \text{ is upper bounded by } O(nm).
\]

→ We can’t output the set if minimal absent words for each factor of size \( m \) in time \( O(n) \).
Minimal absent words on a sliding window

Lemma

\[ \sum_{i=0}^{n-m} |M(y[i \ldots i + m - 1])| \text{ is upper bounded by } O(nm). \]

→ We can’t output the set if minimal absent words for each factor of size \( m \) in time \( O(n) \).

Theorem

The upper bound of

\[ \sum_{i=0}^{n-m-1} |M(y[i \ldots i + m - 1]) \triangle M(y[i+1 \ldots i + m])| \text{ is } O(n). \]
Minimal absent words on a sliding window

Lemma

\[
\sum_{i=0}^{n-m} |M(y[i \ldots i + m - 1])| \text{ is upper bounded by } O(nm).
\]

→ We can’t output the set if minimal absent words for each factor of size \(m\) in time \(O(n)\).

Theorem

The upper bound of

\[
\sum_{i=0}^{n-m-1} |M(y[i \ldots i + m - 1]) \triangle M(y[i + 1 \ldots i + m])| \text{ is } O(n).
\]

→ We need a dynamic structure to go from one set to another efficiently.
Dynamic construction of the suffix tree

- From left to right by Weiner 1973,
- From right to left by McCreight in 1976,
- Ukkonen algorithm in 1995 provides a more intuitive algorithm.
Ukkonen construction algorithm of the suffix tree

\[ S = \emptyset \]
Ukkonen construction algorithm of the suffix tree

\[ S = \emptyset \]

\[
\begin{array}{c}
\downarrow \\
S = A
\end{array}
\]

\[
\begin{array}{c}
\downarrow \\
S = \emptyset
\end{array}
\]

\[
\begin{array}{c}
A(0,0) \\
\downarrow \\
0
\end{array}
\]
Ukkonen construction algorithm of the suffix tree

Minimal absent words over a sliding window
Ukkonen construction algorithm of the suffix tree

\[
S = \emptyset
\]

\[
S = A
\]

\[
S = A\ C
\]

\[
S = A\ C\ A
\]
Minimal absent words over a sliding window

Ukkonen construction algorithm of the suffix tree
Minimal absent words over a sliding window

Ukkonen construction algorithm of the suffix tree

0 1 2
S = ACA

0 1 2 3
S = ACAC

0 1 2 3 4
S = ACACA

CA(1,2)
AC(2,3)
CAC(1,3)
Minimal absent words over a sliding window

Ukkonen construction algorithm of the suffix tree

\[
S = \text{ACACA}
\]

\[
S = \{1, 2, 3, 4, 5\}
\]
Minimal absent words over a sliding window

Ukkonen construction algorithm of the suffix tree

\[ S = \text{A C A C A A} \]

\[ \begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 & \quad 5 \\
A & \quad C & \quad A & \quad C & \quad A & \quad A
\end{align*} \]
Minimal absent words over a sliding window

Ukkonen construction algorithm of the suffix tree

$S = \text{ACACAA}$

$0 1 2 3 4 5$

$S = \text{ACACAA}$
Minimal absent words over a sliding window

Ukkonen construction algorithm of the suffix tree

S = A C A C A A

0 1 2 3 4 5

CA(1,2)

A(0,0)

A(5,5)

CAA(3,5)

CAA(3,5)

CAA(3,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)

CA(1,2)

A(5,5)
Minimal absent words over a sliding window

Ukkonen construction algorithm of the suffix tree

\[ S = \text{ACACAAGC} \]

\[
0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7
\]

Alice Héliou
The suffix tree for a sliding window, Senft 2005

- Remove the leftmost letter,
- Update edge labels.
Minimal absent words over a sliding window

The suffix tree for a sliding window

\( S = \text{ACAACAAGC} \)

\[ S = A \cdot C \cdot A \cdot C \cdot A \cdot A \cdot G \cdot C \]

\[ S = A(0,0) \cdot C(1,1) \cdot A(2,2) \cdot C(3,3) \cdot A(4,4) \cdot A(5,5) \cdot G(6,6) \cdot C(7,7) \]
We have adapted Senft algorithm to compute minimal absent words.
We have adapted Senft algorithm to compute minimal absent words.

- Add on the tree the information of the BWT (the set of letters that precede each factor),
We have adapted Senft algorithm to compute minimal absent words.

- Add on the tree the information of the BWT (the set of letters that precede each factor),
- Add the set of minimal absent words
Minimal absent words over a sliding window

We have adapted Senft algorithm to compute minimal absent words.

- Add on the tree the information of the BWT (the set of letters that precede each factor),
- Add the set of minimal absent words

The mapping $f$ is an injection

$$f : M(z) \rightarrow \Sigma(z) \times V(z) \text{ define by } f(aub) = (a, v_{ub}),$$
where $a \in \Sigma$ and $v_{ub}$ is the node corresponding to $ub$. 
Minimal absent words over a sliding window

Algorithm for maws in a sliding window

\[ z_0 = A C A C A A G C \]

\[ M(z_0) = \{ AAA, GA, AAC, CACAC, CAG, CC, GCA, CG, GG \} \]
Algorithm for maws in a sliding window

\[
z_0 \cdot A = \text{ACACAAAGCA}
\]

\[
M(z_0 \cdot A) = \{\text{AAA, GA, AAC, CACAC, CAG, CC, GCA, GCAA, GCAC, CG, GG}\}
\]
Minimal absent words over a sliding window

Algorithm for maws in a sliding window

\[ A \cdot z_1 = \text{CACACAGCA} \]

\[ M(A \cdot z_1) \setminus M(z_1) = \{\text{CACAC}\} \]
Minimal absent words over a sliding window

Algorithm for maws in a sliding window

\[
z_1 = \text{CA CA A G C A}
\]

\[
M(z_1) = \{\text{AAA, GA, AAC, CAG, CC, GCAA, ACAC, GCAC, CG, GG}\}
\]
Applications to on-line pattern matching

Minimal absent words over a sliding window

For a sequence $S$ of size $n$ and a window of size $m$ we compute:

$$\forall i, 0 \leq i \leq n - m, M(S[i \ldots i + m - 1]),$$

in time $O(n)$ and in space $O(m)$. 

Length Weighted Index (LWI), introduced by Chairungsee in 2012

Metric based on the symmetric difference of minimal absent words

$$LWI(M(x), M(y)) = \sum_{w \in M(x)} \Delta M(y) \cdot |w|^2.$$ 

We obtain the position of minimal distance.
Applications to on-line pattern matching

Minimal absent words over a sliding window

For a sequence $S$ of size $n$ and a window of size $m$ we compute:

$$\forall i, 0 \leq i \leq n - m, M(S[i \ldots i + m - 1]),$$

in time $O(n)$ and in space $O(m)$.

Length Weighted Index (LWI), introduced by Chairungsee in 2012

Metric based on the symmetric difference of minimal absent words sets

$$\text{LWI}(M(x), M(y)) = \sum_{w \in M(x) \Delta M(y)} \frac{1}{|w|^2}.$$
Applications to on-line pattern matching

Minimal absent words over a sliding window

For a sequence $S$ of size $n$ and a window of size $m$ we compute:

$$\forall i, 0 \leq i \leq n - m, M(S[i \ldots i + m - 1]),$$

in time $O(n)$ and in space $O(m)$.

Length Weighted Index (LWI), introduced by Chairungsee in 2012

Metric based on the symmetric difference of minimal absent words sets

$$\text{LWI}(M(x), M(y)) = \sum_{w \in M(x) \triangle M(y)} \frac{1}{|w|^2}.$$  

→ We obtain the position of minimal distance.
Futur works

- Implement the algorithm over a sliding window,
- Compare the results in Bioinformatics for reads alignment.
Remerciements

Merci
Remerciements

LOB
- Hubert Becker
- Hannu Myllykallio
- Roxane Lestini
- Yoann Colliien
- et tous les autres

LIX
- Mireille Régnier
- Yann Ponty
- Philippe Chassignet
- Amélie Héliou
- Afaf Saaidi
- Juraj Michalik
- et tous les autres

Université de Rouen
- Laurent Mouchard
- King’s College London
- Solon Pissis
- Carl Barton

Université d’Helsinki
- Simon Puglisi

Université Paris Est
- Maxime Crochemore
- Gregory Kucherov
- ENS Paris
- Yann Ramusat
# Algorithms to compute minimal absent words

<table>
<thead>
<tr>
<th>References</th>
<th>Structures</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crochemore et al., 1998</td>
<td>Suffix Automata</td>
<td>Expensive in space</td>
</tr>
<tr>
<td>Belazzougui et al. 2013</td>
<td>Bidirectionnal BWT</td>
<td>No implementation available</td>
</tr>
<tr>
<td>Ota et al. 2014</td>
<td>Suffix tree, dynamic approach</td>
<td>Quadratic in time</td>
</tr>
<tr>
<td>Barton et al. 2014</td>
<td>Suffix Array</td>
<td>Linear, fastest available</td>
</tr>
<tr>
<td>Belazzougui et al. 2015</td>
<td>BWT &amp; complementary structures</td>
<td>No implementation</td>
</tr>
</tbody>
</table>