Indexable Elastic Founder Graphs of Minimum Height with Suffix Tree Maneuvers

Nicola Rizzo    Veli Mäkinen

Algorithmic Bioinformatics Research Group
Department of Computer Science, University of Helsinki, Finland
{nicola.rizzo,veli.makinen}@helsinki.fi

CPM 2022

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie ITN ALPACA (grant agreement No 956229).
Outline

1. The Elastic Founder Graph for a MSA
2. Preprocess: Computing the valid segments
3. Preprocess: Minimizing the maximum height
The search for a pangenome data structure

- computational pangenomics: find a data structure for a coherent collection of genomes supporting fast pattern matching
The search for a pangenome data structure

• **computational pangenomics**: find a data structure for a coherent collection of genomes supporting fast pattern matching

• graph-based candidates like Variation Graphs and Elastic Degenerate Strings usually represent a **multiple sequence alignment (MSA)**
The search for a pangenome data structure

- **computational pangenomics**: find a data structure for a coherent collection of genomes supporting fast pattern matching
- graph-based candidates like Variation Graphs and Elastic Degenerate Strings usually represent a multiple sequence alignment (MSA)
- cannot support string matching in subquadratic time under the Orthogonal Vectors Hypothesis for simple classes of graphs
The Elastic Founder Graph

Definition (Mäkinen et al (2020))

A segmentation $S$ of MSA[1..$m$, 1..$n$] induces an elastic block graph $G(S) = (V, E, \ell)$ that we call elastic founder graph (EFG).

Example Segmentation $S = [1..4], [5..8], [9..13]$ induces this EFG $G(S)$. 

- strings become node labels;
- edges are based on local occurrences $\Rightarrow$ recombination;
- linear-time construction algorithms for the gapless setting, non-trivial to extend to the general setting.
The Elastic Founder Graph for a MSA

Preprocess: Computing the valid segments
Preprocess: Minimizing the maximum height

The Elastic Founder Graph

Definition (Mäkinen et al (2020))

A segmentation $S$ of MSA[1..m, 1..n] induces an elastic block graph $G(S) = (V, E, \ell)$ that we call elastic founder graph (EFG).

Example

Segmentation $S = [1..4], [5..8], [9..13]$ induces this EFG $G(S)$. 

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>G</td>
<td>A</td>
<td>C</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>T</td>
<td>T</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

AGCG  ➔  ACTA  ➔  GATAC
AGC   ➔  ATTA   ➔  GTAG
                 ➔  GTTAC
The Elastic Founder Graph

Definition (Mäkinen et al (2020))

A segmentation $S$ of MSA[1..$m$, 1..$n$] induces an elastic block graph $G(S) = (V, E, \ell)$ that we call elastic founder graph (EFG).

Example

Segmentation $S = [1..4], [5..8], [9..13]$ induces this EFG $G(S)$.

• strings become node labels;
The Elastic Founder Graph

Definition (Mäkinen et al (2020))

A segmentation \( S \) of MSA[1..\( m \), 1..\( n \)] induces an elastic block graph \( G(S) = (V, E, \ell) \) that we call elastic founder graph (EFG).

Example

Segmentation \( S = [1..4], [5..8], [9..13] \) induces this EFG \( G(S) \).

- strings become node labels;
- edges are based on local occurrences \( \Rightarrow \) recombination;
The Elastic Founder Graph

Definition (Mäkinen et al (2020))

A segmentation $S$ of MSA$[1..m,1..n]$ induces an elastic block graph $G(S) = (V,E,\ell)$ that we call elastic founder graph (EFG). EFGs respecting the semi-repeat-free property admit a poly-time index for linear-time pattern matching.

Example

Segmentation $S = [1..4], [5..8], [9..13]$ induces this EFG $G(S)$.

- strings become node labels;
- edges are based on local occurrences $\Rightarrow$ recombination;
The Elastic Founder Graph

**Definition (Mäkinen et al. (2020))**

A segmentation $S$ of MSA$[1..m,1..n]$ induces an elastic block graph $G(S) = (V, E, \ell)$ that we call elastic founder graph (EFG). EFGs respecting the semi-repeat-free property admit a poly-time index for linear-time pattern matching.

**Example**

Segmentation $S = [1..4], [5..8], [9..13]$ induces this EFG $G(S)$.

- strings become node labels;
- edges are based on local occurrences $\Rightarrow$ recombination;
The Elastic Founder Graph

Definition (Mäkinen et al (2020))

A segmentation \( S \) of MSA[1..\( m \), 1..\( n \)] induces an elastic block graph \( G(S) = (V, E, \ell) \) that we call elastic founder graph (EFG). EFGs respecting the semi-repeat-free property admit a poly-time index for linear-time pattern matching.

Example

Segmentation \( S = [1..4], [5..8], [9..13] \) induces this EFG \( G(S) \).

- strings become node labels;
- edges are based on local occurrences \(\Rightarrow\) recombination;
- linear-time construction algorithms for the gapless setting, non-trivial to extend to the general setting.
EFG construction algorithms

• we concentrate on constructing a semi-repeat-free EFG minimizing the maximum block height
EFG construction algorithms

- we concentrate on constructing a semi-repeat-free EFG minimizing the maximum block height
- an optimal segmentation is found via dynamic programming (details in the paper) after two important preprocessing steps:
EFG construction algorithms

• we concentrate on constructing a semi-repeat-free EFG minimizing the maximum block height

• an optimal segmentation is found via dynamic programming (details in the paper) after two important preprocessing steps:
  • computing all valid semi-repeat-free segments
  • computing the height information of all valid segments
EFG construction algorithms

- we concentrate on constructing a semi-repeat-free EFG minimizing the maximum block height
- an optimal segmentation is found via dynamic programming (details in the paper) after two important preprocessing steps:
  - computing all valid semi-repeat-free segments
  - computing the height information of all valid segments

Our contributions:

1. preprocessing in time $O(mn\alpha \log |\Sigma|)$, where $\alpha$ is the length of longest aligned common substring
EFG construction algorithms

- we concentrate on constructing a semi-repeat-free EFG minimizing the maximum block height
- an optimal segmentation is found via dynamic programming (details in the paper) after two important preprocessing steps:
  - computing all valid semi-repeat-free segments
  - computing the height information of all valid segments

Our contributions:

1. preprocessing in time $O(mn\alpha \log |\Sigma|)$, where $\alpha$ is the length of longest aligned common substring
2. we studied a refined height definition resulting in an $O(mn)$-time preprocessing and construction algorithm
Representing the valid segments

The first step is computing the valid segments.

**Observation**

If \([x..y]\) is valid, then \([x..y']\) is valid for all \(y' > y\).
Representing the valid segments

The first step is computing the valid segments.

**Observation**

If \([x..y]\) is valid, then 
\([x..y']\) is valid for all 
\(y' > y\).

**Example**

\([3..4]\) is not semi-repeat-free
The Elastic Founder Graph for a MSA

Preprocess: Computing the valid segments

Representing the valid segments

The first step is computing the valid segments.

**Observation**

If \([x..y]\) is valid, then \([x..y']\) is valid for all \(y' > y\).

**Example**

\([3..4]\) is not semi-repeat-free but \([3..5]\) is,
Representing the valid segments

The first step is computing the valid segments.

**Observation**
If \([x..y]\) is valid, then \([x..y']\) is valid for all \(y' > y\).

**Definition**
Given \(x\), the minimal right extension \(f(x)\) marks the first column so that \([x..f(x)]\) is valid.

**Example**
\([3..4]\) is not semi-repeat-free but \([3..5]\) is, so \(f(3) = 5\).
The generalized suffix tree

The main tool we use to compute \([x..f(x)]\) is \(\text{GST}_{\text{MSA}}\), the generalized suffix tree of strings \(\text{spell}(\text{MSA}[i, 1..n]) \cdot \$_i\).
The generalized suffix tree

The main tool we use to compute \([x..f(x)]\) is \(\text{GST}_{\text{MSA}}\), the generalized suffix tree of strings \(\text{spell}(\text{MSA}[i, 1..n]) \cdot \$i\).
The generalized suffix tree

The main tool we use to compute \([x..f(x)]\) is \(\text{GST}_{\text{MSA}}\), the generalized suffix tree of strings \(\text{spell}(\text{MSA}[i, 1..n]) \cdot \$i\).
From the MSA to the suffix tree

- we can break down $f(x)$ to single rows
From the MSA to the suffix tree

- we can break down $f(x)$ to single rows
- suffixes $\Rightarrow$ leaves of $\text{GST}_{\text{MSA}} \Rightarrow$ exclusive ancestors
From the MSA to the suffix tree

- we can break down $f(x)$ to single rows
- suffixes $\Rightarrow$ leaves of $\text{GST}_{\text{MSA}}$ $\Rightarrow$ exclusive ancestors
- we navigate back to the MSA with rank and select queries
From the MSA to the suffix tree

• we can break down $f(x)$ to single rows
• suffixes ⇒ leaves of GST$_{MSA}$ ⇒ exclusive ancestors
• we navigate back to the MSA with rank and select queries
• $f(x)$ can be computed in time $O(m) \Rightarrow$ global $O(mn)$ time
Min max height in the gapless setting

Now that we have the valid segments, we need to compute their height information.
Min max height in the gapless setting

Now that we have the valid segments, we need to compute their height information.

Observation
If MSA[1..m, 1..n] has no gaps, height $H([x..y])$ is increasing with respect to $y$.

<table>
<thead>
<tr>
<th>Segment</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>T T C C C G G</td>
<td>1</td>
</tr>
<tr>
<td>T T A C C G A</td>
<td>2</td>
</tr>
<tr>
<td>T T A C A C A</td>
<td>3</td>
</tr>
<tr>
<td>T T A C A C G</td>
<td>4</td>
</tr>
<tr>
<td>T T A C A A G</td>
<td>5</td>
</tr>
<tr>
<td>G T C A A G G</td>
<td>6</td>
</tr>
</tbody>
</table>

$H([1..y])$: 2 2 3 3 4 5 6 6
Min max height in the gapless setting

Now that we have the valid segments, we need to compute their height information.

**Observation**
If MSA[1..m, 1..n] has no gaps, height $H([x..y])$ is increasing with respect to $y$.

**Definition**
We define the meaningful right extensions $R_x$ as $r_{x,1}, \ldots, r_{x,c_x}$, the positions $y$ where height $H([x..y])$ changes (increases).

![Sequence Alignment]

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>G</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>G</td>
<td>A</td>
<td>$1$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>$2$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>$3$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>G</td>
<td>$4$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>$5$</td>
</tr>
<tr>
<td>G</td>
<td>T</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>$6$</td>
</tr>
</tbody>
</table>

$H([1..y]) = 2 \quad 2 \quad 3 \quad 3 \quad 4 \quad 5 \quad 6 \quad 6$
Min max height in the gapless setting

Now that we have the valid segments, we need to compute their height information.

**Observation**
If MSA\([1..m, 1..n]\) has no gaps, height \(H([x..y])\) is increasing with respect to \(y\).

**Definition**
We define the meaningful right extensions \(R_x\) as \(r_{x,1}, \ldots, r_{x,c_x}\), the positions \(y\) where height \(H([x..y])\) changes (increases).

In the gapless setting, \(|R_x| \leq m\) so \(\sum_{x=1}^{n}|R_x| \in O(mn)\):
Now that we have the valid segments, we need to compute their height information.

**Observation**

If MSA[1..m, 1..n] has no gaps, height $H([x..y])$ is increasing with respect to $y$.

**Definition**

We define the meaningful right extensions $R_x$ as $r_{x,1}, \ldots, r_{x,c_x}$, the positions $y$ where height $H([x..y])$ changes (increases).

In the gapless setting, $|R_x| \leq m$ so $\sum_{x=1}^{n} |R_x| \in O(mn)$:

- **Norri et al (2019)** $O(mn)$-time computation of all $R_x$ w/ height values (in a different context from the semi-repeat-free one)
Min max height in the gapless setting

Now that we have the valid segments, we need to compute their height information.

**Observation**

If MSA[1..m, 1..n] has no gaps, height $H([x..y])$ is increasing with respect to $y$.

**Definition**

We define the meaningful right extensions $R_x$ as $r_{x,1}, \ldots, r_{x,c_x}$, the positions $y$ where height $H([x..y])$ changes (increases).

In the gapless setting, $|R_x| \leq m$ so $\sum_{x=1}^{n} |R_x| \in O(mn)$:

- **Norri et al (2019)** $O(mn)$-time computation of all $R_x$ w/ height values (in a different context from the semi-repeat-free one)
- **Mäkinen et al (2020)** $O(mn)$-time computation of $f(x)$
Min max height in the gapless setting

Now that we have the valid segments, we need to compute their height information.

Observation
If MSA[1..m, 1..n] has no gaps, height $H([x..y])$ is increasing with respect to y.

Definition
We define the meaningful right extensions $R_x$ as $r_{x,1}, \ldots, r_{x,c_x}$, the positions $y$ where height $H([x..y])$ changes (increases).

In the gapless setting, $|R_x| \leq m$ so $\sum_{x=1}^{n} |R_x| \in O(mn)$:

- **Norri et al (2019)** $O(mn)$-time computation of all $R_x$ w/ height values (in a different context from the semi-repeat-free one)

- **Mäkinen et al (2020)** $O(mn)$-time computation of $f(x)$

Result: $O(mn)$-time segmentation algorithm for gapless MSAs.
The more difficult setting with gaps

In the setting with gaps, $R_x \in O(n)$:

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & n-2 & n \\
1 & T & A & - & A & - & A & \ldots & A & - & C \\
\end{array}
\]

\[H([1..y]) = 1 \ 2 \ 1 \ 2 \ 1 \ 2 \ 1 \ \cdots \ 2 \ 1 \ 1\]
The more difficult setting with gaps

In the setting with gaps, \( R_x \in O(n) \):

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & n-2 & n \\
1 & T & A & - & A & - & A & \ldots & A & - & C \\
\end{array}
\]

\( H([1..y]) \) 1 2 1 2 1 2 1 \( \cdots \) 2 1 1

Thus \( \sum_{x=1}^{n} |R_x| \in O(n^2) \)
The more difficult setting with gaps

In the setting with gaps, \( R_x \in O(n) \):

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & n - 2 & n \\
\end{array}
\]

\[
H([1..y]) 1 2 1 2 1 2 1 \ldots 2 1 1
\]

Thus \( \sum_{x=1}^{n} |R_x| \in O(n^2) \):

- computing \( R_x + \) height info na"ively (keyword tries) yields a \( O(mn\alpha \log|\Sigma|) \)-time algorithm, where \( \alpha \) is the length of the longest aligned substring between any two rows
The more difficult setting with gaps

In the setting with gaps, \( R_x \in O(n) \):

\[
\begin{array}{cccccccc}
  & 1 & 2 & 3 & 4 & 5 & 6 & 7 & n-2 & n \\
\end{array}
\]

\[ H([1..y]) = 1 \ 2 \ 1 \ 2 \ 1 \ 2 \ 1 \ \ldots \ 2 \ 1 \ 1 \]

Thus \( \sum_{x=1}^{n} |R_x| \in O(n^2) \):

- computing \( R_x + \) height info na"ively (keyword tries) yields a \( O(mn\alpha \log|\Sigma|) \)-time algorithm, where \( \alpha \) is the length of the longest aligned substring between any two rows
- construction algorithm processes all \( R_x + \) height info as before
The more difficult setting with gaps

In the setting with gaps, \( R_x \in O(n) \):

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & n-2 & n \\
1 & T & A & - & A & - & A & \ldots & A & - & C \\
\end{array}
\]

\[ H([1..y]) \quad 1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 2 \quad 1 \quad \cdots \quad 2 \quad 1 \quad 1 \]

Thus \( \sum_{x=1}^{n} |R_x| \in O(n^2) \):

- computing \( R_x + \text{ height info na"ively (keyword tries) yields a } O(mn\alpha \log|\Sigma|) - \text{time algorithm} \), where \( \alpha \) is the length of the longest aligned substring between any two rows
- construction algorithm processes all \( R_x + \text{ height info as before} \)

Solution is \( O(mn^2 \log|\Sigma|) \): can we do better?
### Prefix-aware height

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC</td>
<td>C</td>
<td>-</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>C</td>
<td>G</td>
<td>-</td>
<td>$1$</td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>AC</td>
<td>-</td>
<td>-</td>
<td>C</td>
<td>-</td>
<td>-</td>
<td>$2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>AC</td>
<td>A</td>
<td>C</td>
<td>-</td>
<td>A</td>
<td>$3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>AC</td>
<td>A</td>
<td>C</td>
<td>G</td>
<td>-</td>
<td>$4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>AC</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>-</td>
<td>$5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GC</td>
<td>A</td>
<td>-</td>
<td>-</td>
<td>A</td>
<td>G</td>
<td>-</td>
<td>$6$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\overline{H}([1..y]) = 2 \quad 3 \quad 3 \quad 3 \quad 5 \quad 5 \quad 6 \quad 6
\]

### Definition

We define $\overline{H}([x..y])$ as the number of distinct strings in $[x..y]$ that are not proper prefixes of other strings in $[x..y]$.
Prefix-aware height

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
T & C & C & C & G & \$1 \\
T & A & C & C & \$2 \\
T & A & C & A & C & A & \$3 \\
T & A & C & A & C & G & \$4 \\
T & A & C & A & A & G & \$5 \\
G & C & A & A & G & \$6 \\
\end{array}
\]

\[\overline{H}([1..y]) = 2 \ 3 \ 3 \ 3 \ 5 \ 5 \ 6 \ 6 \]

**Definition**

We define \(\overline{H}([x..y])\) as the number of distinct strings in \([x..y]\) that are not proper prefixes of other strings in \([x..y]\).

- \(\overline{H}([x..y]) \leq H([x..y])\) so it is a lower bound
Prefix-aware height

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>G</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>T</td>
<td>−</td>
<td>A</td>
<td>C</td>
<td>−</td>
<td>−</td>
<td>C</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>T</td>
<td>−</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>−</td>
<td>$\cdot$</td>
<td>$\cdot$</td>
</tr>
<tr>
<td>T</td>
<td>−</td>
<td>A</td>
<td>C</td>
<td>−</td>
<td>A</td>
<td>G</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>T</td>
<td>−</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>−</td>
<td>G</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>G</td>
<td>C</td>
<td>A</td>
<td>−</td>
<td>A</td>
<td>G</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

$\overline{H}([1..y]) = 2\ 3\ 3\ 3\ 3\ 5\ 5\ 6\ 6$

**Definition**

We define $\overline{H}([x..y])$ as the number of distinct strings in $[x..y]$ that are not proper prefixes of other strings in $[x..y]$.

- $\overline{H}([x..y]) \leq H([x..y])$ so it is a lower bound
- we can define the meaningful prefix-aware extensions $\overline{R}_x$
  \[ \sum_{x=1}^{n} |\overline{R}_x| \in O(mn) \]
Prefix-aware height

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
T & C & - & C & - & - & C & G & - & $1$
T & - & A & C & - & A & C & G & - & $4$
\end{array}
\]

\[\overline{H}([1..y]) = 2\ 3\ 3\ 3\ 3\ 5\ 5\ 6\ 6\]

Definition

We define \(\overline{H}([x..y])\) as the number of distinct strings in \([x..y]\) that are not proper prefixes of other strings in \([x..y]\).

- \(\overline{H}([x..y]) \leq H([x..y])\) so it is a lower bound
- we can define the meaningful prefix-aware extensions \(\overline{R}_x\)
  \[\sum_{x=1}^{n}|\overline{R}_x| \in O(mn)\]
- plug-and-play with the construction algorithm
The suffix tree uncovers the prefix-aware height

Let’s concentrate on the forest of $\text{GST}_{\text{MSA}}$ for $[x..n]$, with $x = 1$
The suffix tree uncovers the prefix-aware height

Let’s concentrate on the forest of $\text{GST}_{\text{MSA}}$ for $[x..n]$, with $x = 1$
The suffix tree uncovers the prefix-aware height

- for each node we find the first ending column of the relative MSA occurrence

\[ \overline{H}([1..y]) \]

\[
\begin{array}{ccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
T & C & - & C & - & - & C & G & - & $1 \\
\end{array}
\]
The suffix tree uncovers the prefix-aware height

- for each node we find the first ending column of the relative MSA occurrence

- most of these ending columns mark a +1 increase in $\bar{H}$
Final suffix tree maneuvers

Computing these values using just GST_{MSA} takes $O(m^2 n)$ time in total.

We obtain a linear-time solution with suffix tree maneuvers:
Final suffix tree maneuvers

Computing these values using just $\text{GST}_{\text{MSA}}$ takes $O(m^2 n)$ time in total.

We obtain a linear-time solution with suffix tree maneuvers:

- we can compute the pos values in $O(mn)$ time with $\text{GPT}_{\text{MSA}}$, the generalized prefix tree of the MSA rows
Final suffix tree maneuvers

Computing these values using just GST\textsubscript{MSA} takes $O(m^2n)$ time in total.

We obtain a linear-time solution with suffix tree maneuvers:

- we can compute the pos values in $O(mn)$ time with GPT\textsubscript{MSA}, the generalized prefix tree of the MSA rows
- $O(1)$-time GST\textsubscript{MSA}-to-GPT\textsubscript{MSA} navigation thanks to weighted ancestor queries/affix trees
Conclusions

Summary of results for optimal EFG construction:

- $O(mn)$-time solution for \textit{min max height} in the gapless setting
- $O(mn \alpha \log |\Sigma|)$-time solution for the setting with gaps
- $O(mn)$-time solution for \textit{min max prefix-aware height}
Conclusions

Summary of results for optimal EFG construction:

- $O(mn)$-time solution for min max height in the gapless setting
- $O(mn\alpha \log|\Sigma|)$-time solution for the setting with gaps
- $O(mn)$-time solution for min max prefix-aware height

Future work:

- extending EFGs to allow segments containing empty strings


Problem

Let $T = (V, E, \text{root})$ be a rooted ordered tree. Given a subset of leaves $L$, find the minimal set $W$ of exclusive ancestors of $L$ in $T$, i.e. covering all leaves in $L$ and only leaves in $L$. 
The exclusive ancestor set problem

Problem

Let \( T = (V, E, \text{root}) \) be a rooted ordered tree. Given a subset of leaves \( L \), find the minimal set \( W \) of exclusive ancestors of \( L \) in \( T \), i.e. covering all leaves in \( L \) and only leaves in \( L \).
The exclusive ancestor set problem

**Problem**

Let $T = (V, E, \text{root})$ be a rooted ordered tree. Given a subset of leaves $L$, find the minimal set $W$ of exclusive ancestors of $L$ in $T$, i.e. covering all leaves in $L$ and only leaves in $L$.

After an $O(|V|)$-time preprocessing of $T$, any instance $L$ can be solved in time $O(|L|)$. 
The solution
The solution

1. partition $L$ in maximal sets of contiguous leaves $L_i$
The solution

1. partition $L$ in maximal sets of contiguous leaves $L_i$
2. for each $L_i$; start from the leftmost leaf and ascend in the tree until failure (checked in $O(1)$)
The solution

1. partition $L$ in maximal sets of contiguous leaves $L_i$
2. for each $L_i$ start from the leftmost leaf and ascend in the tree until failure (checked in $O(1)$)
3. add the last safe ancestor to the solution $W$ and repeat steps 2. and 3. starting from the leftmost uncovered leaf
The solution

1. partition $L$ in maximal sets of contiguous leaves $L_i$
2. for each $L_i$ start from the leftmost leaf and ascend in the tree until failure (checked in $O(1)$)
3. add the last safe ancestor to the solution $W$ and repeat steps 2. and 3. starting from the leftmost uncovered leaf
The solution

1. partition $L$ in maximal sets of contiguous leaves $L_i$
2. for each $L_i$ start from the leftmost leaf and ascend in the tree until failure (checked in $O(1)$)
3. add the last safe ancestor to the solution $W$ and repeat steps 2. and 3. starting from the leftmost uncovered leaf
The solution

1. partition $L$ in maximal sets of contiguous leaves $L_i$
2. for each $L_i$ start from the leftmost leaf and ascend in the tree until failure (checked in $O(1)$)
3. add the last safe ancestor to the solution $W$ and repeat steps 2. and 3. starting from the leftmost uncovered leaf
The solution

1. partition $L$ in maximal sets of contiguous leaves $L_i$
2. for each $L_i$ start from the leftmost leaf and ascend in the tree until failure (checked in $O(1)$)
3. add the last safe ancestor to the solution $W$ and repeat steps 2. and 3. starting from the leftmost uncovered leaf
The solution

1. partition $L$ in maximal sets of contiguous leaves $L_i$
2. for each $L_i$ start from the leftmost leaf and ascend in the tree until failure (checked in $O(1)$)
3. add the last safe ancestor to the solution $W$ and repeat steps 2. and 3. starting from the leftmost uncovered leaf
The final preprocessing algorithm

For $x \in [0..n - 1]$:

- find the exclusive ancestor set of $L_x$, i.e. the leaves of $\text{GST}_{\text{MSA}}$ starting at column $x + 1 \Rightarrow O(m)$;
The final preprocessing algorithm

For $x \in [0..n - 1]$:  

- find the exclusive ancestor set of $L_x$, i.e. the leaves of $\text{GST}_{\text{MSA}}$ starting at column $x + 1 \Rightarrow O(m)$;  
- find values $f^i(x)$ using the gap information (rank and select queries) and compute $f(x) = \max_i f^i(x) \Rightarrow O(m)$;
The final preprocessing algorithm

For $x \in [0..n - 1]$:

- find the exclusive ancestor set of $L_x$, i.e. the leaves of $GST_{MSA}$ starting at column $x + 1 \Rightarrow O(m)$;
- find values $f^i(x)$ using the gap information (rank and select queries) and compute $f(x) = \max_i f^i(x) \Rightarrow O(m)$;
- compute $L_{x+1}$ by following the suffix links $\Rightarrow O(m)$.
The final preprocessing algorithm

For $x \in [0..n - 1]$:

- find the exclusive ancestor set of $L_x$, i.e. the leaves of $\text{GST}_{\text{MSA}}$ starting at column $x + 1 \Rightarrow O(m)$;
- find values $f^i(x)$ using the gap information (rank and select queries) and compute $f(x) = \max_i f^i(x) \Rightarrow O(m)$;
- compute $L_{x+1}$ by following the suffix links $\Rightarrow O(m)$.

Thus, we can compute values $f(x)$ in linear time $O(mn)$. 
Constructing EFGs in the general case

An EFG built from MSA[1..m, 1..n] maximizing the number of blocks (i.e. segments) can be computed recursively:

$$s(j) = \max_{j' : 0 \leq j' < j \text{ s.t.}} s(j') + 1$$

where $s(j)$ is the optimal score of a segmentation of MSA[1..m, 1..j].

**Theorem (Equi et al. (2021))**

*We can compute such EFG in time $O(n)$, after an $O(mn \log m)$-time preprocessing of the MSA.*
The tale of two recursion types

Using only pairs \((x, f(x))\), we can find the score \(s(n)\) of an optimal segmentation with a dynamic programming algorithm analyzing the pairs and at the same time computing \(s(j)\).

Consider adding segment \([x + 1..j']\) to an optimal solution of MSA\([1..m, 1..x]\) having score \(s(x)\).

Thus,

\[
\begin{align*}
s(j) &= \min \left( \min_{(x, f(x)) : f(x) \leq j \leq x + s(x)} s(x), \min_{(x, f(x)) : j > f(x) \land j > x + s(x)} j - x \right) \end{align*}
\]
Simpler data structures

Equi et al. (2021) reached a time complexity of $O(n \log \log n)$, by sorting pairs $(x, f(x))$ in increasing order by their second component and keeping track of the two types of recursions with two data structures. But we can do better:

- for the leader recursion, we just need to store in a variable $S$ the best currently valid value of $j' - x$;
- for the non-leader recursion, we can count the currently valid values $s(x)$ in an array $C$ such that $C[s]$ is the number of valid ranges $[x + 1..j]$ having score $s = s(x)$; a variable $K$ then can store value $\min\{k : C[k] > 0\}$.

Thus $s(j) = \min(S, K)$, but we need to update $S$, $C$, and $K$. 
The linear time solution

• the dynamic management of intervals \([x + 1..x + s(x)]\) (and structure \(C\)) takes time \(O(n)\);
• we don’t have to recompute \(K = \min\{k : C[k] > 0\}\) each time, because when \(C[k]\) gets updated to zero, \(S\) gets updated with value \(K\) and \(S\) increases by at most one at each iteration.

**Theorem**

*We can compute an EFG minimizing the maximum segment length in global time* \(O(mn)\).*