

<https://helda.helsinki.fi>

Indexable Elastic Founder Graphs of Minimum Height

Rizzo, Nicola

Schloss Dagstuhl - Leibniz-Zentrum für Informatik
2022-06

Rizzo , N & Mäkinen , V 2022 , Indexable Elastic Founder Graphs of Minimum Height . in H Bannai & J Holub (eds) , 33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022) . Leibniz International Proceedings in Informatics (LIPIcs) , no. 223 , Schloss Dagstuhl
p̃y - Leibniz - Zentrum für Informatik , Dagstuhl, Germany , pp. 19:1 19:19
on Combinatorial Pattern Matching , Prague , Czech Republic , 27/06/2022 . <https://doi.org/10.4230/LIPIcs.CPM.2022.19>

<http://hdl.handle.net/10138/353005>

<https://doi.org/10.4230/LIPIcs.CPM.2022.19>

cc_by

publishedVersion

Downloaded from Helda, University of Helsinki institutional repository.

This is an electronic reprint of the original article.



This reprint may differ from the original in pagination and typographic detail.

Please cite the original version.

Indexable Elastic Founder Graphs of Minimum Height

Nicola Rizzo  

Department of Computer Science, University of Helsinki, Finland

Veli Mäkinen  

Department of Computer Science, University of Helsinki, Finland

Abstract

Indexable elastic founder graphs have been recently proposed as a data structure for genomics applications supporting fast pattern matching queries. Consider segmenting a multiple sequence alignment $MSA[1..m, 1..n]$ into b blocks $MSA[1..m, 1..j_1]$, $MSA[1..m, j_1 + 1..j_2]$, \dots , $MSA[1..m, j_{b-1} + 1..n]$. The resulting *elastic founder graph* (EFG) is obtained by merging in each block the strings that are equivalent after the removal of gap symbols, taking the strings as the nodes of the block and the original MSA connections as edges. We call an elastic founder graph *indexable* if a node label occurs as a prefix of only those paths that start from a node of the same block. Equi et al. (ISAAC 2021) showed that such EFGs support fast pattern matching and studied their construction maximizing the number of blocks and minimizing the maximum length of a block, but left open the case of minimizing the maximum number of distinct strings in a block that we call *graph height*. For the simplified gapless setting, we give an $O(mn)$ time algorithm to find a segmentation of an MSA minimizing the height of the resulting indexable founder graph, by combining previous results in segmentation algorithms and founder graphs. For the general setting, the known techniques yield a linear-time parameterized solution on constant alphabet Σ , taking time $O(mn^2 \log|\Sigma|)$ in the worst case, so we study the refined measure of *prefix-aware height*, that omits counting strings that are prefixes of another considered string. The indexable EFG minimizing the maximum prefix-aware height provides a lower bound for the original height: by exploiting suffix trees built from the MSA rows and the data structure answering weighted ancestor queries in constant time of Belazzougui et al. (CPM 2021), we give an $O(mn)$ -time algorithm for the optimal EFG under this alternative height.

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Pattern matching; Theory of computation \rightarrow Sorting and searching; Theory of computation \rightarrow Dynamic programming; Applied computing \rightarrow Genomics

Keywords and phrases multiple sequence alignment, pattern matching, data structures, segmentation algorithms, dynamic programming, suffix tree

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.19

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

1 Introduction

String matching in a text and its variants are classic problems in computer science, with a myriad of applications such as biological sequence analysis. The generalization of the string matching problem concerned with searching strings in a labeled graph has gained more and more importance in computational biology, and for a good reason: the rapidly increasing number of sequenced data makes it possible to capture the variation of many species, populations, and cancer genomes, for example forming the so-called *pangenome* of a species [5]. A central challenge of pangenomics is then to provide the computational tools to swap a single reference genome with a pangenomic representation of hundreds – if not thousands – of genomes in the established analysis tasks [19, 24, 25, 13, 17, 7, 20].



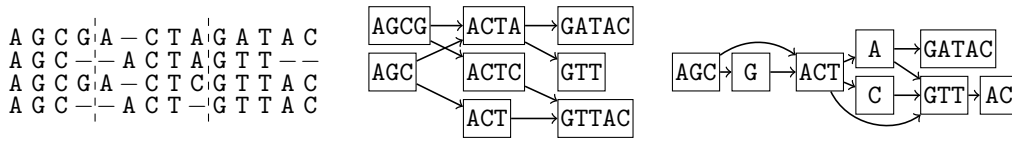
© Nicola Rizzo and Veli Mäkinen;
licensed under Creative Commons License CC-BY 4.0
33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022).

Editors: Hideo Bannai and Jan Holub; Article No. 19; pp. 19:1–19:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** An elastic founder graph (middle) induced from a MSA segmentation (left). This EFG is *semi-repeat-free*, meaning that each node label appears only as prefix of paths starting from the same block. Thus, we say it is indexable since it supports fast pattern matching. On the right, the modification of the EFG suggested by the prefix-aware height, compacting labels that are prefixes of others in the same block: we reserve its study as future work.

The most popular representation for a pangenome is a graph whose paths spell the input genomes, and the basic primitive required on such pangenome graphs is to be able to search occurrences of query strings (short reads) as subpaths of the graph. On this front, research efforts have been met with theoretical roadblocks: string matching in labeled graphs cannot be solved in sub-quadratic time even for simple graph classes, unless the Orthogonal Vectors Hypothesis (OVH) is false [8]; under OVH, no polynomial-time indexing scheme of a graph can support sub-quadratic time queries [9]; identifying if a graph belongs to the class of Wheeler graphs, that are easy to index, is NP-complete [14]. Therefore, practical tools deploy various heuristics or use other pangenome representations as a basis.

The pangenomic representation at the heart of the *elastic founder graph* (EFG), proposed by Mäkinen et al. [18] and Equi et al. [10], is then to assume as input a *multiple sequence alignment*, a matrix $\text{MSA}[1..m, 1..n]$ composed of m rows that are strings of length n , drawn from an alphabet Σ plus a special “gap” symbol where each column represents an aligned position of the characters of the rows. As seen in Figure 1, the EFG is then created by choosing a segmentation of the MSA, that is, a partition of the columns in consecutive blocks: the strings in each block become the nodes of the block, and the edges are defined by the original row connections. If the node labels do not appear as prefix of any other path than those starting at the same block, then the so-called *semi-repeat-free property* holds and the graph supports an index structure for fast pattern matching [18, 10]. Equi et al. [10] also show that an indexability property like this one is required to have pattern matching in sub-quadratic time, since the OVH-based lower bound holds even when restricted to EFGs induced by MSA segmentations. Mäkinen et al. [18] gave an $O(mn)$ time algorithm constructing an indexable EFG minimizing the maximum block length, given a gapless $\text{MSA}[1..m, 1..n]$. Equi et al. [10] extended the result to general MSAs, obtaining $O(mn \log m)$ time algorithms for the same optimization and for maximizing the number of blocks, and we recently improved these two results to $O(mn)$ time [22]. We refer the reader to the aforementioned papers for the connections of the approach to Elastic Degenerate Strings and Wheeler graphs.

In this paper, we continue the study of indexable EFGs and focus on optimizing the height of the resulting graph. This measure is defined as the maximum number of distinct strings (i.e. nodes) in a block, and in the example of Figure 1 (middle) this measure is equal to 3, as the second and last blocks have 3 nodes. In Section 2, we provide the basic definitions around EFGs. In Section 3, we introduce the algorithms to find an MSA segmentation minimizing the height of the resulting indexable EFG: for the gapless case, we show how the left-to-right segmentation algorithm by Norri et al. exploiting *left extensions* [21] can be combined with the computation of the *minimal left extensions* by Equi et al. [18] to obtain an $O(mn)$ algorithm in the case where $|\Sigma| \in O(m)$; since left extensions cannot be used in the general case, we develop an equivalent left-to-right solution exploiting *meaningful right extensions* that is also correct in the case with gaps. In the general case, the number of these

extensions is $O(mn^2)$ and computing them takes $O(mn\alpha \log|\Sigma|)$ time, where α is the length of the longest run in the MSA where a row spells a prefix of the string spelled by another row and, unfortunately, $\alpha \in \Theta(n)$ in the worst case. Hence, we continue with a different generalization of the gapless height that we call *prefix-aware height*, equal to the maximum number of distinct strings in a block but omitting strings that are prefixes of others in such block. In the example of Figure 1, this measure is equal to 2. The number of meaningful right extensions is $O(mn)$ for this refined height, so in Section 4 we obtain an $O(mn)$ time solution on general MSAs: the segmentation minimizing the maximum prefix-aware height provides a useful lower bound on the optimal segmentation under the original height. The linear time is achieved thanks to the computation of the generalized suffix tree built from the MSA rows, its symmetrical prefix tree counterpart, and the constant-time navigation between the two offered by the suffix tree data structure of Belazzougui et al. answering weighted ancestor queries [2]. We leave it for future work to study whether the modified EFG suggested by our refined height, as seen in Figure 1, supports an adapted version of the index for fast pattern matching queries.

2 Definitions

We follow the notation of Equi et al. [10].

Strings. We denote integer intervals by $[x..y]$. Let $\Sigma = [1..\sigma]$ be an alphabet of size $|\Sigma| = \sigma$. A *string* $T[1..n]$ is a sequence of symbols from Σ , i.e. $T \in \Sigma^n$, where Σ^n denotes the set of strings of length n over Σ . In this paper, we assume that σ is always smaller or equal to the length of the strings we are working with. The *reverse* of T , denoted with T^{-1} , is the string T read from right to left. A *suffix* (*prefix*) of string $T[1..n]$ is $T[x..n]$ ($T[1..y]$) for $1 \leq x \leq n$ ($1 \leq y \leq n$). A *substring* of string $T[1..n]$ is $T[x..y]$ for $1 \leq x \leq y \leq n$. The *length* of a string T is denoted $|T|$ and the *empty string* ε is the string of length 0. In particular, substring $T[x..y]$ where $y < x$ is the empty string. For convenience, we denote with Σ^* and Σ^+ the set of finite strings and finite nonempty strings over Σ , respectively. We say that a substring $T[x..y]$ is *proper* if it is non-empty and different from T . String Q *occurs* in T if $Q = T[x..y]$; we say that x is the starting position (occurrence) of Q in T , and y is the ending position (ending occurrence). The *lexicographic order* of two strings A and B is naturally defined by the order of the alphabet: $A < B$ iff $A[1..y] = B[1..y]$ and $A[y+1] < B[y+1]$ for some $y \geq 0$. If $y+1 > \min(|A|, |B|)$, then the shorter one is regarded as smaller. However, we usually avoid this implicit comparison by adding an *end marker* $\$ \notin \Sigma$ to the strings and we consider $\$$ to be the lexicographically smallest character. The concatenation of strings A and B is denoted $A \cdot B$, or just AB .

Elastic founder graphs. MSAs can be compactly represented by elastic founder graphs, the vertex-labeled graphs that we formalize in this section.

A *multiple sequence alignment MSA* $[1..m, 1..n]$ is a matrix with m strings drawn from $\Sigma \cup \{-\}$, each of length n , as its rows. Here, $- \notin \Sigma$ is the *gap* symbol. For a string $X \in (\Sigma \cup \{-\})^*$, we denote $\text{spell}(X)$ the string resulting from removing the gap symbols from X . If an MSA does not contain gaps then we say it is *gapless*, otherwise we say that it is a *general MSA*. Given $I \subseteq [1..m]$, we denote with $\text{MSA}[I, 1..n]$ the MSA obtained by considering only rows $\text{MSA}[i, 1..n]$ with $i \in I$.

Let \mathcal{P} be a *partitioning* of $[1..n]$, that is, a sequence of subintervals $\mathcal{P} = [x_1..y_1], [x_2..y_2], \dots, [x_b..y_b]$ where $x_1 = 1$, $y_b = n$, and $x_j = y_{j-1} + 1$ for all $j > 2$. A *segmentation* S of $\text{MSA}[1..m, 1..n]$ based on partitioning \mathcal{P} is the sequence of b sets $S^k =$

$\{\text{spell}(\text{MSA}[i, x_k..y_k]) \mid 1 \leq i \leq m\}$ for $1 \leq k \leq b$; in addition, we require for a (proper) segmentation that $\text{spell}(\text{MSA}[i, x_k..y_k])$ is not an empty string for any i and k . We call set S^k a *block*, while $\text{MSA}[1..m, x_k..y_k]$ or just $[x_k..y_k]$ is called a *segment*. The *length* of block S^k is $L(S^k) = y_k - x_k + 1$ and its *height* is $H(S^k) = |S^k|$. Since each block is derived from a segment $[x..y]$, we denote *segment length* and *height* with $L(\text{MSA}[1..m, x..y])$ and $H(\text{MSA}[1..m, x..y])$ or just $L([x..y])$ and $H([x..y])$, respectively.

Segmentation naturally leads to the definition of a founder graph through the block graph concept.

► **Definition 1** (Block Graph). *A block graph is a graph $G = (V, E, \ell)$ where $\ell : V \rightarrow \Sigma^+$ is a function that assigns a string label to every node and for which the following properties hold:*

1. *set V can be partitioned into a sequence of b blocks V^1, V^2, \dots, V^b , that is, $V = V^1 \cup V^2 \cup \dots \cup V^b$ and $V^i \cap V^j = \emptyset$ for all $i \neq j$;*
2. *if $(v, w) \in E$ then $v \in V^i$ and $w \in V^{i+1}$ for some $1 \leq i \leq b - 1$; and*
3. *if $v, w \in V^i$ then $|\ell(v)| = |\ell(w)|$, and if $v, w \in V^i$ and $v \neq w$ then $\ell(v) \neq \ell(w)$.*

For *gapless* MSAs, block S^k equals segment $\text{MSA}[1..m, x_k..y_k]$, and in that case the *founder graph* is a block graph induced by segmentation S [18]. The idea is to have a graph in which the nodes represent the strings in S while the edges retain the information of how such strings can be recombined to spell any sequence in the original MSA.

For general MSAs with gaps, we consider the following extension.

► **Definition 2** (Elastic block and founder graphs). *We call a block graph elastic if its third condition is relaxed in the sense that each V^i can contain non-empty variable-length strings. An elastic founder graph (EFG) is an elastic block graph $G(S) = (V, E, \ell)$ induced by a segmentation S of $\text{MSA}[1..m, 1..n]$ as follows: for each $1 \leq k \leq b$ we have $S^k = \{\text{spell}(\text{MSA}[i, x_k..y_k]) \mid 1 \leq i \leq m\} = \{\ell(v) : v \in V^k\}$. It holds that $(v, w) \in E$ if and only if there exist $k \in [1..b-1]$, $i \in [1..m]$ such that $v \in V^k$, $w \in V^{k+1}$, and $\text{spell}(\text{MSA}[i, x_k..y_{k+1}]) = \ell(v)\ell(w)$.*

For example, in the general $\text{MSA}[1..4, 1..13]$ of Figure 1, the segmentation based on partitioning $[1..4]$, $[5..9]$, $[10..14]$ induces an EFG $G(S) = (V^1 \cup V^2 \cup V^3, E, \ell)$ where the nodes in V^1 , V^2 , and V^3 have labels of variable length. As noted by Equi et al. [10], block graphs are connected to Generalized Degenerate Strings [1] and elastic founder graphs are connected to Elastic Degenerate Strings [3].

By definition, (elastic) founder and block graphs are acyclic. For convention, we interpret the direction of the edges as going from left to right. Consider a path P in $G(S)$ between any two nodes. The label $\ell(P)$ of P is the concatenation of the labels of the nodes in the path. Let Q be a query string. We say that Q *occurs* in $G(S)$ if Q is a substring of $\ell(P)$ for any path P of $G(S)$.

► **Definition 3** ([18]). *EFG $G(S)$ is repeat-free if each $\ell(v)$ for $v \in V$ occurs in $G(S)$ only as a prefix of paths starting with v .*

► **Definition 4** ([18]). *EFG $G(S)$ is semi-repeat-free if each $\ell(v)$ for $v \in V$ occurs in $G(S)$ only as a prefix of paths starting with $w \in V$, where w is from the same block as v .*

For example, the EFG of Figure 1 is not repeat-free, since **AGC** occurs as a prefix of two distinct labels of nodes in the same block, but it is semi-repeat-free since all node labels $\ell(v)$ with $v \in V^k$ occur in $G(S)$ only starting from block V^k , or they do not occur at all elsewhere in the graph. These definitions also apply to general elastic block graphs and to elastic degenerate strings as their special case.

Basic tools. A *trie* or *keyword tree* [6] of a set of strings is a rooted directed tree with outgoing edges of each node labeled by distinct symbols such that there is a root-to-leaf path spelling each string in the set; the shared part of the root-to-leaf paths of two different leaves spell the common prefix of the corresponding strings. In a *compact trie*, the maximal non-branching paths of a trie become edges labeled with the concatenation of labels on the path. The *suffix tree* of $T \in \Sigma^*$ is the compact trie of all suffixes of string $T\$$. Such tree takes linear space and can be constructed in linear time so that when reading the leaves from left to right, the suffixes are listed in their lexicographic order [12]. A *generalized suffix tree* is one built on a set of m strings [15]. In this case, string T above is the concatenation of the strings after appending a unique end marker $\$i$ to each string¹, with $1 \leq i \leq m$.

Let $Q[1..m]$ be a query string. If Q occurs in T , then the *locus* or *implicit node* of Q in the suffix tree of T is (v, k) such that $Q = XY$, where X is the string spelled from the root to the parent of v and Y is the prefix of length k of the edge from the parent of v to v . The leaves of the subtree rooted at v , or *the leaves covered by v* , are then all the suffixes sharing the common prefix Q . Let aX and X be paths spelled from the root of a suffix tree to nodes v and w , respectively; then, one can store a *suffix link* from v to w . For suffix trees, a *weighted ancestor query* asks for the computation of the implicit or explicit node corresponding to substring $T[x..y]$ of the text, given x and y .

String $B[1..n]$ from a binary alphabet is called a *bitvector*. Operation $\text{rank}(B, i)$ returns the number of 1s in $B[1..i]$. Operation $\text{select}(B, j)$ returns the index i containing the j -th 1 in B . Both queries can be answered in constant time using an index constructible in linear time and requiring $o(n)$ bits of space in addition to the bitvector itself [16].

3 Construction of EFGs of minimum height

Recall that in the absence of gaps the semi-repeat-free and repeat-free notions (Definitions 3 and 4) are equivalent, and the strings in a block induced by a segment cannot have variable length. In this section, we study the construction of EFGs under the goal of minimizing the maximum block height. Indeed, after showing that semi-repeat-free EFGs are easy to index for fast pattern matching, Equi et al. [10] extended the previous results for the gapless setting showing that semi-repeat-free EFGs are equivalent to specific segmentations of the MSA: the semi-repeat-free property has to be checked only against the MSA, and not the final EFG. We recall these arguments in Section 3.1, along with the resulting recurrence to compute an optimal segmentation under three score functions: *i.* maximizing the number of blocks; *ii.* minimizing the maximum length of a block; and *iii.* minimizing the maximum height of a block.

In the gapless repeat-free setting, scores *i.* and *ii.* admit the construction of indexable founder graphs in $O(mn)$ time, thanks to previous research on founder graphs and MSA segmentations [18, 21, 4]. In Section 3.2 we combine these works to obtain an $O(mn)$ time solution for score *iii.* as well: the optimal segmentation is found mainly by computing the *meaningful left extensions*, that is, the positions $x_1 > \dots > x_k$ where the height of repeat-free segment $[x_i..y]$ increases, with $y \in [1..n]$. In the general and semi-repeat-free setting, extending a segment to the left can violate the semi-repeat-free property and the height can decrease. Thus, Equi et al. in [10] gave $O(n)$ - and $O(n \log \log n)$ -time algorithms for scores *i.* and *ii.*, respectively, exploiting the semi-repeat-free *right extensions* after a

¹ For our purposes, the suffix tree of the concatenated strings is functionally equivalent to the “trimmed” generalized suffix tree seen in Figure 3.

common $O(mn \log m)$ -time preprocessing of the MSA. We recently improved the second algorithm to $O(n)$ time and the preprocessing to $O(mn)$, reaching global linear time [22]. In Section 3.3, we develop a similar algorithm for the construction of a semi-repeat-free segmentation processing the *meaningful right extensions*. Although the number of these extensions is $O(n^2)$ in total, we manage to provide a parameterized linear-time solution providing an upper bound based on the length of the longest run where any two rows spell strings that are one prefix of the other. Instead, an alternative notion of height, the *prefix-aware height*, generates $O(mn)$ *meaningful prefix-aware right extensions*: they can be processed in the same fashion as the original height to obtain an optimal segmentation, and we will show how to compute them efficiently in Section 4.

3.1 Optimal EFGs correspond to optimal segmentations

Consider a segmentation $S = S^1, S^2, \dots, S^b$ inducing a semi-repeat-free EFG $G(S) = (V, E, \ell)$, as per Definition 2. It is easy to see that the strings occurring in $G(S)$ are a superset of the substrings of the MSA rows: for example, string CACTAGA occurs in the EFG of Figure 1 but it does not occur in any row of the original MSA. These new strings, as it was proven by Mäkinen et al. [18] and Equi et al. [10], do not affect the semi-repeat-free property. Intuitively, this is because they involve three or more vertices of $G(S)$.

► **Lemma 5** (Characterization, gapless setting [18]). *We say that a segment $[x..y]$ of a gapless MSA $[1..m, 1..n]$ is repeat-free if string $\text{MSA}[i, x..y]$ occurs in the MSA only at position x of any row. Then $G(S)$ is repeat-free if and only if all segments of S are repeat-free.*

► **Lemma 6** (Characterization [10]). *We say that segment $[x..y]$ of a general MSA $[1..m, 1..n]$ is semi-repeat-free if for any $i, i' \in [1..m]$ string $\text{spell}(\text{MSA}[i, x..y])$ occurs in gaps-removed row $\text{spell}(\text{MSA}[i', 1..n])$ only at position $g(i', x)$, where $g(i', x)$ is equal to x minus the number of gaps in $\text{MSA}[i', 1..x - 1]$. Similarly, $[x..y]$ is repeat-free if the possible occurrence of $\text{spell}(\text{MSA}[i, 1..n])$ in row i' at position $g(i', x)$ also ends at position $g(i', y)$. Then $G(S)$ is semi-repeat-free if and only if all segments of S are semi-repeat-free.*

Thanks to Lemmas 5 and 6, we can compute recursively the score $s(j)$ of an optimal segmentation of prefix $\text{MSA}[1..m, 1..j]$ under our three scoring schemes, using semi-repeat-free segments, that is, respecting the global semi-repeat-free property on the whole MSA:

$$s(j) = \bigoplus_{\substack{j' : 0 \leq j' < j \text{ s.t.} \\ \text{MSA}[1..m, j'+1..j] \text{ is} \\ \text{semi-repeat-free}}} E(s(j'), j', j) \quad (1)$$

where operator \bigoplus and function E extend the optimal partial solutions, and they depend on the desired scoring scheme. Indeed, for $s(j)$ to be equal to the optimal score of a segmentation: *i.* maximizing the number of blocks, set $\bigoplus = \max$ and $g(s(j'), j', j) = s(j') + 1$; for a correct initialization set $s(0) = 0$ and where there is no semi-repeat-free segmentation set $s(j) = -\infty$; *ii.* minimizing the maximum block length, set $\bigoplus = \min$ and $g(s(j'), j', j) = \max(s(j'), L([j' + 1..j])) = \max(s(j'), j - j')$; set $s(0) = 0$ and if there is no semi-repeat-free segmentation set $s(j) = +\infty$. *iii.* minimizing the maximum block height, set $\bigoplus = \min$ and $g(s(j'), j', j) = \max(s(j'), H([j' + 1..j]))$; set $s(0) = 0$ and if there is no semi-repeat-free segmentation set $s(j) = +\infty$.

3.2 The linear time solution for the gapless setting

For gapless MSAs, an $O(mn)$ solution for the construction of segmentations minimizing the maximum block height has been found by Norri et al. [21] for the case where the length of a block is limited by a given lower bound L , rather than with the repeat-free property. This result holds under the assumption that Σ is an integer alphabet of size $O(m)$. In this section, we combine the algorithm by Norri et al. with the computation of values $v(j)$ – that we call the *minimal left extensions* – by Mäkinen et al. [18], obtaining a linear-time solution to the construction of repeat-free founder graphs minimizing the maximum block height.

► **Observation 7** (Monotonicity of left extensions [21, 8]). *Given a gapless MSA[1..m, 1..n], for any $1 \leq x \leq y \leq n$ we say that $[x..y]$ is a left extension of suffix MSA[1..m, y + 1..n]. Then:*

- if $[x..y]$ is repeat-free then $[x'..y]$ is repeat-free for all $x' < x$;
- $m \geq H([x'..y]) \geq H([x..y])$ for all $x' < x$.

Thus, for each $j \in [1..n]$ we define value $v(j)$ as the greatest column index smaller or equal to j such that $[v(j)..j]$ is repeat-free, and we say that $v(j)$ or $[v(j)..j]$ is the minimal left extension of MSA[1..m, j + 1..n]. If there is no valid left extensions then $v(j) = -\infty$.

► **Definition 8** (Meaningful left extensions [21, 8]). *Given a gapless MSA[1..m, 1..n], for any $j \in [1..n]$ we denote with $L_j = \ell_{j,1}, \dots, \ell_{j,c_j}$ the meaningful (repeat-free) left extensions of MSA[1..m, j + 1..n], meaning the strictly decreasing sequence of all positions smaller than or equal to j such that:*

- $\ell_{j,c_j} < \dots < \ell_{j,2} < \ell_{j,1} = v(j)$, so that L_j captures all repeat-free left extensions of MSA[1..m, j + 1..n];
- $H([\ell_{j,k}..j]) > H([\ell_{j,k+1}..j])$ for $2 \leq k \leq c_j$, so that each $\ell_{j,k}$ marks a column where the height of the left extension increases; it follows from Observation 7 that $|L_j| = c_j \leq m$.

If MSA[1..m, j + 1..n] has no repeat-free left extension, we define $L_j = ()$ and $c_j = 0$. Otherwise, for completeness we define $\ell_{j,c_j+1} = -1$.

Under score *iii*. Equation (1) can be rewritten using $L_j = \ell_{j,1}, \dots, \ell_{j,c_j}$ as follows:

$$s(j) = \min_{k \in [1..c_j]} \max \left(\min_{j' \in [\ell_{j,k+1}+1.. \ell_{j,k}]} s(j'), H([\ell_{j,k}..j]) \right) \quad (2)$$

and $s(j) = +\infty$ if $c_j = 0$, so knowing values L_j , $H([\ell_{j,k}..j])$, and $\min_{j' \in [\ell_{j,k+1}+1.. \ell_{j,k}]} s(j')$ for $k \in [1..c_j]$ makes it possible to compute $s(j)$ in $O(m)$ time. On one hand, given a fixed length L , Norri et al. [21] developed an algorithm to compute these values under the variant of Definition 8 considering segments of length at least L – instead of repeat-free segments – in $O(mn)$ total time. On the other hand, Mäkinen et al. [18] developed a linear-time algorithm to compute values $v(j)$ of a gapless MSA. The two solutions can be combined by finding values $v(j)$ with the latter, and by using the values as a dynamic lower bound on the minimum accepted segment length. Since the algorithm we develop in Section 3.3 for the general setting also solves this problem, using the symmetrically defined right extensions, we will not describe such modification in this paper.

► **Theorem 9.** *Given a gapless MSA[1..m, 1..n] from an integer alphabet Σ of size $O(m)$, an optimal repeat-free segmentation of MSA[1..m, 1..n] minimizing the maximum block height can be computed in time $O(mn)$.*

3.3 Revisiting the linear time solution for right extensions

For MSAs with gaps and under the semi-repeat-free notion, the monotonicity of left extensions (Observation 7) fails [11, Table 1]: fixing $j \in [1..n]$, left-extensions $\text{MSA}[1..m, x..y]$ are not always semi-repeat-free, or *valid*, from $x = v(j)$ backwards, and their height could decrease when extending a valid segment. For example, in the MSA of Figure 1, segment $[5..9]$ is semi-repeat-free but segment $[4..9]$ is not, and $H([5..9]) < H([6..9])$. In this section, we resolve the former of the two issues, developing an algorithm exploiting right extensions and computing the optimal MSA segmentation from left to right, in the same fashion as [10, Algorithms 1 and 2] and [23, Algorithm 2]. We will discuss the complexity of computing these right extensions in Section 3.4.

► **Observation 10** (Semi-repeat-free right extensions [10]). *Given general MSA[1..m, 1..n], for any $0 \leq x < y \leq n$ we say that $[x + 1..y]$ is an extension of prefix MSA[1..m, 1..x]. If segment $[x + 1..y]$ is semi-repeat-free, then segment $[x + 1..y']$ is semi-repeat-free for all $y' > y$. Thus, for each $x \in [0..n - 1]$ we define value $f(x)$ as the smallest column index greater than x such that $[x + 1..f(x)]$ is semi-repeat-free, and we say that $f(x)$ or $[x + 1..f(x)]$ is the minimal right extension of MSA[1..m, 1..x]. If there is no valid right extension, then $f(x) = \infty$.*

► **Definition 11** (Meaningful right extensions). *Given general MSA[1..m, 1..n], for any $x \in [0..n - 1]$ we denote with $R_x = r_{x,1}, \dots, r_{x,d_x}$ the meaningful (semi-repeat-free) right extensions of MSA[1..m, 1..x], meaning the strictly increasing sequence of all positions greater than x such that:*

- $f(x) = r_{x,1} < r_{x,2} < \dots < r_{x,d_x}$, so that R_x captures all semi-repeat-free right extensions of MSA[1..m, 1..x];
- $H([x + 1..r_{x,k}]) \neq H([x + 1..r_{x,k} - 1])$ for $2 \leq k \leq d_x$, so that each $r_{x,k}$ marks a column where the height of the right extensions changes.

If MSA[1..m, 1..x] has no semi-repeat-free right extension, then $R_x = ()$ and $d_x = 0$. Otherwise, for completeness we define value $r_{x,d_x+1} = n + 1$.

Since we will treat all R_0, \dots, R_{n-1} together, we complement each value $r_{x,k}$ with column x and the height of the corresponding MSA segment, obtaining triple $(x, r_{x,k}, H([x + 1..r_{x,k}]))$.

Thus, under score *iii*. Equation (1) can be rewritten as follows:

$$s(j) = \min_{\substack{x \in [0..j-1], k \in [1..d_x]: \\ r_{x,k} \leq j < r_{x,k+1}}} \max \left(s(x), H([x + 1..r_{x,k}]) \right). \quad (3)$$

Since each R_x defines non-overlapping ranges $[r_{x,k}, r_{x,k+1} - 1]$ over $[1, n]$, at most one range $[r_{x,k}, r_{x,k+1} - 1]$ per R_x with $x < j$ is involved in the computation of $s(j)$, and the corresponding score depends on which range contains j . Also, note that Equation (3) is simpler than Equation (2). Finally, the algorithm computing the score of an optimal semi-repeat-free segmentation minimizing the maximum block height is described in Algorithm 1, and it works by processing all meaningful right extensions in R_0, \dots, R_{n-1} expressed as triples (x, r, h) and sorted from smallest to largest order by second component. The main strategy is to keep at each iteration j the best scores of the semi-repeat-free segmentations of MSA[1..m, 1..j] ending with a right extension $[1, j]$, $[2, j]$, \dots , or $[j, j]$ described by ranges in R_0, R_1, \dots , or R_{j-1} . Checking each currently valid range individually would result in a quadratic-time solution, so we need to represent these ranges in some other form. Indeed, by counting these scores with an array $\mathbf{C}[1..m]$ such that $\mathbf{C}[i]$ is equal to the number of available solutions having score i , score $s(j)$ can be computed by finding the smallest i such that $\mathbf{C}[i]$ is greater than zero. Array \mathbf{C} needs to be updated only when j reaches some $r_{x,k}$; in other words,

■ **Algorithm 1** Main algorithm to find the optimal score of a semi-repeat-free segmentation minimizing the maximum block height.

Input: Meaningful right extensions $(x_1, r_1, h_1), \dots, (x_k, r_k, h_k)$ sorted from smallest to largest order by second component.

Output: Score of an optimal semi-repeat-free segmentation minimizing the maximum block height.

- 1 Initialize array $R[0..n-1]$ with values in $[0..m] \cup \{\perp\}$ and set all values to \perp ;
- 2 Initialize array $C[1..m]$ with values in $[0..m]$ and set all values to 0;
- 3 $y \leftarrow 1$;
- 4 $\text{minmaxheight}[0] \leftarrow 0$;
- 5 **for** $j \leftarrow 1$ **to** n **do**
- 6 **while** $j = r_y$ **do**
- 7 **if** $R[x_y] \neq \perp$ **then**
- 8 $C[R[x_y]] \leftarrow C[R[x_y]] - 1$; ▷ Remove last solution of R_{x_y}
- 9 $s \leftarrow \max(\text{minmaxheight}[x_y], h_y)$;
- 10 $R[x_y] \leftarrow s$; ▷ Save score corresponding to (x_y, r_y, h_y)
- 11 $C[s] \leftarrow C[s] + 1$; ▷ Add solution corresponding to (x_y, r_y, h_y)
- 12 $y \leftarrow y + 1$;
- 13 $\text{minmaxheight}[j] \leftarrow \min_{i=1}^m \{i : C[i] > 0\}$;
- 14 **return** $\text{minmaxheight}[n]$;

when $j = r$ for some (x, r, h) , the score $\max(s(x), h)$ of an optimal segmentation ending with $[x+1..j]$ must be added to C , and the old score relative to the previous range of R_x must be removed. We can keep track of the scores in an array $R[0..n-1]$ such that $R[x]$ is equal to the score associated with the currently valid extension of R_x . A possible implementation of the solution is described in Algorithm 1. To compute the actual segmentation, instead of just its score, we can use two backtracking arrays B and C_{bt} : $C_{\text{bt}}[i] = x$ where $[r_{x,k}..r_{x,k+1} - 1]$ is a currently valid range of minimum score finishing last, that is, with maximum value of $r_{x,k+1}$; values in C_{bt} can be used to compute $B[j]$, equal to x where $[x+1..j]$ is the last segment of an optimal solution for $\text{MSA}[1..m, 1..j]$. Then, B reconstructs an optimal segmentation. Algorithm 1 can be easily modified to update these arrays, if each meaningful right extension $(x, r_{x,k}, H([x+1..r_{x,k}]))$ is augmented with value $r_{x,k+1} - 1$.

► **Lemma 12.** *Given the meaningful right extensions R_0, R_1, \dots, R_{n-1} of $\text{MSA}[1..m, 1..n]$, we can compute the optimal semi-repeat-free segmentation minimizing the maximum block height in time $O(mn + R)$, with $R := \sum_{x=0}^{n-1} |R_x|$.*

Proof. The correctness follows from Equation (3) and from the arguments above. Sorting the meaningful right extensions (x, r, h) by their second component can be done in time $O(n + R)$, as the meaningful right extensions take value in $[1..n]$. Moreover, the management of arrays R and C takes constant time per meaningful right extensions, and the computation of each $s(j)$ takes $O(m)$ time, reaching the time complexity of $O(mn + R)$. ◀

For the gapless case, this is an alternative solution to that of Section 3.2, since $R \in O(mn)$ and the algorithms by Norri et al. and Equi et al. can be used to compute the meaningful right extensions.

3.4 The complexity of minimizing the maximum block height

As Lemma 12 states, we can process the meaningful right extensions of Definition 11 to compute the score of an optimal segmentation minimizing the maximum block height. Unfortunately, in the general setting with gaps, the total number of meaningful right extensions is $O(n^2)$: as it can be seen in Figure 2, if any two row suffixes starting from the same column x spell the same string but the spelling is interleaved by gaps, then the height of segment $[x..y]$ can change at any column y ; this pattern could involve any two rows in any segment of a general $\text{MSA}[1..m, 1..n]$.

		1	2	3	4	5	6	7	8		$n-2$	n	
	1	T	-	A	-	A	-	A	-	...	A	-	C
	2	T	-	-	A	-	A	-	A		-	A	C
$H([1..y])$	1	1	2	1	2	1	2	1	1	...	2	1	1

■ **Figure 2** Example of $\text{MSA}[1..2, 1..n]$ such that $|R_0| \in O(n)$.

► **Observation 13.** *Given $\text{MSA}[1..m, 1..n]$ over alphabet $\Sigma \cup \{-\}$, we have that $H([x..y]) > H([x..y+1])$ only if there exist rows $i, i' \in [1..m]$ such that $\text{MSA}[i, y+1] = -$, $\text{MSA}[i', y+1] = c$, and $\text{spell}(\text{MSA}[i, x..y]) = \text{spell}(\text{MSA}[i', x..y+1]) = S \cdot c$, with $c \in \Sigma$ and $S \in \Sigma^+$.*

The example of Figure 2 and the context described by Observation 13 seem intuitively artificial, as a high-scoring MSA would try to align the rows to avoid such a situation. Nonetheless, without further assumptions about gaps in the MSA portions reading the same strings, we are left to compute all meaningful right extensions. Indeed, let $K_{x..y}$ be the keyword tree of the set of strings $S_{x..y} := \{\text{spell}(\text{MSA}[i, x..y]) : 1 \leq i \leq m\}$; since $|S_{x..y}| = H([x..y])$, the height of $[x..y]$ is equal to the number of distinct nodes of $K_{x..y}$ corresponding to the strings in $S_{x..y}$. We can obtain a parameterized solution by noting that if $K_{x..y}$ has m leaves then no two strings in $S_{x..y}$ are one prefix of the other and $H([x..y']) = m$ for all $y' > y$.

► **Lemma 14.** *Given general $\text{MSA}[1..m, 1..n]$ over integer alphabet $\Sigma \cup \{-\}$ of size $\sigma \in O(mn)$, we denote with α the maximum length $y - x + 1$ of any segment $[x..y]$ such that $\text{spell}(\text{MSA}[i, x..y])$ is a prefix of $\text{spell}(\text{MSA}[i', x..y])$ for some $i, i' \in [1..m]$. Then, we can compute all meaningful right extensions in time $O(mn\alpha \log \sigma)$.*

Proof. For each $x \in [0..n-1]$, we can find R_x by incrementally computing trees $K_{x+1..x+1}$, $K_{x+1..x+2}$, \dots , $K_{x+1..n}$ using a dynamic keyword tree \mathcal{T} supporting the traversal from the root to the leaves and the insertion of a c -child to an arbitrary node v , with $c \in \Sigma$. A possible implementation of the procedure is described by Algorithm 3 in Appendix A. During the computation, array $V[1..m]$ keeps track of the nodes corresponding to strings $\text{spell}(\text{MSA}[i, x+1..r])$, each variable $v.\text{count}$ counts the number of rows reading the corresponding string, and a variable h counts the number of distinct nodes v such that $v.\text{count}$ is greater than zero; if h changes then the corresponding meaningful right extension of $[x+1..r]$ is (x, r, h) . For each R_x , the algorithm stops if the number of leaves of \mathcal{T} is m , that is it computes at most α keyword trees; no meaningful right extension is missed, thanks to Observation 13 and the above arguments. We can compute R_0, \dots, R_{n-1} in time $O(mn\alpha \log \sigma)$, since the traversal and insertion operations of \mathcal{T} can be implemented in time $O(\log \sigma)$,² the other operations can be supported in constant time. ◀

² Since only insertion is needed and alphabet Σ is fixed, the addition of a children to each node v can be implemented with a dynamic binary tree with height at most $\lceil \log_2 \sigma \rceil$ leaves, growing downwards as the number of children grows.

Since the total number of meaningful right extensions is $O(mn\alpha)$, Lemma 14 and Lemma 12 give the following solution to our segmentation problem.

► **Theorem 15.** *Given general MSA[1..m, 1..n] over an integer alphabet Σ of size $O(mn)$, we can compute the score of an optimal segmentation minimizing the maximum block height in time $O(mn\alpha \log \sigma)$, where α is the length of the longest MSA segment where any two rows spell strings S, S' such that S is a prefix of S' .*

In the worst case, the number of meaningful right extension is $O(mn^2)$, $\alpha \in \Theta(n)$, and the time complexity of Lemma 14 is $\Theta(mn^2 \log \sigma)$: thus, we introduce a different generalization of block height from the gapless setting to the general one.

► **Definition 16 (Prefix-aware height).** *Given MSA[1..m, 1..n], we define the prefix-aware height of a segment $[x..y]$, denoted as $\overline{H}(\text{MSA}[1..m, x..y])$ or just $\overline{H}([x..y])$, as the number of distinct strings S in $\{\text{spell}(\text{MSA}[i, x..y]) : 1 \leq i \leq m\}$ such that S is not a prefix of some other string of the set.*

Since $\overline{H}([x..y])$ is equal to $H([x..y])$ minus the number of strings spelled in $[x..y]$ that are proper prefixes of other strings of the segment, this refined height is always smaller or equal to the original height: the relative optimal segmentation provides a lower bound for the maximum height in the original setting. Moreover, the necessary condition for the decrease in height stated in Observation 13 is no longer valid, and it is easy to see that the monotonicity of prefix-aware right extensions holds (see Observation 7). Indeed, if we define the *meaningful prefix-aware right extensions* $\overline{R}_0, \dots, \overline{R}_{n-1}$ as in Definition 11, it is easy to see that $|\overline{R}_x| \leq m + 1$ for all $x \in [0..n - 1]$, so the number of these extensions is $O(mn)$ in total. Finally, given $\overline{R}_0, \dots, \overline{R}_{n-1}$ as input, Algorithm 1 correctly computes the score of an optimal segmentation under our refined height, since Equation (3) still holds. In Section 4, we will provide an algorithm based on the generalized suffix tree of the gaps-removed MSA rows computing the prefix-aware extensions in time linear in the MSA size, obtaining the following result.

► **Theorem 17.** *Given MSA[1..m, 1..n] over integer alphabet $\Sigma \cup \{-\}$ of size $\sigma \leq mn$, computing a semi-repeat-free segmentation minimizing the maximum prefix-aware block height takes $O(mn)$ time.*

4 Preprocessing the MSA for the prefix-aware height

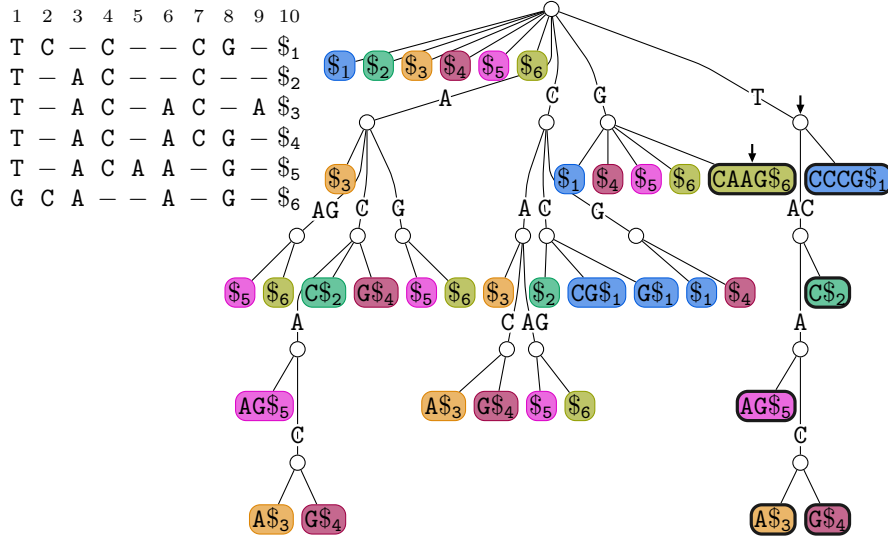
As stated in Section 3.4, the meaningful prefix-aware right extensions are $O(mn)$ in total, so the goal of this section is to compute them in time linear in the MSA size. First, in Section 4.1 we provide an overview of the $O(mn)$ time computation of the minimal right extensions $f(x)$ (Observation 10), that we recently obtained in [22]. The solution consists in solving multiple instances of the *exclusive ancestor problem* – a novel ancestor problem on trees asking for the shallowest nodes covering all and only the given set of leaves – on the following structure, built from the gaps-removed rows of the MSA.

► **Definition 18 ([10, 22]).** *Given a general MSA[1..m, 1..n] from alphabet Σ , we define GST_{MSA} as the generalized suffix tree of the set of strings $\{\text{spell}(\text{MSA}[i, 1..n]) \cdot \$i : 1 \leq i \leq m\}$, with $\$, \dots, \m m new distinct terminator symbols not in Σ .*

An example of GST_{MSA} is given in Figure 3. Then, in Section 4.2, we extend these techniques to show that the forests inside GST_{MSA} identified by the exclusive ancestors can describe the meaningful prefix-aware right extensions: by computing for each node of these forests the

19:12 Elastic Founder Graphs of Minimum Height

position indicating where the first occurrence of the related string ends, and by sorting these positions, we can compute the meaningful right extension in $O(m^2n)$ global time. Finally, in Section 4.3 we describe how these positions can be computed efficiently, thanks to the *generalized prefix tree* of the gap-removed rows and the data structure for weighted ancestor queries of Belazzougui et al. [2]. This structure, after its linear-time construction, makes it possible to navigate from the suffix tree to the prefix tree in constant time, reaching global $O(mn)$ time.



■ **Figure 3** Example of an $\text{MSA}[1..6, 1..9]$ and its GST_{MSA} , where the label to each leaf has been moved inside the leaf itself. Leaves are colored according to the corresponding row. We have also highlighted, with a black outline, the leaves \mathcal{L}_0 corresponding to suffixes $\text{spell}(\text{MSA}[i, 1..n])$; their exclusive ancestors W_0 , the nodes corresponding to $G \cdot \text{CAAG}\$6$ and T , are marked with arrows.

4.1 Computing the minimal right extensions

Consider the generalized suffix tree GST_{MSA} (Definition 18) built from the gaps-removed rows $S_i := \text{spell}(\text{MSA}[i, 1..n])\i , for $i \in [1..m]$. Each suffix $S_i[x..|S_i|]$ corresponds to a unique leaf $\ell_{i,x}$ of GST_{MSA} and vice versa, for $1 \leq x \leq |S_i|$. Moreover, each substring $S_i[x..y]$ corresponds to an explicit or implicit node of GST_{MSA} in the root-to- $\ell_{i,x}$ path, and each explicit or implicit node v of GST_{MSA} corresponds to one or more such substrings, described by the leaves of $\text{GST}_{\text{MSA}}(v)$, the subtree rooted at v . Note that in GST_{MSA} we stripped away essential gap information: we will implicitly add it back by considering a certain set of leaves and of nodes, corresponding to the strings occurring at a certain MSA column x .

But first, the notion of semi-repeat-free segment can be broken down into each single row.

► **Definition 19** (Semi-repeat-free substring [22]). *Given a substring $\text{MSA}[i, x..y]$ such that $\text{spell}(\text{MSA}[i, x..y]) \in \Sigma^+$, we say that $\text{MSA}[i, x..y]$ is semi-repeat-free if, for all $1 \leq i' \leq m$, string $\text{spell}(\text{MSA}[i, x..y])$ occurs in gaps-removed row $\text{spell}(\text{MSA}[i', 1..n])$ only at position $g(i', x)$ (defined as in Lemma 6), or it does not occur at all.*

Indeed, Observation 10 holds also for single rows and we can split the computation of $f(x)$, the smallest integer making segment $[x + 1, f(x)]$ semi-repeat-free: if substring $\text{MSA}[i, x..y]$ is semi-repeat-free, then $\text{MSA}[i, x..y']$ is semi-repeat-free for all $y' > y$. For each $x \in [0..n - 1]$ we can define $f^i(x)$ as the smallest column index greater than x such that $\text{MSA}[i, x + 1..f^i(x)]$ is a semi-repeat-free substring. Then, it is easy to see that $f(x) = \max_{i=1}^m f^i(x)$.

Let \mathcal{L}_x be the set of leaves of GST_{MSA} corresponding to suffixes $\text{spell}(\text{MSA}[i, x+1..n]) \cdot \$_i$, for $i \in [1..m]$. In [22] we proved that the *exclusive ancestors* of these leaves, defined as the shallowest of their ancestors covering only leaves in \mathcal{L}_x , correspond to the shortest semi-repeat-free strings starting from column $x+1$. For example, in the MSA of Figure 3 there are two exclusive ancestors for \mathcal{L}_0 , corresponding to strings $\text{G} \cdot \text{CAAG}\6 and T ; for \mathcal{L}_2 they are four and correspond to strings $\text{AAG} \cdot \$6$, $\text{AC} \cdot \text{A}$, $\text{AC} \cdot \text{C}\2 , and $\text{CC} \cdot \text{G}\1 . Let W_x be the set of exclusive ancestors of \mathcal{L}_x : for each leaf $\ell_{i,x+1}$ covered by some $w \in W_x$, the first character of the label from w 's parent to w in GST_{MSA} – the relative node is implicit or is w itself – corresponds to the smallest semi-repeat-free prefix of $\text{MSA}[i, x+1..n]$.

Thus, we have that $f^i(x)$ corresponds to the k -th non-gap symbol of row i , with $k = \text{rank}(\text{MSA}[i, 1..n], x) + \text{stringdepth}(\text{parent}(w)) + 1$, where $\text{rank}(\text{MSA}[i, 1..n], x)$ is the number of non-gap symbols in $\text{MSA}[i, 1..x]$ and $\text{stringdepth}(u) = |\text{string}(u)|$. For example, in the MSA of Figure 3 we have that values $f^i(0)$ for $i \in [1..6]$ are equal to 1, 1, 1, 1, 1, and 2, and values $f^i(2)$ are equal to 8, 7, 6, 6, 5, and 10; in particular, $f^1(2) = 8$ because CCG is the shortest semi-repeat-free substring of $\text{MSA}[1, 3..10]$, and the last G of CCG corresponds to column position 8 in $\text{MSA}[1, 1..10]$. Each \mathcal{L}_x can be transformed into \mathcal{L}_{x+1} by following the suffix links of rows $i \in [1..m]$ such that $\text{MSA}[i, x+1] \neq -$, and the exclusive ancestor set problem on each \mathcal{L}_x can be solved in time $O(m)$: the minimal right extensions can be computed in time $O(mn)$, provided Σ is an integer alphabet of size $\sigma \leq mn$ [22].

4.2 Computing the meaningful prefix-aware right extensions

Given GST_{MSA} and its leaves \mathcal{L}_x corresponding to the suffixes starting at column $x+1$, the forest with m leaves identified by the exclusive ancestors W_x of \mathcal{L}_x can also be used to study the meaningful prefix-aware right extensions \overline{R}_x (Definitions 11 and 16).

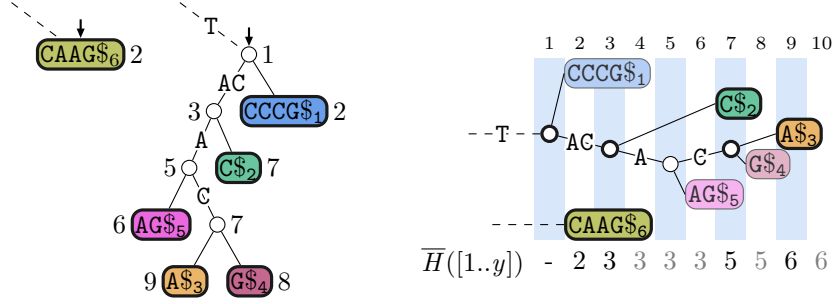
► **Definition 20** (First ending occurrence). *Given GST_{MSA} , let F_x be the set of all explicit nodes of GST_{MSA} belonging to the subtree rooted at some exclusive ancestor $w \in W_x$. Then, for each $v \in F_x$ we define value $\text{pos}(v)$ as the first ending occurrence of string $\text{string}(\text{parent}(v)) \cdot \text{char}(v)$ in the MSA, where $\text{char}(v)$ is the first character of the label from v 's parent to v . In other words, if $S = \text{string}(\text{parent}(v)) \in \Sigma^+$ and $c = \text{char}(v) \in \Sigma$, then $\text{pos}(v)$ is the minimum column index $y \in [1..n]$ such that $Sc = \text{spell}(\text{MSA}[i, x+1..y])$ for some $1 \leq i \leq m$.*

An example of set F_x and of values $\text{pos}(v)$ is shown in Figure 4. After plotting these values in a horizontal line, it is easy to notice that all increases in \overline{H} correspond to some $\text{pos}(v)$, but not the other way around: the pos values that do not affect \overline{H} , because they do not correspond to two or more rows reading strings that are not one prefix of the other, are the first-born children – with respect to the value of pos – of branching nodes.

► **Definition 21** (First-born nodes). *Given GST_{MSA} and its forest F_x corresponding to all semi-repeat-free strings starting from column $x+1$, with $0 \leq x < n$, for each internal node $v \in F_x$ we arbitrarily choose one of its children with minimum value of pos to be a first-born node of F_x . Then, let \widehat{F}_x be the subset of non-first-born nodes of F_x , obtained by removing these nodes from F_x .*

► **Lemma 22.** *Given GST_{MSA} and the set of non-first-born nodes \widehat{F}_x associated with column x , for any $y \in [f(x)..n]$ the prefix-aware height of segment $[x+1..y]$ is equal to the number of nodes in \widehat{F}_x having pos value equal or smaller than y , in symbols $\overline{H}([x+1..y]) = |\{\widehat{v} \in \widehat{F}_x : \text{pos}(\widehat{v}) \leq y\}|$.*

19:14 Elastic Founder Graphs of Minimum Height



■ **Figure 4** On the left, the forest F_0 of the example MSA of Figure 3, annotated with values $\text{pos}(v)$. On the right, the same forest plotted against the MSA columns, with only the non-first-born nodes of \widehat{F}_0 highlighted. Note that $f(0) = f^6(0) = 2$.

Proof. For any $v \in F_x$, let I_v be the set of indexes of the rows whose suffix is covered by v . From the properties of GST_{MSA} it follows that if any $v_1, v_2 \in F_x$ are not one ancestor of the other, then the corresponding strings $\text{string}(\text{parent}(v_1)) \cdot \text{char}(v_1)$ and $\text{string}(\text{parent}(v_2)) \cdot \text{char}(v_2)$ are not one prefix of the other: if $y \geq \text{pos}(v_1)$ and $y \geq \text{pos}(v_2)$, then $\overline{H}(\text{MSA}[I_{v_1} \cup I_{v_2}, x + 1..y]) = \overline{H}(\text{MSA}[I_{v_1}, x + 1..y]) + \overline{H}(\text{MSA}[I_{v_2}, x + 1..y])$. We call this key property the *independence of collateral relatives*.³ In particular, the property holds for any subset U of children of some node $v \in F_x$, provided $y \geq \max_{u \in U} \text{pos}(u)$, and it holds for the exclusive ancestors of $W_x \subseteq F_x$, because $y \geq f(x) \geq \max_{w \in W_x} \text{pos}(w)$:

$$\overline{H}(\text{MSA}[1..m, x + 1..y]) = \sum_{w \in W_x} \overline{H}(\text{MSA}[I_w, x + 1..y]). \quad (4)$$

We can now prove the modification of the thesis restricted to the rows I_v of any node $v \in F_x$. To do so, we introduce one final notation: we denote with \widehat{F}_x^v the set $(\widehat{F}_x \cap \text{GST}_{\text{MSA}}(v)) \cup \{v\}$, that also deals with the case when v is a first-born node. Then, for any $v \in F_x$ and $y \in [\max_{i \in I_v} f^i(x)..n]$ we have that

$$\overline{H}(\text{MSA}[I_v, x + 1..y]) = \begin{cases} 1 & \text{if } y < \text{pos}(v), \\ |\{\hat{v} \in \widehat{F}_x^v : \text{pos}(\hat{v}) \leq y\}| & \text{otherwise.} \end{cases} \quad (5)$$

The proof of Equation (5) proceeds by induction on the height of the subtree rooted at v .

Base case: If v is a leaf then $\widehat{F}_x^v = \{v\}$, $I_v = \{i\}$ for some $i \in [1..m]$, and $\overline{H}(\text{MSA}[\{i\}, x + 1..y]) = 1$, so Equation (5) is easily verified.

Inductive hypothesis: Equation (5) holds for all nodes v such that the subtree rooted at v has height less than or equal to $h \geq 0$.

Inductive step: Let the height of the subtree rooted at v be equal to $h + 1$, and let u_1, \dots, u_p be the $p \geq 2$ children of v , with u_1 the first-born. If $y < \text{pos}(v)$ then all occurrences of $Sc = \text{string}(\text{parent}(v)) \cdot \text{char}(v)$ in the MSA end after column y , so all strings $\text{spell}(\text{MSA}[i, x + 1..y])$ with $i \in I_v$ are prefixes of Sc and $\overline{H}(\text{MSA}[I_v, x + 1..y]) = 1$. Using the same argument, Equation (5) is also verified if $y < \text{pos}(u_1)$, so we can assume $y \geq \text{pos}(u_1) >$

³ In genealogical terms, the ancestor relationship is described as a direct line, opposed to a collateral line for relatives that are not in a direct line.

$\text{pos}(v)$. Consider the children u_k of v such that $y < \text{pos}(u_k)$, for $2 \leq k \leq p$; the strings spelled in the corresponding rows I_{u_k} are prefixes of $\text{string}(\text{parent}(u_1))$, so they are ignored in the prefix-aware height. If $U_{\leq} := \{u_k : 1 \leq k \leq p \wedge \text{pos}(u_k) \leq y\}$ then

$$\begin{aligned} \overline{H}(\text{MSA}[I_v, x + 1..y]) &= \overline{H}\left(\text{MSA}\left[\bigcup_{u \in U_{\leq}} I_u\right][x + 1..y]\right) \\ &= \sum_{u \in U_{\leq}} \overline{H}(\text{MSA}[I_u, x + 1..y]) && \text{indep. collateral relatives} \\ &= \sum_{u \in U_{\leq}} |\{\hat{u} \in \hat{F}_x^u : \text{pos}(\hat{u}) \leq y\}| && \text{inductive hypothesis} \\ &= |\{\hat{v} \in \hat{F}_x^v : \text{pos}(\hat{v}) \leq y\}|. \end{aligned}$$

Note that the last equality holds because $\text{pos}(u_1)$ of $\hat{F}_x^{u_1}$ is replaced by $\text{pos}(v)$ of \hat{F}_x^v . The thesis follows from Equations (4) and (5), because the exclusive ancestors partition the rows $[1..m]$ into $|W_x|$ sets. Also, note that $|\hat{F}_x| = m$. ◀

An example of sets F_x and \hat{F}_x can be seen in Figure 4. Unfortunately, their naive computation takes time $O(m^2)$ if done locally, because GST_{MSA} does not contain the information on the ending occurrences of MSA substrings – and it cannot be easily augmented to do so.

► **Lemma 23.** *Given a general MSA $[1..m, 1..n]$, GST_{MSA} , and the exclusive ancestors W_x , we can compute the meaningful prefix-aware right extensions \overline{R}_x in time $O(m^2)$.*

Proof. For each $v \in F_x$, we can compute $\text{pos}(v)$ by finding for each row $i \in I_v$ the ending position y_i of the occurrence of $Sc = \text{string}(\text{parent}(v)) \cdot \text{char}(v)$ in $\text{MSA}[i, 1..n]$ (Sc is a semi-repeat-free substring so there is at most one occurrence per row). In other words, position y_i corresponds to the k -th non-gap character of row i , where $k = \text{rank}(\text{MSA}[i, 1..n], x) + \text{stringdepth}(\text{parent}(v)) + 1$. Then, $\text{pos}(v) = \min_{i \in I_v} y_i$. The first-born child of v can be found by choosing one of its children with minimum pos values, and the removal of first-born nodes results in the pos values of \hat{F}_x . Given $f(x)$ and the ordered pos values, a simple algorithm like Algorithm 2 in Appendix A considers all columns containing pos values and outputs the relative prefix-aware right extension as a triple $(x, y, \overline{H}([x + 1..y]))$.

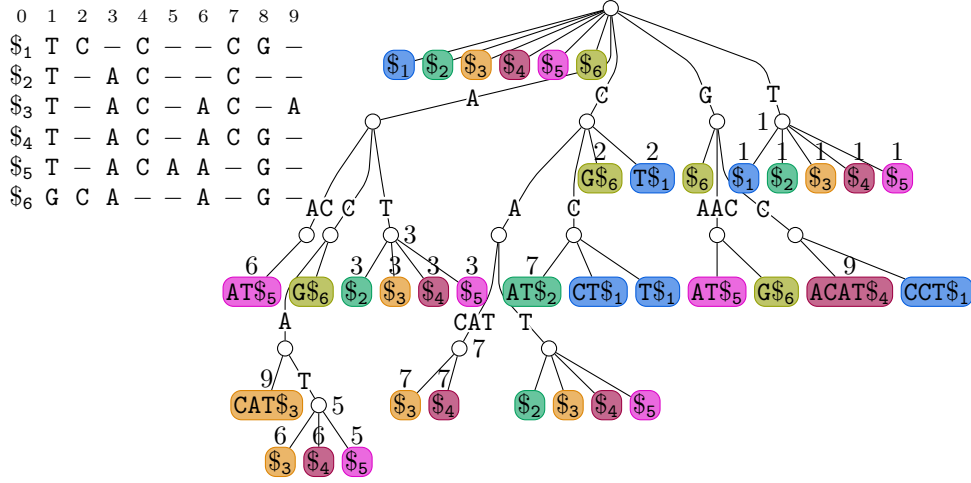
Since we can preprocess in linear time the MSA rows to answer rank and select queries in constant time, the computation of each $\text{pos}(v)$ takes $O(|I_v|)$ time. Forest F_x is composed of compacted trees with m total leaves, so it contains $O(m)$ nodes: the subtrees of F_x can be unbalanced, hence the total time is $O(m^2)$. Then, these values can be sorted in time $O(m \log m)$ and then processed in $O(m)$ time. ◀

4.3 Speedup using weighted ancestor queries

Thanks to Lemma 23, we can compute the meaningful prefix-aware right extensions in $O(m^2n)$ time, the bottleneck being the computation of values $\text{pos}(v)$ (Definition 20). The other tasks can be executed in time $O(mn)$ by generalizing the solution to a global computation: the pos values of all $\hat{F}_0, \dots, \hat{F}_{n-1}$ are $O(mn)$ in total; together they can be sorted in $O(mn)$ time since they take values in $[1..n]$, and they can be separately processed again in total linear time. GST_{MSA} does not contain the information about the ending occurrences of MSA strings, but it does contain the information on the (starting) occurrences: indeed, the sets of leaves $\mathcal{L}_0, \dots, \mathcal{L}_{n-1}$ consider each and every suffix starting from a certain MSA column. This gives us the key idea of symmetry to compute values $\text{pos}(v)$ efficiently.

19:16 Elastic Founder Graphs of Minimum Height

► **Definition 24.** Given a general MSA[1..m, 1..n] from alphabet $\Sigma \cup \{-\}$, we define GPT_{MSA} as the generalized prefix tree of the set of strings $\{\$i \cdot \text{spell}(\text{MSA}[i, 1..n]) : 1 \leq i \leq m\}$, with $\$1, \dots, \m m new distinct terminator symbols not in Σ . Alternatively, GPT_{MSA} can be constructed as the generalized suffix tree of $\{\text{spell}(\text{MSA}[i, 1..n])^{-1} \cdot \$i : 1 \leq i \leq m\}$.



■ **Figure 5** Example of the GPT_{MSA} built from the MSA of Figure 3, annotated with the pos^{-1} values relevant for the computation of R_0 (Figure 4), the meaningful right extensions starting from column 1.

► **Observation 25.** Note that in GPT_{MSA} strings are read from right to left. For each node u of GPT_{MSA} , let $\text{pos}^{-1}(u)$ be the first ending occurrence of $\text{string}(u)$ in some MSA row:

- for any leaf ℓ of GPT_{MSA} corresponding to row $i \in [1..m]$, we have that $\text{pos}^{-1}(\ell)$ is equal to the k -th non-gap character of $\text{MSA}[i, 1..n]$, with $k = |\text{string}(\ell)|$;
- for any internal node u of GPT_{MSA} , let v_1, \dots, v_p be its children; then $\text{pos}^{-1}(u) = \min_{k=1}^p \text{pos}^{-1}(v_k)$;
- given a node v of GST_{MSA} , let v^{-1} be the node of GPT_{MSA} corresponding to string $\text{string}(\text{parent}(v)) \cdot \text{char}(v)$ read from right to left; if this is an implicit node, then we define v^{-1} as the first explicit ancestor in GPT_{MSA} ; then $\text{pos}(v) = \text{pos}^{-1}(v^{-1})$.

For example, if v is the GST_{MSA} node of Figures 3 and 4 corresponding to string $\text{TAC} \cdot \text{A}$, v^{-1} corresponds to ACAT in the GPT_{MSA} of Figure 5 and $\text{pos}^{-1}(v^{-1}) = 5$.

► **Lemma 26.** Given a general MSA[1..m, 1..n], GST_{MSA} , and GPT_{MSA} , values $\text{pos}(v)$ for any node v of GST_{MSA} can be computed in $O(mn)$ time.

Proof. As shown in Observation 25, the tree structure of GPT_{MSA} makes it possible to compute $\text{pos}^{-1}(v)$ recursively: similar to the computation of $\mathcal{L}_0, \dots, \mathcal{L}_{n-1}$ this can be done in $O(mn)$ time. It remains to show that given $v \in \text{GST}_{\text{MSA}}$ we can find $v^{-1} \in \text{GPT}_{\text{MSA}}$ in $O(1)$ time: it is straightforward to locate one occurrence of $\text{string}(\text{parent}(v)) \cdot \text{char}(v)$ in the MSA, so we can find v^{-1} by answering the corresponding weighted ancestor query in GPT_{MSA} . Belazzougui et al. recently proved that we can preprocess suffix trees in linear time to be able to answer weighted ancestor queries in constant time [2]. ◀

This concludes the proof of Theorem 17: as we have already shown in Section 3.4, the meaningful prefix-aware right extensions are a drop-in replacement of the original meaningful extensions, so Lemma 26 implies that the optimal segmentation minimizing the maximum prefix-aware height can be computed in linear time.

References

- 1 Mai Alzamel, Lorraine A. K. Ayad, Giulia Bernardini, Roberto Grossi, Costas S. Iliopoulos, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Degenerate string comparison and applications. In Laxmi Parida and Esko Ukkonen, editors, *18th International Workshop on Algorithms in Bioinformatics, WABI 2018, August 20-22, 2018, Helsinki, Finland*, volume 113 of *LIPICs*, pages 21:1–21:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.WABI.2018.21.
- 2 Djamal Belazzougui, Dmitry Kosolobov, Simon J. Puglisi, and Rajeev Raman. Weighted ancestors in suffix trees revisited. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021, July 5-7, 2021, Wrocław, Poland*, volume 191 of *LIPICs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CPM.2021.8.
- 3 Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Even faster elastic-degenerate string matching via fast matrix multiplication. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.21.
- 4 Bastien Cazaux, Dmitry Kosolobov, Veli Mäkinen, and Tuukka Norri. Linear time maximum segmentation problems in column stream model. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, volume 11811 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2019.
- 5 The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings Bioinform.*, 19(1):118–135, 2018. doi:10.1093/bib/bbw089.
- 6 Rene De La Briandais. File searching using variable length keys. In *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference, IRE-AIEE-ACM '59 (Western)*, pages 295–298, New York, NY, USA, 1959. Association for Computing Machinery. doi:10.1145/1457838.1457895.
- 7 Hannes P. Eggertsson, Snaedis Kristmundsdottir, Doruk Beyter, Hakon Jonsson, Astros Skuladottir, Marteinn T. Hardarson, Daniel F. Gudbjartsson, Kari Stefansson, Bjarni V. Halldorsson, and Pall Melsted. GraphTyper2 enables population-scale genotyping of structural variation using pangenome graphs. *Nature Communications*, 10(1):5402, November 2019. doi:10.1038/s41467-019-13341-9.
- 8 Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of string matching for graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPICs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICALP.2019.55.
- 9 Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. In Tomás Bures, Riccardo Dondi, Johann Gamper, Giovanna Guerrini, Tomasz Jurdzinski, Claus Pahl, Florian Sikora, and Prudence W. H. Wong, editors, *SOFSEM 2021: Theory and Practice of Computer Science - 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy, January 25-29, 2021, Proceedings*, volume 12607 of *Lecture Notes in Computer Science*, pages 608–622. Springer, 2021. doi:10.1007/978-3-030-67731-2_44.
- 10 Massimo Equi, Tuukka Norri, Jarno Alanko, Bastien Cazaux, Alexandru I. Tomescu, and Veli Mäkinen. Algorithms and complexity on indexing elastic founder graphs. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPICs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ISAAC.2021.20.

- 11 Massimo Equi, Tuukka Norri, Jarno Alanko, Bastien Cazaux, Alexandru I. Tomescu, and Veli Mäkinen. Algorithms and complexity on indexing founder graphs. *CoRR*, abs/2102.12822, 2021. [arXiv:2102.12822](https://arxiv.org/abs/2102.12822).
- 12 Martin Farach. Optimal suffix tree construction with large alphabets. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 137–143. IEEE, 1997.
- 13 Erik Garrison, Jouni Sirén, Adam Novak, Glenn Hickey, Jordan Eizenga, Eric Dawson, William Jones, Shilpa Garg, Charles Markello, Michael Lin, and Benedict Paten. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36, August 2018. doi:10.1038/nbt.4227.
- 14 Daniel Gibney and Sharma V. Thankachan. On the hardness and inapproximability of recognizing wheeler graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 51:1–51:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ESA.2019.51.
- 15 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.
- 16 G. Jacobson. Space-efficient static trees and graphs. In *Proc. FOCS*, pages 549–554, 1989.
- 17 Daehwan Kim, Joseph Paggi, Chanhee Park, Christopher Bennett, and Steven Salzberg. Graph-based genome alignment and genotyping with hisat2 and hisat-genotype. *Nature Biotechnology*, 37:1, August 2019. doi:10.1038/s41587-019-0201-4.
- 18 Veli Mäkinen, Bastien Cazaux, Massimo Equi, Tuukka Norri, and Alexandru I. Tomescu. Linear time construction of indexable founder block graphs. In Carl Kingsford and Nadia Pisanti, editors, *20th International Workshop on Algorithms in Bioinformatics, WABI 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 172 of *LIPICs*, pages 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.WABI.2020.7.
- 19 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology*, 17(3):281–308, 2010.
- 20 Tuukka Norri, Bastien Cazaux, Saska Dönges, Daniel Valenzuela, and Veli Mäkinen. Founder reconstruction enables scalable and seamless pangenomic analysis. *Bioinformatics*, 37(24):4611–4619, July 2021. doi:10.1093/bioinformatics/btab516.
- 21 Tuukka Norri, Bastien Cazaux, Dmitry Kosolobov, and Veli Mäkinen. Linear time minimum segmentation enables scalable founder reconstruction. *Algorithms Mol. Biol.*, 14(1):12:1–12:15, 2019.
- 22 Nicola Rizzo and Veli Mäkinen. Linear time construction of indexable elastic founder graphs. In *Proc. 33rd International Workshop on Combinatorial Algorithms (IWOCA 2022)*, 2022. To appear.
- 23 Nicola Rizzo and Veli Mäkinen. Linear time construction of indexable elastic founder graphs. *CoRR*, abs/2201.06492, 2022. [arXiv:2201.06492](https://arxiv.org/abs/2201.06492).
- 24 Korbinian Schneeberger, Jörg Hagmann, Stephan Ossowski, Norman Warthmann, Sandra Gesing, Oliver Kohlbacher, and Detlef Weigel. Simultaneous alignment of short reads against multiple genomes. *Genome Biology*, 10:R98, 2009.
- 25 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014.

A Pseudocode implementations of the algorithms

■ **Algorithm 2** Algorithm computing the meaningful prefix-aware right extensions, given the sorted pos values of \hat{F}_x . Set \bar{F}_x is obtained by removing the first-born nodes from F_x .

Input: Value $f(x)$, values $\text{pos}(w_1), \dots, \text{pos}(w_m)$ of nodes in \bar{F}_x , sorted from smallest to largest order.

Output: Meaningful prefix-aware right extensions \bar{R}_x .

```

1  $h \leftarrow 1$ ;
2 while  $\text{pos}(w_h) < f(x)$  do
3    $h \leftarrow h + 1$ ;
4 while  $h \leq m$  do
5   if  $h = m \vee \text{pos}(w_{h+1}) \neq \text{pos}(w_h)$  then
6      $\text{output}(x, \text{pos}(w_h), h)$ ;
7    $h \leftarrow h + 1$ ;

```

■ **Algorithm 3** Algorithm computing all meaningful right extensions R_0, \dots, R_{n-1} . To efficiently compute keyword trees $K_{x+1..r}$ for $y \in [x+1..n]$, we need a dynamic tree data structure \mathcal{T} supporting navigation and insertions in time $O(\log \sigma)$.

Input: MSA[1..m, 1..n] from an integer alphabet $\Sigma \cup \{-\}$ of size $\sigma \in O(mn)$, minimal right extensions $f(x)$ for $x \in [0..n-1]$.

Output: Meaningful prefix-aware right extensions R_0, \dots, R_{n-1} represented as triples $(x, r_{x,k}, H([x+1..r_{x,k}]))$.

```

1 for  $x \leftarrow 0$  to  $n - 1$  do
2   Initialize empty keyword tree  $\mathcal{T}$ , containing only node root;
3   Initialize array  $V[1..m]$  with values pointers to nodes of  $\mathcal{T}$  and set all values to node root;
4    $h \leftarrow 0$ ;
5    $r \leftarrow x$ ;
6   while  $\mathcal{T}.\text{leaves} < m \wedge r \leq m$  do
7      $h' \leftarrow h$ ;
8     for  $i \leftarrow 1$  to  $m$  do
9       if  $\text{MSA}[i, r] \neq -$  then
10         $V[i].\text{count} \leftarrow V[i].\text{count} - 1$ ;
11        if  $V[i].\text{count} = 0$  then
12           $h \leftarrow h - 1$ ;
13        if  $V[i]$  has an (MSA[ $i, r$ ])-child  $v$  then
14           $V[i] \leftarrow v$ ;
15        else
16          Add new (MSA[ $i, r$ ])-child  $v$  to  $V[i]$ ;
17           $V[i] \leftarrow v$ ;
18        if  $V[i].\text{count} = 0$  then
19           $h \leftarrow h + 1$ ;
20         $V[i].\text{count} \leftarrow V[i].\text{count} + 1$ ;
21   if  $r = f(x)$  then
22      $\text{output}(x, r, h)$ ;
23   else if  $r \geq f(x) \wedge h \neq h'$  then
24      $\text{output}(x, r, h)$ ;
25    $r \leftarrow r + 1$ ;

```
