Linear Time Construction of Indexable Elastic Founder Graphs

Rizzo, Nicola

Springer
2022

Rizzo , N & Mäkinen , V 2022 , Linear Time Construction of Indexable Elastic Founder Graphs . in C Bazgan & H Fernau (eds) , IWOCA 2022: Combinatorial Algorithms . Lecture Notes in Computer Science , vol. 13270 , Springer , Cham , pp. 480-493 , 33rd International Workshop on Combinatorial Algorithms (IWOCA) , Trier , Germany , 07/06/2022 . https://doi.org/10.1007/978-3-031-06678-8_35 , https://doi.org/10.48550/arXiv.2201.06492

http://hdl.handle.net/10138/351030
https://doi.org/10.1007/978-3-031-06678-8_35

unspecified
acceptedVersion

Linear Time Construction of Indexable Elastic Founder Graphs

Nicola Rizzo$^{[0000-0002-2035-6309]}$ and Veli Mäkinen$^{[0000-0003-4454-1493]}$

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

Abstract. The pattern matching of strings in labeled graphs has been widely studied lately due to its importance in genomics applications. Unfortunately, even the simplest problem of deciding if a string appears as a subpath of a graph admits a quadratic lower bound under the Orthogonal Vectors Hypothesis (Equi et al. ICALP 2019, SOFSEM 2021). To avoid this bottleneck, the research has shifted towards more specific graph classes, e.g. those induced from multiple sequence alignments (MSAs). Consider segmenting $\mathsf{MSA}[1..m, 1..n]$ into b blocks $\mathsf{MSA}[1..m, 1..j_1]$, $\mathsf{MSA}[1..m, j_1 + 1..j_2]$, $\ldots$, $\mathsf{MSA}[1..m, j_{b-1} + 1..n]$. The distinct strings in the rows of the blocks, after the removal of gap symbols, form the nodes of an *elastic founder graph* (EFG) where the edges represent the original connections observed in the MSA. An EFG is called *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 gave an $O(mn \log m)$-time algorithm for preprocessing the MSA in a way that allows the construction of indexable EFGs maximizing the number of blocks and, alternatively, minimizing the maximum length of a block, in $O(n)$ and $O(n \log \log n)$ time respectively. Using the suffix tree and solving a novel ancestor problem on trees, we improve the preprocessing to $O(mn)$ time and the $O(n \log \log n)$-time EFG construction to $O(n)$ time, thus showing that both types of indexable EFGs can be constructed in time linear in the input size.

Keywords: multiple sequence alignment · pattern matching · data structures · segmentation algorithms · dynamic programming · suffix tree

Acknowledgements 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

Searching strings in a graph has become a central problem along with the development of high-throughput sequencing techniques. Namely, thousands of human genomes are now available, forming a so-called *pangenome* of a species [20]. Such pangenome can be used to enhance various analysis tasks that have previously been conducted with a single reference genome [13,18,19,8,11,3,14]. The most popular representation for a pangenome is a graph, whose paths spell the input genomes. 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. Unfortunately, even finding exact matches of a query string of length q in a graph with e edges cannot be done significantly faster than $O(qe)$ time, and no index built in polynomial time allows for subquadratic-time string matching, unless the Orthogonal Vectors Hypothesis (OVH) is false [5,4]. Therefore, practical tools deploy various heuristics or use other pangenome representations as a basis.

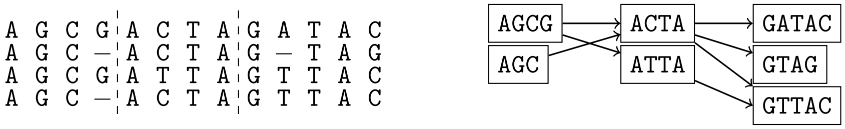

A G C G A C T A G A T A C
A G C – A C T A G – T A G
A G C G A T T A G T T A C
A G C – A C T A G T T A C
AGCG
AGC
ACTA
ATTA
GATAC
GTAG
GTTAC

Fig. 1. An indexable elastic founder graph induced from a segmentation of an MSA. The example is adapted from Equi et al. [6].

Due to the difficulty of string search in general graphs, Mäkinen et al. [12] and Equi et al. [6] studied graphs induced from multiple sequence alignments (MSAs), as we describe in Section 2. Any segmentation of an MSA naturally induces a graph consisting of nodes partitioned into blocks with edges connecting consecutive blocks. Such *elastic founder graph* (EFG) is illustrated in Figure 1. The key observation is that if the resulting node labels do not appear as a prefix of any other path than those starting at the same block, then there is an index structure for the graph that supports fast pattern matching [12,6]. Equi et al. [6] also showed that such indexability property is required, as the OVH-based lower bound holds for EFGs derived from MSAs. Mäkinen et al. [12] gave an $O(mn)$ time algorithm to construct an indexable EFG with minimum maximum block length, given a gapless $\mathsf{MSA}[1..m, 1..n]$. Equi et al. [6] extended the result to general MSAs. They obtained an $O(mn \log m)$-time preprocessing algorithm which allows the construction of indexable EFGs maximizing the number of blocks and, alternatively, minimizing the maximum length of a block, in $O(n)$ and in $O(n \log\log n)$ time, respectively. We recall these results in Section 3.

In this paper, we improve the preprocessing algorithm of Equi et al. to $O(mn)$ by performing an in-depth analysis of their solution based on the generalized suffix tree $\mathsf{GST}_{\mathsf{MSA}}$ built from the gaps-removed rows of the MSA (Section 4). Although removing gaps constitutes a loss of essential information, this information can be fed back into the structure by considering the right subsets of its nodes or leaves. Then, the main step in preprocessing the MSA is solving a novel ancestor problem on the tree structure of $\mathsf{GST}_{\mathsf{MSA}}$ that we call the *exclusive ancestor set problem*, and as our main contribution, we identify such problem and provide a linear-time solution. This directly improves the solution by Equi et al. for constructing indexable EFGs maximizing the number of blocks from $O(mn \log m)$ to $O(mn)$ time. Moreover, in Section 5 we give a new algorithm that after the $O(mn)$-time preprocessing can construct indexable EFGs minimizing the maximum block length in $O(n)$ time. In our subsequent work [16], we extend these techniques to minimize the maximum block height.

2 Definitions

We follow the notation of Equi et al. [6].

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. A *suffix* (*prefix*) of string $T[1..n]$ is $T[i..n]$ ($T[1..i]$) for $1 \leq i \leq n$ and we say it is *proper* if $i > 1$ ($i < n$). The *length* of a string T is denoted $|T|$ and the *empty string* ε is the string of length 0. In particular, substring $T[i..j]$ where $j < i$ is the empty string. For convenience, we denote with Σ^* and Σ^+ the set of finite strings and finite non-empty strings over Σ, respectively. The *lexicographic order* of two strings A and B is naturally defined by the order of the alphabet: $A < B$ iff $A[1..i] = B[1..i]$ and $A[i+1] < B[i+1]$ for some $i \geq 0$. If $i + 1 > \min(|A|, |B|)$, then the shorter one is regarded as smaller. However, we usually avoid this implicit comparison by adding an *end marker* \$ to the strings and we consider \$ to be the smallest character lexicographically. The concatenation of strings A and B is denoted as $A \cdot B$, 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* $\mathsf{MSA}[1..m, 1..n]$ 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 $\mathsf{spell}(X)$ 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. Let $\mathcal{P}$ be a *partitioning* of $[1..n]$, that is, a sequence of subintervals $\mathcal{P} = [x_1..y_1], [x_2..y_2], \ldots, [x_b..y_b]$ where $x_1 = 1$, $y_b = n$, and for all $j > 2$, $x_j = y_{j-1} + 1$. A *segmentation* S of $\mathsf{MSA}[1..m, 1..n]$ based on partitioning $\mathcal{P}$ is the sequence of b sets $S^k = \{\mathsf{spell}(\mathsf{MSA}[i, x_k..y_k]) \mid 1 \leq i \leq m\}$ for $1 \leq k \leq b$; in addition, we require for a (proper) segmentation that $\mathsf{spell}(\mathsf{MSA}[i, x_k..y_k]) \neq \varepsilon$ for any i and k. We call set S^k a *block*, while $\mathsf{MSA}[1..m, x_k..y_k]$ or just $[x_k..y_k]$ is called a *segment*. The *length* of block S^k or its segment $[x_k..y_k]$ is $L(S^k) = L([x_k..y_k]) = y_k - x_k + 1$.

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 *$V^1, V^2, \ldots, V^b$, that is, $V = V^1 \cup V^2 \cup \cdots \cup V^b$ 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 $|\ell(v)| = |\ell(w)|$ for each $1 \leq i \leq b$ and if $v \neq w$, $\ell(v) \neq \ell(w)$.*

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 as follows: for each $1 \leq k \leq b$ we have $S^k = \{\mathsf{spell}(\mathsf{MSA}[i, x_k..y_k]) \mid 1 \leq i \leq m\} = \{\ell(v) : v \in V^k\}$. 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 $\mathsf{spell}(\mathsf{MSA}[i, x_k..y_{k+1}]) = \ell(v)\ell(w)$.*

For example, in the general $\mathsf{MSA}[1..4, 1..13]$ of Figure 1, the segmentation based on partitioning $[1..4], [5..8], [9..13]$ induces an EFG $G(S) = (V^1 \cup V^2 \cup V^3, E, \ell)$ where the nodes in V^1 and V^3 have labels of variable length.

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 ([12]). *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 ([12]). *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. We will discuss these two indexability properties together as the (semi-)repeat-free property, when applicable.

Basic tools. A *trie* [2] 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 $T\$$. In this case, the edge labels are substrings of T and can be represented in constant space as an interval. Such tree takes linear space and can be constructed in linear time, assuming that $\sigma \leq |T|$, so that when reading the leaves from left to right the suffixes are listed in their lexicographic order. [21,7] We say that two or more leaves of the suffix tree are *adjacent* if they succeed one another when reading them left to right. A *generalized suffix tree* is one built on a set of strings. In this case, string T above is the concatenation of the strings with symbol \$ between each.

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 path 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 in 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 the 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.

String $B[1..n]$ from a binary alphabet is called a *bitvector*. The operation $\text{rank}(B, i)$ returns the number of 1s in $B[1..i]$, whereas the 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 requiring $o(n)$ bits in addition to the bitvector itself and computable in linear time [10,9].

3 Overview of EFG construction algorithms

Equi et al. have shown that (semi-)repeat-free EFGs are easy to index for fast pattern matching [6], and as we describe in Section 3.1 they extended the previous research for the gapless and repeat-free setting showing that finding (semi-)repeat-free elastic founder graphs is equivalent to finding (semi-)repeat-free MSA segmentations. Moreover, to show that the (semi-)repeat-free property does not hinder the flexibility in choosing the resulting EFGs, they considered the following score functions for MSA segmentations: *i.* maximizing the number of blocks, and *ii.* minimizing the maximum length of a block.

In the gapless and 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 [12,15,1]. In the general and semi-repeat-free setting, Equi et al. have given $O(mn \log m)$ and $O(mn \log m + n \log \log n)$-time algorithms for scores *i.* and *ii.*, respectively, based on a common preprocessing of the MSA that we review in Section 3.2.

3.1 Segmentation characterization for indexable EFGs

Consider a segmentation $S = S^1, S^2, \ldots, S^b$ that induces a (semi-)repeat-free EFG $G(S) = (V, E, \ell)$, as per Definition 2. The strings occurring in graph $G(S)$ are a superset of the strings occurring in the original MSA rows because each node label can represent *multiple* rows and each edge $(v, w) \in E$ means the existence of *some* row spelling $\ell(v)\ell(w)$ in the corresponding consecutive segments. For example, string GACTAGT occurs in the EFG of Figure 1 but it does not occur in any row of the original MSA.

The (semi-)repeat-free property involves graph $G(S)$, but luckily it does not depend on the new strings added in the founder graph and can be checked only against the MSA and segmentation S. This simplifies choosing a segmentation resulting in an indexable founder graph and it was initially proven by Mäkinen et al. in the gapless and repeat-free setting.

Lemma 1 (Characterization, gapless setting [12]). *We say that a segment* $[x..y]$ *of a gapless* $\mathsf{MSA}[1..m, 1..n]$ *is repeat-free if string* $\mathsf{MSA}[i, x..y]$ *occurs in the MSA only at position* x *of some row, for all* $1 \leq i \leq m$. *Then* $G(S)$ *is repeat-free if and only if all segments defining* S *are repeat-free.*

Equi et al. in [6] refined this property for MSAs with gaps, but did not provide an explicit proof. Since it is essential for the correctness of the construction algorithms, we provide such a proof in the full version of this paper [17].

Lemma 2 (Characterization [6]). *We say that segment* $[x..y]$ *of a general* $\mathsf{MSA}[1..m, 1..n]$ *is semi-repeat-free if for any* $i, i' \in [1..m]$ *string* $\mathsf{spell}(\mathsf{MSA}[i, x..y])$ *occurs in gaps-removed row* $\mathsf{spell}(\mathsf{MSA}[i', 1..n])$ *only at position* $g(i', x)$, *where* $g(i', x)$ *is equal to* x *minus the number of gaps in* $\mathsf{MSA}[i', 1..x]$. *Similarly,* $[x..y]$ *is repeat-free if the eventual occurrence of* $\mathsf{spell}(\mathsf{MSA}[i, x..y])$ *at position* $g(i', x)$ *in row* i' *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.*

3.2 EFG construction algorithms

Just as in the gapless and repeat-free setting, Lemma 2 implies that the optimal score $s(j)$ of a (semi-)repeat-free segmentation of the general MSA prefix $\mathsf{MSA}[1..m, 1..j]$ can be computed recursively for a variety of scoring schemes:

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

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

Equi et al. studied the computation of semi-repeat-free segmentations optimizing for these two scores [6]. The algorithms they developed—and that we will improve in Sections 4 and 5—are based on a common preprocessing of the valid semi-repeat-free segmentation ranges, based on the following observation.

Observation 1 (Semi-repeat-free right extensions [6]). *Given a general* $\mathsf{MSA}[1..m, 1..n]$, *for any* $x < y$ *we say that segment* $[x + 1..y]$ *is an extension of prefix* $\mathsf{MSA}[1..m, 1..x]$. *If extension* $[x + 1..y]$ *is semi-repeat-free, then extension* $[x + 1..y']$ *is semi-repeat-free for all* $y < y' \leq n$.

Note that in the presence of gaps Observation 1 does not hold if we swap the semi-repeat-free notion with the repeat-free one, or if we swap the right extensions with the symmetrically defined left extensions.

To compute $s(j)$, Equation (1) considers all semi-repeat-free right extensions $[j'+1..j]$ ending at column j. Equi et al. discovered that the computation of values $s(j)$ can be done efficiently by considering that each semi-repeat-free right extension $[j'+1..j]$ has as prefix a minimal (semi-repeat-free) right extension $[j'+1..f(j')]$, with function f defined as follows.

Definition 5 (Minimal right extensions [6]). *Given* $\mathsf{MSA}[1..m, 1..n]$*, for each* $0 \leq x \leq n-1$ *we define value* $f(x)$ *as the smallest integer greater than* x *such that segment* $[x+1..f(x)]$ *is semi-repeat-free, or, in other words,* $[x+1..f(x)]$ *is the minimal (semi-repeat-free) right extension of prefix* $\mathsf{MSA}[1..m, 1..x]$*. If there is no semi-repeat-free extension, we define* $f(x) = \infty$*.*

Indeed, Equi et al. in [6] developed an algorithm computing values $f(x)$ in time $O(mn \log m)$. Using only these values, described by a list of pairs $(x, f(x))$ sorted in increasing order by the second component, they developed two algorithms computing the score of an optimal semi-repeat-free segmentation: in time $O(n)$ for the maximum number of blocks score and in time $O(n \log \log n)$ for the maximum block length score. We will explain in detail how the latter works in Section 5, as we will improve its run time to $O(n)$.

4 Preprocessing the MSA in linear time

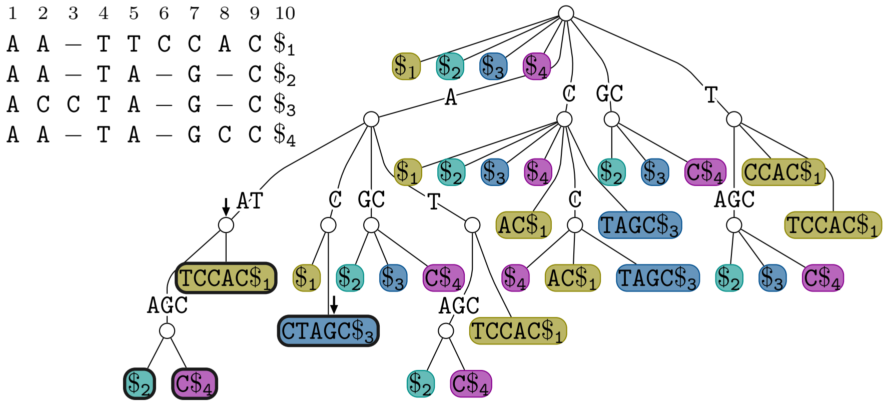
In this section, we study the computation of the minimal right extensions $f(x)$, for $0 \leq x \leq n-1$ (Definition 5). Equi et al. in [6] proposed an $O(nm \log m)$-time solution using the following structure, built from the gaps-removed MSA rows.

Definition 6. *Given* $\mathsf{MSA}[1..m, 1..n]$ *from alphabet* $\Sigma \cup \{-\}$*, we define* $\mathsf{GST}_{\mathsf{MSA}}$ *as the generalized suffix tree of the set of strings* $\{\mathsf{spell}(\mathsf{MSA}[i, 1..n]) \cdot \$_i : 1 \leq i \leq m\}$*, with* $\$_1, \ldots, \$_m$ m *new distinct terminator symbols not in* Σ*.*[1]

An example of $\mathsf{GST}_{\mathsf{MSA}}$ is given in Figure 2. From the suffix tree properties, it follows that for any gaps-removed row $\alpha_i \coloneqq \mathsf{spell}(\mathsf{MSA}[i, 1..n])\$_i$, with $1 \leq i \leq m$: each suffix $\alpha_i[x..|\alpha_i|]$ corresponds to a unique leaf $\ell_{i,x}$ of $\mathsf{GST}_{\mathsf{MSA}}$ and vice versa, with $1 \leq x \leq |\alpha_i|$; each substring $\alpha_i[x..y]$ corresponds to an explicit or implicit node of $\mathsf{GST}_{\mathsf{MSA}}$ in the root-to-$\ell_{i,x}$ path; and each explicit or implicit node corresponds to one or more such substrings, uniquely identifiable thanks to the leaves covered by the node. Also, note that $\mathsf{GST}_{\mathsf{MSA}}$ does not contain any information about the gap symbols of the MSA, as this information will be added back into the structure thanks to the set of leaves and nodes considered.

In Section 4.1 we perform an analysis of $\mathsf{GST}_{\mathsf{MSA}}$ similar to that of Equi et al., showing that semi-repeat-free segments of the MSA correspond to a specific set of nodes of $\mathsf{GST}_{\mathsf{MSA}}$ covering exactly m leaves. Then, in Section 4.2, we show

[1] We added the m new distinct terminators for simplicity, whereas Equi et al. used the suffix tree of the concatenation of all gaps-removed rows with a single new symbol \$ between each. The suffix tree of this string, if a second unique terminator # is concatenated to this string, is equivalent to $\mathsf{GST}_{\mathsf{MSA}}$ for our purposes.



1	2	3	4	5	6	7	8	9	10
A	A	–	T	T	C	C	A	C	$\$_1$
A	A	–	T	A	–	G	–	C	$\$_2$
A	C	C	T	A	–	G	–	C	$\$_3$
A	A	–	T	A	–	G	C	C	$\$_4$

Fig. 2. Example of an $\mathsf{MSA}[1..4, 1..10]$ and its $\mathsf{GST_{MSA}}$, where the label to each leaf has been moved inside the leaf itself. We have also highlighted the leaves corresponding to suffixes $\mathsf{spell}(\mathsf{MSA}[i, 1..n])$ (black outline) and its exclusive ancestors (arrows).

that the novel resulting problem on the tree structure of $\mathsf{GST_{MSA}}$, that we call the *exclusive ancestor set problem*, can be solved efficiently, resulting in an algorithm computing the minimal right extensions in linear time, described in Section 4.3.

4.1 Semi-repeat-free segments in the generalized suffix tree

The following has been implicitly stated and exploited in [6].

Definition 7 (Semi-repeat-free substrings). *Recall the definition of semi-repeat-free segment (Lemma 2). Given substring* $\mathsf{MSA}[i, x..y]$ *of* $\mathsf{MSA}[1..m, 1..n]$ *such that* $\mathsf{spell}(\mathsf{MSA}[i, x..y]) \in \Sigma^+$*, we say that* $\mathsf{MSA}[i, x..y]$ *is semi-repeat-free if, for all* $1 \le i' \le m$*, string* $\mathsf{spell}(\mathsf{MSA}[i, x..y])$ *occurs in gaps-removed row* i' *only at position* $g(i', x)$ *(or it does not occur at all).*

Observation 2. *Segment* $[x..y]$ *is semi-repeat-free if and only if all substrings* $\mathsf{MSA}[i, x..y]$ *are semi-repeat-free, for* $1 \le i \le m$*. If* $\mathsf{MSA}[i, x..y]$ *is semi-repeat-free, then* $\mathsf{MSA}[i, x..y']$ *is semi-repeat-free for all* $y < y' \le n$*. Let* $f^i(x)$ *be the smallest integer greater than* x *such that substring* $\mathsf{MSA}[i, x+1..f^i(x)]$ *is semi-repeat-free: it is easy to see that* $f(x) = \max_{i=1}^{m} f^i(x)$*.*

This translates into a specific set of implicit or explicit nodes of $\mathsf{GST_{MSA}}$. The fact that we added a unique terminator symbol to each row is equivalent to the addition of an MSA column spelling $\$_1 \cdots \$_m$ at position $n+1$, which means that $[x+1..n+1]$ is always semi-repeat-free and the minimal right extensions such that $f(x) = \infty$ become $f(x) = n+1$.

Lemma 3. *Given* m *row substrings* $\mathsf{MSA}[i, x..y_i]$ *of* $\mathsf{MSA}[1..m, 1..n]$ *such that* $\mathsf{spell}(\mathsf{MSA}[i, x..y_i]) \in \Sigma^+$ *for* $1 \le i \le m$*, let* $W = \{w_1, \ldots, w_k\}$ *be the set of implicit or explicit nodes of* $\mathsf{GST_{MSA}}$ *corresponding to strings* $\{\mathsf{spell}(\mathsf{MSA}[i, x..y_i]) :$

$1 \leq i \leq m\}$. *Then* $\mathsf{MSA}[i, x..y_i]$ *is semi-repeat-free for all* $1 \leq i \leq m$ *if and only if* W *covers exactly* m *leaves in* $\mathsf{GST}_{\mathsf{MSA}}$.

Proof. By construction of $\mathsf{GST}_{\mathsf{MSA}}$, W covers the m leaves $\ell_{1,z_1}, \ldots, \ell_{m,z_m}$, with $z_i = g(i, x)$, so we only need to prove that if some $\mathsf{MSA}[i, x..y_i]$ is not semi-repeat-free, or *invalid*, then W covers more than m leaves, and vice versa.

($\Leftarrow$) Let $\mathsf{MSA}[i, x..y_i]$ be invalid, i.e. $\mathsf{spell}(\mathsf{MSA}[i, x..y_i])$ occurs in $\alpha_{i'}$ at some position $\hat{z}$ other than $z_{i'}$, for some row $1 \leq i' \leq m$. Then the node of $\mathsf{GST}_{\mathsf{MSA}}$ corresponding to string $\mathsf{spell}(\mathsf{MSA}[i, x..y_i])$ covers leaf $\ell_{i',\hat{z}} \neq \ell_{i',z_{i'}}$, thus W covers more than m leaves.

($\Rightarrow$) Let $\ell_{i',\hat{z}}$ be a leaf of $\mathsf{GST}_{\mathsf{MSA}}$ other than leaves $\ell_{1,z_1}, \ldots, \ell_{m,z_m}$ covered by some node $w \in W$. By construction, w corresponds to $\mathsf{spell}(\mathsf{MSA}[i, x..y_i])$ for some $1 \leq i \leq m$, so we have that $\mathsf{spell}(\mathsf{MSA}[i, x..y_i])$ occurs in $\alpha_{i'}$ at some position other than $g(i', x)$, since $\ell_{i',\hat{z}} \neq \ell_{i',z_{i'}}$. Thus, $\mathsf{MSA}[i', x..y_i]$ is invalid.

Note that the correctness of Lemma 3 does not hold if we swap the semi-repeat-free notion with the repeat-free one.

Lemma 3, combined with Observation 2, implies that the problem of computing values $f^i(x)$ for all $i \in [1..m]$ can be solved by analyzing the tree structure of $\mathsf{GST}_{\mathsf{MSA}}$ against the MSA suffixes. Indeed, let $L_x := \{\ell_{i,z_i} : 1 \leq i \leq m, z_i = g(i, x+1)\}$ be the leaves of $\mathsf{GST}_{\mathsf{MSA}}$ corresponding to the suffixes $\mathsf{spell}(\mathsf{MSA}[i, x+1..n])$. For each row $1 \leq i \leq m$, the first semi-repeat-free prefix of $\mathsf{spell}(\mathsf{MSA}[i, x+1..n])$ corresponds to the first implicit or explicit node v of $\mathsf{GST}_{\mathsf{MSA}}$ in the root-to-ℓ_{i,z_i} path such that v covers only leaves in L_x. The fact that $\mathsf{GST}_{\mathsf{MSA}}$ is a compacted trie is not an issue: the parent of v in the suffix trie is branching, since it covers more leaves than v, so the first explicit node of $\mathsf{GST}_{\mathsf{MSA}}$ in the root-to-ℓ_{i,z_i} path covering only leaves in L_x is the first explicit descendant w of v, thus we can identify v by finding w. Finally, $f^i(x)$ is computed by retrieving the smallest column index y such that $\mathsf{spell}(\mathsf{MSA}[i, x+1..y]) = \mathrm{string}(\mathrm{parent}(w)) \cdot \mathrm{char}(w)$, where $\mathrm{string}(u)$ is the concatenation of edge labels of the root-to-u path, and $\mathrm{char}(u)$ is the first symbol of the edge label from $\mathrm{parent}(u)$ to u. In other words, y corresponds to the k-th non-gap symbol of MSA row i, with $k = \mathrm{rank}(\mathsf{MSA}[i, 1..n], x) + \mathrm{stringdepth}(\mathrm{parent}(w)) + 1$, where $\mathrm{rank}(\mathsf{MSA}[i, 1..n], x)$ is the number of non-gap symbols in $\mathsf{MSA}[i, 1..x]$ and $\mathrm{stringdepth}(u) = |\mathrm{string}(u)|$. For example, in Figure 2 the leaves of L_0 have been marked and so have the shallowest ancestors covering only leaves in L_0.

4.2 Exclusive ancestor set

The results of the previous section show that we can compute the minimal right extensions by solving multiple instances of the following problem on the tree structure of $\mathsf{GST}_{\mathsf{MSA}}$.

Problem 1 (Exclusive ancestor set). Let $T = (V, E, \mathrm{root})$ be a rooted ordered tree, with $L^T \subseteq V$ the set of its leaves. Given T and a subset of leaves $L \subseteq L^T$, find the minimal set W of exclusive ancestors of L in T, i.e. the minimal set $W \subseteq V$ such that W covers all leaves in L and only leaves in L. Can T be preprocessed to support the efficient solving of multiple instances of the problem?

As is the case for $\mathsf{GST}_{\mathsf{MSA}}$, we can assume that each internal node of T has at least two children, otherwise, a linear-time processing of T can be employed to compact its unary paths. Indeed, after a linear-time preprocessing of T, any instance of exclusive ancestor set can be solved in time $O(|L|)$ by a careful traversal of the tree with the following procedure, that we describe informally:

1. partition L in k maximal sets $L_1, \ldots, L_k$ of leaves contiguous in the ordered traversal of T, to be processed independently (if two leaves belong to different contiguous sets, any common ancestor cannot be part of the solution);
2. for each L_i, with $1 \leq i \leq k$, start from the leftmost leaf ℓ_i and ascend in the tree until the closest ancestor of ℓ_i that covers some leaf not in L_i;
3. upon failure in step 2., add the last safe ancestor to the solution W and if there are still uncovered leaves in L_i repeat steps 2. and 3. starting from the leftmost uncovered leaf.

An example of the procedure is shown in Figure 3. The failure condition of step 2. can be evaluated by checking if both the leftmost leaf and the rightmost leaf in the subtree of the candidate replacement are still in set L_i, and step 2. always terminates if we assume that L is a nontrivial instance: if $L \subset L^T$, then the root of T is not the solution to the problem.

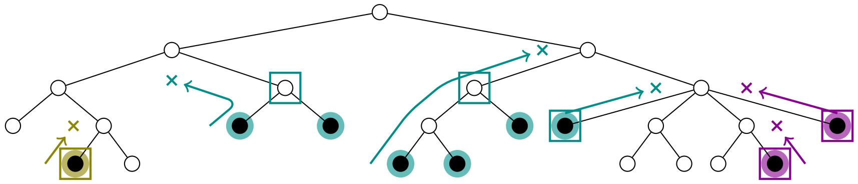


Fig. 3. Example of an instance of exclusive ancestor set, where the set of leaves L corresponds to the black leaves: the algorithm partitions L into sets of contiguous leaves (shown as brown, blue, and purple leaves), and for each set it finds the exclusive ancestors (marked with rectangles). Each arrow shows the ascent of step 2. up the tree until the node corresponding to the failure condition, marked with a cross.

Assuming the leaves of T are sorted, step 1. can be implemented efficiently: we can partition L into sets of contiguous leaves by coloring leaves in L and finding all the leaves with the preceding leaf not in L. We can easily preprocess T to support the required operations in constant time, leading to a time complexity of $O(|L|)$, since any forest built on top of leaves L has $O(|L|)$ nodes.

Lemma 4. *The exclusive ancestor set problem on a rooted ordered tree* $T = (V, E, \mathrm{root})$ *and a subset* L *of its leaves can be solved in time* $O(|L|)$*, after a* $O(|V|)$*-time preprocessing to support operations* $v.\mathrm{leftmostleaf}$*,* $v.\mathrm{rightmostleaf}$ *on any node* $v \in V$ *and operations* $\ell.\mathrm{prevleaf}$*,* $\ell.\mathrm{nextleaf}$*, and the binary coloring of any leaf* $\ell \in L^T$ *in constant time.*

4.3 Computing the minimal right extensions

Returning to the problem of computing values $f(x)$, the representation of $\mathsf{GST}_{\mathsf{MSA}}$ needs to support the operations on its tree structure described by Lemma 4 plus operations $v.$stringdepth, returning the length of the string corresponding to the root-to-v path in $\mathsf{GST}_{\mathsf{MSA}}$ of an explicit node v, and ℓ.suffixlink, implementing the suffix links of the leaves. The final algorithm, described in the full version of this paper [17], computes leaf sets L_0, L_1, $\ldots$, L_{n-1} corresponding to the MSA suffixes starting at column $1, 2, \ldots, n$, respectively, and for each L_x with $0 \leq x < n$:

1. it marks the leaves in L_x and partitions them in sets of contiguous leaves, by finding all their left boundaries ℓ such that ℓ.prevleaf is not marked;
2. it solves the exclusive ancestor set problem on each set of contiguous leaves and whenever it finds an exclusive ancestor, covering leaves $\ell_{i_1}, \ldots, \ell_{i_k}$, it computes values $f^i(x)$ for $i \in \{i_1, \ldots, i_k\}$ (see the conclusion of Section 4.1);
3. after processing all leaves, it finally computes $f(x) = \max_{i=1}^{m} f^i(x)$ and transforms L_x into L_{x+1} by taking the suffix links[2] of only leaves ℓ_i such that $\mathsf{MSA}[i, x+1] \neq -$.

Theorem 1. *Given* $\mathsf{MSA}[1..m, 1..n]$*, we can compute the minimal right extensions* $f(x)$ *for* $0 \leq x \leq n-1$ *in time* $O(mn)$*.*

Proof. The correctness is given by Observation 2 and Lemmas 3 and 4. The construction of $\mathsf{GST}_{\mathsf{MSA}}$ is equivalent to building the suffix tree of a string of length smaller than or equal to $(m+1)n$: a suffix tree supporting the required operations in constant time can be constructed in $O(mn)$ time, since we assume $|\Sigma| \leq mn$. Also, we can preprocess the MSA rows to answer in constant time rank and select queries on the position of gap and non-gap symbols. Thus, the computation of each $f(x)$ takes time $O(|L_x| + m) = O(m)$, so $O(mn)$ time in total.

Corollary 1. *Given* $\mathsf{MSA}[1..m, 1..n]$ *from* $\Sigma \cup \{-\}$*, with* $\Sigma = [1..\sigma]$ *and* $\sigma \leq mn$*, the construction of an optimal semi-repeat-free segmentation minimizing the maximum number of blocks can be done in time* $O(mn)$*.*

Proof. Algorithm [6, Algorithm 1] by Equi et al. solves the problem in $O(n)$ time, assuming it is given the minimal right extensions $(x, f(x))$ sorted in increasing order by the second component, which we can now compute and sort in time $O(mn)$ thanks to Theorem 1.

5 Minimizing the maximum block length

The improvement on the computation of the minimal right extensions in the case of general MSAs from $O(nm \log m)$ to $O(nm)$ gives us the motivation to

[2] As noted by an anonymous reviewer, the support for suffix links is not strictly necessary, since we are exploring leaves only. Indeed, a traversal of the tree can easily fill an $m \times n$ table containing L_0, $\ldots$, L_{n-1}, that we then have to store.

improve the $O(n \log\log n)$-time algorithm of Equi et al. [6, Algorithm 2] for an optimal semi-repeat-free segmentation minimizing the maximum block length. As mentioned in Section 3.2, we can compute $s(j)$ by processing the recursive solutions corresponding to all right extensions $(x, f(x))$ with $f(x) \le j$. For the maximum block length there are two types of recursion for an optimal solution of $\mathsf{MSA}[1..m, 1..j']$ using semi-repeat-free $[x+1..j']$ as its last segment:

- **non-leader recursion:** if $j' \le x + s(x)$ then the score of $s(j')$ is equal to $s(x)$, because the length of segment $[x+1..j']$ is less than or equal to $s(x)$; in this case, we say that $[x+1..j']$ is a *non-leader segment*;
- **leader recursion:** otherwise, if $j' > x + s(x)$, we say that $[x+1..j']$ is a *leader segment*, since it gives score $j' - x$ to an optimal solution constrained to use it as its last segment.

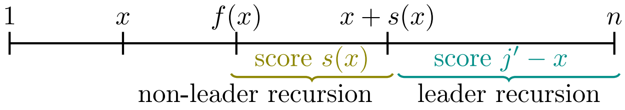


Fig. 4. Scheme for the score of an optimal semi-repeat-free segmentation of $\mathsf{MSA}[1..m, 1..j']$ constrained to use $[x+1..j']$ as its last segment.

Note that if $x + s(x) < f(x)$ then the non-leader recursion does not occur for $(x, f(x))$. Then, it is easy to see that

$$s(j) = \min\left(\min_{\substack{(x,f(x)):\\ f(x)\le j\le x+s(x)}} s(x), \min_{\substack{(x,f(x)):\\ j>f(x)\ \wedge\ j>x+s(x)}} j - x \right) \tag{2}$$

so Equi et al. correctly solve the problem by keeping track of the two types of recursions with two one-dimensional search trees: the first keeps track of ranges $[f(x)..x+s(x)]$ with score $s(x)$, the second tracks ranges $[x+s(x)+1..n]$ where the leader recursion must be used, saving only the $-x$ part of score $j' - x$. With two semi-infinite range minimum queries, for ranges $[j+1..+\infty]$ and $[-\infty..j]$ respectively, we can compute $s(j)$ and solve the problem in time $O(n \log\log n)$.

Instead, we can reach a linear time complexity using simpler data structures, thanks to the following observations: the data structure for the leader recursion can be replaced by a single variable S holding value $\min\{j - x : j > f(x) \wedge j > x + s(x)\}$, so that S is the best score of a segmentation ending with a leader segment $[x+1..j]$; for the non-leader recursion, we can swap the structure of Equi et al. with an equivalent array $\mathtt{C}[1..n]$ such that $\mathtt{C}[k]$ counts the number of available solutions with score k using the non-leader recursion so that a variable $K = \min\{k : \mathtt{C}[k] > 0\}$ is equal to the best score of a segmentation ending with a non-leader segment $[x+1..j]$. The final and crucial observation is that the two types of recursion are closely related: when $[x+1..j]$ goes from being a non-leader segment to a leader segment, that is, $j = x + s(x) + 1$, we decrease $\mathtt{C}[s(x)]$ by one and update S with value $s(x) + 1 = j - x$ if needed. Therefore, when the best score of $\mathtt{C}[1..n]$ is removed in this way, we do not need to update K to $\min\{k : \mathtt{C}[i] > 0\}$, but it is sufficient to increment K by 1 to ensure that $s(j) = \min(K, S)$, unless other updates of $\mathtt{C}$ and S result in a better score.

Theorem 2. *Given the minimal right extensions $(x, f(x))$ of $\mathsf{MSA}[1..m, 1..n]$, we can compute in time $O(n)$ the score of an optimal semi-repeat-free segmentation minimizing the maximum block length.*

Proof. The correctness of the algorithm, described in the full version of this paper [17], follows from that of [6, Algorithm 2] and from the fact that when $\mathtt{C}[K] = 0$ we have that $\mathtt{C}[j'] = 0$ for $1 \leq j' \leq K$ and $S \leq K + 1$. Similarly, the processing of minimal right extensions $(x, f(x))$ and the dynamic management of intervals $[f(x)..s(x) + j']$ takes time $O(n)$ in total, thus the algorithm takes linear time.

Combined with Theorem 1, we get our second main result.

Corollary 2. *Given $\mathsf{MSA}[1..m, 1..n]$ from $\Sigma \cup \{-\}$, with $\Sigma = [1..\sigma]$ and $\sigma \leq mn$, the construction of an optimal semi-repeat-free segmentation minimizing the maximum block length can be done in time $O(mn)$.*

References

1. Cazaux, B., Kosolobov, D., Mäkinen, V., Norri, T.: Linear time maximum segmentation problems in column stream model. In: Brisaboa, N.R., Puglisi, S.J. (eds.) String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11811, pp. 322–336. Springer (2019)
2. De La Briandais, R.: File searching using variable length keys. In: Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference. p. 295–298. IRE-AIEE-ACM '59 (Western), Association for Computing Machinery, New York, NY, USA (1959). https://doi.org/10.1145/1457838.1457895, https://doi.org/10.1145/1457838.1457895
3. Eggertsson, H.P., Kristmundsdottir, S., Beyter, D., Jonsson, H., Skuladottir, A., Hardarson, M.T., Gudbjartsson, D.F., Stefansson, K., Halldorsson, B.V., Melsted, P.: Graphtyper2 enables population-scale genotyping of structural variation using pangenome graphs. Nature Communications **10**(1), 5402 (Nov 2019). https://doi.org/10.1038/s41467-019-13341-9, https://doi.org/10.1038/s41467-019-13341-9
4. Equi, M., Grossi, R., Mäkinen, V., Tomescu, A.I.: On the complexity of string matching for graphs. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S. (eds.) 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece. LIPIcs, vol. 132, pp. 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
5. Equi, M., Mäkinen, V., Tomescu, A.I.: Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless seth fails. In: SOFSEM 2021: Theory and Practice of Computer Science. pp. 608–622. Springer International Publishing, Cham (2021)

6. Equi, M., Norri, T., Alanko, J., Cazaux, B., Tomescu, A.I., Mäkinen, V.: Algorithms and complexity on indexing elastic founder graphs. In: Ahn, H., Sadakane, K. (eds.) 32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan. LIPIcs, vol. 212, pp. 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.ISAAC.2021.20, https://doi.org/10.4230/LIPIcs.ISAAC.2021.20
7. Farach, M.: Optimal suffix tree construction with large alphabets. In: Proceedings 38th Annual Symposium on Foundations of Computer Science. pp. 137–143. IEEE (1997)
8. Garrison, E., Sirén, J., Novak, A., Hickey, G., Eizenga, J., Dawson, E., Jones, W., Garg, S., Markello, C., Lin, M., Paten, B.: Variation graph toolkit improves read mapping by representing genetic variation in the reference. Nature Biotechnology **36** (08 2018). https://doi.org/10.1038/nbt.4227
9. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. FOCS. pp. 549–554 (1989)
10. Jacobson, G.J.: Succinct static data structures. Carnegie Mellon University (1988)
11. Kim, D., Paggi, J., Park, C., Bennett, C., Salzberg, S.: Graph-based genome alignment and genotyping with hisat2 and hisat-genotype. Nature Biotechnology **37**, 1 (08 2019). https://doi.org/10.1038/s41587-019-0201-4
12. Mäkinen, V., Cazaux, B., Equi, M., Norri, T., Tomescu, A.I.: Linear time construction of indexable founder block graphs. In: Kingsford, C., Pisanti, N. (eds.) 20th International Workshop on Algorithms in Bioinformatics, WABI 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference). LIPIcs, vol. 172, pp. 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.WABI.2020.7, https://doi.org/10.4230/LIPIcs.WABI.2020.7
13. Mäkinen, V., Navarro, G., Sirén, J., Välimäki, N.: Storage and retrieval of highly repetitive sequence collections. Journal of Computational Biology **17**(3), 281–308 (2010)
14. Norri, T., Cazaux, B., Dönges, S., Valenzuela, D., Mäkinen, V.: Founder reconstruction enables scalable and seamless pangenomic analysis. Bioinformatics **37**(24), 4611–4619 (07 2021). https://doi.org/10.1093/bioinformatics/btab516, https://doi.org/10.1093/bioinformatics/btab516
15. Norri, T., Cazaux, B., Kosolobov, D., Mäkinen, V.: Linear time minimum segmentation enables scalable founder reconstruction. Algorithms Mol. Biol. **14**(1), 12:1–12:15 (2019)
16. Rizzo, N., Mäkinen, V.: Indexable elastic founder graphs of minimum height. In: Proc. 33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022) (2022), to appear.
17. Rizzo, N., Mäkinen, V.: Linear time construction of indexable elastic founder graphs. CoRR **abs/2201.06492** (2022), https://arxiv.org/abs/2201.06492
18. Schneeberger, K., Hagmann, J., Ossowski, S., Warthmann, N., Gesing, S., Kohlbacher, O., Weigel, D.: Simultaneous alignment of short reads against multiple genomes. Genome Biology **10**, R98 (2009)
19. Sirén, J., Välimäki, N., Mäkinen, V.: Indexing graphs for path queries with applications in genome research. IEEE/ACM Transactions on Computational Biology and Bioinformatics **11**(2), 375–388 (2014)
20. The Computational Pan-Genomics Consortium: Computational pan-genomics: status, promises and challenges. Briefings Bioinform. **19**(1), 118–135 (2018). https://doi.org/10.1093/bib/bbw089, https://doi.org/10.1093/bib/bbw089

21. Ukkonen, E.: On-line construction of suffix trees. Algorithmica **14**(3), 249–260 (1995). https://doi.org/10.1007/BF01206331, https://doi.org/10.1007/BF01206331