Frontiers in Bioinformatics

TYPE Original Research
PUBLISHED 26 September 2024
DOI 10.3389/fbinf.2024.1397036

Check for updates

OPEN ACCESS

EDITED BY
Leena Salmela,
University of Helsinki, Finland

REVIEWED BY
Jing Li,
Integrated DNA Technologies, United States
Cinzia Pizzi,
University of Padua, Italy

*CORRESPONDENCE
Nadia Pisanti,
nadia.pisanti@unipi.it

RECEIVED 06 March 2024
ACCEPTED 23 August 2024
PUBLISHED 26 September 2024

CITATION
Gabory E, Mwaniki MN, Pisanti N, Pissis SP, Radoszewski J, Sweering M and Zuba W (2024) Pangenome comparison via ED strings. *Front. Bioinform.* 4:1397036.
doi: 10.3389/fbinf.2024.1397036

Pangenome comparison via ED strings

Esteban Gabory[1], Moses Njagi Mwaniki[2], Nadia Pisanti[2]*, Solon P. Pissis[1,3], Jakub Radoszewski[4], Michelle Sweering[1] and Wiktor Zuba[1]

[1]Centrum Wiskunde & Informatica, Amsterdam, Netherlands, [2]Department of Computer Science, University of Pisa, Pisa, Italy, [3]Department of Computer Science, Vrije Universiteit, Amsterdam, Netherlands, [4]Institute of Informatics, University of Warsaw, Warsaw, Poland

Introduction: An elastic-degenerate (ED) string is a sequence of sets of strings. It can also be seen as a directed acyclic graph whose edges are labeled by strings. The notion of ED strings was introduced as a simple alternative to variation and sequence graphs for representing a pangenome, that is, a collection of genomic sequences to be analyzed jointly or to be used as a reference.

Methods: In this study, we define notions of *matching statistics* of two ED strings as similarity measures between pangenomes and, consequently infer a corresponding distance measure. We then show that both measures can be computed efficiently, in both theory and practice, by employing the *intersection graph* of two ED strings.

Results: We also implemented our methods as a software tool for pangenome comparison and evaluated their efficiency and effectiveness using both synthetic and real datasets.

Discussion: As for efficiency, we compare the runtime of the intersection graph method against the classic product automaton construction showing that the intersection graph is faster by up to one order of magnitude. For showing effectiveness, we used real SARS-CoV-2 datasets and our matching statistics similarity measure to reproduce a well-established clade classification of SARS-CoV-2, thus demonstrating that the classification obtained by our method is in accordance with the existing one.

KEYWORDS

elastic-degenerate string, intersection graph, pangenome comparison, matching statistics, SARS-CoV-2

1 Introduction

Many biomedical applications of bioinformatics face the twofold challenge of analyzing an ever-increasing number of genome sequences and the need to choose which genome should be used as the *reference*. Generalizing other definitions, in The computational Pangenomics Consortium (2018), a *pangenome* was defined as "any collection of genomic sequences to be analyzed jointly or to be used as a reference," somewhat merging the two above-mentioned challenges into that of analyzing a pangenome. When projected within a single species, a pangenome represents a collection of sequences that are (part of whole) genomes originating from different individuals or strains of a single clade or population.

Currently, *pangenomics* constitutes an important paradigm shift within genomics to deal with the widespread availability of human sequencing data and the discovery of

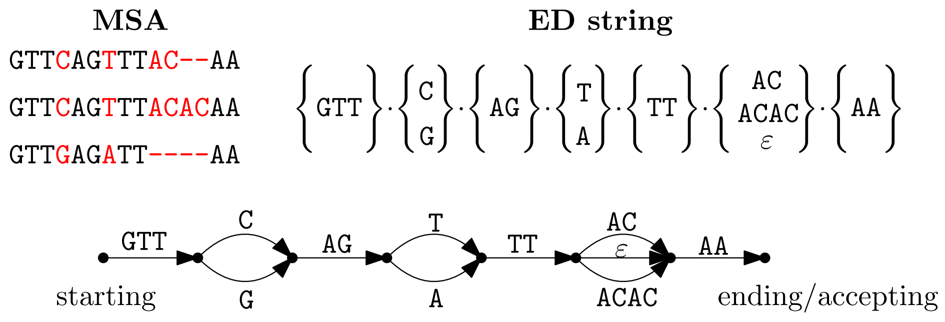

FIGURE 1
An example of an MSA (top left) and its corresponding (non-unique) ED string T of length $n = 7$, cardinality $m = 11$ and size $N = 20$ (top right), and edge-labeled DAG for T. Note that ε denotes the empty string. The DAG can also be viewed as an NFA with extended (multi-letter) transitions.

large-scale genomic variation in many eukaryotic species; see Paten et al. (2017); Liao et al. (2023). In contrast to a *linear* reference, a pangenome reference aims to compactly represent the variation within a population by encoding the commonalities and differences among the underlying sequences. This gives rise to different pangenome representations, usually edge- or node-labeled directed graphs (Baaijens et al., 2022; Carletti et al., 2019). The most widely-used pangenome representations are *variation graphs* (see Garrison et al., 2018a; Eizenga et al., 2021) and *sequence graphs* (see Rakocevic et al., 2019).

The computational challenges posed by pangenomes often result in a trade-off between the efficiency and accuracy of the methods and the information content of the chosen representation. A simpler *acyclic* alternative to the aforementioned representations is the notion of *elastic-degenerate string* (ED string); see Iliopoulos et al. (2021). When compared to more powerful representations, ED strings have the algorithmic advantage of supporting both theoretically (Aoyama et al., 2018; Bernardini et al., 2019; 2022)) and practically (Grossi et al., 2017; Pissis and Retha, 2018; Cisłak et al., 2018) fast *on-line* pattern matching, also for the approximate case (Bernardini et al., 2017; 2020). Moreover, they support fast dynamic-programming-based alignment, as shown in Mwaniki et al. (2023); Mwaniki and Pisanti (2022), and short-read mapping; see Büchler et al. (2023).

An ED string is a concatenation of n sets of strings (inspect Figure 1). Every set of strings encodes a collection of *consecutive columns* of an underlying multiple sequence alignment (MSA), from left to right. Every set encodes the commonalities or differences of the underlying sequences which are represented by the MSA. Formally, an ED string T is a sequence of n sets $T[1],\ldots,T[n]$ containing m strings in total whose cumulative length is N. We call n, m, and N the *length*, the *cardinality* and the *size* of T, respectively. An ED string T compactly represents all strings that can be spelled concatenating, for $1 \le i \le n$, a string chosen from set $T[i]$. For example, the string GTTCAGATTACAA is one of the strings represented by the ED string of Figure 1. Every ED string can also be viewed as an edge-labeled directed acyclic graph (DAG). As an example, Figure 1 shows the DAG of a simple ED string: the DAG has $n+1$ nodes as the ED string has length $n = 7$. The ED string may also be viewed as a nondeterministic finite automaton (NFA) Hopcroft et al. (2003) with extended transitions. By *extended*, we mean multi-letter transitions, instead of single-letter ones.

Any pangenome representation aims to improve the downstream analyses of genomic data by removing biases inherent in the use of a linear single-genome representation (Baaijens et al., 2022). For example, pangenomes allow for representing haplotype-resolved data with genome phasing, as shown in Bonizzoni et al. (2016). Using linear genomes as a reference, determining at which chromosomal copy (i.e., paternal or maternal) the different alleles are located, may be erroneous or incomplete due to reference bias. Genotyping (the task of reconstructing the allele variants that characterize an individual) can be improved by using a pangenome representation as a reference, which removes the bias of using a single linear genome as a reference to map the reads coming from an individual's sample. Pangenomes also allow for accurate read alignment as certain genome regions are important yet challenging to assemble using classic read alignment tools, because of the bias of using a single linear reference genome (Garrison et al., 2018b; Liao et al., 2023). This explains the high level of attention paid in recent years to the task of sequence-to-graph comparison; see, e.g., Büchler et al. (2023); Equi et al. (2023); Mwaniki et al. (2023); Li et al. (2020); Gibney et al. (2022); Jain et al. (2020); Rautiainen et al. (2019); Rautiainen and Marschal (2020). In phylogenetic analyses, the aim is to study the evolutionary relationships among different groups of organisms (e.g., species or population variants). This was traditionally performed using a sample sequence per organism that is somewhat representative of the group or population, and then inferring a tree or a graph based on pairwise distances or similarities among these samples. This task can be biased if the sample linear genome turns out not to be the most representative, whereas a pangenome can compactly represent the entire population.

Our Contributions. Here, we make an important step from the above-mentioned *sequence-to-graph* (i.e., sequence-to-pangenome) comparison towards graph-to-graph (pangenome-to-pangenome) comparison[1]. In particular, we assume that the two pangenomes to be compared are represented by means of two ED strings: T_1 and T_2.

1 A preliminary step on degenerate strings (D strings), that is, a restricted version of ED strings, was made by Alzamel et al. (2018), (2020).

We first recall a very fast and low memory consumption method from Gabory et al. (2023) to determine whether T_1 and T_2 have a nonempty intersection, that is, whether the two pangenomes share a genome. This method relies on a powerful representation of all string fragments that are *in both* T_1 *and* T_2, that is, of the complete set of sequences shared by the two pangenomes; this representation was named by Gabory et al. (2023) an *intersection graph* of T_1 and T_2. From thereon, we develop a novel method for pangenome comparison via ED strings, based on an extension to ED strings of the well-known notion of Matching Statistics on standard strings (*cf.* the book Gusfield (1997)). We define two versions of Matching Statistics for ED strings.

- **ED Matching Statistics** as the theoretically most straightforward notion that extends the standard one: for every $i \in [1, n_1]$ of T_1, where n_1 is the length of T_1, we report in $\mathsf{MS}_{T_1,T_2}[i]$ the length of the longest string starting at the ith set of T_1 that is also a substring of T_2.
- **Breakpoint Matching Statistics** as a notion specifically designed for genomic variants: for every $i \in [1, n_1]$ of T_1, we report in $\mathsf{BMS}_{T_1,T_2}[i]$ the length of the longest string starting at the ith set of T_1 that is also a substring of T_2 and that is within pairs of breakpoints that have been detected by the multiple alignment underlying the pangenome.

The output is the Matching Statistics array MS_{T_1,T_2} (resp. BMS_{T_1,T_2}) of size n_1 that specifies maximal local matches of T_1 with respect to T_2. The Matching Statistics array MS_{T_2,T_1} (resp. BMS_{T_2,T_1}) of T_2 with respect to T_1 is defined dually: for every $j \in [1, n_2]$ of T_2, where n_2 is the length of T_2, we report the longest string starting at the jth set of T_2 that fulfills the corresponding requirements with a substring of T_1. We then suggest to use the Matching Statistics to define the following measures.

- Similarity measure $\mathcal{MS}(T_1, T_2)$ (resp. $\mathcal{BMS}(T_1, T_2)$) between T_1 and T_2 as the sum of the average values of arrays MS_{T_1,T_2} and MS_{T_2,T_1} (resp. of arrays BMS_{T_1,T_2} and BMS_{T_2,T_1});
- Distance measure d (resp. bd) between T_1 and T_2 based on $\mathcal{MS}(T_1, T_2)$ (resp. on $\mathcal{BMS}(T_1, T_2)$).

Both distance measures can be trivially computed in $\mathcal{O}(1)$ time from $\mathcal{MS}(T_1, T_2)$ and $\mathcal{BMS}(T_1, T_2)$, and both similarity measures can be trivially computed in $\mathcal{O}(n_1 + n_2)$ time from MS_{T_1,T_2} and MS_{T_2,T_1} (resp. BMS_{T_1,T_2} and BMS_{T_2,T_1}). The algorithmic challenge is thus: *how can we efficiently compute the Matching Statistics arrays*? While an algorithm based on classic product automaton techniques (Lawson, 2004) would require $\Omega(N_1 N_2)$ time in the worst case, our method achieves this in $\mathcal{O}(N_1 m_2 + N_2 m_1)$ worst-case time, where N_1 and N_2 are the sizes of T_1 and T_2, respectively, and m_1 and m_2 are the cardinalities of T_1 and T_2, respectively. We achieve this via the above-mentioned intersection graph of T_1 and T_2. The running time of our algorithm is good in the following sense: if each of the ED strings, T_1 and T_2, represents a pangenome of closely-related genomes, then the cardinalities m_1 and m_2 are expected to be asymptotically much smaller than the sizes N_1 and N_2, respectively.

We also implemented the entire pipeline in C++ as a software tool for pangenome comparison, which is publicly available at https://github.com/urbanslug/junctions under GNU GPL v3.0.

We evaluated the efficiency and effectiveness of the developed methods using both synthetic and real datasets. For efficiency, we compared the runtime of the intersection graph against the classic product automaton construction. As expected, the intersection graph is faster by up to one order of magnitude. For effectiveness, we used real SARS-CoV-2 datasets and our *Breakpoint Matching Statistics* method to reproduce a well-established clades classification of SARS-CoV-2, thus demonstrating that the classification obtained by our method is in accordance with the existing classification.

In Section 2, we describe and analyze our methods. In Section 3, we present our results. We conclude this paper in Section 4.

2 Methods

In this section we recall from Gabory et al. (2023) the ED STRING INTERSECTION problem for two ED strings (Section 2.1) and the notion of *intersection graph* (Section 2.2). We then extend the MATCHING STATISTICS problem on two standard strings (Gusfield, 1997) to two ED strings defining the ED MATCHING STATISTICS and BREAKPOINT MATCHING STATISTICS problems, and show how to solve them using an intersection graph (Section 2.3). We also formally define our similarity and distance measures for ED strings (Section 2.4).

Let us begin with some basic definitions and notations from Gabory et al. (2023). An *alphabet* Σ is a finite nonempty set of elements called *letters*. By Σ^* we denote the set of all strings over Σ including the *empty string* ε of length 0. An *elastic-degenerate string* (ED string, in short) T is a sequence $T = T[1]\cdots T[n]$ of n finite sets, where $T[i]$ is a subset of Σ^*. The total size of T is defined as $N = N_\varepsilon + \sum_{i=1}^{n}\sum_{S\in T[i]}|S|$, where N_ε is the total number of empty strings in T. By m we denote the total number of strings in all $T[i]$, i.e., $m = \sum_{i=1}^{n}|T[i]|$. We say that T has *length* $n = |T|$, *cardinality* m and *size* N. The *language* of T is defined as $\mathcal{L}(T) = \{S_1 \cdot \ldots \cdot S_n : S_i \in T[i] \text{ for all } i \in [1, n]\}$, where $\cdot$ denotes the string concatenation.

2.1 ED strings intersection

Let us start by formally defining the ED STRING INTERSECTION problem.

ED STRING INTERSECTION (EDSI)

Input: Two ED strings, T_1 of length n_1, cardinality m_1 and size N_1, and T_2 of length n_2, cardinality m_2 and size N_2.

Output: YES if $\mathcal{L}(T_1)$ and $\mathcal{L}(T_2)$ have a nonempty intersection, NO otherwise.

This EDSI problem can be efficiently solved using the notion of *compacted nondeterministic finite automaton* (compacted *NFA*). A compacted NFA is a 5-tuple $(Q, \Sigma, \delta^{ext}, q_0, F)$, where Q is a finite set of *states*; Σ is an alphabet; $\delta^{ext}: Q \times \Sigma^* \to \mathcal{P}(Q)$, is an *extended transition* function, $\mathcal{P}(Q)$ is the power set of Q; $q_0 \in Q$ is the *starting* state; and $F \subseteq Q$ is the set of *accepting* states. Such an NFA can also be represented by a standard (uncompacted) NFA, where each extended transition is subdivided into standard one-letter transitions (and ε-transitions), $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$. The states of a compacted NFA are called *explicit*, whereas the states obtained from these subdivisions are called *implicit*. In both cases, given an (compacted or not) NFA A, we define the *language accepted* by A, denoted by $\mathcal{L}(A)$, as the set of strings that can be read from the starting state to an accepting state.

We next define the path-automaton of an ED string.

Definition 1. (Path-automaton of an ED string). *Let T be an ED string of length n, cardinality m, and size N. The path-automaton of T is the compacted NFA consisting of V states and E transitions defined as follows:*

- *$V = n + 1$ is the number of explicit states, numbered from one through $n + 1$. State one is the starting state and state $n + 1$ is the accepting state. State $i \in [2, n]$ is the state in-between $T[i - 1]$ and $T[i]$.*
- *$E = m = \sum_i m_i$, where $m_i = |T[i]|$ is the number of extended transitions from state i to state $i + 1$ labeled with the strings in $T[i]$.*

The path-automaton of T accepts exactly $\mathcal{L}(T)$. The uncompacted version of this path-automaton has $V^u = \mathcal{O}(N)$ states and $E^u = N$ transitions.

In the following, we are interested only in the graph underlying the path-automaton, that is, the directed acyclic graph (DAG), where every *node* represents an explicit state and every labeled directed *edge* represents an extended transition of the path-automaton (inspect also Figure 1). Indeed, it should be noticed that the path-automata of the ED strings are DAGs (direct acyclic graphs) as they are always acyclic, but may contain ε-transitions. For example, the DAG shown in Figure 1 is the path-automaton for T.

Checking whether two NFA have a nonempty intersection can be done in $\mathcal{O}(N_1N_2)$ time[2],[3], where N_1 and N_2 are the sizes of the two NFA, and therefore a naïve method can check whether $\mathcal{L}(T_1) \cap \mathcal{L}(T_2) \neq \varnothing$ in time $\mathcal{O}(N_1N_2)$, that is, quadratic in the sizes of T_1 and T_2. The compacted NFA representation allows for a more efficient algorithm for computing and representing the intersection. The idea is to use compacted transitions that directly compare whole string-segments instead of single letters, which can be performed efficiently using classic tools such as suffix trees or LCP queries.

Lemma 1. Gabory et al. (2023). Given two compacted NFA A_1 and A_2, with V_1 and V_2 explicit states and E_1 and E_2 extended transitions, respectively, a compacted NFA representing the intersection of A_1 and A_2 with $\mathcal{O}(V_1^uV_2 + V_1V_2^u)$ explicit states and $\mathcal{O}(E_1^uE_2 + E_1E_2^u)$ extended transitions can be computed in $\mathcal{O}(E_1^uE_2 + E_1E_2^u)$ time.

Lemma 1 thus implies the following result.

Corollary 1. The compacted NFA representing the intersection of two path-automata with $\mathcal{O}(N_1n_2 + N_2n_1)$ explicit states and $\mathcal{O}(N_1m_2 + N_2m_1)$ extended transitions can be constructed in $\mathcal{O}(N_1m_2 + N_2m_1)$ time.

2 A *breakpoint* in a genome is a location on a chromosome where DNA might get deleted, inverted, or swapped around Sankoff and Blanchette (1998).

3 This can be done using the folklore product automaton construction (Lawson, 2004).

Consequently, we can compute the compacted NFA of the intersection of two ED strings T_1 and T_2 (with cardinalities m_1 and m_2 and sizes N_1 and N_2) in $\mathcal{O}(N_1m_2 + N_2m_1)$ time. We remark that this compacted NFA is also an acyclic graph which we name the *intersection graph* of the two ED strings. As shown by Gabory et al. (2023), one can solve EDSI in practice without effectively constructing the entire graph, but rather part of it. However, since the intersection graph is crucial for the other methods that we describe in Section 2.3 and apply in this paper, we dedicate the following section to its description.

2.2 The intersection graph

In this section we describe the notion of the *intersection graph* G from the DAGs (of the two path-automata) G_1 and G_2 of T_1 and T_2 and how it can be used to solve the EDSI problem. We will do this by means of a running example of two ED strings T_1 and T_2. We refer the reader to Gabory et al. (2023) for the formal definition of G and the full details of an efficient $\mathcal{O}(N_1m_2 + N_2m_1)$-time construction algorithm.

Let us consider the two ED strings $T_1 = \left\{\begin{matrix}\texttt{AC}\\ \texttt{A}\\ \texttt{TGCT}\end{matrix}\right\} \cdot \left\{\begin{matrix}\varepsilon\\ \texttt{CA}\end{matrix}\right\}$ and $T_2 = \left\{\begin{matrix}\texttt{T}\\ \varepsilon\end{matrix}\right\} \cdot \left\{\begin{matrix}\texttt{AC}\\ \texttt{GCA}\end{matrix}\right\}$ that have a nonempty intersection as the string AC belongs to $\mathcal{L}(T_1) \cap \mathcal{L}(T_2)$. Their path-automata are represented by graphs G_1 and G_2 shown in Figure 2.

The nodes of the intersection graph correspond to pairs (i, j) from G_1 and G_2, respectively, where at least one of them must be an explicit state. As a consequence, we draw the intersection graph using, as a layout, a grid (in dotted lines) of $n_2 + 1$ copies of G_1 and $n_1 + 1$ copies of G_2. Therein, the possible nodes for the intersection graph G are pairs, as described above. Let (i, j) and (i', j') be nodes in G, with i, i' from G_1 and j, j' from G_2. We observe an extended transition from (i, j) to (i', j') with label S if one can read a string S both from i to i' in G_1 and from j to j' in G_2. Figure 3 shows the intersection graph G for T_1 and T_2: the intersection is nonempty and contains a single string AC that can be read on the red path.

2.3 Matching statistics for ED strings

When the ED strings T_1 and T_2 come from real pangenomes, being able to quickly tell whether $\mathcal{L}(T_1) \cap \mathcal{L}(T_2)$ is nonempty might not be informative enough for practical applications. Indeed, two pangenomes may still share a lot of fragments even if the two ED strings that represent them are such that $\mathcal{L}(T_1) \cap \mathcal{L}(T_2) = \varnothing$. Thus, to be more sensitive to local similarities and detect shared *fragments* of pangenomes, we consider *Matching Statistics* (Gusfield, 1997), a more elaborate ED string comparison task that is heavily employed for standard strings (under the Hamming distance model, i.e., with mismatches) in practical applications, especially in bioinformatics (Ulitsky et al., 2006; Apostolico et al., 2014; Leimeister and Morgenstern, 2014; Apostolico et al., 2016; Pizzi, 2016; Thankachan et al., 2017).

Let us first recall the classic MATCHING STATISTICS problem on standard strings.

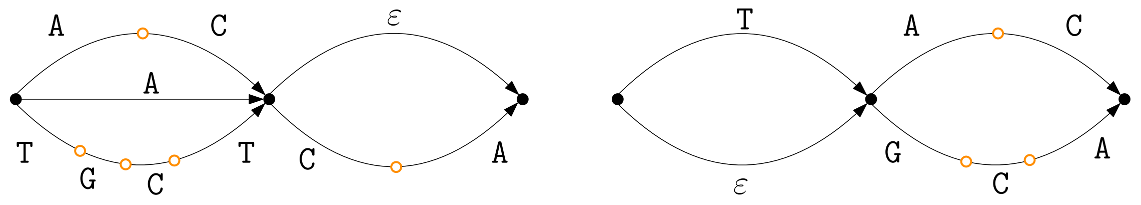

FIGURE 2
The two DAGs G_1 and G_2 for ED strings T_1 and T_2. The filled black nodes are explicit states, while the orange empty nodes are implicit states.

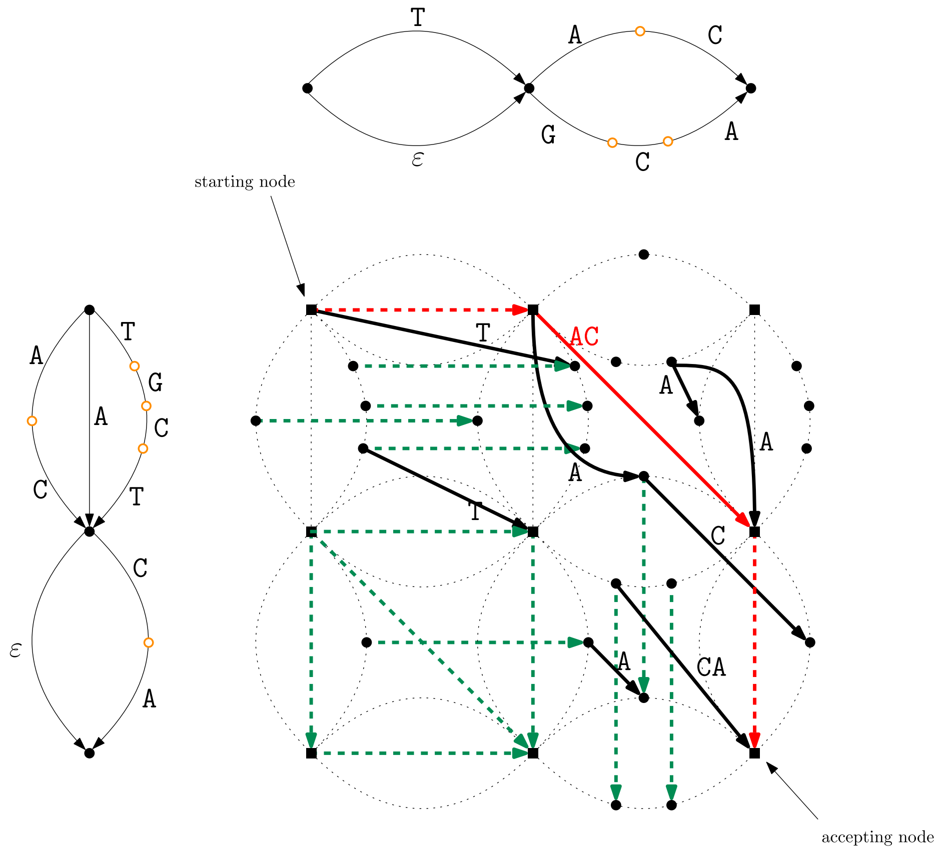

FIGURE 3
Intersection graph G for T_1 and T_2, where G_1 and G_2 are shown at the left and on the top, respectively, to simplify the understanding of G. A node (i,j) in the intersection is represented by a square if both i and j are explicit nodes, and by a circle if only one of them is. The dashed edges of the intersection graph G correspond to ε-transitions (namely, transitions such that no letter is read when traversed), while the solid edges correspond to the other extended transitions. A string in $\mathcal{L}(T_1) \cap \mathcal{L}(T_2)$ corresponds to a path from the starting node of G to the accepting node. Here the intersection is nonempty and contains a single string AC, which can be read on the red path.

MATCHING STATISTICS

Input: Two strings, S_1 of length n_1, and S_2 of length n_2.

Output: For each $i \in [1, n_1]$, the length $\mathsf{MS}_{S_1,S_2}[i]$ of the longest prefix of $S_1[i..n_1]$, which is a substring of S_2.

For example, the Matching Statistics of S_1 = AGTGCATTG of length nine and S_2 = TTG are the following array of size $|S_1| = 9$: $\mathsf{MS}_{S_1,S_2} = [0, 1, 2, 1, 0, 0, 3, 2, 1]$. The MATCHING STATISTICS problem can be solved in linear $\mathcal{O}(n_1 + n_2)$ time using the suffix tree of S_1 (Gusfield, 1997). In this section we extend the Matching Statistics notion, as well as the problem of computing them, to ED strings, in the direction of representing local similarities for pangenomes. We suggest two possible definitions of Matching Statistics for ED strings: the first one (Section 2.3.1) is the most inclusive notion, that is, it takes into account local matches that are prefixes of a string in $\mathcal{L}(T_1[i..n_1])$ and occur in some string from $\mathcal{L}(T_2)$; the second notion (Section 2.3.2) has a biologically motivated further condition: it assumes that relevant fragments are those that begin at positions of the genomes that the multiple alignment has detected as a *breakpoint*, meaning a locus of the resulting pangenome in which a variant (or a conserved fragment) either begins or ends. We will name the former notion *ED Matching Statistics* and the latter *Breakpoint Matching Statistics*.

2.3.1 ED matching statistics

The *ED Matching Statistics* between two ED strings T_1 of length n_1 and T_2 of length n_2, is an array MS_{T_1,T_2} of length n_1 storing, for each $i \in [1, n_1]$, the length of the longest local match between T_1 and T_2 which is a prefix of a string in $\mathcal{L}(T_1[i..n_1])$.

ED MATCHING STATISTICS

Input: Two ED strings, T_1 of length n_1, cardinality m_1 and size N_1, and T_2 of length n_2, cardinality m_2 and size N_2.

Output: For each $i \in [1, n_1]$, the length $\mathsf{MS}_{T_1,T_2}[i]$ of the longest prefix of a string in $\mathcal{L}(T_1[i..n_1])$, which is a substring of a string in $\mathcal{L}(T_2)$.

Example 1. Let us consider again $T_1 = \left\{\begin{matrix}\texttt{AC}\\\texttt{A}\\\texttt{TGCT}\end{matrix}\right\}\cdot\left\{\begin{matrix}\varepsilon\\\texttt{CA}\end{matrix}\right\}$ and $T_2 = \left\{\begin{matrix}\texttt{T}\\\varepsilon\end{matrix}\right\}\cdot\left\{\begin{matrix}\texttt{AC}\\\texttt{GCA}\end{matrix}\right\}$ of our running example. We have that $\mathsf{MS}_{T_1,T_2}[1] = 3$ and $\mathsf{MS}_{T_1,T_2}[2] = 2$. Indeed, the longest prefix of a string in $\mathcal{L}(T_1)$ that occurs in $\mathcal{L}(T_2)$ is TGC, having length 3, and the longest prefix of a string in $\mathcal{L}(T_1[2])$ that occurs in $\mathcal{L}(T_2)$ is CA, having length 2.

We observe that, in the intersection graph G of T_1 and T_2, the sought match starts at a node (i, j) where i is an explicit state of T_1. As a consequence, the intersection graph G can be used to efficiently compute the Matching Statistics of two ED strings, using the following algorithm:

We consider a slightly augmented version of the intersection graph obtained from an uncompacted intersection automaton. We first construct the automaton as in Corollary 1, and then we additionally compute some extra nodes and transitions as follows: when we process a state corresponding to a pair (i, j) (where i is from G_1 and j from G_2), and we have two transitions s and t having a nonempty common prefix and going out from i and j, respectively, then we construct the corresponding transition to the state (i', j'), where i' (resp. j') from G_1 (resp. from G_2) is the state that can be reached through the longest common prefix of s and t, even if both i' and j' are implicit.

We observe that, even in this case, the overall number of the transition pair checks remains the same; therefore the total size of the constructed underlying graph G remains $\mathcal{O}(N_1m_2 + N_2m_1)$. Indeed, in the final intersection graph, all the additional nodes are at most one edge away from a previously existing node; therefore the number of additional edges outgoing from an existing node is bounded by the number of strings that can be read from that node, that is, $\min(m_1, m_2)$.

We then assign to each edge the weight w storing the length of its string label and process the nodes in reversed topological order to compute, for each node k, the value $M(k)$ as follows: we set $M(k) = 0$ for the nodes that do not have successors (for example, the accepting node or nodes corresponding to a pair of implicit states), and then $M(k) = \max_{k'}(M(k') + w(k, k'))$ where k' iterates over all successors of k. By construction, for an explicit state i of G_1 and any state j of G_2, we have $M((i, j)) = \ell$ if and only if ℓ is equal to the maximal LCP between two strings S_1 and S_2, where $S_1 \in \mathcal{L}(T_1[i..n])$ and S_2 is spelled starting at (explicit or implicit) state j in G_2. Finally, for every explicit state i of G_1 we compute $\mathsf{MS}_{T_1,T_2}[i] = \max_v M((i, v))$ over all (explicit or implicit) states v of G_2 to obtain the output.

This ends the description of the proposed algorithm for the computation of ED Matching Statistics, which proves the following result.

Theorem 1. The ED MATCHING STATISTICS problem can be solved in $\mathcal{O}(N_1m_2 + N_2m_1)$ time by using an intersection graph of T_1 and T_2, which can be constructed within the same complexity.

Figure 4 shows how the Matching Statistics of T_1 and T_2 of our running example can be computed on their intersection graph G. For example, to compute $\mathsf{MS}_{T_1,T_2}[1]$, we look at the paths starting at nodes $(i = 1, j)$ in the path-automaton of T_1, and return the length of the longest label of such a path. The longest one of such paths (drawn in blue) corresponds to the string TGC having length 3, and thus $\mathsf{MS}_{T_1,T_2}[1] = 3$.

2.3.2 Breakpoint Matching Statistics

The *Breakpoint Matching Statistics* between two ED strings T_1 and T_2 refer to the notion of *breakpoint* in the genome rearrangement literature; see Baudet et al. (2010). An ED string T representing a pangenome results from a multiple sequence alignment of several genomes with the length n of T corresponding to the loci where either multiple variants or a conserved fragment begin: these are the *breakpoints* that the alignment has detected. The assumption underlying Breakpoint Matching Statistics is that biologically relevant fragments in pangenomes are those that begin at a breakpoint. The *Breakpoint Matching Statistics* between T_1 of length n_1 and T_2 of length n_2, is an array BMS_{T_1,T_2} of length n_1 storing, for each $i \in [1, n_1]$, the length of the longest local match between T_1 and T_2 that is a prefix of a string in $\mathcal{L}(T_1[i..n_1])$ and hence starts at a breakpoint in T_1, with the additional constraint that this must be part of a match that starts at a breakpoint in both T_1 and T_2 and ends at a breakpoint in at least one of the ED strings.

Formally, for any two ED strings, T_1 and T_2, a *breakpoint match* (b-match) of T_1 and T_2, for some $1 \le i_1 \le i_2 \le n_1$ and $1 \le j_1 \le j_2 \le n_2$, is a string S such that $S \in \mathcal{L}(T_1[i_1..i_2])$ and $S \in \mathcal{L}(T_2[j_1..j_2])$. Intuitively, S starts *and* ends at a breakpoint in both T_1 and T_2.

A string S is a *left-breakpoint match* (lb-match) if (i) $S \in \mathcal{L}(T_1[i_1..i_2])$ and S is a prefix of a string in $\mathcal{L}(T_2[j_1..n_2])$ *or* (ii) S is a prefix of a string in $\mathcal{L}(T_1[i_1..n_1])$ and $S \in \mathcal{L}(T_2[j_1..j_2])$. Intuitively, S begins at a breakpoint in both ED strings and ends at a breakpoint in at least one of the two ED strings. We denote this specific instance of S by $S^{i_2,j_2}_{i_1,j_1}$. Note that any b-match is also an lb-match.

Let us now formalize the problem of computing the Breakpoint Matching Statistics that we solve in this section.

BREAKPOINT MATCHING STATISTICS

Input: Two ED strings, T_1 of length n_1, cardinality m_1 and size N_1, and T_2 of length n_2, cardinality m_2 and size N_2.

Output: For each $i \in [1, n_1]$, the length $\mathsf{BMS}_{T_1,T_2}[i]$ of a longest prefix P of a string in $\mathcal{L}(T_1[i..n_1])$ that can be left-extended with a string from $\mathcal{L}(T_1[i_1..i-1])$ into an lb-match $S^{i_2,j_2}_{i_1,j_1}$, for some i_1, i_2, j_1, j_2.

The motivation for allowing a left-extension to an lb-match and not forcing P to be an lb-match is to maintain the property of the

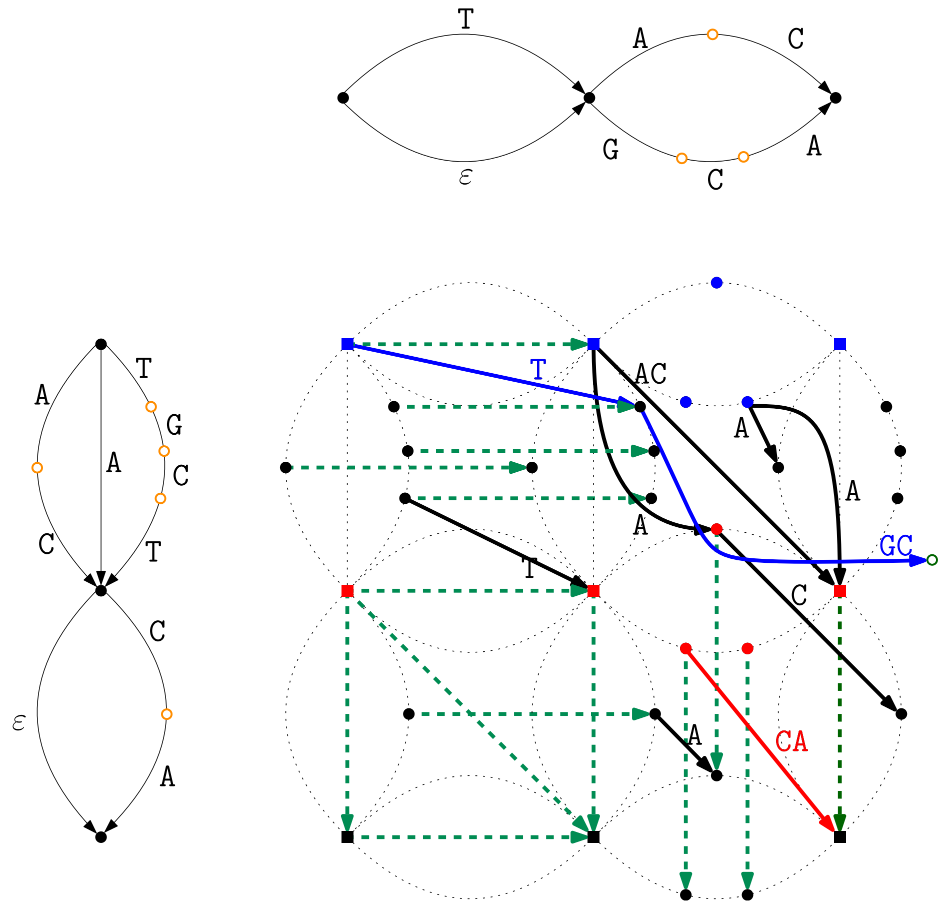

FIGURE 4
Matching Statistics of T_1 and T_2 of our running example on their intersection graph G, where, again - to simplify the understanding - we also draw G_1 and G_2 at the left and on the top, respectively. Note that this time, the pairs of implicit nodes that are reachable in a single extended transition from a pair that was previously computed are added. In the figure, there is only one such extra node, which is represented by a green open circle at the right of the graph. Here we highlight the paths that are relevant for computing the Matching Statistics array MS_{T_1,T_2}. To compute $\mathsf{MS}_{T_1,T_2}[1]$, we look at the paths starting at nodes (i,j) where i is the explicit state one in the path-automaton of T_1, and return the length of the longest label of such a path. These are the paths starting in one of the blue nodes (these are the nodes that correspond to the uppermost explicit node of G_1 paired with any node of G_2, that is, they correspond to the uppermost dotted copy of G_2). The longest one of such paths (also drawn in blue) corresponds to the string TGC having length 3; therefore, $\mathsf{MS}_{T_1,T_2}[1] = 3$. For $\mathsf{MS}_{T_1,T_2}[2]$ we do the same but using as starting nodes those in red that correspond to the internal explicit node of G_1 paired with any node of G_2 (i.e., the nodes of the middle dotted copy of G_2). Here the longest path is drawn in red and it spells the string CA, and therefore we set $\mathsf{MS}_{T_1,T_2}[2] = 2$. Computing MS_{T_2,T_1} can be performed in a dual manner on the same graph, but using as starting nodes those of the leftmost dotted copy of G_1 for $\mathsf{MS}_{T_2,T_1}[1]$, and those of the middle dotted copy of G_1 for $\mathsf{MS}_{T_2,T_1}[2]$.

standard Matching Statistics of not decreasing rapidly from $\mathsf{MS}_{S_1,S_2}[i]$ to $\mathsf{MS}_{S_1,S_2}[i+1]$.

Example 2. Let $T_1 = \left\{\begin{matrix}\mathtt{AC}\\ \mathtt{A}\\ \mathtt{TGCT}\end{matrix}\right\} \cdot \left\{\begin{matrix}\varepsilon\\ \mathtt{CA}\end{matrix}\right\}$ and $T_2 = \left\{\begin{matrix}\mathtt{T}\\ \varepsilon\end{matrix}\right\} \cdot \left\{\begin{matrix}\mathtt{AC}\\ \mathtt{GCA}\end{matrix}\right\}$ as in Example 1. We have that $\mathsf{BMS}_{T_1,T_2}[1] = 2$, because $P = \mathtt{AC}$ starting at position $i = 1$ is equal to $S = \mathtt{AC}$, which is a b-match (for $i_1 = i_2 = 1$ and $j_1 = j_2 = 2$) and hence an lb-match. We also have $\mathsf{BMS}_{T_1,T_2}[2] = 1$, because $P = \mathtt{C}$ starting at position $i = 2$ is left-extended to $S = \mathtt{AC}$, which is an lb-match (for $i_1 = 1$, $i_2 = 2$ and $j_1 = j_2 = 2$). For both cases ($i = 1$ and $i = 2$), we have that P is the longest possible such prefix.

We now show that the intersection graph G of T_1 and T_2 can also be used to efficiently compute their Breakpoint Matching Statistics. Indeed, the local matches that the Breakpoint Matching Statistics account for can be characterized in the intersection graph G of T_1 and T_2 as the common substrings that start at a node (i, j) where i is an explicit state of T_1, and j can be either explicit or implicit in T_2. Additionally, (i, j) must be reachable in G from a node (i_1, j_1) where i_1 and j_1 are both explicit: they correspond to a common breakpoint of the two pangenomes. Moreover, if such a substring ends at node (i_2, j_2), then at least one state among i_2 and j_2 must be explicit. Notice that, should j be an explicit node, then the reachability condition above can be fulfilled by $j = j_1$ itself; in that case we also have $i = i_1$. On the other hand, if j is implicit, then it must be that $i_1 \neq i$ and $j_1 \neq j$.

Figure 5 shows the computation of the Breakpoint Matching Statistics in the intersection graph G of our running example. For example, for $\mathsf{BMS}_{T_1,T_2}[1]$, we use the occurrence of AC (blue edge) starting at a node that corresponds to a pair of explicit states and ending at a node that also corresponds to a pair of explicit states (a

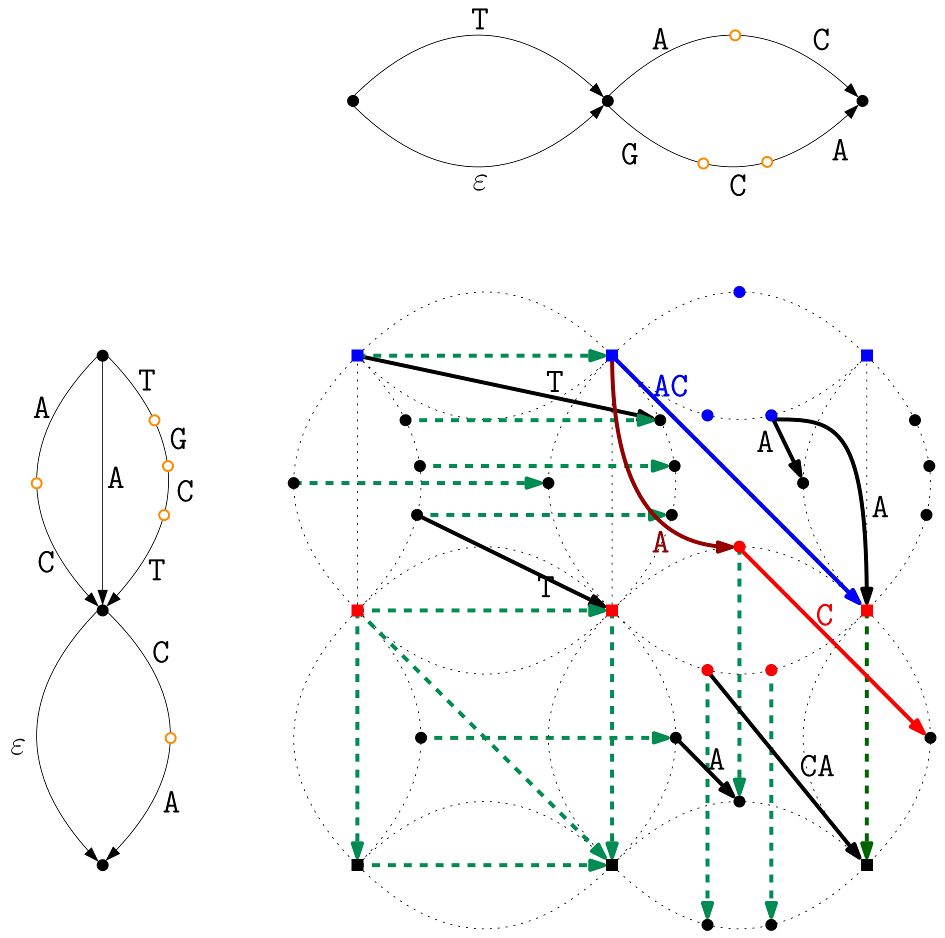

FIGURE 5
Breakpoint Matching Statistics computation in the intersection graph G of T_1 and T_2. To compute $\mathsf{BMS}_{T_1,T_2}[1]$, the candidate starting nodes of the match in G are those in blue: nodes (i,j) where i is an explicit state of T_1 in the uppermost dotted copy of G_2, and j is either an explicit state of T_2 (squared blue nodes) or an implicit one (circled blue nodes). Note that TGC is the longest match that starts at the first set of T_1 but it does not fulfill the conditions for the Breakpoint Matching Statistics because it does not end at any breakpoint; for the same reason, TG is also not a good candidate match. The occurrence of AC corresponding to the blue edge starts at a blue square node; hence it is reachable from the node itself that corresponds to a pair of explicit states, and it ends at a node that is again a pair of explicit states, and hence a breakpoint for both T_1 and T_2. There is no longer match satisfying these conditions; therefore we set $\mathsf{BMS}_{T_1,T_2}[1] = 2$. For $\mathsf{BMS}_{T_1,T_2}[2]$ we do the same but use as starting nodes those in red that correspond to the internal explicit node of G_1 paired with any node of G_2 (i.e., the nodes of the middle dotted copy of G_2). The red path spelling C: (i) is a prefix in $T_1[2]$ starting at an explicit node of T_1; (ii) is reachable from a square node in G by spelling A in both strings (curved brown red edge labeled with A); and (iii) ends where $T_2[2]$ does, that is, at a breakpoint. Since this is the longest such path in G, we set $\mathsf{BMS}_{T_1,T_2}[2] = 1$. Note, for example, that the match CA that occurs in $T_1[2]$ and inside $T_2[2]$ cannot be used for $\mathsf{BMS}_{T_1,T_2}[2]$ because it starts at a node that is not reachable from a pair of explicit nodes, meaning that it is not upperbounded by a breakpoint in T_2. Computing BMS_{T_2,T_1}, which is of size $n_2 = 2$, can be done in a dual manner on the very same graph, using as starting nodes those of the leftmost dotted copy of G_1 for $\mathsf{BMS}_{T_2,T_1}[1] = 2$ (obtained by traversing an ε-transition and then AC), and those of the middle dotted copy of G_1 for $\mathsf{BMS}_{T_2,T_1}[2] = 2$ (AC again).

breakpoint for both T_1 and T_2). No longer match satisfies these conditions; hence, we set $\mathsf{BMS}_{T_1,T_2}[1] = 2$.

Notice that the Breakpoint Matching Statistics require more restricted matches tī ED Matching Statistics. Indeed we have that for any two ED strings T_1 and T_2, it holds that $\mathsf{BMS}_{T_1,T_2}[i] \leq \mathsf{MS}_{T_1,T_2}[i]$ for all $1 \leq i \leq n_1$. It should be clear that Breakpoint Matching Statistics can be computed within the same complexities as the ones described in Theorem 1. We thus obtain the following.

Theorem 2. *The* BREAKPOINT MATCHING STATISTICS *problem can be solved in $\mathcal{O}(N_1 m_2 + N_2 m_1)$ time by using an intersection graph of T_1 and T_2, which can be constructed within the same complexity.*

2.4 Our measures for comparing pangenomes

In this section we describe a similarity and a distance measure for pangenome comparison. These measures can be derived from either the MS or the BMS array. We assume that these have been pre-computed.

We consider both arrays MS_{T_1,T_2} and MS_{T_2,T_1} (resp. BMS_{T_1,T_2} and BMS_{T_2,T_1}) as the Matching Statistics is not *per se* a symmetric notion: the two arrays do not even need to have the same size (when $n_1 \neq n_2$). A simple solution for a similarity measure is to consider the sum of the two averages: each array is turned into a number being the average of its values, and the sum makes everything symmetric.

Thus, we define the *similarity measure* $\mathcal{MS}(T_1,T_2)$ (resp. $\mathcal{BMS}(T_1,T_2)$) between T_1 and T_2 as follows:

$$\mathcal{MS}(T_1,T_2) = \mathcal{MS}(T_2,T_1) = \frac{\sum_{i\in[1,n_1]}\mathsf{MS}_{T_1,T_2}[i]}{n_1} + \frac{\sum_{j\in[1,n_2]}\mathsf{MS}_{T_2,T_1}[j]}{n_2}$$

and

$$\mathcal{BMS}(T_1,T_2) = \mathcal{BMS}(T_2,T_1) = \frac{\sum_{i\in[1,n_1]}\mathsf{BMS}_{T_1,T_2}[i]}{n_1} + \frac{\sum_{j\in[1,n_2]}\mathsf{BMS}_{T_2,T_1}[j]}{n_2}$$

We now move further in order to design a notion of *distance* between pangenomes based on $\mathcal{MS}(T_1,T_2)$ (resp. $\mathcal{BMS}(T_1,T_2)$). Unlike the notion of similarity, the distance has to decrease when the two pangenomes get more similar; hence, following a standard procedure, we invert the similarity measure while normalizing over the logarithm of the size of the pangenome. The reason for the normalization is that the values inside arrays MS_{T_1,T_2} and BMS_{T_1,T_2} are affected by the sizes of both strings–a very short ED string cannot contain a long match. Therefore, for a given T_1, to account for the size N_2 of T_2, we normalize $\mathcal{MS}(T_1,T_2)$ (resp. $\mathcal{BMS}(T_1,T_2)$) by $\log N_2$ and then invert[4], thus obtaining $\log N_2/\mathcal{MS}(T_1,T_2)$ (resp. $\log N_2/\mathcal{BMS}(T_1,T_2)$). Again, this gives rise to a non-symmetric notion, while symmetry is a desired property for a distance. Another desired property is reflexivity, requiring any pangenome to have distance zero from itself. The latter can be ensured by subtracting[5] a "correction term" as follows:

$$\bar{d}(T_1,T_2) = \frac{\log N_2}{\mathcal{MS}(T_1,T_2)} - \frac{\log N_1}{\mathcal{MS}(T_1,T_1)}.$$

and

$$\overline{bd}(T_1,T_2) = \frac{\log N_2}{\mathcal{BMS}(T_1,T_2)} - \frac{\log N_1}{\mathcal{BMS}(T_1,T_1)}.$$

thus guaranteeing that $\bar{d}(T_1,T_1) = \overline{bd}(T_1,T_1) = 0$ for any nonempty T_1. However, both $\bar{d}$ and $\overline{bd}$ are not symmetric yet, and hence, we finally ensure that our distance is *symmetric* resorting again to the sum. Therefore, we set:

$$d(T_1,T_2) = d(T_2,T_1) = \bar{d}(T_1,T_2) + \bar{d}(T_2,T_1).$$

and

$$bd(T_1,T_2) = bd(T_2,T_1) = \overline{bd}(T_1,T_2) + \overline{bd}(T_2,T_1).$$

Our d and bd distances resemble an analogous widely used distance measure for standard sequences such as *MissMax* (Pizzi, 2016) that is based on Matching Statistics with mismatches, and *kmacs* (Ulitsky et al., 2006; Leimeister and Morgenstern, 2014; Thankachan et al., 2017) that is based on an approximation of Matching Statistics with mismatches.

4 We assume that $\mathcal{MS}(T_1,T_2), \mathcal{BMS}(T_1,T_2) > 0$.

5 For a nonempty T_1, the quantities $\mathcal{MS}(T_1,T_1)$ and $\mathcal{BMS}(T_1,T_1)$ are always greater than zero

3 Experiments

We implemented the $\mathcal{O}(N_1m_2 + N_2m_1)$-time algorithm for solving EDSI as well as an $\mathcal{O}(N_1N_2)$-time algorithm for EDSI based on the classic product automaton construction. We also implemented the $\mathcal{O}(N_1m_2 + N_2m_1)$-time algorithm computing both Matching Statistics notions as well as the downstream similarities and distance measures for any two ED strings. All implementations were written in `C++` and the source code is freely available at https://github.com/urbanslug/junctions. We compiled all implementations with `gcc` version 12.2.0 at optimization level (`-O3`).

3.1 Efficiency on simulated data

In this section, we compare the running time of our EDSI with that of the naïve method and with the parameters that define the characteristics of the input ED strings, with the purpose of validating the time efficiency of our algorithm and show how it actually improves in practice with respect to the baseline quadratic running time. In order to do that, we use synthetic data generated on the alphabet {`A`, `C`, `G`, `T`}.

The synthetic ED strings were generated using another tool of ours named SIMED (https://github.com/urbanslug/simed). The tool starts by generating a random standard string of length W over the DNA alphabet, assuming a uniform distribution of letters. This is considered to be an initial sequence. We can view this as an ED string with $N = W$ and $n = m = 1$. The SIMED tool assumes a very simple evolutionary model (where each position has an equal chance of mutating, and each letter has the same probability of occurring at any position) and generates an ED string from the initial sequence based on the following input parameters.

- W as the length of the initial random (not ED yet) string;
- d as the number of positions where a set of multiple strings occurs, given as a percentage of W (that is, d is the fraction of degenerate positions);
- S as the maximum number of strings in any set of the resulting ED string;
- L as the maximum length of the strings in any set of the resulting ED string.

As aforementioned, the tool first generates a standard string uniformly at random, which we denote by X ($|X| = W$). It then samples $\delta = \lceil dW \rceil$ non-overlapping length-L substrings of X uniformly at random. We denote these by $X[i_1..i_1 + L - 1], \ldots, X[i_\delta..i_\delta + L - 1]$, where $i_j + L \le i_{j+1}$ for $j \in [1, \delta - 1]$. For every $j \in [1, \delta]$, it picks a uniformly random integer s from $[1, S]$ and produces $s - 1$ strings of uniformly random lengths from $[0, L]$, each string generated uniformly at random; these $s - 1$ strings and the original fragment $X[i_j..i_j + L - 1]$ form a set D_{i_j} of strings. Finally it sets T as $X[1..i_1 - 1] \cdot D_{i_1} \cdot X[i_1 + L..i_2 - 1] \cdot D_{i_2} \cdots D_{i_\delta} \cdot X[i_\delta + L..W]$. Note that T is indeed an ED string; we denote its length, cardinality, and size by n, m, N, respectively. If we choose d, S, W such that $(\delta + \delta \cdot S) \ll W$ then we have that $m \le (\delta + \delta \cdot S) \ll W \le N \Rightarrow m \ll N$. It is worth noting that if the same initial string X is used to generate two distinct ED strings, then X will appear in their (nonempty) intersection.

TABLE 1 Results with simulation parameters: d = 0.1% with S = 3, L = 3 and with S = 5, L = 5.

W	N_1	m_1	N_2	m_2	Naïve (s)	EDSI (s)
			S = 3 and L = 3			
10k	10,019	36	10,023	38	0.69	0.04
30k	30,062	107	30,071	107	6.20	0.14
50k	50,106	173	50,110	172	17.57	0.29
100k	100,225	354	100,203	344	72.81	0.47
			S = 5 and L = 5			
10k	10,084	49	10,066	50	0.68	0.06
30k	30,198	144	30,212	148	6.21	0.16
50k	50,381	244	50,358	250	18.00	0.29
100k	100,837	515	100,776	500	74.04	0.65

Starting from the same base sequence X of length W, in each experiment described in this section, we used the same parameters d, L and S to generate both T_1 and T_2. Thus, X guarantees a nonempty intersection between T_1 and T_2, and both implementations always answered YES (as expected) without a premature ending (hence, detecting their worst-case running time). As expected, the improved $\mathcal{O}(N_1m_2 + N_2m_1)$-time implementation of EDSI was faster than the naïve $\mathcal{O}(N_1N_2)$-time implementation in all datasets, especially with longer variants and/or with short widely interspersed variants, that is for ED strings where $m \ll N$. Results are shown below.

We report a table for each set of parameters, and in each table, we show the data for different starting synthetic string lengths W, up to $|W|$ = 100k bases. The data reported in the columns of Tables 1, 2 are: the length W of the initial string, the size N_1 and cardinality m_1 of the first synthetic ED string, the size N_2 and cardinality m_2 of the second synthetic ED string, and the time taken by the Naïve method and by EDSI, both measured in seconds. The parameters d (frequency of positions with multiple variants), S (maximum number of variants in such positions), and L (maximum length of such variants) determine the degree of *degeneracy* of the ED strings. As shown below, we have $m \ll N$ because (i) wherever the sequence is not degenerate, N grows linearly with W while m is constant, and (ii) wherever there is a degenerate position, $N \in \mathcal{O}(S \times L)$ while $m \in \mathcal{O}(S)$. This explains why our $\mathcal{O}(N_1m_2 + N_2m_1)$-time algorithm is much faster than the $\mathcal{O}(N_1N_2)$-time one.

The tables show that the Naïve scales quadratically in the size of the ED strings while EDSI is much faster as $m \ll N$. A comparison of the second and third experiments reported in Tables 2 highlights how, when only L grows (it doubles from 5 to 10 while d and S remain 1% and 5, respectively), our tool has basically the same performance whereas the Naïve becomes slower. The explanation is that when L grows, only N grows while m does not (as we can see), and hence m and N diverge even more. Finally, we remark that the parameter that most affects m/N (i.e., the ratio of our asymptotic gain with respect to the Naïve) is d, as the comparison of Tables 1 and 2 shows for the corresponding values of S and L.

These experiments were conducted on a laptop with a 64 bit 11th Gen Intel(R) Core(TM) i7-11800H 8 core processor and 16 GB of

TABLE 2 Simulation parameters: d = 1% with S = 3, L = 3, with S = 5, L = 5, and with S = 5, L = 10.

W	N_1	m_1	N_2	m_2	Naïve (s)	EDSI (s)
			S = 3 and L = 3			
10k	10,218	346	10,232	348	0.70	0.05
30k	30,688	1,064	30,659	1,040	6.46	0.21
50k	51,155	1758	51,104	1752	18.84	0.48
100k	102,227	3,469	102,258	3,497	77.86	1.71
			S = 5 and L = 5			
10k	10,838	504	10,796	494	0.80	0.06
30k	32,362	1,479	32,415	1,505	7.36	0.25
50k	54,098	2,508	54,146	2,525	20.84	0.56
100k	108,071	4,987	107,947	4,986	84.62	1.89
			S = 5 and L = 10			
10k	11,696	498	11,803	500	0.96	0.06
30k	35,405	1,531	35,140	1,495	8.83	0.25
50k	58,745	2,503	58,659	2,484	25.22	0.59
100k	117,444	4,985	117,417	4,989	101.10	1.97

RAM. Timings were recorded using `std::chrono` from the C++ standard library.

3.2 Efficiency on human genome data

In this section, we present some experiments for the running time of EDSI on real human genome data with variants. The goal is to show that our tool is fast even on real data, as the ratio between m and N is not too large for real pangenomes built out of real human genome fragments and their Variant Call Format (VCF) data. We built ED strings for human genome data from the GRCh38.p13 dataset, specifically from HLA-B in chromosome VI as well as the actin beta (ACTB) gene in chromosome VII. We used human genomic sequence data in the FASTA format and variation data in the Variant Call Format (.vcf file) from the following three databases: 1000G https://www.internationalgenome.org/ (2024), TOPmed https://topmed.nhlbi.nih.gov/ (2024), and gnomAD https://gnomad.broadinstitute.org/ (2024).

The human leukocyte antigen (HLA) gene is contained in the major histocompatibility complex on the p arm (chromosomal region 6p21.33) of Chromosome VI which is known to be one of the most polymorphic regions of the human genome. The HLA gene is involved in immune system regulation (Crux and Elahi, 2017; Romero-Sánchez et al., 2021) and is found in the region between positions 31,353,872 and 31,367,067 (hence it is 13 kb long). ACTB is a highly conserved gene in humans that codes for several proteins involved in cell structure and integrity. For each genome fragment (HLA and ACTB data), and for each database (out of the three 1000G, TOPmed, and gnomAD), we selected from the .vcf file only the variants that are recorded in that specific database, and we updated the regions containing variation, as denoted in the.vcf file

TABLE 3 ED strings with databases and VCF and sizes, and time (in seconds) required by Naïve and by EDSI for HLA data.

DB_1	VCF_1	N_1	m_1	DB_2	VCF_2	N_2	m_2	Naïve (s)	EDSI (s)
1000G	all	13,332	224	1000G	SNP	13,247	161	1.25	0.05
TOPMed	all	15,090	3,452	gNomad	SNP	13,941	1785	2.11	1.06
gNomad	all	14,436	2044	TOPMed	SNP	14,355	3,170	2.10	1.13

TABLE 4 ED strings with databases and VCF and sizes, and time (in seconds) required by Naïve and by EDSI for ACTB data.

DB_1	VCF_1	N_1	m_1	DB_2	VCF_2	N_2	m_2	Naïve (s)	EDSI (s)
1000G	all	37,019	644	gNomad	SNP	37,876	3,146	9.82	0.53

into sets containing both the original in the reference and the variants contained in.vcf, thus creating an ED string. We performed this in two ways: one for all variants of that database for that genome fragment, and one by selecting the SNPs variants only. We then used AEDSO (https://github.com/urbanslug/aedso) to build the ED strings. Data download links and commands used are available at https://github.com/urbanslug/junctions/blob/master/test_data/human.org.

In summary, we have two ED strings (one with all variants and one with SNPs only) per each database, and each genome fragment. We remark that all of these ED strings include the original non-mutated string in the language; hence for each pair *the intersection is nonempty*. This ensures detecting a running time of EDSI without a premature ending due to empty intersection: we ran EDSI for all pairs. For the HLA data, Table 3 shows the sizes (and types) of the ED strings and the running times of a few of these experiments. Table 4 shows results for the ACTB data. We observe that EDSI improves over Naïve by one order of magnitude whenever $m \ll N$ (1000G and gNomad variants datasets), and still improves over the Naïve even when N/m is a small constant, like with TOPMed data.

Finally, to conduct an experiment on these data with larger inputs, we picked a larger fragment of reference from Chromosome VI (spanning over the HLA region) of length 100Kb, and we repeated the same procedure as above. Table 5 presents the results of the experiment. We observe that even for these longer ED strings, EDSI is generally significantly faster than the Naïve method, especially on data such as that of the 1000G variants dataset–therein the ratio between N and m is larger than in the other data.

These experiments were conducted on a laptop with a 64 bit 11th Gen Intel(R) Core(TM) i7-11800H 8 core processor and 16 GB of RAM. Timings were recorded using `std::chrono` from the C++ standard library.

3.3 Similarity of SARS-CoV-2 clades

To demonstrate the effectiveness of the Breakpoint Matching Statistics and the similarity measure based on them, we computed the BMS arrays and the $\mathcal{BMS}$ similarity measures for all pairs of clades of real SARS-CoV-2 data downloaded from NextStrain[6] Hadfield et al. (2018), a platform collecting SARS-CoV-2 evolution analyses, built out of GenBank https://www.ncbi.nlm.nih.gov/genbank/ (2024) data. As an example of the NextStrain report, Figure 7 shows a phylogenetic tree of 3357 SARS-CoV-2 genomes sampled between December 2019 and August 2023, constructed using a complex pipeline described in https://docs.nextstrain.org/en/latest/learn/parts.html (2024).

We selected 35 SARS-CoV-2 clades as classified by NextStrain in https://nextstrain.org/ncov/open/global/alltime (2024) and, within each of them, we downloaded randomly selected individual genome samples (10 when available, and less otherwise), ruling out a few clades with too few samples: we were left with 31 clades. We provide the raw datasets at https://github.com/urbanslug/junctions/tree/master/test/_data/SARS/_CoV/_2.

For each clade, we constructed a multiple sequence alignment (MSA) using abPOA (Gao et al., 2020) and, from each such MSA, we generated the corresponding ED string using MSA2EDS[7]. Our tool MSA2EDS constructs ED strings from an MSA by simply collapsing 100% conserved columns (that is, columns with the same letter in all variants) into solid letters in the ED string, and into sets of distinct variants otherwise.

For all pairs h, k of these 31 SARS-CoV-2 clades, we computed the Breakpoint Matching Statistics arrays BMS_{T_k,T_h} and BMS_{T_h,T_k}, and the consequent pairwise similarity $\mathcal{BMS}(T_k, T_h)$[8].

Figure 6 shows the graph generated using NetworkX's *spring_layout* toolkit (Hagberg et al., 2008) when given all pairwise $\mathcal{BMS}$ among the 31 clades as input parameters. NetworkX's algorithm simulates a force-directed representation of the network, treating nodes as repelling objects and edges as springs that hold the nodes close according to the input similarity measures. The simulation continues until the positions are close to an equilibrium, which results in a graph in which closely-related (that is, similar according to our $\mathcal{BMS}$ measure) clades are clustered together. We also computed, for all pairs h, k of the clades, the Matching Statistics arrays MS_{T_k,T_h} and MS_{T_h,T_k}, and the consequent pairwise similarity

6 https://nextstrain.org/ncov/open/global/all-time

7 https://github.com/urbanslug/junctions

8 A complete table with all pairwise $\mathcal{BMS}$ scores among clades is available at the following url:https://github.com/urbanslug/junctions/blob/master/test_data/SARS/CoV/2/BPMS/similarity.tsv.

TABLE 5 ED strings with databases and VCF and sizes, and time (in seconds) required by Naïve and by EDSI for a fragment of data.

DB_1	VCF_1	N_1	m_1	DB_2	VCF_2	N_2	m_2	Naïve (s)	EDSI (s)
1000G	all	100,951	3,730	1000G	SNP	101,252	2,753	73	1.12
TOPMed	all	113,111	21,253	TOPMed	SNP	108,669	18,931	99.04	21.44
gNomad	all	130,918	42,572	gNomad	SNP	117,793	38,877	150.72	109.83

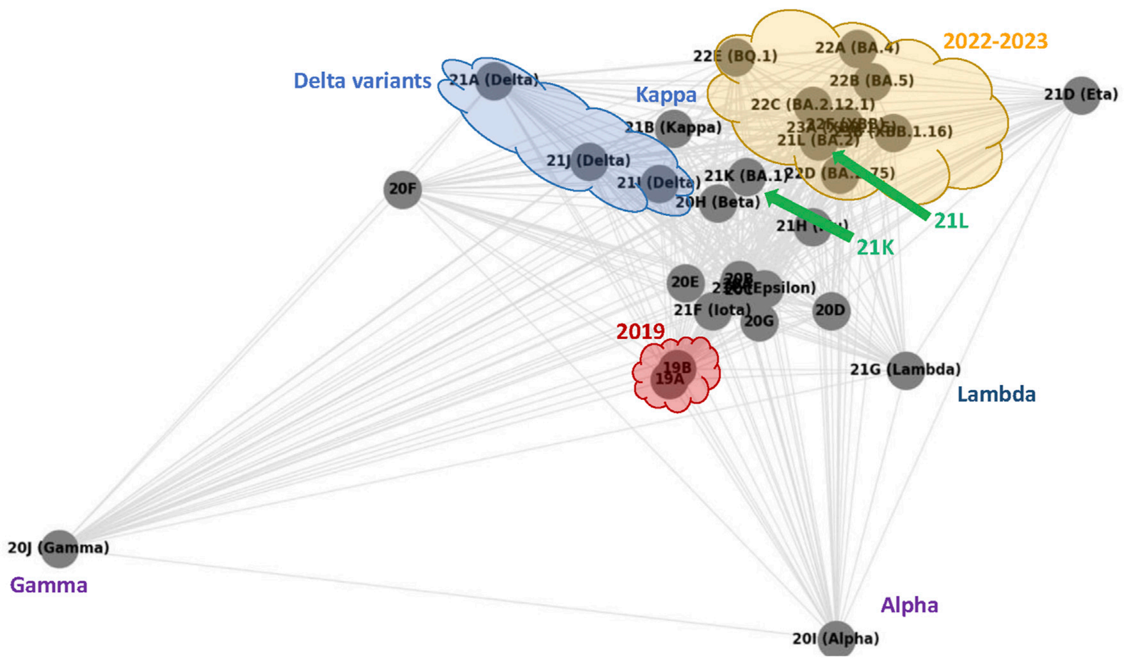

FIGURE 6
SARS-CoV-2 clades pairwise similarity graph generated according to average Breakpoint Matching Statistics. The annotation (all non grey nor black graphics and text) highlights similarities with Figure 7.

$\mathcal{MS}(T_k, T_h)$. However, the graph built using NetworkX's *spring_layout* with the Matching Statistics similarity measures (data not shown here) did not exhibit clade clusters as significant as those in Figure 6 generated with the Breakpoint Matching Statistics similarity.

As our annotation shows, the graph in Figure 6 sketches a phylogenetic network of SARS-CoV-2 clades that essentially reproduces the NextStrain phylogeny shown in Figure 7. The former is a complete graph in which all edges are present, and their length is related to our similarity measure (the closer, the more similar); the latter is a typical unrooted phylogenetic tree structure in which clades similarities group subpopulations into subtrees (the closer the common ancestor is, the more similar).

- **2019 clades.** The two 2019 clades 19A and 19B, are very close in Figure 6 (circled by a red cloud shape), and belong to the same subtree in Figure 7 of the NextStrain phylogeny.
- **Delta and Kappa variant clades.** The three clades 21A, 21I and 21J, which all belong to the *Delta variant* are clustered together and highlighted by a (blue) cloud shape at the top of Figure 6 as they turn out to have a higher similarity compared to each other. The grouping of these clades reproduces that of the NextStrain phylogeny shown in Figure 7, where the three Delta clades are the cluster of blue branches at the bottom right. Moreover, in both figures clade 21B of the *Kappa variant* is very close to the Delta variants.
- **20F, Gamma and Lambda variant clades.** In the graph in Figure 6, the 20F clade and the *Lambda* and *Gamma variants* seem to be outliers, as they stand slightly away from everything else. Indeed, by looking at the data and maps in NextStrain[6], it turns out that variant gamma is, in fact, peculiar, as it has lasted over a year with a quite regular but limited incidence, and checking its location on the world map of NextStrain[6], we can actually see that it was diffused almost exclusively in South America, thus explaining its isolated position in our graph. Similarly, the *Lambda variant* has only been observed in western South America. Finally, clade 20F turns out to have been observed basically only in center Australia. These type of isolated clades are also highlighted as independent of each other in Figure 7, where subtrees that include their sample are all rooted in the main thick branch of the phylogeny (as it is better visible in NextStrain[6] than in Figure 7).

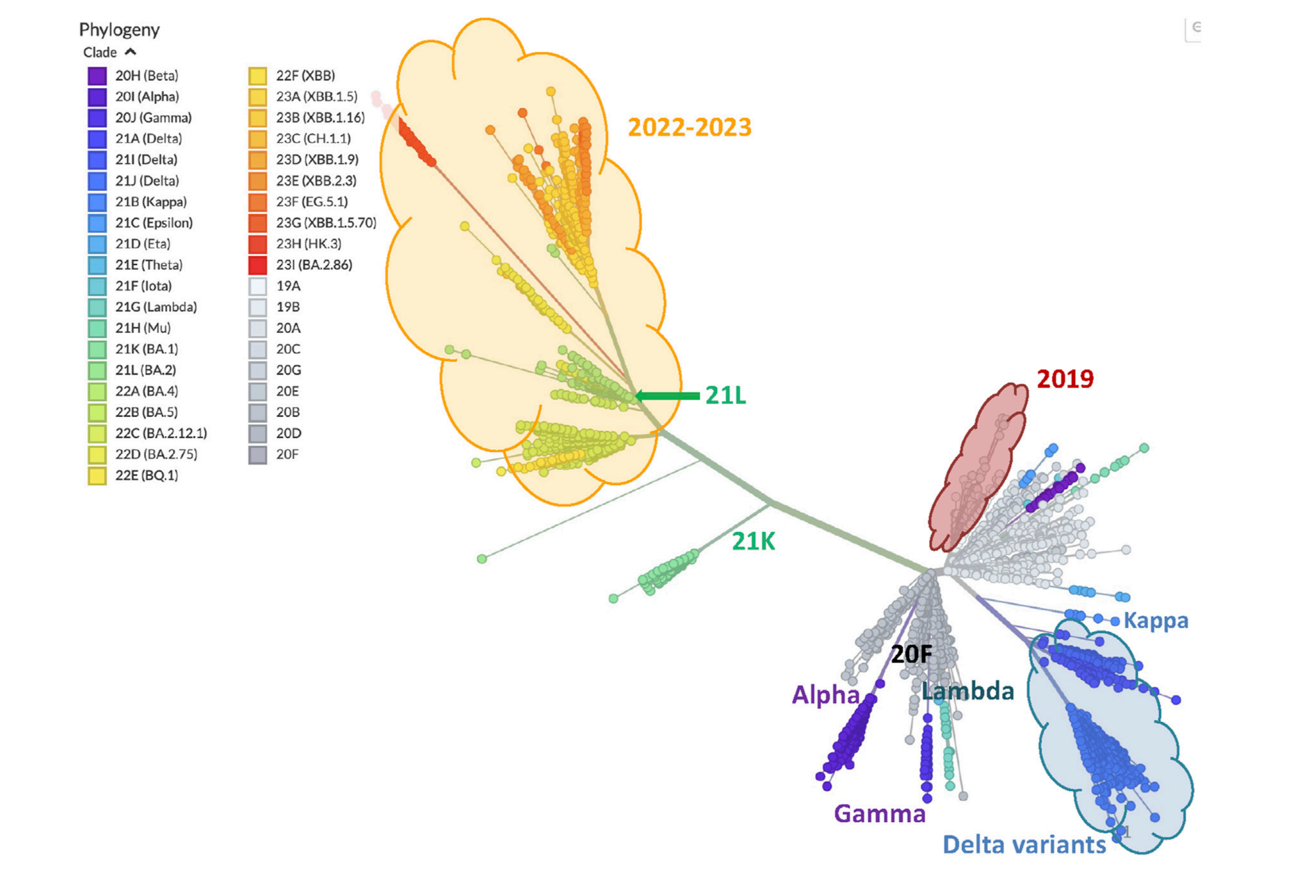

FIGURE 7
Phylogeny of 3357 SARS-CoV-2 genomes samples. The figure is generated and downloaded from Nextstrain https://nextstrain.org/ncov/open/global/all-time (2024), and some annotation is added here to highlight similarities with the graph of Figure 6.

- **Alpha variant clade**. The *alpha variant* of SARS-CoV-2 has spread worldwide, with a high incidence for over 2 years. In Figure 7 its many samples all lay in a subtree rooted (like those of the *Gamma* and *Lambda variants*) directly at the main branch of the phylogeny; accordingly, in the graph of Figure 6, the node corresponding to the *Alpha variant* pangenome is not specifically close to any other clade.
- **2020 clades**. Apart from the aforementioned variants, not surprisingly, all the other 2020 clades appear in Figure 6 and are all clustered in the center of the graph.
- **2022 and 2023 clades**. Finally, the late variants of 2023 and their 2022 ancestors are all clustered at the top right of our graph, again highlighted by a cloud (yellow) shape. This is again in agreement with the NextStrain phylogeny, as we observe these clades at the top of Figure 7. In both figures, we also observe that two 2021 clades appear as outliers inside (21L) or close to (21K) the areas of these late variants (they are pointed by green arrows in both figures).

Thus, we can conclude that our $\mathcal{BMS}$ similarity measure allowed us to reproduce the clade clustering of an established phylogeny for SARS-CoV-2 data. For the computing performance, for each pair of clades, we recorded the CPU time required to build the two ED strings out of the MSAs, compute their intersection graph, and compute the BMS array and its average. The 465 pairwise similarity computations required 17 h of CPU time in total and 2 min on average (30 min for the slowest pair); we remark that these computations can be performed in parallel. Memory usage ranged from 22MB to 361 MB. These values show the moderate resource requirements of our methods.[9] These experiments were run on a DELL PowerEdge R750 machine, used in non-exclusive mode, with 2 Intel(R) Xeon(R) Gold 5318Y CPUs, each running at 2.10 GHz and using 24 physical cores (and 48 hyperthreading cores). The main memory is shared and is of size 992 GB. The operating system used is Ubuntu 22.04.2 LTS.

4 Future work

We plan to investigate methods for local comparison of ED strings, that is, devising efficient methods to find fragments that are common to two or more ED strings (like the fragments we detect with Matching Statistics) but that are not necessarily identical in all of their occurrences (unlike those we detect with Matching Statistics). This could be achieved by means of a preliminary

9 Precise measurements are listed here: https://github.com/urbanslug/junctions/blob/master/test_data/covid.org.

preprocessing filtering step such as those of Peterlongo et al. (2009) for edit distance and Peterlongo et al. (2005), (2008) for Hamming distance. This filtering step could possibly be paired with suitable notions of maximality in frequency (Federico and Pisanti, 2009) or in conservation (Grossi et al., 2009; 2011) for the common fragments in order to avoid redundancy in the output results.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

EG: Writing–original draft, Writing–review and editing. MM: Writing–original draft, Writing–review and editing. NP: Writing–original draft, Writing–review and editing. SP: Writing–original draft, Writing–review and editing. JR: Writing–original draft, Writing–review and editing. MS: Writing–original draft, Writing–review and editing. WZ: Writing–original draft, Writing–review and editing.

Funding

The author(s) declare that financial support was received for the research, authorship, and/or publication of this article. This work was partially supported by the PANGAIA, ALPACA and NETWORKS projects that have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No. 872539, 956229 and 101034253, respectively. Nadia Pisanti was partially supported by MUR PRIN 2022 YRB97K PINC and by NextGeneration EU programme PNRR ECS00000017 Tuscany Health Ecosystem. Jakub Radoszewski was supported by the Polish National Science Center, grant no. 2022/46/E/ST6/00463.

Acknowledgments

We are also very grateful to NextStrain for maintaining and allowing access to their platform and the reports therein.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Alzamel, M., Ayad, L. A. K., Bernardini, G., Grossi, R., Iliopoulos, C. S., Pisanti, N., et al. (2018). "Degenerate string comparison and applications,". *18th international workshop on algorithms in bioinformatics, WABI 2018, August 20-22, 2018, Helsinki, Finland*. Editors L. Parida and E. Ukkonen (Schloss Dagstuhl: LIPIcs), 21, 1–21:14. 113 of *LIPIcs*. doi:10.4230/LIPIcs.WABI.2018.21

Alzamel, M., Ayad, L. A. K., Bernardini, G., Grossi, R., Iliopoulos, C. S., Pisanti, N., et al. (2020). Comparing degenerate strings. *Fundam. Inf.* 175, 41–58. doi:10.3233/FI-2020-1947

Aoyama, K., Nakashima, Y., I, T., Inenaga, S., Bannai, H., and Takeda, M. (2018). "Faster online elastic degenerate string matching,". *Annual symposium on combinatorial pattern matching, CPM 2018, july 2-4, 2018 - qingdao, China*. Editors G. Navarro, D. Sankoff, and B. Zhu (Schloss Dagstuhl: LIPIcs), 9, 1–9:10. doi:10.4230/LIPIcs.CPM.2018.9

Apostolico, A., Guerra, C., Landau, G. M., and Pizzi, C. (2016). Sequence similarity measures based on bounded hamming distance. *Theor. Comput. Sci.* 638, 76–90. doi:10.1016/J.TCS.2016.01.023

Apostolico, A., Guerra, C., and Pizzi, C. (2014). "Alignment free sequence similarity with bounded hamming distance," in *Data compression conference, DCC 2014, snowbird, UT, USA, 26-28 march, 2014*. Editors A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer (IEEE), 183–192. doi:10.1109/DCC.2014.57

Baaijens, J. A., Bonizzoni, P., Boucher, C., Vedova, G. D., Pirola, Y., Rizzi, R., et al. (2022). Computational graph pangenomics: a tutorial on data structures and their applications. *Nat. Comput.* 21, 81–108. doi:10.1007/s11047-022-09882-6

Baudet, C., Lemaitre, C., Dias, Z., Gautier, C., Tannier, E., and Sagot, M. (2010). Cassis: detection of genomic rearrangement breakpoints. *Bioinform* 26, 1897–1898. doi:10.1093/bioinformatics/btq301

Bernardini, G., Gawrychowski, P., Pisanti, N., Pissis, S. P., and Rosone, G. (2019). "Even faster elastic-degenerate string matching via fast matrix multiplication,". *46th international colloquium on automata, languages, and programming, ICALP 2019, july 9-12, 2019, patras, Greece*. Editors C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi (Schloss Dagstuhl: LIPIcs), 21, 1–21:15. doi:10.4230/LIPIcs.ICALP.2019.21

Bernardini, G., Gawrychowski, P., Pisanti, N., Pissis, S. P., and Rosone, G. (2022). Elastic-degenerate string matching via fast matrix multiplication. *SIAM J. Comput.* 51, 549–576. doi:10.1137/20M1368033

Bernardini, G., Pisanti, N., Pissis, S., and Rosone, G. (2017). "Pattern matching on elastic-degenerate text with errors," in *24th international symposium on string processing and information retrieval (SPIRE)*, 74–90. doi:10.1007/978-3-319-67428-5_7

Bernardini, G., Pisanti, N., Pissis, S. P., and Rosone, G. (2020). Approximate pattern matching on elastic-degenerate text. *Theor. Comput. Sci.* 812, 109–122. doi:10.1016/j.tcs.2019.08.012

Bonizzoni, P., Dondi, R., Klau, G. W., Pirola, Y., Pisanti, N., and Zaccaria, S. (2016). On the minimum error correction problem for haplotype assembly in diploid and polyploid genomes. *J. Comput. Biol.* 23, 718–736. doi:10.1089/cmb.2015.0220

Büchler, T., Olbrich, J., and Ohlebusch, E. (2023). Efficient short read mapping to a pangenome that is represented by a graph of ED strings. *Bioinformatics* 39, btad320. doi:10.1093/bioinformatics/btad320

Carletti, V., Foggia, P., Garrison, E., Greco, L., Ritrovato, P., and Vento, M. (2019). "Graph-based representations for supporting genome data analysis and visualization: opportunities and challenges," in *Graph-based representations in pattern recognition - 12th IAPR-TC-15 international workshop, GbRPR 2019, tours, France, june 19-21, 2019, proceedings*. Editors D. Conte, J. Ramel, and P. Foggia (Springer), 237–246. 11510 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-030-20081-7_23

Cisłak, A., Grabowski, S., and Holub, J. (2018). SOPanG: online text searching over a pan-genome. *Bioinformatics* 34, 4290–4292. doi:10.1093/bioinformatics/bty506

Crux, N. B., and Elahi, S. (2017). Human leukocyte antigen (HLA) and immune regulation: how do classical and non-classical HLA alleles modulate immune response to human immunodeficiency virus and hepatitis C virus infections? *Front. Immunol.* 8, 832. doi:10.3389/fimmu.2017.00832

Eizenga, J. M., Novak, A. M., Kobayashi, E., Villani, F., Cisar, C., Heumos, S., et al. (2021). Efficient dynamic variation graphs. *Bioinform* 36, 5139–5144. doi:10.1093/bioinformatics/btaa640

Equi, M., Mäkinen, V., Tomescu, A. I., and Grossi, R. (2023). On the complexity of string matching for graphs. *ACM Trans. Algorithms* 19 (21), 1–25. doi:10.1145/3588334

Federico, M., and Pisanti, N. (2009). Suffix tree characterization of maximal motifs in biological sequences. *Theor. Comput. Sci.* 410, 4391–4401. doi:10.1016/J.TCS.2009.07.020

Gabory, E., Mwaniki, N. M., Pisanti, N., Pissis, S. P., Radoszewski, J., Sweering, M., et al. (2023). "Comparing elastic-degenerate strings: algorithms, lower bounds, and applications,". *34th annual symposium on combinatorial pattern matching, CPM 2023, june 26-28, 2023, marne-la-vallée, France*. Editors L. Bulteau and Z. Lipták (Schloss Dagstuhl: LIPIcs), 11, 1–1120. doi:10.4230/LIPIcs.CPM.2023.11

Gao, Y., Liu, Y., Ma, Y., Liu, B., Wang, Y., and Xing, Y. (2020). abPOA: an SIMD-based C library for fast partial order alignment using adaptive band. *Bioinformatics* 37, 2209–2211. doi:10.1093/bioinformatics/btaa963

Garrison, E., Sirén, J., Novak, A. M., Hickey, G., Eizenga, J. M., Dawson, E. T., et al. (2018a). Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nat. Biotechnol.* 36, 875–879. doi:10.1038/nbt.4227

Garrison, E., Sirén, J., Novak, A. M., Hickey, G., Eizenga, J. M., Dawson, E. T., et al. (2018b). Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nat. Biotechnol.* 36, 875–879. doi:10.1038/nbt.4227

Gibney, D., Thankachan, S. V., and Aluru, S. (2022). On the hardness of sequence alignment on de bruijn graphs. *J. Comput. Biol.* 29, 1377–1396. doi:10.1089/cmb.2022.0411

Grossi, R., Iliopoulos, C. S., Liu, C., Pisanti, N., Pissis, S. P., Retha, A., et al. (2017). "On-line pattern matching on similar texts,". *28th annual symposium on combinatorial pattern matching, CPM 2017, july 4-6, 2017, Warsaw, Poland*. Editors J. Kärkkäinen, J. Radoszewski, and W. Rytter (Schloss Dagstuhl: LIPIcs), 9, 1–9:14. doi:10.4230/LIPIcs.CPM.2017.9

Grossi, R., Pietracaprina, A., Pisanti, N., Pucci, G., Upfal, E., and Vandin, F. (2009). "MADMX: a novel strategy for maximal dense motif extraction," in *Algorithms in bioinformatics, 9th international workshop, WABI 2009, Philadelphia, PA, USA, september 12-13, 2009. Proceedings*. Editors S. Salzberg and T. J. Warnow (Springer), 362–374. 5724 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-642-04241-6_30

Grossi, R., Pietracaprina, A., Pisanti, N., Pucci, G., Upfal, E., and Vandin, F. (2011). MADMX: a strategy for maximal dense motif extraction. *J. Comput. Biol.* 18, 535–545. doi:10.1089/CMB.2010.0177

Gusfield, D. (1997). *Algorithms on strings, trees, and sequences - computer science and computational biology*. Cambridge University Press. doi:10.1017/cbo9780511574931

Hadfield, J., Megill, C., Bell, S. M., Huddleston, J., Potter, B., Callender, C., et al. (2018). Nextstrain: real-time tracking of pathogen evolution. *Bioinform* 34, 4121–4123. doi:10.1093/bioinformatics/bty407

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in science conference*. Editors G. Varoquaux, T. Vaught, and J. Millman (Pasadena, CA USA), 11–15.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2003). *Introduction to automata theory, languages, and computation - international edition*. 2nd Edition. Addison-Wesley.

Iliopoulos, C. S., Kundu, R., and Pissis, S. P. (2021). Efficient pattern matching in elastic-degenerate strings. *Inf. Comput.* 279, 104616. doi:10.1016/j.ic.2020.104616

Jain, C., Zhang, H., Gao, Y., and Aluru, S. (2020). On the complexity of sequence-to-graph alignment. *J. Comput. Biol.* 27, 640–654. doi:10.1089/cmb.2019.0066

Lawson, M. V. (2004). *Finite automata*. Chapman and Hall/CRC.

Leimeister, C., and Morgenstern, B. (2014). kmacs: the *k*-mismatch average common substring approach to alignment-free sequence comparison. *Bioinform* 30, 2000–2008. doi:10.1093/bioinformatics/btu331

Li, H., Feng, X., and Chu, C. (2020). The design and construction of reference pangenome graphs with minigraph. *Genome Biol.* 21, 265. doi:10.1186/s13059-020-02168-z

Liao, W.-W., Asri, M., Ebler, J., Doerr, D., Haukness, M., Hickey, G., et al. (2023). A draft human pangenome reference. *Nature* 617, 312–324. doi:10.1038/s41586-023-05896-x

Mwaniki, N. M., Garrison, E., and Pisanti, N. (2023). "Fast exact string to D-texts alignments," in *Proceedings of the 16th international joint conference on biomedical engineering systems and Technologies, BIOSTEC 2023, volume 3: BIOINFORMATICS, Lisbon, Portugal, february 16-18, 2023*. Editors H. Ali, N. Deng, A. L. N. Fred, and H. Gamboa, 70–79. doi:10.5220/0011666900003414

Mwaniki, N. M., and Pisanti, N. (2022). "Optimal sequence alignment to ED-strings," in *Bioinformatics research and applications - 18th international symposium, ISBRA 2022, haifa, Israel, november 14-17, 2022, proceedings*. Editors M. S. Bansal, Z. Cai, and S. Mangul (Springer), 204–216. 13760 of *Lecture Notes in Computer Science*. doi:10.1007/978-3-031-23198-8_19

Paten, B., Novak, A. M., Eizenga, J. M., and Garrison, E. (2017). Genome graphs and the evolution of genome inference. *Genome Res.* 27, 665–676. doi:10.1101/gr.214155.116

Peterlongo, P., Pisanti, N., Boyer, F., do Lago, A. P., and Sagot, M. (2008). Lossless filter for multiple repetitions with hamming distance. *J. Discrete Algorithms* 6, 497–509. doi:10.1016/J.JDA.2007.03.003

Peterlongo, P., Pisanti, N., Boyer, F., and Sagot, M. (2005). "Lossless filter for finding long multiple approximate repetitions using a new data structure, the bi-factor array," in *String processing and information retrieval, 12th international conference, SPIRE 2005, Buenos Aires, Argentina, november 2-4, 2005, proceedings*. Editors M. P. Consens and G. Navarro (Springer), 179–190. 3772 of *Lecture Notes in Computer Science*. doi:10.1007/11575832_20

Peterlongo, P., Sacomoto, G. A. T., do Lago, A. P., Pisanti, N., and Sagot, M. (2009). Lossless filter for multiple repeats with bounded edit distance. *Algorithms Mol. Biol.* 4, 3. doi:10.1186/1748-7188-4-3

Pissis, S. P., and Retha, A. (2018). "Dictionary matching in elastic-degenerate texts with applications in searching VCF files on-line,". *17th international symposium on experimental algorithms, SEA 2018, june 27-29, 2018, L'aquila, Italy*. Editor G. D'Angelo (Schloss Dagstuhl: LIPIcs), 16, 1–16:14. doi:10.4230/LIPIcs.SEA.2018.16

Pizzi, C. (2016). Missmax: alignment-free sequence comparison with mismatches through filtering and heuristics. *Algorithms Mol. Biol.* 11, 6. doi:10.1186/S13015-016-0072-X

Rakocevic, G., Semenyuk, V., Lee, W.-P., Spencer, J., Browning, J., Johnson, I. J., et al. (2019). Fast and accurate genomic analyses using genome graphs. *Nat. Genet.* 51, 354–362. doi:10.1038/s41588-018-0316-4

Rautiainen, M., Mäkinen, V., and Marschall, T. (2019). Bit-parallel sequence-to-graph alignment. *Bioinform* 35, 3599–3607. doi:10.1093/bioinformatics/btz162

Rautiainen, M., and Marschal, T. (2020). GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome Biol.* 21, 253. doi:10.1186/s13059-020-02157-2

Romero-Sánchez, C., Hernández, N., Chila-Moreno, L., Jiménez, K., Padilla, D., Bello-Gualtero, J. M., et al. (2021). HLA-B allele, genotype, and haplotype frequencies in a group of healthy individuals in Colombia. *J. Clin. Rheumatol.* 27, S148–S152. doi:10.1097/rhu.0000000000001671

Sankoff, D., and Blanchette, M. (1998). Multiple genome rearrangement and breakpoint phylogeny. *J. Comput. Biol.* 5, 555–570. doi:10.1089/cmb.1998.5.555

Thankachan, S. V., Chockalingam, S. P., Liu, Y., Krishnan, A., and Aluru, S. (2017). A greedy alignment-free distance estimator for phylogenetic inference. *BMC Bioinform* 18 (238), 238–8. doi:10.1186/s12859-017-1658-0

The Computational Pan-Genomics Consortium (2018). Computational pan-genomics: status, promises and challenges. *Briefings Bioinforma.* 19, 118–135. doi:10.1093/bib/bbw089

Ulitsky, I., Burstein, D., Tuller, T., and Chor, B. (2006). The average common substring approach to phylogenomic reconstruction. *J. Comput. Biol.* 13, 336–350. doi:10.1089/cmb.2006.13.336