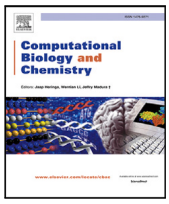
Computational Biology and Chemistry 122 (2026) 108870

Contents lists available at ScienceDirect

Computational Biology and Chemistry

journal homepage: www.elsevier.com/locate/cbac



Research Article

Measuring genomic data with prefix-free parsing

Simone Lucà [a],*, Francesco Masillo [b], Zsuzsanna Lipták [a]

[a] University of Verona, Department of Computer Science, Strada le Grazie, 15, Verona, 37134, Italy
[b] Technische Universität Dortmund, Faculty of Computer Science, Otto-Hahn-Straße, 14, Dortmund, 44227, Germany

ARTICLE INFO

Keywords:
Compression
Repetitiveness measures
Prefix-free parsing
Pangenome openness
Text indexing
Massive data

ABSTRACT

Summary: Prefix-free parsing (Boucher et al., 2019) is a highly effective heuristic for computing text indexes for very large amounts of biological data. The algorithm constructs a data structure, the prefix-free parse (PFP) of the input, consisting of a dictionary and a parse, which is then used to speed up computation of the final index. In this paper, we study the *size* of the PFP, which we refer to as π, and show that it is a powerful tool in its own right. To show this, we present two use cases. We first study the application of π as a *repetitiveness measure* of the input text, and compare it to other currently used repetitiveness measures, including z (the number of Lempel–Ziv phrases), r (the number of runs of the Burrows–Wheeler Transform), and δ (the text's substring complexity). We then turn to the use of π as a measure for *pangenome openness*. In both applications, our results are similar to existing measures, but our tool, in almost all cases, is more efficient than those computing the other measures, both in terms of time and space, sometimes by orders of magnitude. We close the paper with a detailed systematic study of the parameter choice for PFP (window size w and modulus p). This gives rise to interesting open questions.
Availability and implementation: The source code is available at https://github.com/simolucaa/piPFP. The accession codes for all the datasets used and the raw results are available at https://github.com/simolucaa/piPFP_experiments.

1. Introduction

With the availability of unprecedented amounts of genomic data today, it is becoming increasingly important to be able to store, handle, and process this data efficiently (Pan-Genomics Consortium, 2018). One central issue is that datasets, in addition to their size, can have different properties, such as number of sequences, length of individual sequences, or level of repetitiveness. Current software is frequently tailor-made for specific types of data, taking advantage of certain properties of the input. Therefore, it is paramount that users be able to test their data's properties.

One such central property is *repetitiveness*. Intuitively, this is a measure of how much of the sequence data occurs more than once in the input. This concept is intimately related to our ability to store the data using compressed text indexes, which require little storage space while at the same time enabling efficient string processing tasks such as pattern matching (Navarro, 2022b). To date there is no generally accepted measure of repetitiveness (Navarro, 2022a), and different measures are being used, such as the number z of the phrases of the Lempel–Ziv 77 parse (Lempel and Ziv, 1976; Ziv and Lempel, 1977), the number r of runs of the Burrows–Wheeler Transform (BWT) (Burrows and Wheeler, 1994), or the more recent δ-measure (Raskhodnikova et al., 2013) (also referred to as substring complexity).

Another property of interest is *pangenome openness* (Tettelin et al., 2008). A pangenome is a set of genomes of individual of the same species. Pangenome openness essentially measures the amount of additional data necessary to have complete information about a species, given a dataset of individual genomes (i.e., a pangenome). Several methods for measuring pangenome openness were introduced recently (Parmigiani et al., 2024; Bonnie et al., 2024; Page et al., 2015; Jonkheer et al., 2022; Chaudhari et al., 2016).

In this paper, we introduce a new method of measuring genomic data, based on prefix-free parsing, and show that it can be successfully applied in both of these areas: as a repetitiveness measure and for measuring pangenome openness.

Prefix-free parsing (Boucher et al., 2019) (PFP) is a highly effective heuristic for computing text indexes for very large amounts of biological data. The algorithm computes a data structure, the prefix-free parse (PFP) of the input, consisting of a dictionary and a parse, which is then used to speed up computation of the final index. Originally introduced as a preprocessing step to speed up the computation of the

* Corresponding author.
E-mail addresses: simone.luca@univr.it (S. Lucà), francesco.masillo@tu-dortmund.de (F. Masillo), zsuzsanna.liptak@univr.it (Z. Lipták).

https://doi.org/10.1016/j.compbiolchem.2025.108870
Received 6 November 2025; Received in revised form 23 December 2025; Accepted 26 December 2025
Available online 30 December 2025

Burrows–Wheeler Transform (BWT) of very large genomic sequence data, it has since been extended to computing other data structures, as well. These include the suffix array (Kuhnle et al., 2020); the extended BWT (Boucher et al., 2021a, 2024) of Mantaci et al. (2007); and the XBWT (Gagie et al., 2021), an extension of the BWT to labeled trees. PFP has also been introduced as a text index in its own right (Boucher et al., 2021b; Oliva et al., 2022), or as a way to enhance the FM-index to boost query performance (Hong et al., 2024); and a recursive variant for BWT construction was proposed (Oliva et al., 2023) in order to further scale the computation with huge amounts of data. PFP has also shown competitive results for building grammars (Gagie et al., 2019; Kim et al., 2024), computing the Lempel–Ziv factorization (Hong et al., 2023), and computing the smallest suffixient set (Cenzato et al., 2025). Other applications of PFP for constructing pangenome-sized data structures include building the r-index of Gagie et al. (2020) augmented with thresholds (Bannai et al., 2020). This combination of data structures enable the computation of matching statistics, which in turn allows finding Maximal Exact Matches (MEMs). MEMs are a crucial ingredient for speeding up the alignment phase of sets of reads to a reference genome. Recent tools, such as `MONI` (Rossi et al., 2022), `PHONI` (Boucher et al., 2021c), `AUG-PHONI` (Martínez-Guardiola et al., 2023), `LAZY-PHONI` (Goga et al., 2024), `SPUMONI` (Ahmed et al., 2021), `SPUMONI2` (Ahmed et al., 2023), `SIGMONI` (Shivakumar et al., 2024), `MOVI` (Zakeri et al., 2024), all use PFP as preprocessing step for building the underlying r-index. Latest work includes the use of PFP for computing multiple sequence alignments (Olbrich et al., 2025) and for merging very large BWTs (Díaz-Domínguez et al., 2025).

Here, we explore the power of PFP for measuring certain properties of the input data. We introduce the measure π, the *size* of the PFP data structure, and show that it is a powerful tool in its own right. Intuitively, the more repetitive the input, the fewer unique phrases there are, and therefore, the smaller the data structure. Our experiments in fact indicate that π might be a good candidate for a new measure both as a repetitiveness measure, and for measuring pangenome openness. It gives similar results regarding repetitiveness as the established repetitiveness measures z (the number of Lempel–Ziv phrases), r (the number of runs of the Burrows–Wheeler Transform), and δ (the text's substring complexity). When used to decide pangenome openness, on almost all of the benchmark datasets, classification using π gives the same result as recent competitor measures.

We also present an open-source tool, `piPFP`, which computes π. Our experiments show that `piPFP` is highly efficient both as regards to computation time and memory requirements, and outperforms other methods, sometimes by orders of magnitude. We believe that using π as an alternative method for measuring these properties can be beneficial to users, as it requires little computational resources and has similar accuracy as other, more complex and/or slower, methods. This is because the dictionary is able to capture the redundant information, leading to a small memory footprint, which in turns helps with caching and processing speed.

Finally, we turn our attention to parameter choice in PFP. The PFP algorithm has two user-defined parameters, the window size w and the modulus p. In the original paper (Boucher et al., 2019), a brief study of parameter choice was presented, on the basis of which the default parameters of the accompanying tool were chosen. Another parameter comparison was included in the PhD thesis (Oliva, 2023) (see Sections 3.1 and 3.3 for detailed discussions). Here, we explore these parameters in depth, concluding that for certain applications, very small window sizes may not be appropriate. Our study on the parameter choices leads to several open problems of a theoretical nature.

Summarizing, our contributions are threefold:

1. We introduce the size of the PFP for measuring certain properties of the input data.
2. We present a tool, `piPFP`, which computes this measure and show that it is highly efficient.
3. We present an in-depth study of the parameter choice for PFP.

The rest of the paper is organized as follows. In Section 2, we give the necessary technical background and present our experimental setup. Section 3 contains the experimental results: as a repetitiveness measure, for pangenome openness, and exploring parameter choice for PFP. We close with a discussion and open problems in Section 4.

2. Materials and methods

In this section, we first give the necessary background on prefix-free parsing (Section 2.1), the repetitiveness measures z, r, and δ (Section 2.2), and pangenome openness (Section 2.4). This is followed by the introduction of our PFP-based measure (Section 2.5), the presentation of the datasets used in our experiments (Section 2.6), and details of the implementation (Section 2.7).

2.1. Prefix-free parsing

Prefix-free parsing (*PFP*), introduced in Boucher et al. (2019), is a technique that processes a string T of length n using two integers, w and p, both greater than 1. The method produces a parse of T consisting of overlapping phrases, each uniquely stored in a dictionary denoted as D. The resulting parse P is represented as a meta-string, where each phrase is replaced by its rank in the lexicographically sorted dictionary. We refer to the prefix-free parse of T as $PFP(T)$, consisting of the dictionary D and the parse P. Next, we give an outline of this method.

The PFP process begins by appending w occurrences of a special symbol #, which is smaller than all characters and does not appear elsewhere in the string, to both the beginning and end of T. We refer to the resulting string as $T' = \#^w T \#^w$. Next, we define a set of *trigger strings*, E, that will guide the parsing of T, as follows: We compute the Karp–Rabin hash $H_q(t)$ for all substrings of length w in T' using a large prime q. A substring $t = T'[i..i+w-1]$ is then in E if $H_p(t) = H_q(t) \bmod p = 0$, or if $t = \#^w$. This can be computed efficiently via the well-known rolling hash algorithm of Karp and Rabin (Karp and Rabin, 1987). These trigger strings are then used to parse T', as follows.

The dictionary D for PFP consists of substrings of T' that start with a trigger string and end with the next trigger string in T'. These substrings, referred to as *phrases*, can be identified by scanning T' and adding a new phrase to D whenever the next trigger string is encountered. Once the dictionary D is built, it is sorted lexicographically. Using the sorted dictionary D and the input string T, we generate the parse P, which consists of these overlapping phrases from D. Each phrase overlaps with the next by w characters, corresponding to a trigger string. This parsing process creates an array of integers in P, where each integer is the rank of the corresponding phrase in D. With D and P, the original string T can be reconstructed.

A set of strings is called *prefix-free* if it does not contain any two strings which are one a proper prefix of the other. It can be shown that the set of the suffixes of phrases (i.e., elements of the dictionary D) are prefix-free, as formalized in the following lemma:

Lemma 1 (*Boucher et al., 2019, Lemma 1*). *For a string T and its prefix-free parse $PFP(T)$, which consists of a dictionary D and a parse P, the set S of suffixes of phrases in D of length at least w forms a prefix-free set.*

This is a crucial property of PFP, necessary for the fast building of text indexes such as the BWT. It was shown in Boucher et al. (2019) how to construct the Burrows–Wheeler Transform (BWT) (Burrows and Wheeler, 1994) using space and time proportional to the size of the dictionary D and parse P. Later, in Kuhnle et al. (2020), it was shown how to compute the suffix array samples at run-boundaries while computing the BWT to produce the r-index (Gagie et al., 2020) within the same time and space bounds.

Example 1. Let the parameters be $w = 2$ and $p = 11$. (We ignore q here, as it is a large prime, and has therefore no impact on the computation, i.e., $H_q(u) = u$ for short strings u.) Using $b = 256$ and the ASCII-256 encoding, where A = 65, C = 67, G = 71, and T = 84, the only two w-length words that map to 0 are GT and TC:

$$H_p(\texttt{GT}) = (71 \cdot 256 + 84) \bmod 11 = 0,$$
$$H_p(\texttt{TC}) = (84 \cdot 256 + 67) \bmod 11 = 0,$$

and therefore, $E = \{\#\#, \texttt{GT}, \texttt{TC}\}$.

Consider the string

$$T = \texttt{TCCGGGCTGGCATCTGCCGTAAAGGTGCGGATATCCGGGCTGGCATCA$$
$$T' = \#\#\texttt{TCCGGGCTGGCATCTGCCGTAAAGGTGCGGATATCCGGGCTGGCATCA}\#\#$$

With $E = \{\#\#, \texttt{GT}, \texttt{TC}\}$, we compute the corresponding lexicographically sorted dictionary: $D = \{[0 : \#\#\texttt{TC}], [1 : \texttt{GTAAAGGT}], [2 : \texttt{GTGCGGATATC}], [3 : \texttt{TCA}\#\#], [4 : \texttt{TCCGGGCTGGCATC}], [5 : \texttt{TCTGCCGT}]\}$. The parse that is obtained from T' and D is $P = [0, 4, 5, 1, 2, 4, 3]$.

2.2. Repetitiveness measures

The study of string repetitiveness focuses on quantifying the complexity and redundancy present in a string. The question of which measure best captures string repetitiveness has been the focus of much recent research. Among the most commonly used measures are z, the number of phrases of the Lempel–Ziv factorization, r, the number of runs of the Burrows–Wheeler Transform, and δ, the so-called substring complexity. For more details on these measures, their properties, including advantages and disadvantages, as well as the relationships among the wide spectrum of repetitiveness measures, we point the interested reader to the comprehensive survey (Navarro, 2022a).

Here, we give a short introduction to these three measures, which we will use as a baseline in our comparative experimental studies.

2.2.1. Number of phrases in the Lempel–Ziv factorization

The Lempel–Ziv compression algorithms (Lempel and Ziv, 1976; Ziv and Lempel, 1977, 1978) are fundamental to understanding repetitiveness in strings. In particular, the LZ77 compression algorithm (Lempel and Ziv, 1976; Ziv and Lempel, 1977) parses a string T into a sequence of phrases, where each phrase is either a new character or a substring of maximum length that appears also earlier in the text. The number of phrases, denoted as z, is a good measure of repetitiveness, since highly repetitive strings tend to have fewer phrases.

Lempel–Ziv parsing is significant not only as a compression technique but also as a way to define theoretical bounds for other measures of repetitiveness. It captures the idea of redundancy in a string by representing repeated patterns succinctly.

2.2.2. Number of runs of the Burrows–Wheeler Transform

The Burrows–Wheeler Transform (BWT) (Burrows and Wheeler, 1994) is another method which is widely used in string processing and compression. Given a string T, the BWT is computed by sorting lexicographically the set of suffixes of T and then concatenating the preceding character of the suffixes. A well-known property of the BWT is that repetitive strings tend to produce long runs of identical characters in the transformed string. The number of runs, denoted as r, can thus be used as a measure of repetitiveness, since smaller values of r indicate higher levels of repetitiveness in the original string.

The number of runs r is closely related to the compressibility of T and can be exploited for building text indexes, such as the FM-index (Ferragina and Manzini, 2000), or the more recent r-index (Gagie et al., 2020) or extended r-index (Boucher et al., 2024).

2.3. Substring complexity

The measure δ, known as *substring complexity* (Raskhodnikova et al., 2013), quantifies the repetitiveness of a string based on the normalized number of distinct substrings of fixed length k, maximized over all possible k. Formally, for a string T of length n, the measure δ is defined as:

$$\delta(T) = \max_{1 \le k \le n} \frac{S(k)}{k},$$

where $S(k)$ is the number of distinct k-length substrings of T.

The measure δ is known to lower-bound both z and r for every text T (Kociumaka et al., 2023).

2.4. Pangenome openness

The term pangenome openness (Tettelin et al., 2008) refers to a method to estimate a pangenome's total size, given a set of genomes of individuals from the same species. This estimation requires calculating the pangenome growth, which monitors the number of distinct attributes observed as a function of the number of entities considered. Heap's Law (Heaps, 1978), known in the information retrieval field, can be used to describe this phenomenon as follows: the rate at which new attributes are found decreases as one considers more and more entities, as this rate is proportional to $kN^{\gamma-1} = kN^{-\alpha}$, where $\alpha = 1-\gamma$ and k are constants that depend on the particular text, and N is the total number of attributes. In other words, as sampling proceeds, discovering a new attribute becomes increasingly harder (Tettelin et al., 2008; Heaps, 1978; Parmigiani et al., 2024). In pangenomics, the entities are the genomes, and the attributes can be genes, open reading frames (ORFs), or genome intervals of fixed size (e.g., k-mers Parmigiani et al., 2024). The parameter α is used to classify a genome as either *closed* or *open*:

- If $\alpha < 1$, the pangenome is classified as *open*. This means that the number of attributes discovered increases significantly with the addition of new genomes. This indicates a high level of genomic diversity, suggesting that each newly sequenced genome contributes unique genes.
- If $\alpha > 1$, the pangenome is classified as *closed*. This indicates that the number of new attributes reaches a plateau as more genomes are added to the set.

This definition of pangenome growth, however, makes α dependent on the order in which the genomes are added. To address this issue, one can average over all possible permutations; however, this is prohibitive, even for a small number of genomes. One solution is to estimate the growth by selecting a subset of all possible permutations (Tettelin et al., 2008; Bonnie et al., 2024). We will instead employ a solution presented by Parmigiani et al. (2024), which is based on a commonly used method in ecology (Heck et al., 1975). In the following, we follow the definitions given in Parmigiani et al. (2024). (Note that their definition of openness differs slightly from the original one given in Tettelin et al. (2008), as explained in Parmigiani et al. (2024, p. 5).)

Given n entities $G_1, \ldots, G_n$, we define the average total cardinality $f_{tot}(m)$, for $1 \le m \le n$, of the union of m of these entities, and the average number $f_{new}(m)$ of new attributes that are added when adding an mth entity to $m-1$ entities, as follows:

Definition 1. Let $\mathcal{U}$ be a universe (of *attributes*) and $\mathcal{G} = \{G_1, \ldots, G_n\}$ a set of *entities*, i.e. $G_i \subseteq \mathcal{U}$ for $1 \le i \le m$. In addition, let $h(i)$ be the number of items occurring in exactly i input genomes, with $1 \le i \le n$, and $n^{\underline{m}} = n(n-1)(n-2)\cdots(n-m+1)$ is the falling factorial. Then

$$f_{tot}(m) = \frac{1}{\binom{n}{m}} \sum_{|\mathcal{G}'|=m} \Big| \bigcup_{G \in \mathcal{G}'} G \Big| = \sum_{i=1}^{n} h(i)\Big(1 - \frac{(n-i)^{\underline{m}}}{n^{\underline{m}}}\Big),$$

$$f_{new}(m) = \begin{cases} 0 & \text{if } m = 0, \\ f_{tot}(m) - f_{tot}(m-1) & \text{otherwise.} \end{cases}$$

We also write f_{tot} for $f_{tot}(n)$ and f_{new} for $f_{new}(n)$.

It is known Parmigiani et al. (2024) that f_{new} follows Heap's Law, therefore it is possible to fit $kN^{-\alpha}$ on f_{new} and derive the parameter α.

2.5. How we use PFP for measuring genomic data

To evaluate the repetitiveness of a sequence, we rely on the fact that PFP phrases in the dictionary are unique. Ideally, a more repetitive sequence will result in fewer phrases in the dictionary, with the parameters w and p influencing the number of phrases. Intuitively, the more repetitive the sequence, the fewer unique phrases there will be, resulting in a smaller dictionary and overall less storage space required by the PFP data structure. In order to better study this question, we introduce a measure of the size of the PFP which encompasses the two parameters w and p, as follows:

Definition 2. Let w, p be two integers, where $w \geq 4$ and $p \geq 10$. Given a text T, let $D_{w,p}$ and $P_{w,p}$ be the dictionary and parse, respectively, of the PFP using window size w and modulus p. We define

$$\pi_{w,p}(T) = \|D_{w,p}\| + |P_{w,p}|,$$

where $\|D_{w,p}\|$ denotes the total length of the phrases in $D_{w,p}$ and $|P_{w,p}|$ the length of the parse. When T is clear from the context, we also write $\pi_{w,p}$ for $\pi_{w,p}(T)$.

Example 2. Consider the string

$$T = \texttt{TCCGGGCTGGCATCTGCCGTAAAGGTGCGGATATCCGGGCTGGCATCA}$$

and the parameters $w = 2$ and $p = 11$.

As shown in Example 1, the dictionary and parse of T with these parameters are as follows: $D_{2,11}$ = {[0 : ##TC], [1 : GTAAAGGT], [2 : GTGCGGATATC], [3 : TCA##], [4 : TCCGGGCTGGCATC], [5 : TCTGCCGT]} and $P_{2,11} = [0, 4, 5, 1, 2, 4, 3]$.

The total length of the phrases in $D_{2,11}$ is $\|D_{2,11}\| = 4+8+11+5+14+8 = 50$, and the length of the parse is $|P_{2,11}| = 7$. Therefore, we have $\pi_{2,11}(T) = \|D_{2,11}\| + |P_{2,11}| = 50 + 7 = 57$.

To assess pangenome openness, we applied the approach of Parmigiani et al. (2024) described in Section 2.4, using PFP phrases as attributes rather than k-mers. Intuitively, the number of unique phrases, i.e., the cardinality of the dictionary, should increase less and less as new genomes are added, capturing the transition from open to closed pangenome in a similar way as do k-mers. On the technical level, we compute a histogram $h(i)$, which, for every i, counts how many PFP phrases are present in exactly i genomes. This histogram is then used to compute f_{tot}. Finally, we compute f_{new} from f_{tot} and fit the function $kN^{-\alpha}$ to f_{new} to estimate α.

2.6. Datasets used

In this section, we present an overview of the datasets used in our experiments. We conducted three main sets of experiments, each using different datasets. These datasets are accessible through NCBI using the provided accession codes: https://github.com/simolucaa/piPFP_experiments. A summary of the datasets is presented in Table 1.

We downloaded the FASTA files through NCBI with the following filters: "Assembly level: complete", "Exclude: atypical genomes, metagenome-assembled genomes (MAGs), genomes from large multi-isolate projects". No preprocessing was done.

Table 1
Table summarizing the datasets used in the three main sets of experiments. Genome size is given in mega-bases (Mb), and σ is the size of the alphabet. The datasets have different alphabetsize σ due to the different metacharacters used.

	Species	Number of genomes	Average genome size (Mb)	Total size (Mb)	σ
1	*A. thaliana*	21	136.64	2 869.43	16
2	*E. coli*	120	5.74	665.63	8
3	SARS-CoV-2	500	0.03	14.90	11
4	*S. cerevisiae*	28	12.04	337.23	9
5	*S. enterica*	60	4.80	288.25	14
6	*H. sapiens*	9	3081.83	26 856.70	16
7	*E. coli*	7725	5.15	38 414.10	15
8	*B. aphidicola*	35	0.61	20.45	5
9	*B. cereus*	83	5.65	452.63	13
10	*C. botulinum*	50	3.95	190.76	5
11	*C. burnetii*	13	2.06	25.86	4
12	*C. jejuni*	234	1.68	378.74	15
13	*F. tularensis*	57	1.90	104.65	5
14	*H. pylori*	234	1.63	368.63	15
15	*P. marinus*	9	1.73	15.04	4
16	*R. palustris*	8	5.38	41.53	4
17	*S. pneumoniae*	88	2.11	179.43	8
18	*S. pyogenes*	247	1.85	440.21	11
19	*Y. pestis*	56	4.71	254.80	6

Datasets used for π as a repetitiveness measure

To evaluate the effectiveness of π as a repetitiveness measure, we performed three experiments:

1. Comparison of π to other repetitiveness measures. For this comparison, we used the same 60 *Salmonella enterica* genomes as Bonnie et al. (2024) (entry 5 in Table 1).
2. Comparison of π of real and random sequences. We generated random sequences (detail in Section 3.1) and compared their π values to those of real sequences from three datasets: *Escherichia coli*, *Saccharomyces cerevisiae*, and *S. enterica* (see entries 2, 4, and 5 in Table 1).
3. Performance assessment. For this experiment, we tested the tools on the same datasets as (2) and on larger datasets, namely 9 *Homo sapiens* genomes and 7725 *E. coli* genomes (entries 6 and 7 of Table 1).

Datasets used for pangenome openness

To evaluate our tool, we compared our results to those of other tools in two ways:

1. Prediction accuracy. We compared our predictions to those of other tools. For these experiments, we used the same dataset as Parmigiani et al. (2024), which comprises 12 prokaryotic genomes (see entries 8–19 of Table 1).
2. Performance assessment. For this experiment, we compared our tool to others. We used the same datasets as (1) and larger datasets, namely 9 *H. sapiens* genomes and 7725 *E. coli* (entries 6 and 7 of Table 1).

Datasets used for parameter choice for π

For this experiment, we selected five datasets to represent five different biological categories: a complex animal (*H. sapiens*), a plant (*Arabidopsis thaliana*), a simple eukaryote (*S. cerevisiae*), two prokaryotic organisms (*S. enterica* and *E. coli*), and a virus (SARS-CoV-2). More information about these datasets can be found in Table 1 (entries 1–6).

2.7. Implementation details

Our tool `piPFP` is available at https://github.com/simolucaa/piPFP. We implemented both the computation of the repetitiveness measure

$\pi_{w,p}$ and the pangenome openness measure, building on the original implementation of PFP.[1]

We modified the original implementation in order to speed up the computation and to save working space. In the dictionary, we only store the hash of the phrases rather than the full explicit phrases. This is because we only need the number of distinct phrases and their length, not the actual phrases themselves. In practice, we maintain a hashmap that stores the hashes of the phrases as keys and their lengths as items. We also avoid storing the parse explicitly on disk by counting the total number of phrases found during computation, reducing the parse to a single integer counter. This is done in order to avoid costly I/O operations.

When using multiple threads, we also slightly deviate from the original implementation by using a concurrent hashmap, `growt`, from Maier et al. (2019) instead of the standard implementation found in the C++ library. This allows faster parallel insertions of new entries in the hashmap without an explicit lock mechanism.

For the computation of pangenome openness (see Sections 2.4 and 2.5), we based our implementation on Parmigiani et al. (2024), replacing the k-mers with PFP phrases. To enable fast computation across different threads, we assign to each thread a range of sequences to process and populate a thread-specific hashset. This thread-specific hashset stores only the hashes of the phrases. After having populated the hashset for one genome, there is a merging phase in which we merge the populated thread-specific hashset into a collective hashmap. This collective hashmap stores hashes of phrases as keys and the corresponding counter as items. The collective hashmap is then processed to output the final histogram after every genome is processed. In our program, we use the standard C++ library implementation for the hashsets, while we again use the `growt` implementation for the collective hashmap.

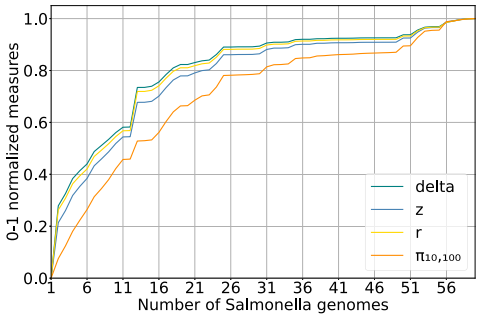
3. Experimental results and discussion

For all experiments presented in this section, we used a machine equipped with an Intel(R) Core(R) i9-11900 @ 2.50 GHz (with turbo speed @ 5 GHz) with 16 MB of cache and 8 cores (16 threads via hyperthreading), 64 GB of DDR4-3200MHz RAM and 1 TB of SSD disk space. The operating system was Ubuntu 22.04 LTS, and the compiler used was `g++` version 11.4.0. The code was compiled with options `-std=c++20 -O3 -march=native -ldl -pthread` enabled. Measurements regarding running times and space consumption were recorded using the built-in command of the operating system `/usr/bin/time -v`. Runtime and peak memory values correspond to a single run.

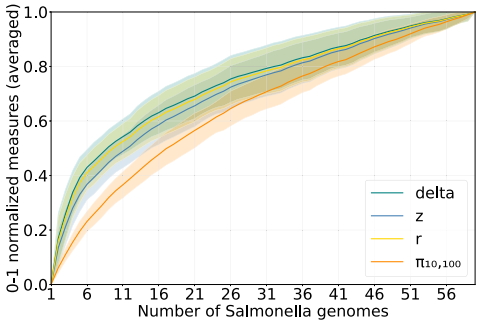
3.1. PFP as a repetitiveness measure

In this section, we evaluate our proposed repetitiveness measure, $\pi_{w,p}$, through three sets of experiments. First, we compare $\pi_{w,p}$ with the established repetitiveness measures r, z, and δ. Previous work has already compared the working space of PFP with r. In particular, Oliva (2023) found a linear correlation between the size of the PFP and r in one set of highly repetitive data. Additionally, Lucà (2023) studied how the degree of sequence repetitiveness influenced the sizes of D and P across different parameter combinations in various real and artificial datasets, and compared $\|D\|$ to r. Second, we examine the behavior of $\pi_{w,p}$ on real and randomly generated sequences. Third, we evaluate the performance of our tool and other tools for different measures, using both small and large datasets.

In the first experiment, we compare $\pi_{w,p}$ (computed with `piPFP`) to the repetitiveness measures z (computed with `PFP_LZ77`), r (`optimalBWT`), and δ (`substring-complexity`, exact version). We ran



(a) One permutation (number 66).



(b) Averaged over 100 permutations.

Fig. 1. 0–1 normalized measures. δ, z, r and $\pi_{10,100}$ show a similar pattern of growth with the progressive addition of 60 genomes.

`piPFP` with the default parameters $w = 10$ and $p = 100$, the default parameters chosen in the original publication (Boucher et al., 2019), and since π does not vary significantly with $w > 10$ and $p > 50$ (see Fig. 7). We used the dataset provided in Bonnie et al. (2024), as detailed in Section 2.6 and Table 1.

We considered 100 different orderings of the input set of genomes. For each ordering, we computed the repetitiveness measures z, r, δ and $\pi_{10,100}$ on increasingly larger subsets of *S. enterica* genomes, from 1 to 60, and applied 0–1 normalization for each measure individually:

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}},$$

where X is the original value, $X_{\min}$ and $X_{\max}$ are the minimum and maximum values of the measure, respectively, and X' is the normalized value.

We report the results in Fig. 1. The left plot shows the normalized measures for one permutation (permutation number 66, for details see Appendix A), while the right plot shows the same measures averaged over all 100 permutations.

As we can observe, $\pi_{10,100}$ closely follows the trend of the other repetitiveness measures in our experiments. We were also able to confirm the linear correlation between π and r, as observed by Oliva (2023) (see Fig. 2; for the permutation shown, the Pearson correlation coefficient is 0.982.) This seems to imply that, at least on microbial data, our repetitiveness measure behaves similarly to the more established measures.

In a second set of experiments, we compared $\pi_{w,p}$ of random sequences to that of real biological sequences of the same size to gain a better understanding of the value of $\pi_{w,p}$ as a repetitiveness measure. We performed this experiment on three datasets, including both prokaryotic and eukaryotic data. In particular, we used 60 *S. enterica* genomes, 120 *E. coli* genomes, and 28 *S. cerevisiae* genomes. For each dataset, we generated three types of random sequences:

[1] https://gitlab.com/manzai/Big-BWT.

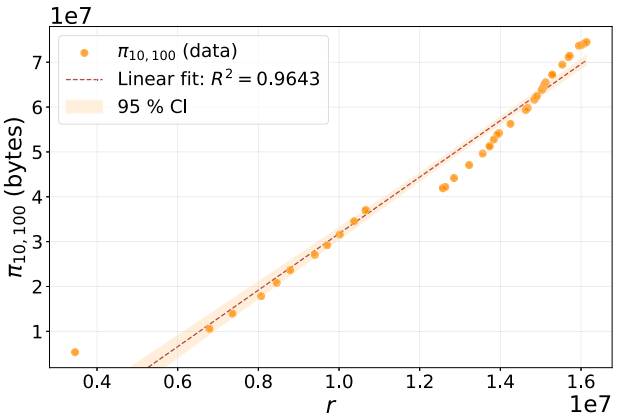


Fig. 2. Linear correlation between $\pi_{10,100}$ and r. Points are measurements from permutation number 66. The dashed line is a least-squares fit with slope 6.29968 and intercept -3.121×10^7 ($R^2 = 0.9643$); the shaded area shows the 95% confidence interval.

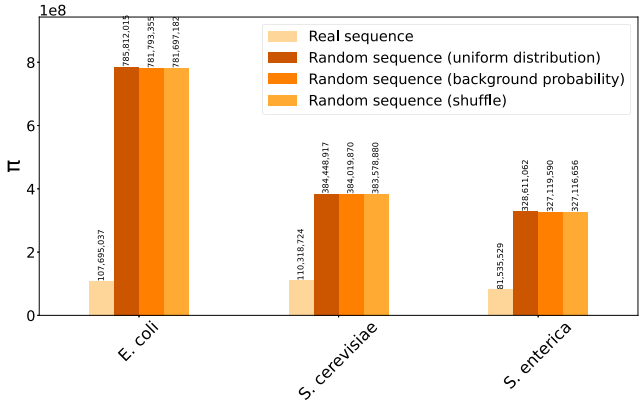


Fig. 3. Comparison of $\pi_{w,p}$ of real sequences and random sequences of the same size. For each dataset, we report $\pi_{10,100}$ for the real set of sequences and for three sets of randomized sequences (see Section 3.1 for details). For each random sequence, the plotted value is the mean over 100 independently generated sequences; the corresponding standard deviation is negligible at the scale of the plot (data not shown, see Table A.5 for mean and standard deviation values).

1. Each character is chosen according to the uniform distribution, i.e., with probability $1/\sigma$.
2. Each character is chosen according to a background distribution, based on the character frequencies in the original sequence.
3. The sequence is a random permutation of the original sequence (C++ `shuffle`).

For each randomization method, we generated 100 sets of sequences, with the same cardinality and same total length as the biological data, and computed $\pi_{10,100}$ for each one, as well as for the biological sequence set. We report the results in Fig. 3. In all three datasets, $\pi_{10,100}$ for the randomized sequences is substantially larger than for the corresponding real sequence, thus showing that π successfully distinguishes between real-life sequences and random sequences.

In a third set of experiments, we evaluated the efficiency of `piPFP` by comparing its running time and space consumption with those of other tools. Because we are now focusing on efficiency, we have added further tools to those used previously (namely, `PFP_LZ77`, `optimalBWT`, and `substring-complexity`). In particular we added: `kkp3`, `ropeBWT2`, `ropeBWT3`, `substring-complexity` (streaming version), and `DandD` (dashing mode). All tools used for the repetitiveness measure experiments are summarized in Table 2.

We ran every tool using a single thread on the full set of 60 *S. enterica* genomes and on another dataset consisting of 120 *E. coli* genomes. As shown in Fig. 4(a), our tool outperforms its competitors in terms of running time by at least an order of magnitude. It is worth noting that our tool also allows for high parallelization: using 16 logical threads, the time is reduced from 2.20 s to 0.34 s for *S. enterica* and from 5.34 s to 0.87 s for *E. coli*. Regarding space consumption, `piPFP` uses considerably less space than most of the other tools, except `DandD` and the streaming version of `substring-complexity`.

3.2. PFP for estimating pangenome openness

We now move our attention to the topic of pangenome openness. To study the effectiveness of `piPFP` in estimating pangenome openness, we performed three sets of experiments. The first set involved a comparative analysis of `piPFP` against other tools. The second set focused on examining the impact of parameter selection on the accuracy of `piPFP`. The third set evaluated the computational efficiency of our tool. For the first two sets of experiments, we used the same datasets employed by Parmigiani et al. (2024). For the third set, we incorporated larger datasets, as detailed in Section 2.6.

In the first experiment, we analyzed 12 prokaryotic genomes (Table 1, entries 8–19) using three different tools: `Pangrowth`[2] v1.0.0 (Parmigiani et al., 2024), `DandD` (Bonnie et al., 2024), and `piPFP` (our tool), with parameters $(4, 10)$ and $(5, 10)$ for (w, p).

In Table 3, we report the α values that indicate whether a pangenome is classified as open or closed. For completeness, we include the results from the gene-based tools reported in Parmigiani et al. (2024) (Table 2 there): `Roary`[3] (Page et al., 2015), `Pantools`[4] (Jonkheer et al., 2022) and `BPGA`[5] (Chaudhari et al., 2016). Out of the 12 datasets analyzed, `piPFP` classifies 10 as open, which aligns with the results from the other tools, while 2 are classified as closed. Specifically, the datasets designated as closed are *C. burnetii* ($\alpha = 1.44$) and *Y. pestis* ($\alpha = 1.69$). For *C. burnetii*, our tool's classification agrees with `DandD` ($\alpha = 1.25$) only, whereas for *Y. pestis*, our tool disagrees with all others.

To explain our results, we note that here we chose small w and p for `piPFP` in order to get smaller phrases to better capture the differences between genomes. One might be tempted to view this as sampling the k-mer space, for $k = p + w - 1$. This is because, on average, we would expect every pth window to map to 0, resulting in roughly n/p phrases of length roughly $p + w - 1$. However, we found that the phrases from our experiments for small w and p result in significant variability of phrase lengths, which is far from a regular sampling of the k-mers present in the genomes (data not shown). It is intriguing to see that, nonetheless, in most cases, the results we obtain with these parameters closely follow those from other tools.

In the second set of experiments, we studied whether the exact choice of parameters has a significant impact on the results. In Fig. 5, we show that varying the parameters of `piPFP` does not have a noteworthy impact on the output α value. We tried every combination of $w = \{4, 5, 6, 7\}$ and $p = \{10, 11, 12, 13, 14, 15\}$. As depicted in the boxplots, the distribution of the different α values has a very low variance. The standard deviation in all cases lay between 0.016 and 0.042 (see Table 4).

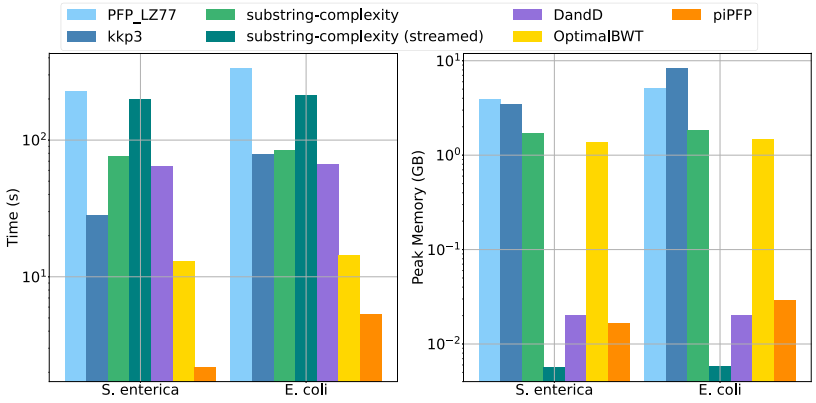
In the third set of experiments, we evaluated the efficiency of `piPFP` against `Pangrowth` and `DandD`. The results for `Pangrowth` and `piPFP` are shown in Fig. 6. `DandD`'s results are excluded because its computation time was significantly higher than that of the other two; this is due to its method for calculating pangenome growth, which

[2] https://github.com/gi-bielefeld/pangrowth.
[3] https://github.com/sanger-pathogens/Roary.
[4] https://git.wur.nl/bioinformatics/pantools.
[5] https://sourceforge.net/projects/bpgatool/.

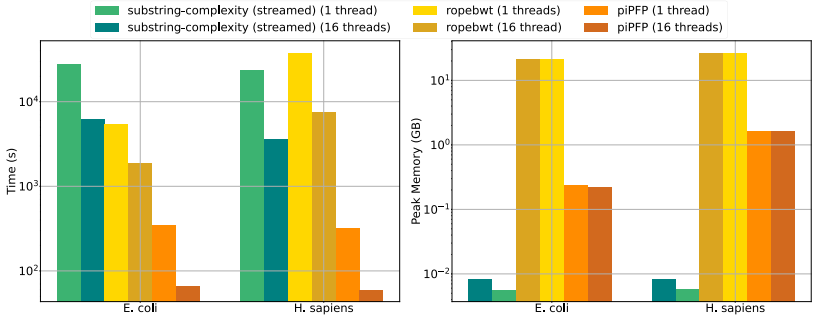
Table 2
List of tools used for the repetitiveness experiments. The tools are grouped according to the measure they compute.

	Tool	Parameters and notes
z	`PFP_LZ77 v0.0.1`[a] `kkp3`[b]	Parameters `-w 10 -p 100`.
r	`optimalBWT`[c] `ropeBWT3 v3.9`[d]	Parameters `-a sais`.
δ	`substring- complexity`[e] `DandD v1.0.0`[f]	Exact version with the executable `delta`. Streaming version with the executable `delta-stream`. Dashing mode. Command `dandd tree`. Parameters `-k 15 -n 10`.
$\pi_{w,p}$	`piPFP`[g]	Parameters `-w 10 -p 100`.

[a] https://github.com/AaronHong1024/PFP_LZ77 (Hong et al., 2023).
[b] https://www.cs.helsinki.fi/group/pads/lz77.html (Kärkkäinen et al., 2013).
[c] https://github.com/davidecenzato/optimalBWT (Cenzato et al., 2023).
[d] https://github.com/lh3/ropebwt3 (Li, 2024).
[e] https://github.com/regindex/substring-complexity (Kociumaka et al., 2023; Becker et al., 2024).
[f] https://github.com/jessicabonnie/dandd (Bonnie et al., 2024).
[g] https://github.com/simolucaa/piPFP.



(a) Efficiency comparison of tools on small datasets, namely 60 *S. enterica* genomes and 120 *E. coli* genomes (y-axis in log-scale).



(b) Efficiency comparison of tools on big datasets, namely 7725 *E. coli* genomes and 9 human genomes (y-axis in log-scale). The experiments on LZ did not finish, since none of the tools scaled to this data size.

Fig. 4. Efficiency comparisons on small and big datasets. The parameters choice for `piPFP` is $w = 10$ and $p = 100$.

involves averaging over a subset of permutations of genome unions. In this regard, we selected a subset of size 20, as the datasets are small and Bonnie et al. chose this parameter in the original DandD paper (Bonnie et al., 2024).

In our experiments on smaller datasets (Table 1, entries 8–19), `DandD` consistently used less space compared to the other tools; however, the running times ranged from around 20 s to 45 min, depending on the dataset. While using a larger amount of space, our tool consistently stays below 1 GB of RAM, yet runs considerably faster. Compared to `Pangrowth`, `piPFP` consistently proved to be faster, used less space, and scaled well with the data (Fig. 6).

3.3. Parameters of PFP

The authors of the original paper, which introduced the PFP (Boucher et al., 2019), included a small study on the choice of the two parameters, the window size w and the modulus p. They gave experimental results on the Pizza and Chili repetitive corpus (Pizza and Chili repetitive corpus, 2025) with respect to compression (i.e., our π) with three parameter settings $(6, 20), (8, 50), (10, 100)$, and on a biological dataset of 2.7 GB (1000 *S. enterica* genomes) with respect to running time and peak memory, with 15 combinations of parameters ($w = 6, 8, 10$ and $p = 50, 100, 200, 400, 800$). Based on their results, they chose the parameters $(10, 100)$ as the default setting of their tool

Table 3
α values for piPFP, Pangrowth, DandD, Roary, Pantools and BPGA for each species.

Species	#genes	piPFP (4,10)	piPFP (5,10)	Pangrowth	DandD	Roary	Pantools	BPGA
B. aphidicola	2911	0.22	0.24	0.30	0.25	0.20	0.21	0.20
B. cereus	295	0.68	0.71	0.78	0.79	0.68	0.68	0.69
C. botulinum	902	0.63	0.64	0.74	0.72	0.66	0.70	0.68
C. burnetii	2319	1.44	1.51	0.90	1.25	0.87	0.92	0.83
C. jejuni	242	0.84	0.87	0.87	0.90	0.76	0.78	0.69
F. tularensis	767	0.88	0.90	0.76	0.85	0.71	0.73	0.64
H. pylori	790	0.55	0.62	0.68	0.86	0.49	0.54	0.50
P. marinus	421	0.24	0.27	0.28	0.27	0.29	0.32	0.29
R. palustris	788	0.40	0.48	0.56	0.48	0.49	0.49	0.51
S. pneumoniae	818	0.85	0.88	0.75	0.78	0.73	0.77	0.69
S. pyogenes	674	0.76	0.78	0.75	0.76	0.81	0.74	0.68
Y. pestis	889	1.69	1.76	0.83	0.97	0.87	0.62	0.64

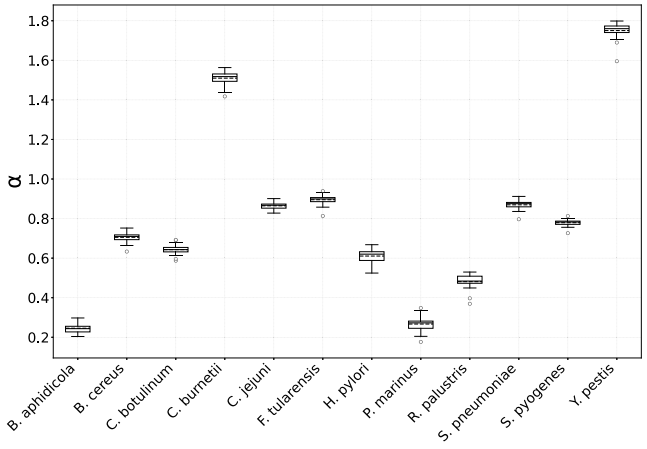


Fig. 5. Distribution of α values on the 12 datasets. The values are taken over 24 parameter choices. See Section 3.2 for more details.

Table 4
Standard deviation and mean of α values.

Species	Standard deviation	Average α
S. pneumoniae	0.023	0.87
B. aphidicola	0.027	0.24
F. tularensis	0.026	0.89
P. marinus	0.039	0.28
S. pyogenes	0.017	0.78
C. jejuni	0.020	0.86
B. cereus	0.026	0.70
Y. pestis	0.042	1.75
H. pylori	0.040	0.61
C. burnetii	0.034	1.51
R. palustris	0.037	0.48
C. botulinum	0.025	0.64

BigBWT. Oliva (2023) further studied how varying the parameters w and p affected $\|D\|$, $|P|$, and the average length of a phrase. In his experiment, he tested combinations of $w = 10, 15, \ldots, 40$ and $p = 50$, 100, 150, 200, 300, 400, 500, 600, 1200, 1800 on a dataset of 1000 diploid haplotypes of Chromosome 19, taken from the 1000 Genomes Project. Lastly, Lucà (2023) replicated the original study's approach by testing the same 15 parameter combinations on three real datasets: *S. enterica*, *E. coli*, and SARS-CoV-2, alongside 36 artificial datasets derived from these real ones (which exhibited varying levels of repetitiveness).

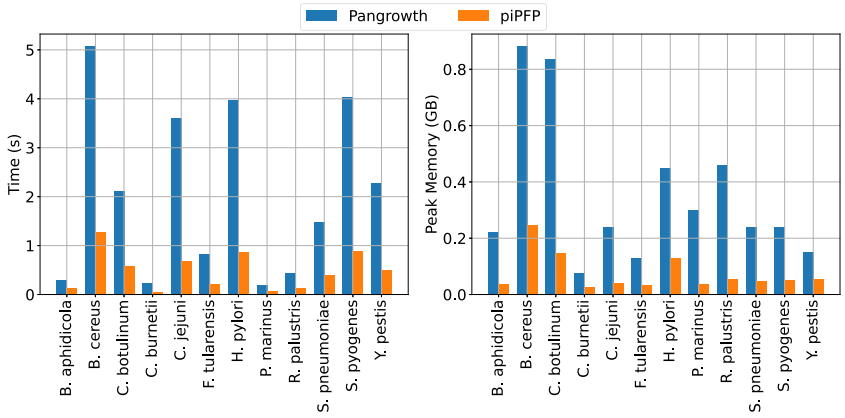
We decided to study the choice of parameters more in depth. First, similarly to the other experiments, we chose regular intervals for our parameters, computing $\pi_{w,p}$ for $w = 5, 10, \ldots, 30$ and $p = 40, 50, 60, \ldots, 290, 300$. The results showed an overall smooth function (as evidenced by discrete approximation of the derivatives for fixed w), in accordance with the results of previous studies. However, we observed a very irregular behavior with $w = 5$ (data not shown).

Therefore, in a second set of experiments, we decided to use randomly chosen parameters to avoid artifacts due to possible number theoretic issues such as divisibility (see Section 3.3.1). On each of our six datasets, we computed $\pi_{w,p}$ for over 1000 parameter combinations, choosing in each case 15 values for w from the interval $[4, 100]$ and 70 values for p from the interval $[10, 500]$. We plot the size $\pi_{w,p}$ of the resulting $PFP(T)$ in Fig. 7. The values for w can be seen in the plots, while those for p can be found in the raw results file provided in the results GitHub repository (https://github.com/simolucaa/piPFP_experiments). We can see that in most cases, the graph for a fixed window size w and increasing modulus p shows an initial decrease, a minimum around $p = 50$ or $p = 100$, followed by a very gradual increase. This is because with modulus p, on average, there should be n/p phrases, where n is the total length of the dataset. Therefore, the parse will require $4n/p$ bytes, since in the parse, the phrases are referred to by their rank in the dictionary, and each integer requires 4 bytes. On the other hand, the dictionary will contain unique phrases only. If the sequences are very similar, a slight increase in phrase length should result in a slightly larger dictionary. These two effects result in a slight increase in the total size $\pi_{w,p}$ for increasing p. Indeed, our experiments show that the parse length is very close to n/p for larger window sizes. (We discuss small window sizes separately below.) We observe that the SARS-CoV-2 data behaves more irregularly than the other datasets for larger window and modulus values (again, as seen by approximating first and second derivatives). This is possibly due to the fact that viruses mutate frequently, and therefore, their genomes have more differences compared to those of other species.

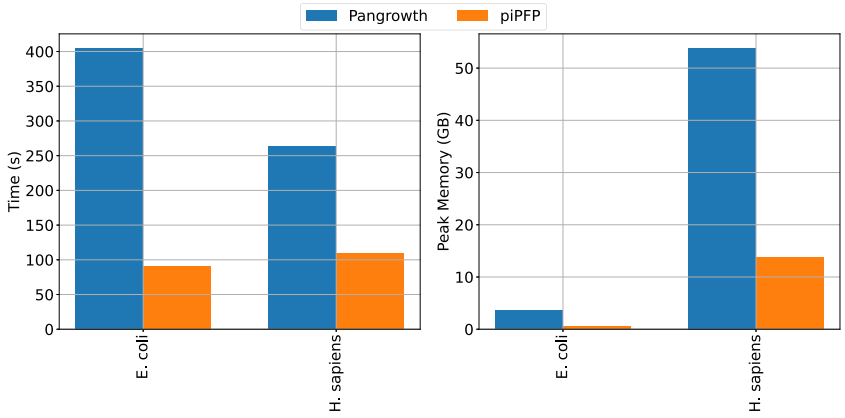
To deepen this study, we also analyzed the behavior of π with fixed p and found that π increases with increasing w (data not shown). This may be due to the fact that increasing the parameter w increases the size of the overlaps between subsequent phrases. Indeed, if we assume that all phrases are unique, then $|P|$ will be approximately n/p, independent of w, while the total size of the dictionary is $\|D_{w,p}\| = |T'| + w(|D_{w,p}| - 1)$, with the last term accounting for the fact that the overlaps are repeated between consecutive phrases. Therefore, if the number of phrases remains the same, then with increasing w, we get an increase in the total size of the overlaps, and consequently of $\|D_{w,p}\|$. Our experimental results indicate that a similar argument holds even if not all phrases are unique.

3.3.1. Small window sizes

As can be seen in several of the plots in Fig. 7, there appears to be an irregular behavior of the size of the data structure for small window sizes. This is because for certain values of p, there are no or very few trigger strings, resulting in one long phrase, and thus no compression at all. This phenomenon can be seen much more in detail in Fig. 8, where we plot, for the same six datasets, the measure $\pi_{w,p}$ using window sizes $w = 4, 5, \ldots, 10$ and modulus p taking values of multiples of 10 between



(a) Efficiency comparison on small datasets.



(b) Efficiency comparison on large datasets.

Fig. 6. Efficiency comparison of `Pangrowth` and `piPFP`. The parameters choice for `piPFP` is $w = 4$ and $p = 10$. The maximum number of threads used by the two tools was set to 16.

40 and 300. For example, on *S. enterica*, there are no trigger strings in the text at all for $w = 5$ and $p = 160, 170$, or for $w = 4$ and $p = 40, 80, 120, 160, 170, 200, 230, 240, 250, 280$, resulting in one long phrase.

This effect is due to a phenomenon which we refer to as *coefficient scarcity*. Since the alphabet is very small, and the characters are encoded according to their ASCII-code, it can happen that no string of size w is hashed to 0 modulo p. For example, if p is a multiple of 40, then no string of length 4 over the alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ maps to 0 modulo p, as we show in Appendix D. To alleviate this phenomenon, one could consider changing the corresponding values to which the individual characters are mapped, such as mapping them to random or prime numbers. However, in most implementations, the standard ASCII encoding is used by default.

4. Conclusion

In this paper, we presented π, the size of the PFP data structure, and proposed it as a measure of repetitiveness of the text on the one hand, and as a method for deciding pangenome openness, on the other. For both uses, we showed that our new measure gives comparable results to other, more established, measures. We also presented a tool, `piPFP`, computing it, and showed that it outperforms other tools both w.r.t. running time and space. To be fair, our tool is tailor-made to compute only π, while several (but not all) of the other methods compute the measures only as a by-product. Nonetheless, we would argue that it makes sense to use a more efficient tool if one only needs the value itself.

On the theoretical side, our experiments show that the PFP heuristic does a good job in capturing the string's repetitiveness, similar to more structured (and well-defined) measures such as z, r, or δ. From our study of the pangenome openness, PFP shows promising results for simulating k-mer counts.

It remains an open problem to compute $\pi^*(T)$, the minimum size of the PFP of the text T, taken over all choices of parameter pairs (w, p). Our results from Section 3.3 can serve as a starting point here. In particular, we saw that while the coefficient scarcity problem seems to be restricted to small window sizes for most of our datasets, on the SARS-CoV-2 data, we observed an irregular behavior for all window sizes, possibly due to the fact that this dataset, being a viral genome, is far more variable than the others. This seems to imply that on more variable data, the choice of parameters has a higher impact on π.

Yet another interesting avenue of further research is the choice of the hash function: Does π vary by a significant amount, or can there be a better trade-off in terms of practical and theoretical running time, say by using highly-engineered hash functions such as xxHash or MurmurHash?

Future work includes exploring further applications of π, such as for pairwise string similarity resp. distance, i.e., as a non-alignment based string similarity measure. Other applications in which we can substitute k-mers with PFP phrases (e.g. read mapping, genome assembly, …) could also be an interesting future direction.

Finally, we believe that studying the dictionary size alone could be of interest, especially as a measure of inter-sequence similarity within a dataset of pairwise similar sequences, such as a pangenome.

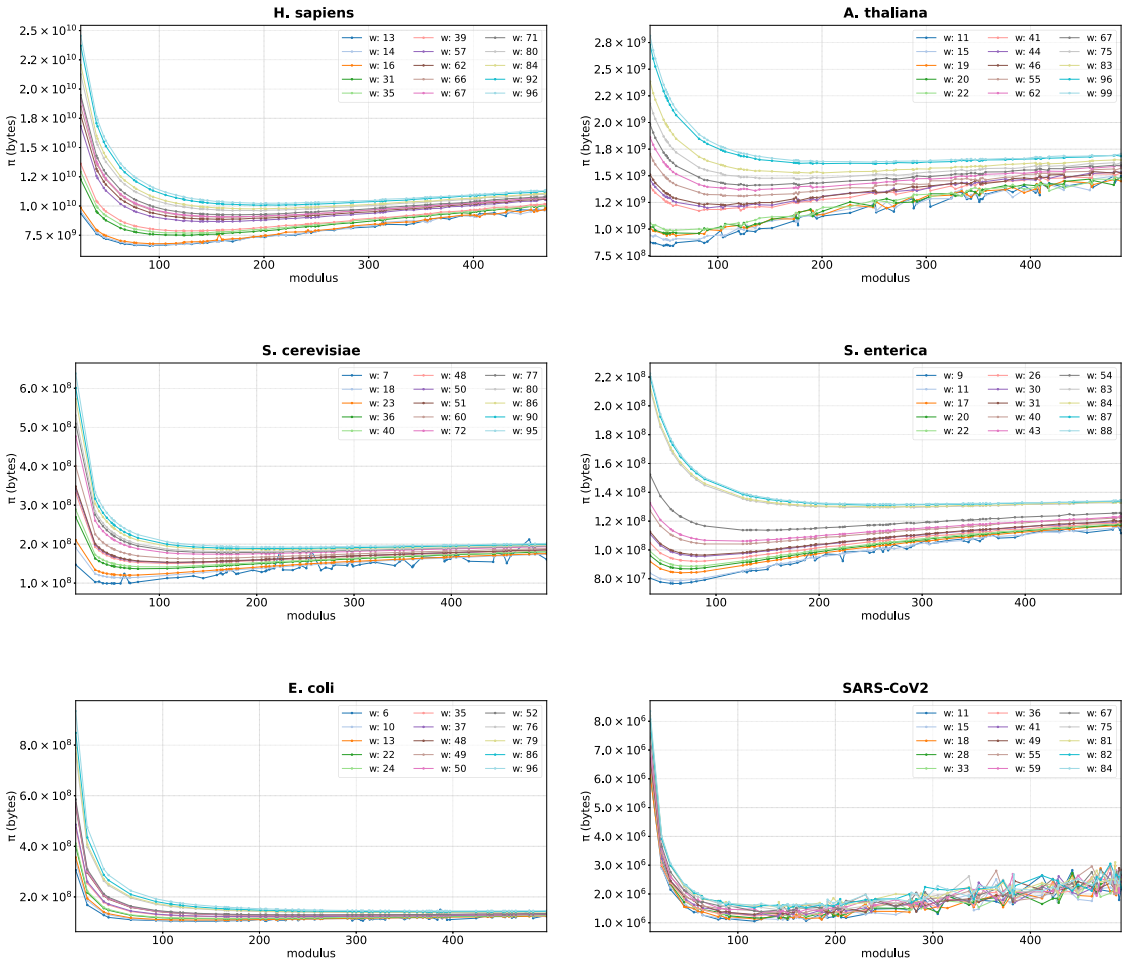


Fig. 7. $\pi_{w,p}$ with randomly chosen w and p. For each species, we chose 15 values for w from the range $[4, 100]$ and 70 values for p from $[10, 500]$.

CRediT authorship contribution statement

Simone Lucà: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Data curation. **Francesco Masillo:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology. **Zsuzsanna Lipták:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

No competing interest is declared.

Acknowledgments

We thank the anonymous reviewers for helpful comments and suggestions. ZsL is partially funded by the MUR PRIN project no. 2022YRB97K 'PINC' ("Pangenome INformatiCs: From Theory to Applications"), and by the INdAM-GNCS project no. E53C24001950001.

Appendix A. Additional details for the repetitiveness experiments

A.1. Comparison of π to other repetitiveness measures

The first experiment in Section 3.1 relies on randomized permutations of the order of genomes. Here we describe the randomization procedure and settings in detail to ensure full reproducibility.

A fixed based seed is used. For permutation $i \in \{1, \ldots, 100\}$, the seed is defined as $\text{seed}_i = 1234 + i - 1$. As a result, permutation $i = 1$ uses seed 1234, permutation $i = 2$ uses seed 1235, and so on.

Permutations are generated using Python's built-in `random.Random` class, which implements the Mersenne Twister pseudo-random number generator (MT19937).

For completeness, we also report explicitly the genome ordering represented in Fig. 1, which corresponds to permutation $i = 66$:

`NZ_CP010283.fna, NZ_CP009561.fna, NC_021810.fna, NC_020307.fna, NZ_CP039502.fna, NC_016831.fna, NC_021818.fna, NZ_CP047542.fna, NC_010102.fna, NC_006511.fna, NC_012125.fna, NC_021812.fna, NC_010067.fna, NC_011147.fna, NZ_CP010279.fna, NC_011205.fna, NC_003198.fna, NZ_CP037893.fna, NC_021176.fna, NC_004631.fna, NZ_CP019035.fna, NC_011274.fna, NC_016857.fna, NZ_CP047544.fna, NC_011094.fna, NZ_CP047529.fna, NZ_CP010280.fna, NZ_CP047540.fna, NC_016832.fna, NC_003197.fna, NZ_CP019186.fna, NZ_CP011394.fna, NZ_CP047550.fna, NZ_CP012347.fna, NC_006905.fna, NZ_CP007528.fna, NZ_CP047531.fna, NC_011294.fna, NZ_CP010282.fna, NC_021820.fna, NC_017623.fna, NC_016810.fna, NC_011080.fna, NZ_CP007598.fna, NZ_CP047525.fna, NZ_CP047535.fna, NZ_CP010281.fna, NZ_CP047546.fna, NC_017046.fna, NZ_CP010284.fna, NC_016863.fna, NZ_CP007530.fna, NZ_CP015923.fna, NC_016854.fna, NZ_CP047553.fna, NC_011149.fna, NC_011083.fna,`

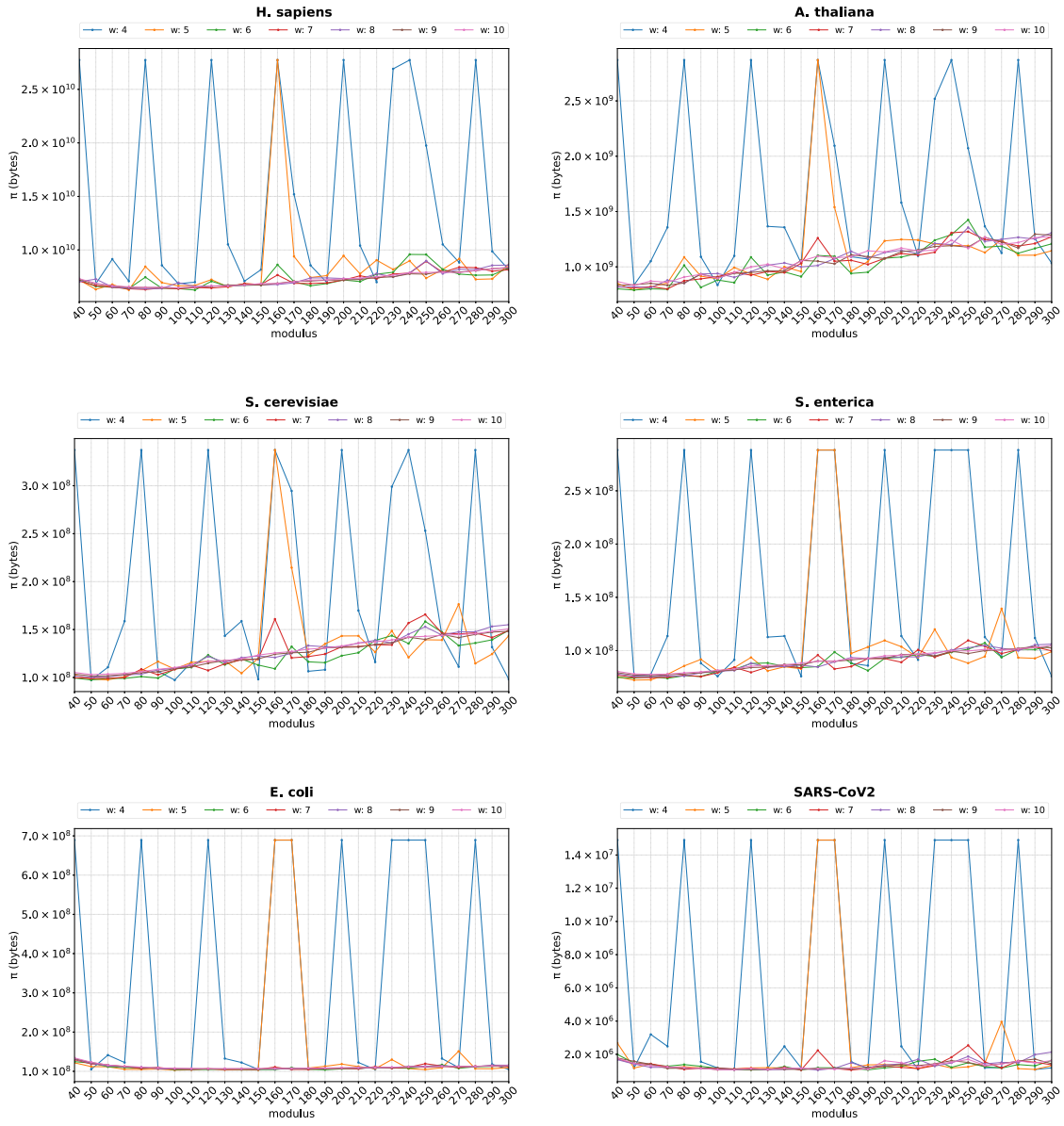


Fig. 8. Values of $\pi_{w,p}$ with small w. This highlights the unstable behavior of π with small w, especially with $w = 4, 5$.

`NZ_CP015924.fna, NC_021814.fna, NC_016856.fna.`

All other permutations can be reproduced exactly using the procedure and seeds described above.

A.2. Comparison of π of real and random sequences

In the second set of experiments, in which we compare $\pi_{w,p}$ of real sequences to randomized sequences (Section 3.1, Fig. 3), we generated 100 independent sequences for each of the three randomization methods and computed $\pi_{10,100}$ on each of them.

Random sequences are generated with a C++ program, using the pseudorandom generator `std::mt19937` (Mersenne Twister). We use a fixed base seed $s_0 = 12345$ and compute the successive seeds as follows:

$$s(\text{mode}, i) = s_0 + 10000 \cdot \text{mode} + i,$$

where $\text{mode} \in \{1, 3, 4\}$ corresponds to the uniform distribution, the background distribution, and shuffle, respectively.

Table A.5
Mean $\pi_{10,100}$ and standard deviation for the randomized sequences. For each dataset and randomization method (Section 3.1) we report the mean and standard deviation of $\pi_{10,100}$ over 100 randomly generated sets of sequences.

Organism	Randomization method	Average $\pi_{10,100}$	Standard deviation
E. coli	Uniform distribution	785 812 015	40 754.33
E. coli	Background distribution	781 793 355	34 491.56
E. coli	Shuffle	781 697 182	34 842.71
S. cerevisiae	Uniform distribution	384 448 917	24 609.11
S. cerevisiae	Background distribution	384 019 870	23 215.71
S. cerevisiae	Shuffle	383 578 880	25 017.84
S. enterica	Uniform distribution	328 611 062	23 325.19
S. enterica	Background distribution	327 119 590	23 875.48
S. enterica	Shuffle	327 116 656	21 339.85

For completeness, we also report the mean $\pi_{10,100}$ and the standard deviation over the 100 randomly generated sets of sequences in Table A.5.

Appendix B. Additional details on the computation of α

In Section 3.2 we estimate the parameter α by fitting Heaps law to f_{new}. Here, we provide additional information about the curve-fitting method.

The fitting is performed using the Python library `NumPy`, more specifically, with the function `np.polyfit`. By default, `np.polyfit` performs a least-squares fit with uniform weighting. The script used to compute α is provided in the same repository as our tool `piPFP`.

Appendix C. Additional details on the experiment on random parameters

In this part, we give additional details for the contents of Section 3.3.

C.1. List of randomly chosen moduli

Here, we provide a list of the randomly chosen moduli for $w \in [4, 100]$ and $p \in [10, 500]$.

- *H. sapiens*: p = 25, 40, 41, 47, 49, 63, 67, 76, 78, 91, 94, 106, 116, 123, 124, 131, 139, 141, 142, 146, 155, 157, 160, 161, 170, 177, 190, 197, 201, 202, 214, 221, 230, 240, 242, 245, 249, 252, 267, 283, 285, 306, 311, 313, 314, 316, 336, 343, 354, 356, 359, 369, 377, 389, 397, 401, 403, 406, 411, 420, 431, 432, 445, 455, 458, 460, 461, 468, 469
- *A. thaliana*: p = 35, 38, 40, 48, 50, 51, 52, 54, 57, 60, 82, 87, 90, 103, 105, 107, 120, 122, 125, 128, 136, 140, 152, 173, 175, 177, 178, 189, 191, 195, 201, 227, 249, 251, 252, 262, 269, 271, 287, 288, 291, 303, 304, 324, 328, 334, 339, 343, 347, 351, 352, 354, 357, 376, 383, 394, 405, 406, 409, 410, 417, 418, 432, 433, 457, 459, 460, 480, 481, 487
- *S. cerevisiae*: p = 14, 34, 38, 42, 46, 51, 53, 54, 60, 64, 69, 70, 78, 108, 119, 135, 145, 159, 163, 165, 169, 172, 174, 179, 181, 196, 205, 209, 210, 213, 218, 224, 227, 237, 244, 245, 251, 256, 266, 271, 277, 279, 280, 293, 297, 298, 299, 306, 317, 322, 342, 347, 351, 360, 361, 370, 392, 393, 395, 399, 405, 418, 443, 445, 454, 457, 461, 480, 486, 498
- *S. enterica*: p = 36, 46, 56, 58, 65, 66, 76, 81, 89, 126, 130, 133, 134, 139, 150, 151, 159, 163, 171, 172, 179, 186, 192, 196, 197, 207, 220, 223, 225, 247, 249, 253, 254, 259, 262, 265, 267, 269, 273, 285, 289, 292, 304, 305, 307, 313, 315, 324, 339, 340, 345, 347, 349, 356, 359, 362, 366, 374, 390, 397, 416, 433, 443, 447, 460, 485, 486, 488, 492
- *E. coli*: p = 10, 22, 39, 44, 67, 93, 108, 109, 118, 121, 134, 155, 164, 166, 168, 177, 181, 189, 201, 206, 207, 217, 218, 219, 227, 234, 240, 241, 246, 257, 263, 265, 291, 296, 300, 305, 307, 313, 314, 317, 325, 330, 334, 350, 355, 363, 369, 373, 375, 376, 378, 479, 483, 384, 387, 389, 392, 394, 397, 424, 436, 439, 440, 441, 452, 454, 458, 461, 465
- SARS-CoV2 : p = 10, 22, 31, 45, 47, 55, 59, 63, 64, 80, 82, 88, 91, 92, 117, 124, 141, 142, 144, 146, 151, 152, 157, 160, 163, 170, 184, 186, 191, 192, 198, 203, 205, 206, 211, 231, 237, 241, 262, 276, 278, 283, 288, 292, 305, 315, 337, 342, 347, 365, 372, 379, 382, 383, 384, 412, 423, 428, 429, 443, 449, 463, 468, 472, 482, 486, 487, 488, 491

C.2. Selection of the parameters

The values were selected by drawing values independently for w and p from the predefined ranges. For each run, parameter values were sampled, using a random seed, from uniform distributions over their respective ranges.

Appendix D. Proof of coefficient scarcity

Let $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, $w = 4$, and let p be a multiple of 40. Consider a string $S = s_1 s_2 s_3 s_4 \in \Sigma^4$, and define $x_i = f(s_i)$, the ASCII-256 encoding of character s_i. In particular,

$$f(\mathtt{A}) = 65, \quad f(\mathtt{C}) = 67, \quad f(\mathtt{G}) = 71, \quad f(\mathtt{T}) = 84.$$

Let $b = 256$ be the base. Then S is a trigger string if and only if

$$x_1 b^3 + x_2 b^2 + x_3 b + x_4 \equiv 0 \pmod{p}. \tag{D.1}$$

Since p is a multiple of 40, Eq. (D.1) implies

$$x_1 b^3 + x_2 b^2 + x_3 b + x_4 \equiv 0 \pmod{40}. \tag{D.2}$$

As can be easily verified, b, b^2, b^3 are all congruent 16 (mod 40), and therefore, Eq. (D.2) equation reduces to

$$16y + z \equiv 0 \pmod{40},$$

where $y = x_1 + x_2 + x_3 \pmod{40}$ and $z = x_4 \pmod{40}$. Now note that the possible values for z are 25, 27, 31, and 4, as these are the values of $f(c) \pmod{40}$ for the four characters $c = \mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}$. Since $16y$ is the additive inverse modulo 40 of z, it must therefore be equal to one of 15, 13, 9, or 36. However, the set generated by the element 16 in $\mathbb{Z}_{40}$ is $\langle 16 \rangle = \{0, 16, 32, 8, 24\}$, and, in particular, does not contain any of those numbers. This proves that Eq. (D.1) has no solution.

Note also that we could ignore the large prime q used for the Karp–Rabin hash in this argument because with such small window sizes, all numbers in question are smaller than q.

As can be seen from Fig. 7, the coefficient scarcity effect disappears with increasing window size.

References

Ahmed, O., Rossi, M., Gagie, T., Boucher, C., Langmead, B., 2023. SPUMONI 2: improved classification using a pangenome index of minimizer digests. Genome Biol. 24 (1), 122. http://dx.doi.org/10.1186/s13059-023-02958-1.

Ahmed, O., Rossi, M., Kovaka, S., Schatz, M.C., Gagie, T., Boucher, C., Langmead, B., 2021. Pan-genomic matching statistics for targeted nanopore sequencing. Iscience 24 (6), http://dx.doi.org/10.1016/j.isci.2021.102696.

Bannai, H., Gagie, T., I., T., 2020. Refining the *r*-index. Theoret. Comput. Sci. 812, 96–108. http://dx.doi.org/10.1016/j.tcs.2019.08.005.

Becker, R., Canton, M., Cenzato, D., Kim, S., Kodric, B., Prezza, N., 2024. Sketching and streaming for dictionary compression. In: Proc. of the 2024 Data Compression Conference. DCC 2024, IEEE, pp. 213–222. http://dx.doi.org/10.1109/DCC58796.2024.00029.

Bonnie, J.K., Ahmed, O.Y., Langmead, B., 2024. DandD: efficient measurement of sequence growth and similarity. IScience 27 (3), http://dx.doi.org/10.1016/j.isci.2024.109054.

Boucher, C., Cenzato, D., Lipták, Zs., Rossi, M., Sciortino, M., 2021a. Computing the original eBWT faster, simpler, and with less memory. In: International Symposium on String Processing and Information Retrieval. Springer, pp. 129–142. http://dx.doi.org/10.1007/978-3-030-86692-1_11.

Boucher, C., Cenzato, D., Lipták, Zs., Rossi, M., Sciortino, M., 2024. *r*-Indexing the eBWT. Inform. and Comput. 298, 105155. http://dx.doi.org/10.1007/978-3-030-86692-1_1.

Boucher, C., Cvacho, O., Gagie, T., Holub, J., Manzini, G., Navarro, G., Rossi, M., 2021b. PFP compressed suffix trees. In: Proc. of the 2021 Workshop on Algorithm Engineering and Experiments. ALENEX 2021, SIAM, pp. 60–72. http://dx.doi.org/10.1137/1.9781611976472.5.

Boucher, C., Gagie, T., I, T., Köppl, D., Langmead, B., Manzini, G., Navarro, G., Pacheco, A., Rossi, M., 2021c. PHONI: Streamed matching statistics with multi-genome references. In: 2021 Data Compression Conference. DCC, IEEE, pp. 193–202. http://dx.doi.org/10.1109/DCC50243.2021.00027.

Boucher, C., Gagie, T., Kuhnle, A., Langmead, B., Manzini, G., Mun, T., 2019. Prefix-free parsing for building big BWTs. Algorithms Mol. Biol. 14 (1), 13:1–13:15. http://dx.doi.org/10.1186/S13015-019-0148-5.

Burrows, M., Wheeler, D.J., 1994. A Block-Sorting Lossless Data Compression Algorithm. Technical Report, DIGITAL System Research Center.

Cenzato, D., Guerrini, V., Lipták, Zs., Rosone, G., 2023. Computing the optimal BWT of very large string collections. In: 2023 Data Compression Conference. DCC, IEEE, pp. 71–80. http://dx.doi.org/10.1109/DCC55655.2023.00015.

Cenzato, D., Olivares, F., Prezza, N., 2025. On computing the smallest suffixient set. In: String Processing and Information Retrieval. Springer Nature Switzerland, pp. 73–87. http://dx.doi.org/10.1007/978-3-031-72200-4_6.

Chaudhari, N.M., Gupta, V.K., Dutta, C., 2016. BPGA–an ultra-fast pan-genome analysis pipeline. Sci. Rep. 6 (1), 24373. http://dx.doi.org/10.1038/srep24373.

Díaz-Domínguez, D., Gagie, T., Guerrini, V., Langmead, B., Lipták, Zs., Manzini, G., Masillo, F., Shivakumar, V., 2025. Prefix-free parsing for merging big BWTs. In: Proc. of the 32nd International Symposium on String Processing and Information Retrieval, SPIRE 2025, London (UK), 8-10 Sept. 2025. In: Lecture Notes in Computer Science, Springer.

Ferragina, P., Manzini, G., 2000. Opportunistic data structures with applications. In: Proc. of the 41st Annual Symposium on Foundations of Computer Science. FOCS 2000, IEEE Computer Society, pp. 390–398. http://dx.doi.org/10.1109/SFCS.2000.892127.

Gagie, T., Gourdel, G., Manzini, G., 2021. Compressing and indexing aligned readsets. In: Proc. of the 21st International Workshop on Algorithms in Bioinformatics. WABI 2021, In: LIPIcs, vol. 201, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 13:1–13:21. http://dx.doi.org/10.4230/LIPICS.WABI.2021.13.

Gagie, T., I, T., Manzini, G., Navarro, G., Sakamoto, H., Takabatake, Y., 2019. Rpair: Rescaling RePair with rsync. In: Proc. of the 26th International Symposium on String Processing and Information Retrieval. SPIRE 2019, In: Lecture Notes in Computer Science, vol. 11811, Springer, pp. 35–44. http://dx.doi.org/10.1007/978-3-030-32686-9_3.

Gagie, T., Navarro, G., Prezza, N., 2020. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. J. ACM 67 (1), 2:1–2:54. http://dx.doi.org/10.1145/3375890.

Goga, A., Depuydt, L., Brown, N.K., Fostier, J., Gagie, T., Navarro, G., 2024. Faster maximal exact matches with lazy LCP evaluation. In: Proc. of the 2024 Data Compression Conference. DCC 2024, IEEE, pp. 123–132. http://dx.doi.org/10.1109/DCC58796.2024.00020.

Heaps, H.S., 1978. Information Retrieval: Computational and Theoretical Aspects. Academic Press, Inc, pp. 206–208.

Heck, Jr., K.L., van Belle, G., Simberloff, D., 1975. Explicit calculation of the rarefaction diversity measurement and the determination of sufficient sample size. Ecology 56 (6), 1459–1461. http://dx.doi.org/10.2307/1934716.

Hong, A., Oliva, M., Köppl, D., Bannai, H., Boucher, C., Gagie, T., 2024. Pfp-fm: an accelerated FM-index. Algorithms Mol. Biol. 19 (1), 15. http://dx.doi.org/10.1186/s13015-024-00260-8.

Hong, A., Rossi, M., Boucher, C., 2023. LZ77 via prefix-free parsing. In: Proc. of the 25th Symposium on Algorithm Engineering and Experiments. ALENEX 2023, SIAM, pp. 123–134. http://dx.doi.org/10.1137/1.9781611977561.CH11.

Jonkheer, E.M., van Workum, D.-J.M., Anari, S.S., Brankovics, B., de Haan, J.R., Berke, L., van der Lee, T.A.J., de Ridder, D., Smit, S., 2022. PanTools v3: functional annotation, classification and phylogenomics. Bioinformatics 38 (18), 4403–4405. http://dx.doi.org/10.1093/bioinformatics/btac506.

Kärkkäinen, J., Kempa, D., Puglisi, S.J., 2013. Linear time Lempel-Ziv factorization: Simple, fast, small. In: Proc. of the 24th Annual Symposium on Combinatorial Pattern Matching. CPM 2013, In: Lecture Notes in Computer Science, vol. 7922, Springer, pp. 189–200. http://dx.doi.org/10.1007/978-3-642s-38905-4_19.

Karp, R.M., Rabin, M.O., 1987. Efficient randomized pattern-matching algorithms. IBM J. Res. Dev. 31 (2), 249–260. http://dx.doi.org/10.1147/rd.312.0249.

Kim, J., Varki, R., Oliva, M., Boucher, C., 2024. Re^{2}pair: Increasing the scalability of RePair by decreasing memory usage. In: 32nd Annual European Symposium on Algorithms. ESA 2024, In: Leibniz International Proceedings in Informatics (LIPIcs), vol. 308, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 78:1–78:15. http://dx.doi.org/10.4230/LIPICS.ESA.2024.78.

Kociumaka, T., Navarro, G., Prezza, N., 2023. Toward a definitive compressibility measure for repetitive sequences. IEEE Trans. Inf. Theory 69 (4), 2074–2092. http://dx.doi.org/10.1109/TIT.2022.3224382.

Kuhnle, A., Mun, T., Boucher, C., Gagie, T., Langmead, B., Manzini, G., 2020. Efficient construction of a complete index for pan-genomics read alignment. J. Comput. Biol. 27 (4), 500–513. http://dx.doi.org/10.1089/cmb.2019.0309.

Lempel, A., Ziv, J., 1976. On the complexity of finite sequences. IEEE Trans. Inform. Theory 22 (1), 75–81. http://dx.doi.org/10.1109/TIT.1976.1055501.

Li, H., 2024. BWT construction and search at the terabase scale. Bioinformatics 40 (12), btae717. http://dx.doi.org/10.1093/bioinformatics/btae717.

Lucà, M., 2023. Parameters for Prefix-Free Parsing: Analysis and Experimentation (Master's thesis). University of Verona, URL: https://github.com/simolucaa/luca_masterthesis.

Maier, T., Sanders, P., Dementiev, R., 2019. Concurrent hash tables: Fast and general (?)!. ACM Trans. Parallel Comput. (TOPC) 5 (4), 1–32. http://dx.doi.org/10.1145/3309206.

Mantaci, S., Restivo, A., Rosone, G., Sciortino, M., 2007. An extension of the Burrows-Wheeler Transform. Theoret. Comput. Sci. 387 (3), 298–312. http://dx.doi.org/10.1016/J.TCS.2007.07.014.

Martínez-Guardiola, C., Brown, N.K., Silva-Coira, F., Köppl, D., Gagie, T., Ladra, S., 2023. Augmented thresholds for MONI. In: Proc. of the 2023 Data Compression Conference. DCC 2023, IEEE, pp. 268–277. http://dx.doi.org/10.1109/DCC55655.2023.00035.

Navarro, G., 2022a. Indexing highly repetitive string collections, Part I: repetitiveness measures. ACM Comput. Surv. 54 (2), 29:1–29:31. http://dx.doi.org/10.1145/3434399.

Navarro, G., 2022b. Indexing highly repetitive string collections, Part II: Compressed indexes. ACM Comput. Surv. 54 (2), 26:1–26:32. http://dx.doi.org/10.1145/3432999.

Olbrich, J., Büchler, T., Ohlebusch, E., 2025. Generating multiple alignments of genomes of the same species. In: Kim, J., Conceição, R.C., Yousef, M., Bhavsar, A., Pelayo, S., Fred, A., Gamboa, H. (Eds.), Proceedings of the 18th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2025 - Volume 1, Porto, Portugal, February 20-22, 2025. SCITEPRESS, pp. 459–468. http://dx.doi.org/10.5220/0013165400003911.

Oliva, M., 2023. Building Succinct Data Structures for Pangenomics (Ph.D. thesis). University of Florida.

Oliva, M., Cenzato, D., Rossi, M., Lipták, Zs., Gagie, T., Boucher, C., 2022. CSTs for terabyte-sized data. In: Proc. of the 2022 Data Compression Conference. DCC 2022, IEEE, pp. 93–102. http://dx.doi.org/10.1109/DCC52660.2022.00017.

Oliva, M., Gagie, T., Boucher, C., 2023. Recursive prefix-free parsing for building big BWTs. In: Proc. of the 2023 Data Compression Conference. DCC 2023, IEEE, pp. 62–70. http://dx.doi.org/10.1109/DCC55655.2023.00014.

Page, A.J., Cummins, C.A., Hunt, M., Wong, V.K., Reuter, S., Holden, M.T.G., Fookes, M., Falush, D., Keane, J.A., Parkhill, J., 2015. Roary: rapid large-scale prokaryote pan genome analysis. Bioinformatics 31 (22), 3691–3693. http://dx.doi.org/10.1093/bioinformatics/btv421.

Pan-Genomics Consortium, T.C., 2018. Computational pan-genomics: status, promises and challenges. Brief. Bioinform. 19 (1), 118–135. http://dx.doi.org/10.1093/bib/bbw089.

Parmigiani, L., Wittler, R., Stoye, J., 2024. Revisiting pangenome openness with k-mers. Peer Community J. 4, http://dx.doi.org/10.24072/pcjournal.415.

Pizza and Chili repetitive corpus, 2025. http://pizzachili.dcc.uchile.cl/repcorpus.html. (Last Accessed 22 January 2025).

Raskhodnikova, S., Ron, D., Rubinfeld, R., Smith, A.D., 2013. Sublinear algorithms for approximating string compressibility. Algorithmica 65 (3), 685–709. http://dx.doi.org/10.1007/s00453-012-9618-6.

Rossi, M., Oliva, M., Langmead, B., Gagie, T., Boucher, C., 2022. MONI: a pangenomic index for finding maximal exact matches. J. Comput. Biol. 29 (2), 169–187. http://dx.doi.org/10.1089/CMB.2021.0290.

Shivakumar, V.S., Ahmed, O.Y., Kovaka, S., Zakeri, M., Langmead, B., 2024. Sigmoni: classification of nanopore signal with a compressed pangenome index. Bioinformatics 40 (Supplement 1), i287–i296. http://dx.doi.org/10.1093/bioinformatics/btae213.

Tettelin, H., Riley, D., Cattuto, C., Medini, D., 2008. Comparative genomics: the bacterial pan-genome. Curr. Opin. Microbiol. 11 (5), 472–477. http://dx.doi.org/10.1016/j.mib.2008.09.006.

Zakeri, M., Brown, N.K., Ahmed, O.Y., Gagie, T., Langmead, B., 2024. Movi: A fast and cache-efficient full-text pangenome index. IScience 27 (12), http://dx.doi.org/10.1016/j.isci.2024.111464.

Ziv, J., Lempel, A., 1977. A universal algorithm for sequential data compression. IEEE Trans. Inform. Theory 23 (3), 337–343.

Ziv, J., Lempel, A., 1978. Compression of individual sequences via variable-rate coding. IEEE Trans. Inf. Theory 24 (5), 530–536. http://dx.doi.org/10.1109/TIT.1978.1055934.