Bioinformatics, 2024, **40(7)**, btae333
https://doi.org/10.1093/bioinformatics/btae333
Advance Access Publication Date: 24 May 2024
Review

OXFORD

Sequence analysis

A survey of BWT variants for string collections

Davide Cenzato [1,†] **and Zsuzsanna Lipták** [2,†,*]

[1]Department of Environmental Sciences, Informatics and Statistics, Ca' Foscari University, Venice, 30123, Italy
[2]Department of Computer Science, University of Verona, Verona, 37134, Italy

*Corresponding author. Department of Computer Science, University of Verona, Strada le Grazie, 15, Verona, 37134, Italy. E-mail: zsuzsanna.liptak@univr.it (Zs.L.)

†Equal contribution.

Associate Editor: Peter Robinson

Abstract

Motivation: In recent years, the focus of bioinformatics research has moved from individual sequences to collections of sequences. Given the fundamental role of the Burrows–Wheeler transform (BWT) in string processing, a number of dedicated tools have been developed for computing the BWT of string collections. While the focus has been on improving efficiency, both in space and time, the exact definition of the BWT used has not been at the center of attention. As we show in this paper, the different tools in use often compute non-equivalent BWT variants: the resulting transforms can differ from each other significantly, including the number r of runs, a central parameter of the BWT. Moreover, with many tools, the transform depends on the input order of the collection. In other words, on the same dataset, the same tool may output different transforms if the dataset is given in a different order.

Results: We studied 18 dedicated tools for computing the BWT of string collections and were able to identify 6 different BWT variants computed by these tools. We review the differences between these BWT variants, both from a theoretical and from a practical point of view, comparing them on eight real-life biological datasets with different characteristics. We find that the differences can be extensive, depending on the datasets, and are largest on collections of many similar short sequences. The parameter r, the number of runs of the BWT, also shows notable variation between the different BWT variants; on our datasets, it varied by a multiplicative factor of up to 4.2.

Availability and implementation: Source code and scripts to replicate the results and download the data used in the article are available at https://github.com/davidecenzato/BWT-variants-for-string-collections.

1 Introduction

The Burrows–Wheeler transform (BWT) (Burrows and Wheeler 1994) is a fundamental string transformation which is at the heart of many modern compressed data structures for text processing, in particular in bioinformatics (Langmead *et al.* 2009, Li and Durbin 2010, Langmead and Salzberg 2012). With the increasing availability of low-cost high-throughput sequencing technologies, the focus has moved from single strings to large string collections, such as the 1000 Genomes Project (The 1000 Genomes Project Consortium 2015), the 10 000 Genomes Project (Genome 10K Community of Scientists 2009), the 100 000 Human Genome Project (Turnbull *et al.* 2018), the 1001 Arabidopsis Project (The 1001 Genomes Consortium 2016), and the 3000 Rice Genomes Project (3K RGP) (Sun *et al.* 2017). This has led to a widespread use of compressed data structures on inputs which are collections of sequences, rather than individual sequences.

A number of tools have been developed in recent years for computing the BWT of a collection (multiset) of strings. The focus has been on efficiently processing datasets of ever increasing sizes, but little attention has been paid to the actual method used to compute the BWT. This is an issue, as the BWT was originally defined for a single string, and it is not immediately clear how to define it for a collection (multiset) of strings. In fact, there exists more than one way to compute a Burrows–Wheeler-type transform of multiple strings. Even though all these methods maintain the properties necessary for building string indexes on top of the BWT, such as reversibility and LF-property, they differ in other, important, ways.

As we will show in this paper, different tools not only apply different algorithms to compute the BWT of the input collection, but they output different transforms. Studying 18 publicly available tools, we identified six distinct BWT-variants which are computed by these tools. The tools included in this study are: `BEETL` (Bauer *et al.* 2013), `BCR_LCP_GSA` (Bauer *et al.* 2013), `ropebwt2` (Li 2014), `nvSetBWT` (Pantaleoni 2014), `msbwt` (Holt and McMillan 2014), `Merge-BWT` (Sirén 2016), `eGSA` (Louza *et al.* 2017), `BigBWT` (Boucher *et al.* 2019), `bwt-lcp-parallel` (Bonizzoni *et al.* 2019), `eGAP` (Egidi *et al.* 2019), `gsufsort` (Louza *et al.* 2020), `G2BWT` (Díaz-Domínguez and Navarro 2021), `grlBWT` (Díaz-Domínguez and Navarro 2023), `pfpebwt` (Boucher *et al.* 2021a), `cais` (Boucher *et al.* 2021a), `r-pfbwt` (Oliva *et al.* 2023), `CMS-BWT` (Masillo 2023), and `optimalBWT` (Cenzato *et al.* 2023). In Table 1, we give the BWT variants as computed by these 18 tools on a toy example of five DNA-strings.

The size of BWT-based compressed data structures such as the RLFM-index (Mäkinen and Navarro 2005) or the r-index (Gagie *et al.* 2020) is typically measured in the number of runs (maximal substrings consisting of the same letter) of the

Received: 4 February 2024; **Revised:** 13 April 2024; **Editorial Decision:** 11 May 2024; **Accepted:** 23 May 2024

Table 1. Overview of some properties of the six BWT variants considered in this paper and the tools computing them.[a]

BWT variant	result on example	order of shared suffixes	tools
non separator-based			
eBWT($\mathcal{M}$)	CGGGATGTACGTTAAAAA	omega-order of strings	`pfpebwt`, `cais`, (`lFGSACA`)
separator-based			
dolEBWT($\mathcal{M}$)	GGAAACGG$$$TTACTGTAAA	lexicographic order of strings	`G2BWT`, `pfpebwt`, `cais`, `msbwt`
mdolBWT($\mathcal{M}$)	GAGAAGCG$$$TTATCTGAAA	input order of strings	`BEETL`, `ropebwt2`, `nvSetBWT`, `eGSA`, `eGAP`, `Merge-BWT`, `bwt-lcp-parallel`, `gsufsort`, `grlBWT`, `BCR_LCP_GSA`
colexBWT($\mathcal{M}$)	AAAGGCGG$$$TTACTGTAAA	colexicographic order of strings	`ropebwt2`, `BCR_LCP_GSA`
optBWT($\mathcal{M}$)	AAAGGGGC$$$TTACTTGAAA	order of as of Bentley et al.'s algo	`optimalBWT`
concBWT($\mathcal{M}$)	AAGAGGGC$$$TTACTGTAAA	lexicographic order of subsequent strings in the input	`BigBWT`, `r-pfbwt`, `CMS-BWT`, tools for single-string BWT

[a] The colors in the example BWTs correspond to interesting intervals in separator-based variants; the same characters are highlighted in the eBWT to show their positions, see Section 3.2. (We included lFGSACA, which also computes the eBWT, in brackets, since it is a library and not a command-line tool, as opposed to all others listed.)

BWT, commonly denoted r. This parameter r has become central as a measure of the storage space required by these data structures. Additionally, much recent research effort has concentrated on the construction of data structures which cannot only store but query, process, and mine strings in space and time proportional to r (Bannai *et al.* 2020, Gagie *et al.* 2020, Cobas *et al.* 2021, Oliva *et al.* 2021, Boucher *et al.* 2024). Moreover, the parameter r (or the related n/r, the average runlength of the BWT) is also being increasingly seen as a measure of repetitiveness of the string or strings, with several recent works theoretically exploring its suitability as such a measure, as well as its relationship to other repetitiveness measures (Giuliani *et al.* 2021, Navarro 2021, Kempa and Kociumaka 2022, Akagi *et al.* 2023). The parameter r is now also being used as a property of the dataset itself, e.g. (Bannai *et al.* 2020, Boucher *et al.* 2021b, Cobas *et al.* 2021).

However, the number of runs varies between the different BWT-variants, as can be seen on our toy example. This has important implications not only for the storage space required for BWT-based compressed data structures, but also for claims about the level of repetitiveness of the dataset. With competing non-equivalent methods around, this measure is not well defined. We will explore this question further (Section 4) and suggest resolving the issue by standardizing the definition.

1.1 Overview of methods for defining multi-string BWT

The classical way of computing text indexes of more than one string is to concatenate them, adding a different end-of-string-symbol at the end of each string, and then to compute the index for the concatenated string. This is the method traditionally used for generating classical data structures such as suffix trees resp. suffix arrays for multiple strings, and results in the so-called *generalized suffix tree* resp. *generalized suffix array*, see e.g. (Gusfield 1997, Ohlebusch 2013). Applied directly, this method would lead to an unacceptable increase in the size of the alphabet, from σ, often a small constant in applications, to $\sigma+k$, where k is the number of strings in the collection, typically in the thousands or even tens or hundreds of thousands. One way to avoid this is to use only conceptually different end-of-string-symbols, i.e. to have only one dollar-sign and apply string input order to break ties. This is the method used by most tools, including `ropebwt2` (Li 2014), `BCR` (Bauer *et al.* 2013) (with a different algorithm which results in the same output), and many others. Another method to avoid increasing the alphabet size is to separate the input strings using the same end-of-string-symbol; in this case, a different end-of-string-symbol has to be added to the end of the concatenated string to ensure correctness, as e.g. done by `BigBWT` (Boucher *et al.* 2019). An equivalent solution is to concatenate the input strings without removing the end-of-line or end-of-file characters, since these act as separators; or to concatenate them without separators and use a bitvector to mark the end of each string. Many studies nowadays use string collections in experiments without turning to dedicated tools for multi-string BWT, e.g. (Bannai *et al.* 2020, Kuhnle *et al.* 2020, Puglisi and Zhukova 2021); often the input strings are turned into one single sequence using one of the methods described above, and then the single-string BWT is computed; it is, however, not always stated explicitly which was the method used to obtain one sequence. Underlying this is the implicit assumption that all methods are equivalent.

In 2007, Mantaci *et al.* (2007) introduced the *extended Burrows–Wheeler transform* (eBWT), which generalizes the BWT to a multiset of strings. The eBWT, like the BWT, is reversible, and maintains other properties of the BWT such as fast pattern matching functionality. The eBWT can handle both linear and circular strings and is thus particularly well applicable in bioinformatics, since many genomic sequences are circular, including mitochondrial DNA, bacterial, and some viral DNA (Boucher *et al.* 2024).

Since then, however, the term "extended BWT" has come to be used as a generic term to denote the BWT of a collection of sequences. This is unfortunate, as the eBWT has several properties, such as independence from the input order, which the other methods do not share; and it is defined using a different order relation from the classical BWT (see Section 2).

Two of the tools listed compute the eBWT according to the original definition, `pfpebwt` and `cais`, both (Boucher *et al.* 2021a). The very recent software `lFGSACA` (Olbrich *et al.* 2024, in press), which was brought to our attention during the review process, also computes the original eBWT; it is, however, a C++ library rather than a command-line tool. All other tools listed append an end-of-string character to the input strings,

explicitly or implicitly, and as a consequence, the resulting transforms differ from the one defined in Mantaci *et al.* (2007). Moreover, the output in most cases depends on the input order of the sequences (except for those tools that compute what we term dolEBWT, colexBWT, or optBWT, see Section 3). The exact nature of this dependence differs from one transform to another. The result is that the BWT variants computed by different tools on the same dataset, or by the same tool on the same dataset but given in a different order, may vary considerably.

1.2 Multidollar BWT

The BWT-variant which we term mdolBWT is the most general one, in the sense that all others, except for the eBWT, can be simulated by mdolBWT. This is also the variant output by most tools, and it is dependent on the input order: both the transform itself and the number of runs varies depending on the order in which the strings are concatenated. Bentley, Gibney, and Thankachan recently gave a linear-time algorithm for computing mdolBWT with the minimum number of runs amongst all input string orders (Bentley *et al.* 2020). (The same paper includes an NP-hardness result regarding finding the best order on the alphabet, which is a different problem.) In order to study the variation of the parameter r, we first implemented a variant of this algorithm counting the minimal number of runs, which later led to a new tool, `optimalBWT`, computing the optBWT (Cenzato *et al.* 2023). We use this tool in our experiments as a baseline for the number of runs of the other BWT-variants. On our real-life biological datasets, the parameter r varies by up to a multiplicative factor of 4.2 between the different variants. It was shown in Cenzato *et al.* (2023) that an improvement by a multiplicative factor of up to 31 can be obtained between the input order and optBWT, again on real-life biological datasets.

1.3 What is not covered

This paper deals with tools for string collections, so we did not include any tool that computes the BWT of a single string, such as libdivsufsort, sais-lite-lcp, libsais, or bwtdisk (Ferragina *et al.* 2012). Although in many cases, these are the tools used for collections of strings, the transform they compute depends on the method with which the string collection was turned into a single string, as explained above. Nor did we include other BWT variants for single strings such as the bijective BWT (Gil and Scott 2012, Köppl *et al.* 2020), since, again, these were not designed for string collections.

We did not include Big-xBWT (Gagie *et al.* 2021), a tool for compressing and indexing read collections, which computes the xBWT of Ferragina *et al.* (2005, 2009), and requires a reference sequence in addition to the string collection. The xBWT is not necessarily a permutation of the input characters. Nor is the tool (Ohlebusch *et al.* 2018) for reference-free xBWT included in this review: even though it does not require a reference sequence, it, too, computes the xBWT and not the BWT. Finally, we did not include (Cazaux and Rivals 2019), since its method for concatenating the input strings (using the same separator symbol but without an additional end-of-string character) has not been implemented.

1.4 Our contributions

1) We identify six distinct BWT variants which are computed by 18 publicly available tools, specifically designed for string collections. We formally describe the differences between these, identifying specific intervals to which differences are restricted.
2) We describe the impact on the number r of runs of the BWT and give an upper bound on how much the colexicographic order (sometimes referred to as "reverse lexicographic order") can differ from the optimal order of Bentley *et al.* (2020).
3) We complement our theoretical analysis with extensive experiments, comparing the BWT variants on eight real-life datasets with different characteristics.
4) We suggest a way of standardizing the parameter r, thus showing how to eliminate the ambiguity caused by the presence of different BWT-variants.

To the best of our knowledge, this is the first systematic study of the different BWT variants in use for collections of strings. In the following, we give the necessary definitions in Section 2. In Section 3, we present the BWT variants and analyze their differences; we discuss the effects on the repetitiveness measure r in Section 4. A summary of our experimental results is given in Section 5. We draw some conclusions from our study in Section 6. Proofs, details of the experimental setup, along with the full tables with detailed results on all eight datasets are included in the Supplementary Material. Source code and scripts to replicate the results and download the data used in the article are available at https://github.com/davidecenzato/BWT-variants-for-string-collections.

A preliminary version of this article appeared in Cenzato and Lipták (2022).

2 Preliminaries

Let Σ be a finite ordered alphabet of size σ. We use the notation $T = T[1..n]$ for a string T of length n over Σ, $T[i]$ for the ith character, and $T[i..j]$ for the substring $T[i]\cdots T[j]$ of T, where $i \leq j$; the length of T is denoted $|T|$, and the empty string is denoted ε. For a string T over Σ and an integer $m > 0$, we write T^m for the m-fold concatenation of T. A string T is called *primitive* if $T = U^m$ implies $T = U$ and $m = 1$. Every string T can be written uniquely as $T = U^m$, where U is primitive; in this case, we refer to U as $\mathrm{root}(T)$ and to m as $\exp(T)$. In other words, for every string T it holds that $T = \mathrm{root}(T)^{\exp(T)}$. Often, an end-of-string character (usually denoted \$) is appended to the end of T; this character is not an element of Σ and is smaller than all characters from Σ. Note that appending a \$ makes any string primitive.

String S is a *conjugate* of string T if $S = T[i..n]T[1..i-1]$ for some $i \in \{1,\ldots,n\}$ (also called the *ith rotation* of T). It is easy to see that a string of length n has n distinct conjugates if and only if it is primitive. A *run* in string T is a maximal substring consisting of the same character; we denote by $\mathrm{runs}(T)$ the number of runs of T. For example, $\mathrm{runs}(\mathtt{CAAGGGA}) = 4$.

For two strings S,T, the *(unit-cost) edit distance*, or *Levenshtein distance*, $\mathrm{dist}_{\mathrm{edit}}(S,T)$ is defined as the minimum number of operations necessary to transform S into T, where an operation can be deletion or insertion of a character, or substitution of a character by another. The *Hamming distance* $\mathrm{dist}_{\mathrm{H}}(S,T)$, defined only if $|S| = |T|$, is the number of positions i such that $S[i] \neq T[i]$. The *lexicographic order* on Σ^* is defined as follows: $S <_{\mathrm{lex}} T$ if S is a proper prefix of T, or if there exists an index j s.t. $S[j] < T[j]$ and for all $i < j$, $S[i] = T[i]$. The *colexicographic order*, or *colex-order* (referred to as

reverse lexicographic order or *rlo* in Li (2014) and Cox *et al.* (2012)), is defined as follows: $S <_{\text{colex}} T$ if $S^{\text{rev}} <_{\text{lex}} T^{\text{rev}}$, where $X^{\text{rev}} = X[n]X[n-1]\cdots X[1]$ denotes the reverse of the string $X = X[1..n]$.

For a string $T = T[1..n]$ over Σ, the BWT (Burrows and Wheeler 1994), BWT(T), is a permutation of the characters of T, given by concatenating the last characters of the lexicographically sorted conjugates of T. In Table 2, we give two examples: BWT(CAGAGA) = GGCAAA, and BWT(CAGAGA$) = AGGC$AA.

It follows from the definition of the BWT that two strings S, T are conjugates if and only if BWT(S) = BWT(T). Indeed, the BWT is reversible up to conjugates: if a string L is the BWT of some string T, then a string S can be computed in linear time such that L = BWT(S), and thus S is a conjugate of T. To make the BWT uniquely reversible, one can add an index to it, marking the lexicographic rank of the conjugate in input. For example, BWT(CAGAGA) = GGCAAA, and the index 4 specifies that the input was the 4th conjugate in lexicographic order. Alternatively, one adds a $ to the end of T, which makes the input unique: BWT(CAGAGA$) = AGGC$AA, and CAGAGA$ is the only string ending in $ with this BWT. Note on the example that BWT with and without end-of-string symbol can be quite different.

An important parameter of the BWT of string T is the number of runs $r(T)$ = runs(BWT(T)). It is well-known that on repetitive inputs, the BWT tends to produce long runs of the same character, making it amenable to compression via runlength-encoding (RLE). In our example, r(CAGAGA) = 3, while the original string has 6 runs. This property, referred to as the *clustering effect* of the BWT, is taken advantage of by compressed data structures such as the RLFM-index (Mäkinen and Navarro 2005) or the r-index (Gagie *et al.* 2020).

Next we define the *omega-order* (Mantaci *et al.* 2007) on Σ^*: $S \prec_\omega T$ if root(S) = root(T) and exp(S) < exp(T), or if $S^\omega <_{\text{lex}} T^\omega$ (implying root(S) ≠ root(T)), where T^ω denotes the infinite string obtained by concatenating T infinitely many times. The omega-order relation coincides with the lexicographic order if neither of the two strings is a proper prefix of the other. The two orders can differ otherwise, e.g. GT $<_{\text{lex}}$ GTC but GTC $\prec_\omega$ GT.

For a multiset of strings $\mathcal{M} = \{T_1, \ldots, T_k\}$, the eBWT, eBWT($\mathcal{M}$) (Mantaci *et al.* 2007), is a permutation of the characters of the strings in $\mathcal{M}$, given by concatenating the last characters of the conjugates of each T_i, for $i = 1, \ldots, k$, listed in omega-order. For example, the omega-sorted conjugates of $\mathcal{M}$ = {GTC, GT} are: CGT, GTC, GT, TCG, TG, hence, eBWT($\mathcal{M}$) = TCTGG, see Table 2. Again, adding the indices of the input conjugates, in this case 2 and 3, makes the eBWT reversible, see Mantaci *et al.* (2007) for details.

Table 2. BWT of the strings CAGAGA and CAGAGA$, and eBWTof the string collections {GTC, GT}, and {GTC$,GT$}.

CAGAGA	BWT	CAGAGA$	BWT	{GTC, GT}	eBWT	{GTC$, GT$}	eBWT
ACAGAG	G	$CAGAGA	A	CGT	T	$GT	T
AGACAG	G	A$CAGAG	G	GTC	C	$GTC	C
AGAGAC	C	AGA$CAG	G	GT	T	C$GT	T
CAGAGA	A	AGAGA$C	C	TCG	G	GT$	$
GACAGA	A	CAGAGA$	$	TG	G	GTC$	$
GAGACA	A	GA$CAGA	A			T$G	G
		GAGA$CA	A			TC$G	G

3 BWT variants for string collections

We identified six distinct transforms, listed below, which were computed by the tools given in Table 1. Let $\mathcal{M} = \{T_1, \ldots, T_k\}$ be a multiset of strings, with total length $N_\mathcal{M} = \sum_{i=1}^{k} |T_i|$. Since several of the data structures depend on the order in which the strings are listed, we implicitly regard $\mathcal{M}$ as a list $[T_1, \ldots, T_k]$, and write $\rho(\mathcal{M})$ for a specific input order ρ.

1) **extended BWT:** eBWT($\mathcal{M}$) of Mantaci *et al.* (2007) (see Section 2)
2) **dollar-eBWT:** dolEBWT($\mathcal{M}$) = eBWT($\{T_i\$ | \mathrm{T_i} \in \mathcal{M}\}$)
3) **multidollar BWT:** mdolBWT($\mathcal{M}$) = BWT($T_1\$_1T_2\$_2 \cdots T_k\$_k$), where dollars are assumed to be smaller than characters from Σ and $\$_1 < \$_2 < \ldots < \$_k$
4) **colexicographic BWT:** colexBWT($\mathcal{M}$) = mdolBWT($\gamma(\mathcal{M})$), where γ is the colexicographic ("reverse lexicographic," rlo) order of the strings in $\mathcal{M}$.
5) **optimal BWT:** optBWT($\mathcal{M}$) = mdolBWT($opt(\mathcal{M})$), where $opt(\mathcal{M})$ is the order given by the algorithm of Bentley *et al.* (2020), which minimizes the number of runs (see Section 4 for details).
6) **concatenated BWT:** concBWT($\mathcal{M}$) = BWT($T_1\$T_2\$ \cdots T_k\$\#$), where # < $.

Because all BWT variants except the eBWT use additional end-of-string symbols as string separators, we refer to these by the collective term *separator-based BWT variants*. In Table 1, we show the six transforms on our running example of five DNA-strings, and give first properties of these transforms. For ease of exposition and comparison, we replaced all separator-symbols by the same dollar-sign $, even where, conceptually or concretely, different dollar-signs are assumed to terminate the individual strings. This is the case for mdolBWT and its special cases, colexBWT and optBWT. Moreover, the concBWT contains one additional character, the final end-of-string symbol, here denoted by #, which is smaller than all other characters; thus, the additional rotation starting with # is the smallest and results in an additional dollar in the first position of the transform. To facilitate the comparison with the other transforms, we remove this first symbol from concBWT and replace the # by $.

It is important to point out that the programs listed in Table 1 do not necessarily use the definitions given here; however, in each case, the resulting transform is the one claimed, up to renaming or removing separator characters, see Sections 3.1 and 3.2.

3.1 The effect of adding separator symbols

The first obvious difference between the eBWT and the separator-based variants is their length: eBWT($\mathcal{M}$) has length $N_\mathcal{M}$, while all other variants have length $N_\mathcal{M} + k$, since they contain an additional character (the separator) for each input string.

In all separator-based transforms, the k-length prefix consists of a permutation of the last characters of the input strings. This is because the rotations starting with the dollars are the first k lexicographically. On the other hand, in the eBWT, these k characters occur interspersed with the rest of the transform; namely, in the positions corresponding to the omega-ranks of the input strings T_i (see Tables 1 and 3).

In general, adding a $ to the end of the strings introduces a distinction, not present in the eBWT, between suffixes and

Table 3. From left to right we show the eBWT, the dolEBWT, the mdolBWT, the colexBWT, the optBWT, and the concBWT of the string collection $\mathcal{M} = \{\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}\}$.[a]

i	eBWT	rotation
(4,4)	C	AATC
(3,1)	G	ACG
(5,3)	G	AGG
(1,1)	G	ATATG
(4,1)	A	ATCA
(1,3)	T	ATGAT
(2,3)	G	ATG
(4,3)	T	CAAT
(3,2)	A	CGA
(3,3)	C	GAC
(5,2)	G	GAG
(1,5)	T	GATAT
(2,2)	T	GAT
(5,1)	A	GGA
(1,2)	A	TATGA
(4,2)	A	TCAA
(1,4)	A	TGATA
(2,1)	A	TGA

i	dolE	rotation
(3,4)	G	$ACG
(1,6)	G	$ATATG
(4,5)	A	$ATCA
(5,4)	A	$GGA
(2,4)	A	$TGA
(4,4)	C	A$ATC
(5,3)	G	A$GG
(2,3)	G	A$TG
(3,1)	$	ACG$
(1,1)	$	ATATG$
(4,1)	$	ATCA$
(1,3)	T	ATG$AT
(4,3)	T	CA$AT
(3,2)	A	CG$A
(3,3)	C	G$AC
(1,5)	T	G$ATAT
(5,2)	G	GA$G
(2,2)	T	GA$T
(5,1)	$	GGA$
(1,2)	A	TATG$A
(4,2)	A	TCA$A
(1,4)	A	TG$ATA
(2,1)	$	TGA$

i	mdol	rotation
(1,6)	G	$_1$ATATG
(2,4)	A	$_2$TGA
(3,4)	G	$_3$ACG
(4,5)	A	$_4$ATCA
(5,4)	A	$_5$GGA
(2,3)	G	A$$_2$TG
(4,4)	C	A$$_4$ATC
(5,3)	G	A$$_5$GG
(3,1)	$_3$	ACG$$_3$
(1,1)	$_1$	ATATG$$_1$
(4,1)	$_4$	ATCA$$_4$
(1,3)	T	ATG$$_1$AT
(4,3)	T	CA$$_4$AT
(3,2)	A	CG$$_3$A
(1,5)	T	G$$_1$ATAT
(3,3)	C	G$$_3$AC
(2,2)	T	GA$$_2$T
(5,2)	G	GA$$_5$G
(5,1)	$_5$	GGA$$_5$
(1,2)	A	TATG$$_1$A
(4,2)	A	TCA$$_4$A
(1,4)	A	TG$$_1$ATA
(2,1)	$_2$	TGA$$_2$

i	colex	rotation
(4,5)	A	$_4$ATCA
(5,4)	A	$_5$GGA
(2,4)	A	$_2$TGA
(3,4)	G	$_3$ACG
(1,6)	G	$_1$ATATG
(4,4)	C	A$_4$ATC
(5,3)	G	A$_5$GG
(2,3)	G	A$_2$TG
(3,1)	$_3$	ACG$_3$
(1,1)	$_1$	ATATG$_1$
(4,1)	$_4$	ATCA$_4$
(1,3)	T	ATG$_1$AT
(4,3)	T	CA$_4$AT
(3,2)	A	CG$_3$A
(3,3)	C	G$_3$AC
(1,5)	T	G$_1$ATAT
(5,2)	G	GA$_5$G
(2,2)	T	GA$_2$T
(5,1)	$_5$	GGA$_5$
(1,2)	A	TATG$_1$A
(4,2)	A	TCA$_4$A
(1,4)	A	TG$_1$ATA
(2,1)	$_2$	TGA$_2$

i	opt	rotation
(2,4)	A	$_2$TGA
(5,4)	A	$_5$GGA
(4,5)	A	$_4$ATCA
(3,4)	G	$_3$ACG
(1,6)	G	$_1$ATATG
(2,3)	G	A$_2$TG
(5,3)	G	A$_5$GG
(4,4)	C	A$_4$ATC
(3,1)	$_3$	ACG$_3$
(1,1)	$_1$	ATATG$_1$
(4,1)	$_4$	ATCA$_4$
(1,3)	T	ATG$_1$AT
(4,3)	T	CA$_4$AT
(3,2)	A	CG$_3$A
(3,3)	C	G$_3$AC
(1,5)	T	G$_1$ATAT
(2,2)	T	GA$_2$T
(5,2)	G	GA$_5$G
(5,1)	$_5$	GGA$_5$
(1,2)	A	TATG$_1$A
(4,2)	A	TCA$_4$A
(1,4)	A	TG$_1$ATA
(2,1)	$_2$	TGA$_2$

i	conc	rotation
23	A	$#ATATG
10	A	ACGAT
14	G	$ATCA$G
19	A	GGA#A
6	G	TGAAC
22	G	A$#ATAT
9	G	AACGA
18	C	AGGA#
11	$	ACG$ATC
1	$	ATATG$T
15	$	ATCA$GG
3	T	ATG$TGA
17	T	CAGGA
12	A	CG$ATCA
13	C	G$ATCA$
5	T	GTGAA
21	G	GA$#ATA
8	T	GAACG
20	$	GGA$#AT
2	A	TATG$TG
16	A	TCA$GGA
4	A	TGTGA
7	$	TGA$ACG

[a] Indices are given with reference to the numbering $T_1 = \texttt{ATATG}, T_2 = \texttt{TGA}, T_3 = \texttt{ACG}, T_4 = \texttt{ATCA}, T_5 = \texttt{GGA}$. Note that we give the rotations according to Lemma 1.

other substrings: since the separators are smaller than all other characters, occurrences of a substring as suffix will be listed en bloc before all other occurrences of the same substring, while in the eBWT, these occurrences are listed interspersed with the other occurrences of the same substring.

Example 1. Let $\mathcal{M} = \{\texttt{AACGAC}, \texttt{TCAC}\}$ and $U = \texttt{AC}$. U occurs both as a suffix and as an internal factor; the characters preceding it are A (internal substring) and C, G (suffix), and we have $\text{eBWT}(\mathcal{M}) = \texttt{CGACATAACC}$, $\text{dolEBWT}(\mathcal{M}) = \texttt{CC\$GCAAATAC\$}$.

Finally, it should be noted that adding end-of-string symbols to the input strings changes the definition of the order applied. As observed above, the omega-order coincides with the lexicographic order on all pairs of strings S, T where neither is a proper prefix of the other; but with end-of-strings characters, no input string can be a proper prefix of another. Thus, on rotations of the $T_i\$$'s, the omega-order equals the lexicographic order. As an example, consider the multiset $\mathcal{M} = \{\texttt{GTC\$}, \texttt{GT\$}\}$ from Section 2: we have the following omega-order among the rotations: GT, GTC, CGT, GT, GTC$, TG, TCG (see Table 2), which coincides with the lexicographic order. Similarly, adding *different* dollars $\$_1$, $\$_2$, ..., $\$_k$ and applying the omega-order results again in the lexicographic order between the rotations, with different dollar symbols considered as distinct characters. This implies:

Lemma 1. *Let $\mathcal{M} = \{T_1, T_2, \ldots, T_k\}$ be a string collection. Then*

1) *$dolEBWT(\mathcal{M}) = mdolBWT(\text{lex}(\mathcal{M}))$, where $\text{lex}(\mathcal{M})$ denotes the lexicographic order of the strings in $\mathcal{M}$;*
2) *$mdolBWT(\mathcal{M}) = \text{eBWT}(\{T_i\$_i \mid i = 1, \ldots, k\})$, up to renaming of dollars.*

Regarding the differences among the separator-based BWT variants, we will show that all differences occur in certain well-defined intervals of the BWT, and that the differences themselves depend only on a specific permutation of $\{1, \ldots, k\}$, given by the combination of the input order, the lexicographic order of the input strings, and the BWT variant applied. In Table 3, we give the full BWT matrices for all separator-based BWT variants.

3.2 Interesting intervals

Let us call a string U a *shared suffix* w.r.t. multiset $\mathcal{M}$ if it is the suffix of at least two strings in $\mathcal{M}$. Let b be the lexicographic rank of the smallest rotation beginning with $U\$$ and e the lexicographic rank of the largest rotation beginning with $U\$$, among all rotations of strings $T\$$, where $T \in \mathcal{M}$. (One can think of $[b, e]$ as the suffix-array interval of $U\$$.) We call $[b, e]$ an *interesting interval* if there exist $i \neq j$ s.t. U is a suffix of both T_i and T_j, and the preceding characters in T_i and T_j are different, i.e. the two occurrences of U as suffix of T_i and T_j constitute a left-maximal repeat. (Put in different terms, interesting intervals correspond to internal nodes in the suffix tree of the reverse string, within the subtree of \$.) Clearly, $[1, k]$ is an interesting interval unless all strings end with the same character. Note that interesting intervals differ both from the *SAP-intervals* of Cox *et al.* (2012) and from the *tuples* of Bentley *et al.* (2020) [called *maximal row ranges* in Manzini (2016)]: the former are the intervals corresponding to *all* shared suffixes U, even if not left-maximal, while the latter include also suffixes U that are not shared.

Lemma 2. *Any two distinct interesting intervals are disjoint.*

We can now narrow down the differences between any two separator-based BWTs of the same multiset. The next proposition states that these can only occur in interesting

intervals (part 1). This implies that the dollar-symbols appear in the same positions in all separator-based variants except for one very specific case (part 2). Moreover, we get an upper bound on the Hamming distance between two separator-based BWTs (part 3).

Proposition 1. *Let L_1 and L_2 be two separator-based BWTs of the same multiset $\mathcal{M}$.*

1) *If $L_1[i] \neq L_2[i]$ then $i \in [b,e]$ for some interesting interval $[b,e]$.*
2) *Let $\mathcal{I}_1$ resp. $\mathcal{I}_2$ be the positions of the dollars in L_1 resp. L_2. If $\mathcal{I}_1 \neq \mathcal{I}_2$ then there are $i \neq j$ such that T_i is a proper suffix of T_j.*
3) $\text{dist}_H(L_1, L_2) \leq \sum_{[b,e] \text{ interesting}} (e-b+1)$.

Proposition 1 implies that the variation of the different transforms can be explained based solely on what rule is used to break ties for shared suffixes. We will see next how the different BWT variants determine this tie-breaking rule.

4 Effects on the parameter *r*

What is the effect of the differences in the BWT variants on the number of runs of the BWT? As the following example shows, the number of runs can differ significantly between different variants.

Example 2. Let $\mathcal{M} = \{$AAAA, AGCA, GCAA, GTCA, CAAA, CGCA, TCAA, TTCA$\}$. Then there are 28 runs in mdolBWT($\mathcal{M}$) = AAAAAAAAACACACACACAC$$GTGTGT$$ AC$$GT$$; colexBWT($\mathcal{M}$) = AAAAAAAAAAACCCCAACCAC$$GGTTGT$$ AC$$GT$$ has 18 runs.

The results of Section 3 give us a method to measure the degree to which the BWT variants can differ.

Lemma 3. *Let $[b,e]$ be an interesting interval, and $(n_1, \ldots, n_\sigma)$ the Parikh vector of $L[b..e]$, i.e. n_i is the number of occurrences of the ith character. Let $\mathtt{a}$ be such that $n_\mathtt{a} = \max_i n_i$, and $N_\mathtt{a} = (e-b+1) - n_\mathtt{a}$, the sum of the other character multiplicities. Then the maximum number of runs in interval $[b,e]$ is $e-b+1$ if $n_\mathtt{a} - 1 \leq N_\mathtt{a}$, and $2N_\mathtt{a}+1$ otherwise.*

Definition 1. Let $\mathcal{M}$ be a multiset and $var([b,e])$ be the bound on number of runs in $[b,e]$ from Lemma 3. The *variability* of $\mathcal{M}$ is

$$var(\mathcal{M}) = \frac{\sum_{[b,e] \text{ interesting}} var([b,e])}{\sum_{[b,e] \text{ interesting}} (e-b+1)}.$$

The colexBWT has been shown experimentally to yield a low number of runs of the BWT (Cox *et al.* 2012, Li 2014). This is because it groups the characters in the interesting intervals in at most σ runs. Even though it does not always minimize r, we can bound its distance from the optimum.

Proposition 2. *Let L be the colexBWT of multiset $\mathcal{M}$, and let r_{OPT} denote the minimum number of runs of any separator-based BWT of $\mathcal{M}$. Then $\text{runs}(L) \leq r_{\text{OPT}} + 2 \cdot c_\mathcal{M}$, where $c_\mathcal{M}$ is the number of interesting intervals.*

The algorithm of Bentley *et al.* (2020) for the optimal order for mdolBWT is based on the idea of starting from the colex-order and then adjusting, where possible, the order of the runs within interesting intervals in order to minimize character changes at the borders, i.e. such that the first and the last run of each interesting interval is identical to the run preceding and following that interesting interval. This is equivalent to sorting groups of sequences sharing the same left-maximal suffix. This sorting can be done on each interesting interval independently without affecting the other interesting intervals. In Table 3, we show the result on our toy example, where it reduces the number of runs by 2 w.r.t. colex order. In the next section, we compare the number of runs of the non-separator based BWT variants to the optimum.

5 Experimental results

We computed the five BWT variants eBWT, dolEBWT, mdolBWT, concBWT, and colexBWT, on eight different genomic datasets. We used the tool `optimalBWT` to compute the minimum number of runs (i.e. that of optBWT) and used this as a baseline for comparison with the r parameter of the other BWT-variants. For mdolBWT and concBWT, we used the default input order in which the dataset was downloaded. The eight datasets have different characteristics: four contain short reads [SARS-CoV-2 short (Starr *et al.* 2020), Simons Diversity reads (Mallick *et al.* 2016), 16S rRNA short (Winand *et al.* 2019), Influenza A reads (Van den Hoecke *et al.* 2015)], and four long sequences [SARS-CoV-2 long (Greaney *et al.* 2022), 16S rRNA long (Edgar 2018), Candida auris reads (Woodworth *et al.* 2019), SARS-CoV-2 genomes (Boucher *et al.* 2021a)], with the last being whole viral genomes.

On each of the datasets, we computed the pairwise Hamming distance between separator-based BWTs. To compare them to the eBWT, we computed the pairwise edit distance on a small subset of the sequences (for computational reasons), and the Hamming distance on the small set, for comparison. We generated the following statistics on each of the datasets: number of interesting intervals, fraction of positions within interesting intervals (total length of interesting intervals divided by total length of the dataset), and the dataset's variability (Definition 1).

In Table 4, we give a brief summary of the experimental results, including these statistics on all eight datasets, as well as the average pairwise distance and average runlength over the five separator-based BWT variants; for full results, see the Supplementary Material.

The experiments showed a high variation in the number of runs in particular on datasets of short sequences. The highest difference was between colexBWT and concBWT, by a multiplicative factor of over 4.2, on the SARS-CoV-2 short dataset. In Fig. 1, we plot the average runlength n/r for the four short sequence datasets, and the percentage increase of the number of runs w.r.t. r_{opt}. The variation is less pronounced

Table 4. Summary of the results on the eight datasets.[a]

Dataset	No. seq	Avg. length	Ratio positions in int. intervals	Variability	Avg. Hamming d. betw. $-sep. BWTs	max *n/r*	min *n/r*	*n/r* optimal
SARS-CoV-2 short	500 000	50	0.792	0.210	0.11754	31.524	7.494	35.125
Simons Diversity reads	500 000	100	0.107	0.976	0.07195	7.873	5.299	8.133
16S rRNA short	500 000	152	0.741	0.058	0.02982	44.253	18.836	44.873
Influenza A reads	500 000	231	0.103	0.363	0.02609	49.172	23.100	50.275
SARS-CoV-2 long	50 000	1075	0.175	0.037	0.00464	73.204	57.568	74.498
16S rRNA long	16 741	1502	0.047	0.104	0.00289	46.879	45.015	47.140
Candida auris reads	50 000	2483	0.007	0.497	0.00246	1.732	1.726	1.732
SARS-CoV-2 genomes	2000	29 805	0.001	0.148	0.00012	521.610	499.549	523.240

[a] From left to right we report dataset names, number of sequences and average sequence length, the ratio of positions in interesting intervals, the variability of the dataset (Definition 1), the average normalized Hamming distance between any two of the separator-based BWT variants (optBWT not included). In the last column, we report the average runlength of the optBWT, and in the previous two columns, the maximum and minimum average runlength (n/r) taken over the other five BWT variants.

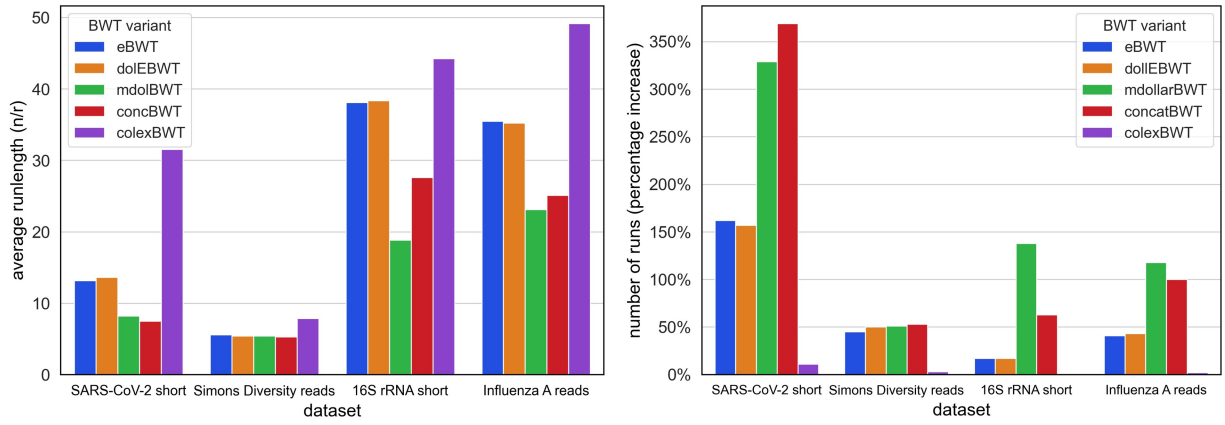

Figure 1. Results regarding r on short sequence datasets, of all BWT variants. Left: average runlength (n/r). Right: number of runs (percentage increase with respect to optBWT).

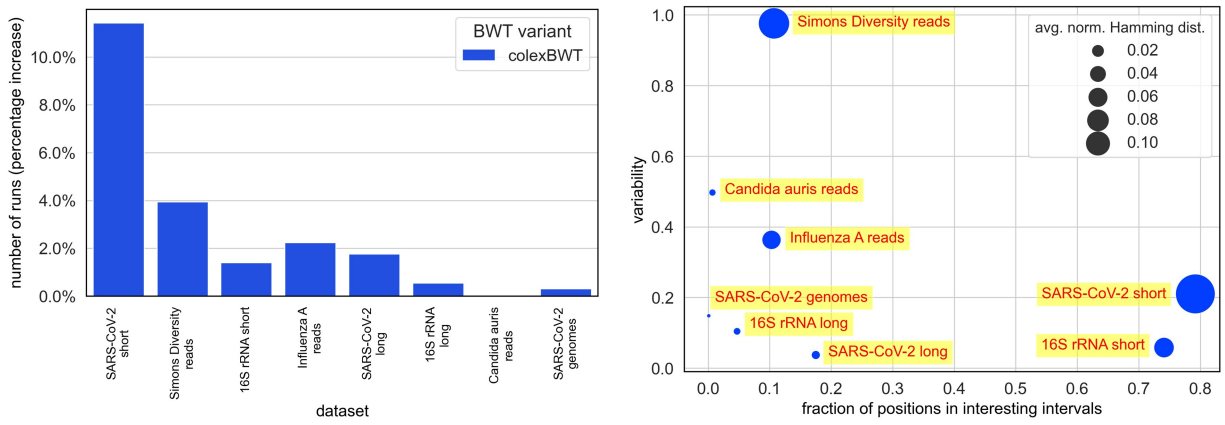

Figure 2. Left: number of runs of the colexBWT with respect to optimal BWT (percentage increase) on all eight datasets. Right: average normalized Hamming distance variations with respect to variability and fraction of positions in interesting intervals on all datasets.

on the one dataset which is less repetitive, namely Simons Diversity reads. Recall that the mdolBWT and concBWT vary depending on the input permutation. On most long sequence datasets, on the other hand, the differences were quite small (see Supplementary Material). To better understand how far the colexBWT is from the optimum w.r.t. the number of runs, we plot in Fig. 2 (left) the number of runs of colexBWT w.r.t. to r_{opt}, on all eight datasets. The strongest increase is on short sequences, where the variation among all BWT variants is high, as well; on the long sequence datasets, with the exception of SARS-CoV-2 long sequences, the colexBWT is very close to the optimum; however, note that on those datasets, all BWTs are close to the optimum.

The average number of runs and the average pairwise Hamming distance strongly depend on the length of the sequences. If the collection has a lot of short sequences which are very similar, then the differences between the BWTs both w.r.t. the number of runs, and as measured by the Hamming

distance, can be large. This is because there are a lot of maximal shared suffixes and so, many positions are in interesting intervals. To better understand this relationship, we plotted, in Fig. 2 (right), the average Hamming distance against the two parameters variability and fraction of positions in interesting intervals. We see that the two datasets with highest average Hamming distance have at least one of the two values very close to 1, while for those datasets where both values are very low, the BWT variants do not differ very much.

The input order used by the mdolBWT and the concBWT is the order in which the input sequences appear when the dataset is downloaded. Our study shows that only a few input permutations can minimize the number of runs of the resulting BWT, namely those orders that group the characters inside the interesting intervals in at most σ runs, such as the order of Bentley *et al.* and the colexicographic order. Since there are $k!$ possible input permutations, selecting an arbitrary input order will likely result in a BWT whose number of runs is much larger than the optimal one, especially on datasets with high variability.

6 Conclusion

We presented the first study of the different variants of the BWT for string collections. We found that the transforms computed by different tools differ not insignificantly, as measured by the pairwise Hamming distance: up to 12% between different BWT variants on the same dataset in our experiments. We showed that most current tools implement BWT variants that are input order dependent, so that the same tool can produce different outputs if the input set is permuted. These differences extend also to the number of runs r, a parameter that is central in the analysis of BWT-based data structures, and which is increasingly being used as a measure of the repetitiveness of the dataset itself.

With string collections replacing individual sequences as the prime object of research and analysis, and thus becoming the standard input for text indexing algorithms, we believe that it is all the more important for users and researchers to be aware that not all methods are equivalent, and to understand the precise nature of the BWT variant produced by a particular tool.

We suggest further to standardize the definition of the parameter r for string collections, using either the colexicographic order—implemented by the tool `ropebwt2` (Li 2014)—or the optimal order of Bentley *et al.* (2020)—implemented by the tool `optimalBWT` (Cenzato *et al.* 2023). In this paper, we found that the number of runs can vary by up to a factor of 4.2 on real-life biological datasets, while in Cenzato *et al.* (2023), a factor of 31 was shown on other biological data. Not only does this heavily impact the space requirements of BWT-based data structures, but it also means that using the average runlength n/r as a repetitiveness measure of a dataset is ambiguous, unless the research community agrees on the BWT variant being used for the definition of this parameter.

Acknowledgements

We thank Massimiliano Rossi for some cleaned and filtered datasets, and the anonymous reviewers for valuable suggestions.

Supplementary data

Supplementary data are available at *Bioinformatics* online.

Conflict of interest

None declared.

Funding

D.C. is funded by the European Union (ERC, REGINDEX, 101039208). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Zs.L. is partially funded by the MUR PRIN Project PINC, Pangenome INformatiCs: from Theory to Applications [Grant No. 2022YRB97K], and by the INdAM—GNCS Project CUP_E53C23001670001. This publication has been made possible thanks to the University of Verona's Special Funds for Open Access Publication.

References

Akagi T, Funakoshi M, Inenaga S. Sensitivity of string compressors and repetitiveness measures. *Inf Comput* 2023;**291**:104999.

Auton A, Brooks LD, Durbin RM *et al.*; 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature* 2015;**526**:68–74. https://doi.org/10.1038/nature15393

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

Bauer MJ, Cox AJ, Rosone G. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theor Comput Sci* 2013;**483**:134–48. https://doi.org/10.1016/j.tcs.2012.02.002.

Bentley JW, Gibney D, Thankachan SV. On the complexity of BWT-runs minimization via alphabet reordering. In: *Proceedings of 28th Annual European Symposium on Algorithms (ESA 2020), Pisa, Italy, September 7-9, 2020. Wadern, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik 2020, Volume 173 of LIPIcs*, **15**:1–15:13. https://doi.org/10.4230/LIPIcs.ESA.2020.15

Bonizzoni P, Vedova GD, Pirola Y *et al.* Multithread multistring Burrows–Wheeler transform and longest common prefix array. *J Comput Biol* 2019;**26**:948–61. https://doi.org/10.1089/cmb.2018.0230

Boucher C, Gagie T, Kuhnle A *et al.* Prefix-free parsing for building big BWTs. *Algorithms Mol Biol* 2019;**14**:13–5. https://doi.org/10.1186/s13015-019-0148-5

Boucher C, Cenzato D, Lipták Zs *et al.* Computing the original eBWT faster, simpler, and with less memory. In: *Proceedings of 28th International Symposium on String Processing and Information Retrieval (SPIRE 2021), Lille, France, October 4-6, 2021. Berlin, Germany: Springer Volume 12944 of LNCS 2021a*, 129–42. https://doi.org/10.1007/978-3-030-86692-1_11

Boucher C, Cvacho O, Gagie T *et al.* PFP compressed suffix trees. In: *Proceedings of 23rd Symposium on Algorithm Engineering and Experiments (ALENEX 2021)*, Virtual Conference, January 10-11, 2021. Philadelphia, PA: SIAM 2021b, 60–72. https://doi.org/10.1137/1.9781611976472.5

Boucher C, Cenzato D, Lipták Zs. *et al.* Indexing the eBWT. *Inf Comput* 2024;**298**:105155. https://doi.org/10.1016/j.ic.2024.105155

Burrows M, Wheeler DJ. *A block sorting lossless data compression algorithm*. Technical Report 124, Digital Equipment Corporation, 1994.

Cazaux B, Rivals E. Linking BWT and XBW via Aho-Corasick automaton: Applications to run-length encoding. In: *Proceedings of 30th Ann. Symp. on Combinatorial Pattern Matching (CPM 2019)*, Pisa,

Italy, June 18-20, 2019. Wadern, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, *Volume 128 of LIPIcs, 2019*, **24**:1–24:20. https://doi.org/10.4230/LIPIcs.CPM.2019.24

Cenzato D, Lipták Zs. A theoretical and experimental analysis of BWT variants for string collections. In *Proceedings of 33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022)*, Prague, Czech Republic, June 27-29, 2022. Wadern, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, *Volume 223 of LIPIcs*, 2022, **25**:1–25:18. https://doi.org/10.4230/LIPICS.CPM.2022.25

Cenzato D, Guerrini V, Lipták Zs *et al.* Computing the optimal BWT of very large string collections. In: *Proceedings of 33rd Data Compression Conference (DCC 2023)*, Snowbird, UT, USA, March 21-24, 2023. New York, NY: IEEE 2023, 71-80.. https://doi.org/10.1109/DCC55655.2023.00015

Cobas D, Gagie T, Navarro G. A fast and small subsampled *r*-index. In: *Proceedings of 32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021), Wrocław, Poland, July 5-7, 2021. Wadern, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Volume 191 of LIPIcs, 2021* **13**:1–13:16. https://doi.org/10.4230/LIPIcs.CPM.2021.13.

Cox AJ, Bauer MJ, Jakobi T *et al.* Large-scale compression of genomic sequence databases with the Burrows–Wheeler transform. *Bioinformatics* 2012;**28**:1415–9. https://doi.org/10.1093/bioinformatics/bts173

Díaz-Domínguez D, Navarro G. Efficient construction of the extended BWT from grammar-compressed DNA sequencing reads. CoRR, abs/2102.03961, 2021, preprint: not peer reviewed.

Díaz-Domínguez D, Navarro G. Efficient construction of the BWT for repetitive text using string compression. *Inf Comput* 2023; **294**:105088.

Edgar RC. Updating the 97% identity threshold for 16S ribosomal RNA OTUs. *Bioinformatics* 2018;**34**:2371–5. https://doi.org/10.1093/bioinformatics/bty113

Egidi L, Louza FA, Manzini G *et al.* External memory BWT and LCP computation for sequence collections with applications. *Algorithms Mol Biol* 2019;**14**:6–6:15. https://doi.org/10.1186/s13015-019-0140-0

Ferragina P, Luccio F, Manzini G Structuring labeled trees for optimal succinctness, and beyond. In: *Proceedings of 46th IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, Pittsburgh, PA, USA, 23-25 October, New York, NY: IEEE 2005., 184–93. https://doi.org/10.1109/SFCS.2005.69

Ferragina P, Luccio F, Manzini G *et al.* Compressing and indexing labeled trees, with applications. *J ACM* 2009;**57**:1–33. https://doi.org/10.1145/1613676.1613680

Ferragina P, Gagie T, Manzini G. Lightweight data indexing and compression in external memory. *Algorithmica* 2012;**63**:707–30. https://doi.org/10.1007/s00453-011-9535-0

Gagie T, Navarro G, Prezza N. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J ACM* 2020;**67**:1–54.

Gagie T, Gourdel G, Manzini G. Compressing and indexing aligned readsets. In: *Proceedings of 21st International Workshop on Algorithms in Bioinformatics (WABI 2021), Virtual Conference, August 2-4, 2021. Wadern, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Volume 201 of LIPIcs 2021*, **13**:1–13:21. https://doi.org/10.4230/LIPIcs.WABI.2021.13

Genome 10K Community of Scientists. A proposal to obtain whole-genome sequence for 10,000 vertebrate species. *J Hered* 2009;**100**: 659–74. https://doi.org/10.1093/jhered/esp086

Gil JY, Scott DA. A bijective string sorting transform. CoRR abs/1201.3077, 2012, preprint: not peer reviewed.

Giuliani S, Inenaga S, Lipták Zs *et al.* Novel results on the number of runs of the Burrows–Wheeler Transform. In: *47th International Conference on Current Trends in Theory and Practice of Comp. Science (SOFSEM 2021), Bolzano-Bozen, Italy, January 25-29, 2021. Berlin, Germany: Springer, Volume 12607 of LNCS 2021*, 249–62. https://doi.org/10.1007/978-3-030-67731-2\18

Greaney AJ, Starr TN, Eguia RT *et al.* A SARS-CoV-2 variant elicits an antibody response with a shifted immunodominance hierarchy. *PLoS Pathog* 2022;**18**:e1010248. https://doi.org/10.1101/2021.10.12.464114

Gusfield D. *Algorithms on Strings, Trees, and Sequences—Computer Science and Computational Biology*. Cambridge, UK: Cambridge University Press, 1997.

Holt J, McMillan L. Merging of multi-string BWTs with applications. *Bioinformatics* 2014;**30**:3524–31. https://doi.org/10.1093/bioinformatics/btu584

Kawakatsu T, Huang S-SC, Jupe F *et al.*; 1001 Genomes Consortium. Epigenomic diversity in a global collection of *Arabidopsis thaliana* accessions. *Cell* 2016;**166**:492–505. https://doi.org/10.1016/j.cell.2016.06.044

Kempa D, Kociumaka T. Resolution of the Burrows–Wheeler transform conjecture. *Commun ACM* 2022;**65**:91–8.

Köppl D, Hashimoto D, Hendrian D *et al.* In-place bijective Burrows–Wheeler transforms. In: *Proceedings of 31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020), Copenhagen, Denmark, June 17-19, 2020. Wadern, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Volume 161 of LIPIcs 2020*, **21**:1–21:15. https://doi.org/10.4230/LIPIcs.CPM.2020.21

Kuhnle A, Mun T, Boucher C *et al.* Efficient construction of a complete index for pan-genomics read alignment. *J Comput Biol* 2020; **27**:500–13.

Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. *Nat Methods* 2012;**9**:357–9. https://doi.org/10.1038/nmeth.1923

Langmead B, Trapnell C, Pop M *et al.* Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol* 2009;**10**:R25. https://doi.org/10.1186/gb-2009-10-3-r25

Li H. Fast construction of FM-index for long sequence reads. *Bioinformatics* 2014;**30**:3274–5. https://doi.org/10.1093/bioinformatics/btu541

Li H, Durbin R. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics* 2010;**26**:589–95. https://doi.org/10.1093/bioinformatics/btp698

Louza FA, Telles GP, Hoffmann S *et al.* Generalized enhanced suffix array construction in external memory. *Algorithms Mol Biol* 2017;**12**: 26. https://doi.org/10.1186/s13015-017-0117-9

Louza FA, Telles GP, Gog S *et al.* gsufsort: constructing suffix arrays, LCP arrays and BWTs for string collections. *Algorithms Mol Biol* 2020;**15**:18. https://doi.org/10.1186/s13015-020-00177-y

Mäkinen V, Navarro G. Succinct suffix arrays based on run-length encoding. *Nordic J Comput* 2005;**12**:40–66.

Mallick S, Li H, Lipson M *et al.* The simons genome diversity project: 300 genomes from 142 diverse populations. *Nature* 2016;**538**: 201–6. https://doi.org/10.1038/nature18964

Mantaci S, Restivo A, Rosone G *et al.* An extension of the Burrows–Wheeler transform. *Theor Comput Sci* 2007;**387**:298–312. https://doi.org/10.1016/j.tcs.2007.07.014

Manzini G. XBWT tricks. In: *Proceedings of 23rd International Symposium on String Processing and Information Retrieval (SPIRE 2016), Beppu, Japan, October 18-20, 2016. Berlin, Germany: Springer, Volume 9954 of LNCS*, 2016 80–92. https://doi.org/10.1007/978-3-319-46049-9

Masillo F. Matching statistics speed up BWT construction. In: *Proceedings of 31st Annual European Symposium on Algorithms (ESA 2023)*, Amsterdam, Netherlands, September 4-6, 2023. Wadern, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Volume 274 of *LIPIcs, 2023*, **83**:1–83:15. https://doi.org/10.4230/LIPICS.ESA.2023.83

Navarro G. Indexing highly repetitive string collections, part I: repetitiveness measures. *ACM Comput Surv* 2021;**54**:1–31. https://doi.org/10.1145/3434399

Ohlebusch E. *Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction*. Bremen, Germany: Oldenbusch Verlag, 2013.

Ohlebusch E, Stauß S, Baier U. Trickier XBWT tricks. In: *Proceedings of 25th International Symposium in String Processing and Information Retrieval (SPIRE 2018), Lima, Peru, October 9-11, 2018. Berlin, Germany: Springer, Volume 11147 of LNCS*, **2018**, 325–33. https://doi.org/10.1007/978-3-030-00479-8\26

Olbrich J, Ohlebusch E, Büchler T. Generic non-recursive sufix array construction. *ACM Trans Algorithms* 2024;**20**:1–42. https://doi.org/10.1145/3641854

Oliva M, Rossi M, Sirén J Efficiently merging r-indexes. In: *Proceedings of 31st Data Compression Conference (DCC 2021)*, Snowbird, UT, USA, March 23-26, 2021. New York, NY: IEEE 2021, 203–12. https://doi.org/10.1109/DCC50243.2021.00028

Oliva M, Gagie T, Boucher C. Recursive prefix-free parsing for building big BWTs. In: *Proceedings of 33rd Data Compression Conference (DCC 2023)*, 2023, 62–70. https://doi.org/10.1109/DCC55655.2023.00014.

Pantaleoni J. A massively parallel algorithm for constructing the BWT of large string sets. CoRR abs/1410.0562, 2014, preprint: not peer reviewed.

Puglisi SJ, Zhukova B. Document retrieval hacks. In: *Proceedings of 19th International Symposium on Experimental Algorithms (SEA 2021), Nice, France, June 7-9, 2021. Wadern, Germany: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Volume 190 of LIPIcs 2021*, **12**:1–12:12. https://doi.org/10.4230/LIPIcs.SEA.2021.12

Sirén J. Burrows–Wheeler Transform for terabases. In *Proceedings of 26th Data Compression Conference (DCC 2016)*, Snowbird, UT, USA, March 30 - April 1, 2016. New York, NY: IEEE 2016, 211–20. https://doi.org/10.1109/DCC.2016.17

Starr TN, Greaney AJ, Hilton SK *et al.* Deep mutational scanning of SARS-CoV-2 receptor binding domain reveals constraints on folding and ACE2 binding. *Cell* 2020;**182**:1295–310.e20. https://doi.org/10.1016/j.cell.2020.08.012

Sun C, Hu Z, Zheng T *et al.* RPAN: rice pan-genome browser for 3000 rice genomes. *Nucleic Acids Res* 2017;**45**:597–605. https://doi.org/10.1093/nar/gkw958

Turnbull C, Scott RH, Thomas E *et al.* The 100,000 genomes project: bringing whole genome sequencing to the NHS. *Br Med J* 2018;**361**:k1687. https://doi.org/10.1136/bmj.k1687

Van den Hoecke S, Verhelst J, Vuylsteke M *et al.* Analysis of the genetic diversity of influenza a viruses using next-generation DNA sequencing. *BMC Genomics* 2015;**16**:79. https://doi.org/10.1186/s12864-015-1284-z

Winand R, Bogaerts B, Hoffman S *et al.* Targeting the 16s rRNA gene for bacterial identification in complex mixed samples: comparative evaluation of second (illumina) and third (oxford nanopore technologies) generation sequencing technologies. *IJMS* 2019;**21**:298. https://doi.org/10.3390/ijms21010298

Woodworth MH, Dynerman D, Crawford ED *et al.* Sentinel case of Candida auris in the Western United States following prolonged occult colonization in a returned traveler from India. *Microb Drug Resist* 2019;**25**:677–80. https://doi.org/10.1089/mdr.2018.0408

Bioinformatics, 2024, 40, 1–10
https://doi.org/10.1093/bioinformatics/btae333
Review