A Theoretical and Experimental Analysis of BWT Variants for String Collections

Davide Cenzato
Department of Computer Science, University of Verona, Italy

Zsuzsanna Lipták
Department of Computer Science, University of Verona, Italy

Abstract

The extended Burrows-Wheeler-Transform (eBWT), introduced by Mantaci et al. [Theor. Comput. Sci., 2007], is a generalization of the Burrows-Wheeler-Transform (BWT) to multisets of strings. While the original BWT is based on the lexicographic order, the eBWT uses the omega-order, which differs from the lexicographic order in important ways. A number of tools are available that compute the BWT of string collections; however, the data structures they generate in most cases differ from the one originally defined, as well as from each other. In this paper, we review the differences between these BWT variants, both from a theoretical and from a practical point of view, comparing them on several real-life datasets with different characteristics. We find that the differences can be extensive, depending on the dataset characteristics, and are largest on collections of many highly similar short sequences. The widely-used 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.

2012 ACM Subject Classification Theory of computation → Data compression; Applied computing → Bioinformatics

Keywords and phrases Burrows-Wheeler-Transform, extended BWT, string collections, repetitiveness measures, r, compression

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.25

Related Version *Full Version*: `https://arxiv.org/abs/2202.13235`

Supplementary Material *Software (Source Code and Data)*: `https://github.com/davidecenzato/BWT-variants-for-string-collections`

Acknowledgements We would like to thank Massimiliano Rossi who supplied us with some cleaned and filtered datasets.

1 Introduction

The Burrows-Wheeler-Transform [9] (BWT) is a fundamental string transformation which is at the heart of many modern compressed data structures for text processing, in particular in bioinformatics [34, 36, 33]. 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 [53], 10,000 Genomes Project [44], the 100,000 Human Genome Project [55], the 1001 Arabidopsis Project [54], and the 3,000 Rice Genomes Project (3K RGP) [52]. This has led to a widespread use of compressed data structures for string collections.

Concurrently, ever increasing text sizes have been driving a trend towards ever smaller data structures. The size of BWT-based data structures is typically measured in the number of runs (maximal substrings consisting of the same letter) of the BWT, commonly denoted r. This parameter r has become fundamental as a measure of storage space required by such

Table 1 The different BWT variants on the multiset $\mathcal{M} = \{\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}\}$. For detailed explanations, see Section 3.

variant	result on example	tools
eBWT	`CGGGATGTACGTTAAAAA`	`pfpebwt` [6]
dolEBWT	`GGAAACGG$$$TTACTGT$AAA$`	`G2BWT` [14], `pfpebwt` [6], `msbwt` [28]
mdolBWT	`GAGAAGCG$$$TTATCTG$AAA$`	`BCR` [3], `ropebwt2` [35], `nvSetBWT` [48], `Merge-BWT` [50], `eGSA` [38], `eGAP` [16], `bwt-lcp-parallel` [5], `gsufsort` [37]
concBWT	`$AAGAGGGC$#$TTACTGT$AAA$`	`BigBWT` [8], tools for single-string BWT
colexBWT	`AAAGGCGG$$$TTACTGT$AAA$`	`ropebwt2` [35]

data structures. Moreover, much recent research effort has concentrated on the construction of data structures which can not only store but query, process, and mine strings in space and time proportional to r [22, 2, 47, 12].

The parameter r is also being increasingly seen as a measure of repetitiveness of the string, with several recent works theoretically exploring its suitability as such a measure, as well as its relationship to other such measures [43, 24, 1].

Several tools exist that compute variants of the BWT for string collections, among these `BCR` [3], `ropebwt2` [35], `nvSetBWT` [48], `msbwt` [28], `Merge-BWT` [50], `eGSA` [38], `BigBWT` [8], `bwt-lcp-parallel` [5], `eGAP` [16], `gsufsort` [37], `G2BWT` [14], and `pfpebwt` [6]. It should be noted though that, when the input is a collection of strings, it is not completely straightforward how to compute the BWT – since the BWT was originally designed for individual strings. In fact, there exists more than one way to compute a Burrows-Wheeler-type transform for a collection of strings, and it turns out that different tools not only use different algorithms, but they output different data structures. As a first example, in Table 1, we give the BWT variants as computed by 12 tools on a toy example of 5 DNA-strings.

The classical way of computing text indexes of string collections is to concatenate the strings, adding a different end-of-string-symbol at the end of each string, and then computing the index for the concatenated string. This is the method traditionally used for generating classical data structures such as suffix trees and suffix arrays for more than one string, and results in the so-called *generalized suffix tree* resp. *generalized suffix array* (see e.g. [27, 45]). The drawback of this method is an increase in the size of the alphabet, from σ, often a small constant in applications, to $\sigma + k$, where k is the number of elements 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 e.g. by `ropebwt2` [35] and by `BCR` [3]. Another method to avoid increasing the alphabet 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. in `BigBWT` [8]. 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 (e.g. [49, 2, 32]); 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. [40] introduced the *extended Burrows-Wheeler-Transform* (eBWT), which generalizes the BWT to a multiset of strings. The eBWT, like the BWT, is reversible; moreover, it is independent of the order in which the strings in the collection are presented. This is not true of any of the other methods mentioned above. Note that the eBWT differs from the BWT in several ways, most importantly in the order relation for sorting conjugates: while the BWT uses lexicographic order, the eBWT uses the so-called omega-order. (For precise definitions, see Section 2.)

The only tool up to date that computes the eBWT according to the original definition is `pfpebwt` [6]; all other tools append an end-of-string character to the input strings, explicitly or implicitly, and as a consequence, the resulting data structures differ from the one defined in [40]. Moreover, the output in most cases depends on the input order of the sequences (except for [14], [28], and, using a specific option, [35]). As a further complication, the exact nature of this dependence differs from one data structure 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. This variability extends to the parameter r, the number of runs of the BWT. This is all the more important given the fact that r (and the related parameter n/r, the average length of a run) is increasingly being used as a parameter characterizing the dataset itself, namely as a measure of its repetitiveness (see e.g. [12, 2, 7]).

1.1 Our contribution

To the best of our knowledge, this is the first systematic treatment of the different BWT variants in use for collections of strings. Our contributions are:

1. We define five distinct BWT variants which are computed by 12 current tools specifically designed for string collections and formally describe the differences between these, identifying specific intervals to which differences are restricted.
2. We show the influence of the input order on the output, in dependence of the BWT variant.
3. We describe the consequences on the number r of runs of the BWT and give an upper bound on the amount by which the colexicographic order (sometimes referred to as "reverse lexicographic order") can differ from the optimal order of Bentley et al. [4].
4. We complement our theoretical analysis with extensive experiments, comparing the five BWT variants on eight real-life datasets with different characteristics.

1.2 Related work

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 [42], sais-lite-lcp [20], libsais [26], bwtdisk [17]. Even though, in many cases, these are the tools used for collections of strings, the data structure they compute depends on the method used for turning the string collection into a single string, as explained above. Nor did we include other BWT variants for single strings such as the bijective BWT [23, 30], since, again, these were not designed for string collections.

The Big-xBWT [21] is a tool for compressing and indexing read collections, using the xBWT of Ferragina et al. [18, 19]. In addition to the string collection, it requires a reference sequence as input, in contrast to the other tools. Moreover, the output is not comparable either, since its length can vary – as opposed to all other BWT variants we review, the xBWT is not a permutation of the input characters but can be shorter, due to the fact that it first maps the input to a tree and then applies the xBWT to it, a BWT-like index for labeled trees, rather than for strings. Likewise, the tool [46] for reference-free xBWT is not included in this review: even though it does not require a reference sequence, it, too, computes the xBWT, which is a data structure that does not fall within the category we focus on. Further, we did not include SPRING [11], a reference-free compressor for FASTQ and FASTA files: even though it employs a BWT-based compressor (BSC) during computation, it does not output the BWT.

There has been considerable interest recently in the parameter r, the number of runs of the BWT: it was put in relation with other measures of repetitiveness in [29], while both [10] and [4] studied the question which permutation of the input strings of the collection results in the lowest value for r. Since the method for concatenating the input strings used in [10] (using the same separator symbol but without an additional end-of-string character) differs from all BWT variants that have been implemented by some tool, we do not include it in this study. The result by Bentley et al. [4], on the other hand, is more general, and we will employ it as a benchmark in our experimental comparisons (see Section 5).

1.3 Overview

We give the necessary definitions in Section 2; note that we assume familiarity of the reader with the Burrows-Wheeler-Transform. In Section 3, we present the BWT variants and analyse their differences. In Section 4 we discuss the effects on the repetitiveness measure r, while our experimental results are presented in Section 5. We draw some conclusions from our study in Section 6. Due to space restrictions, most proofs have been omitted and can be found in the full version, along with the full tables with detailed results on all eight datasets.

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 $i \leq j$; $|T|$ denotes the length of T, and ε the empty string. For a string T over Σ and an integer $m > 0$, T^m denotes 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. We refer to U as $\text{root}(T)$ and to m as $\exp(T)$, i.e., $T = \text{root}(T)^{\exp(T)}$. A *run* in string T is a maximal substring consisting of the same character; we denote by $\text{runs}(T)$ the number of runs of T. Often, an end-of-string character (usually denoted $) is appended to the end of T; this character is not element of Σ and is assumed to be smaller than all characters from Σ. Note that appending a $ makes any string primitive.

For two strings S, T, the *(unit-cost) edit distance* $\text{dist}_{\text{edit}}(S, T)$ 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* $\text{dist}_{\text{H}}(S, T)$, 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 by $S <_{\text{lex}} T$ 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* in [35, 13]) is defined by $S <_{\text{colex}} T$ 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 $X = X[1..n]$. 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).

Given a string $T = T[1..n]$ over Σ, the *Burrows-Wheeler-Transform* [9], $\mathrm{BWT}(T)$, is a permutation of the characters of T, given by concatenating the last characters of the lexicographically sorted conjugates of T. The number of runs of the BWT of string T is denoted $r(T)$, i.e. $r(T) = \mathrm{runs}(\mathrm{BWT}(T))$. To make the BWT uniquely reversible, one can add an index to it marking the lexicographic rank of the conjugate in input. For example, $\mathrm{BWT}(\texttt{banana}) = \texttt{nnbaaa}$, hence $r(\texttt{banana}) = 3$, 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: $\mathrm{BWT}(\texttt{banana\$}) = \texttt{annb\$aa}$. Note that BWT with and without end-of-string symbol can be quite different.

Next we define the *omega-order* [40] on Σ^*: $S \prec_\omega T$ if $\mathrm{root}(S) = \mathrm{root}(T)$ and $\exp(S) < \exp(T)$, or if $S^\omega <_{\mathrm{lex}} T^\omega$ (implying $\mathrm{root}(S) \neq \mathrm{root}(T)$), 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. $\texttt{GT} <_{\mathrm{lex}} \texttt{GTC}$ but $\texttt{GTC} \prec_\omega \texttt{GT}$.

Given a multiset of strings $\mathcal{M} = \{T_1, \ldots, T_k\}$, the *extended Burrows-Wheeler-Transform*, $\mathrm{eBWT}(\mathcal{M})$ [40], 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 $\mathcal{M} = \{\texttt{GTC}, \texttt{GT}\}$ are: `CGT, GTC, GT, TCG, TG`, hence, $\mathrm{eBWT}(\mathcal{M}) = \texttt{TCTGG}$. Again, adding the indices of the input conjugates, in this case $2, 3$, makes the eBWT uniquely reversible.

3 BWT variants for string collections

We identified five distinct transforms, which we list below, that were computed by the programs listed above. 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 $(\mathcal{M}, \pi)$ explicitly for a specific permutation π in which the strings are presented.

1. $\mathrm{eBWT}(\mathcal{M})$: the extended BWT of $\mathcal{M}$ of Mantaci et al. [40]
2. $\mathrm{dolEBWT}(\mathcal{M}) = \mathrm{eBWT}(\{T_i\$ \mid T_i \in \mathcal{M}\})$ ("dollar-eBWT")
3. $\mathrm{mdolBWT}(\mathcal{M}) = \mathrm{BWT}(T_1\$_1 T_2\$_2 \cdots T_k\$_k)$, where dollars are assumed to be smaller than characters from Σ and $\$_1 < \$_2 < \ldots < \$_k$ ("multidollar BWT")
4. $\mathrm{concBWT}(\mathcal{M}) = \mathrm{BWT}(T_1\$T_2\$ \cdots T_k\$\#)$, where $\# < \$$ ("concatenated BWT")
5. $\mathrm{colexBWT}(\mathcal{M}) = \mathrm{mdolBWT}(\mathcal{M}, \gamma)$, where γ is the permutation corresponding to the colexicographic ('reverse lexicographic') order of the strings in $\mathcal{M}$.

Because all BWT variants except the eBWT use additional end-of-string symbols as string separators, we refer to these four by the collective term *separator-based BWT variants*. In Table 2 we show the five data structures on our running example of 5 DNA-strings, and give first properties. For ease of exposition and comparison, we replaced all separator-symbols by the same dollar-sign `$` for all string separator symbols, even where, conceptually or concretely, different dollar-signs are assumed to terminate the individual strings, as is the case for mdolBWT. Moreover, the concBWT contains one additional character, the final end-of-string symbol, here denoted by `#`, which is smaller than all other characters; thus, the additional rotation starting with `#` is the smallest and results in an additional dollar in the first position of the transform. For ease of comparison, we remove this first symbol from concBWT and replace the `#` by `$`.

Table 2 Overview of properties of the five BWT variants considered in this paper. The colors in the example BWTs correspond to interesting intervals in separator-based variants, see Section 3.2.

BWT variant	example	order of shared suffixes	independent of input order?
non-sep.based			
eBWT($\mathcal{M}$)	`CGGGATGTACGTTAAAAA`	omega-order of strings	yes
separator-based			
dolEBWT($\mathcal{M}$)	`GGAAACGG$$$TTACTGT$AAA$`	lexicographic order of strings	yes
mdolBWT($\mathcal{M}$)	`GAGAAGCG$$$TTATCTG$AAA$`	input order of strings	no
concBWT($\mathcal{M}$)	`AAGAGGGC$$$TTACTGT$AAA$`	lexicographic order of subsequent strings in input	no
colexBWT($\mathcal{M}$)	`AAAGGCGG$$$TTACTGT$AAA$`	colexicographic order	yes

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 Section 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 four 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; 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 Table 2).

The next point is that adding a `$` to the end of the strings introduces a distinction, not present in the eBWT, between suffixes and 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. On the other hand, in the eBWT, these occurrences will be listed interspersed with the other occurrences of the same substring.

▶ **Example 1.** Let $\mathcal{M} = \{$`AACGAC`$,$ `TCAC`$\}$ and $U =$ `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 eBWT($\mathcal{M}$) = `CGACATAACC`, dolEBWT($\mathcal{M}$) = `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 $\mathcal{M} = \{$`GTC$,` `GT$`$\}$ from Section 2: we have the following omega-order among the rotations: `$GT, $GTC, C$GT, GT$, GTC$, T$G, TC$G`, which coincides with the lexicographic order. Similarly, adding *different* dollars $\$_1$, $\$_2$, $\ldots$, $\$_k$ and applying the omega-order results again in the lexicographic order between the rotations, with different dollar symbols considered as distinct characters. Indeed, if we append a different dollar-sign to each input string, then the omega-order, the lexicographic order, and the order of the suffixes of the concatenated string (i.e. our mdolBWT) are all equivalent.

Regarding the differences among the four 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 Tables 3 and 4 we give the full BWT matrices for all five BWT variants, along with the optimal one minimizing the number of runs, see Section 4.

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

index	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$\$_2$

index	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$

index	conc	rotation
23	A	$#ATATG$TGA$ACG$ATCA$GGA
10	A	$ACG$ATCA$GGA$#ATATG$TGA
14	G	$ATCA$GGA$#ATATG$TGA$ACG
19	A	$GGA$#ATATG$TGA$ACG$ATCA
6	G	$TGA$ACG$ATCA$GGA$#ATATG
22	G	A$#ATATG$TGA$ACG$ATCA$GG
9	G	A$ACG$ATCA$GGA$#ATATG$TG
18	C	A$GGA$#ATATG$TGA$ACG$ATC
11	$	ACG$ATCA$GGA$#ATATG$TGA$
1	$	ATATG$TGA$ACG$ATCA$GGA$#
15	$	ATCA$GGA$#ATATG$TGA$ACG$
3	T	ATG$TGA$ACG$ATCA$GGA$#AT
17	T	CA$GGA$#ATATG$TGA$ACG$AT
12	A	CG$ATCA$GGA$#ATATG$TGA$A
13	C	G$ATCA$GGA$#ATATG$TGA$AC
5	T	G$TGA$ACG$ATCA$GGA$#ATAT
21	G	GA$#ATATG$TGA$ACG$ATCA$G
8	T	GA$ACG$ATCA$GGA$#ATATG$T
20	$	GGA$#ATATG$TGA$ACG$ATCA$
2	A	TATG$TGA$ACG$ATCA$GGA$#A
16	A	TCA$GGA$#ATATG$TGA$ACG$A
4	A	TG$TGA$ACG$ATCA$GGA$#ATA
7	$	TGA$ACG$ATCA$GGA$#ATATG$

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. (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 [13] and from the *tuples* of [4] (called *maximal row ranges* in [41]): 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. The next lemma follows from the fact that no two substrings ending in \$ can be one prefix of the other.

Table 4 From left to right we show the eBWT, the colexBWT, and the optimal BWT of the string collection $\mathcal{M} = \{\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}\}$, see Section 4.

index	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

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

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

▶ **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 to interesting intervals. This implies that the dollar-symbols appear in the same positions in all separator-based variants except for one very specific case. Moreover, we get an upper bound on the Hamming distance between two separator-based BWTs:

▶ **Proposition 3.** *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 exist $i \neq j$ such that T_i is a proper suffix of T_j.*
3. $\mathrm{dist}_{\mathrm{H}}(L_1, L_2) \leq \sum_{[b,e] \text{ interesting interval}} (e - b + 1)$.

Proof. *1.* Let $L_1[i] = \texttt{x}$ and $L_2[i] = \texttt{y}$. Since all separator-based BWT variants use the lexicographical order of the rotations, this means that there exists a substring U which is preceded by $\texttt{x}$ in one string T_j and by $\texttt{y}$ in another $T_{j'}$, the first occurrence has rank i in one BWT and the other has rank i in the other BWT variant. This implies that the two occurrences are followed by two dollars, and either the two dollars are different, or they are the same dollar, and the subsequent substrings are different. Therefore, U defines an interesting interval. Parts *2.* and *3.* follow from *1.* ◀

Proposition 3 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.

3.3 Permutations induced by separator-based BWT variants

Let us now restrict ourselves to $\mathcal{M}$ being a set, i.e., no string occurs more than once. (This is just for convenience since now the input order uniquely defines a permutation w.r.t. lexicographic order; the results of this section apply equally to multisets $\mathcal{M}$.) As we showed in the previous subsection, the only differences between the different separator-based BWT variants are given by the order in which shared suffixes are listed. It is also clear that the same order applies in each interesting interval, as well as to the k-length prefix of the transform, whether or not it is an interesting interval.

Since the strings are all distinct, they each have a unique lexicographic rank within the set $\mathcal{M}$. Thus the input order can be seen as a permutation ρ of the lexicographic ranks[1]; if the strings are input in lexicographic order, then $\rho = id$. For our toy example $\mathcal{M} = [\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}]$, we have $\rho = 25134$.

Let us now define as *output permutation* π the permutation of the last characters of the input strings, as found in the k-length prefix of the BWT variant in question. We will denote the output permutations of the dolEBWT, mdolBWT, concBWT, and colexBWT by $\pi_{de}, \pi_{md}, \pi_{conc}$, and π_{colex}, respectively. Again, we give these permutations w.r.t. the lexicographic ranks of the strings. In our running example, we have $\pi_{de} = 12345$, $\pi_{md} = 25134$, $\pi_{conc} = 45132$, and $\pi_{colex} = 34512$.

It is easy to see that the permutation π_{md} is equal to ρ, since the dollar-symbols are ordered according to ρ. For the dolEBWT, the rank of $\texttt{\$}T_i$ equals the lexicographic rank of T_i among all input strings, i.e., $\pi_{de} = id$. Further, $\pi_{colex} = \gamma$ by definition, where γ denotes the colexicographic order of the input strings. The situation is more complex in the case of concBWT. Since the $\texttt{\#}$ is the smallest character, the last string of the input will be the first, while for the others, the lexicographic rank *of the following string* decides the order. In our running example, $\pi_{conc} = 45132$. We next formalize this.

Let Φ_ρ be the *linking permutation* [31] of ρ, defined by $\Phi_\rho(i) = \rho(\rho^{-1}(i)+1)$, for $i \neq \rho(k)$, and $\Phi_\rho(\rho(k)) = \rho(1)$, the permutation that maps each element to the element in the next position and the last element to the first. Let us also define, for $j \in \{1,\ldots,k\}$ and $i \neq j$, $f_j(i)$ by $f_j(i) = i$ if $i < j$ and $i-1$ otherwise. The next lemma gives the precise relationship between ρ and π_{conc}. It says[2], essentially, that π_{conc} is the BWT of ρ.

▶ **Lemma 4.** *Let ρ be the permutation of the input order w.r.t. the lexicographic order, i.e. the ith input string has lexicographic rank $\rho(i)$. Then $\pi_{conc} = \pi_{conc}(\rho)$ is given by:*

$$\pi_{conc}(1) = \rho(k), \qquad \textit{and for } i \neq \rho(k) : \pi^{-1}_{conc}(i) = f_{\rho(1)}(\Phi_\rho(i)) + 1. \tag{1}$$

▶ **Example 5.** The mapping $\rho \mapsto \pi_{conc}$ for $k = 3$ is as follows: $123 \mapsto 312$, $132 \mapsto 231$, $312 \mapsto 231$, $213 \mapsto 321$, $231 \mapsto 132$, and $321 \mapsto 123$. Note that no ρ maps to 213.

As can be seen already for $k = 3$, not all permutations π are reached by this mapping. We will call a permutation π *feasible* if there exists an input order ρ such that $\pi_{conc}(\rho) = \pi$. For $k = 4$, there are 18 feasible permutations (out of 24), for $k = 5$, 82 (out of 120). In

[1] For those used to thinking about suffix arrays, ρ can be seen as the inverse suffix array of the input if the strings are thought of as meta-characters.
[2] We thank Massimiliano Rossi for this observation.

Table 5, we give the percentage of feasible permutations π, for k up to 11. The lexicographic order is always feasible, namely with $\rho = k, k-1, \ldots, 2, 1$; however, the colex order is not always feasible, as the following example shows.

▶ **Example 6.** Let $\mathcal{M} = \{\texttt{GAA}, \texttt{ACA}, \texttt{TGA}\}$, thus $\gamma = 213$, but as we have seen, no permutation of the strings in $\mathcal{M}$ will yield this order for concBWT. In addition, the colexBWT($\mathcal{M}$) = `AAAACGG$AT$$` has 7 runs, while all feasible ones have at least 8: `AAAGACG$AT$$`, `AAACGAG$AT$$`, `AAAAGCG$AT$$`, `AAAGCAG$AT$$`, `AAACAGG$AT$$`.

Table 5 Percentage of feasible permutations w.r.t. concBWT.

no. of seq's k	3	4	5	6	7	8	9	10	11
	83.33%	75.0%	68.33%	63.89%	60.12%	57.29%	54.8%	52.81%	51.0%

An important consequence is that the permutations induced by mdolBWT and concBWT are always different: $\pi_{md} \neq \pi_{conc}$ holds always, since $\pi_{conc}(1) = \rho(k)$. This means that, in whatever order the strings are given w.r.t. lexicographic order, on most string sets the resulting transforms mdolBWT and concBWT will differ.

4 Effects on the parameter r

What is the effect of the different permutations π of the strings in $\mathcal{M}$, induced by these 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 7.** Let $\mathcal{M} = \{\texttt{AAAA}, \texttt{AGCA}, \texttt{GCAA}, \texttt{GTCA}, \texttt{CAAA}, \texttt{CGCA}, \texttt{TCAA}, \texttt{TTCA}\}$. Then mdolBWT($\mathcal{M}$) = `AAAAAAAAACACACACACACAC$$GTGTGT$$AC$$GT$$` has 28 runs, while colexBWT($\mathcal{M}$) = `AAAAAAAAAAAAACCCCAACCAC$$GGTTGT$$AC$$GT$$` has 18 runs.

▶ **Lemma 8.** *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* a *be such that $n_{\texttt{a}} = \max_i n_i$, and $N_{\texttt{a}} = (e - b + 1) - n_{\texttt{a}}$, the sum of the other character multiplicities. Then the maximum number of runs in interval $[b, e]$ is $e - b + 1$ if $n_{\texttt{a}} - 1 \leq N_{\texttt{a}}$, and $2N_{\texttt{a}} + 1$ otherwise.*

We will use this lemma to measure the variability of a dataset:

▶ **Definition 9.** *Let $\mathcal{M}$ be a multiset. For an interesting interval $[b, e]$, let $var([b, e])$ be the upper bound on the number of runs in $[b, e]$ from Lemma 8. Then the* variability *of $\mathcal{M}$ is*

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

Which of the BWT variants produces the fewest runs? As we have shown, this depends on the input order with most BWT variants, and the only possible variation is within interesting intervals. The colexBWT has been shown experimentally to yield a low number of runs of the BWT [35, 13]. Even though it does not always minimize r (one can easily create small examples where other permutations yield a lower number of runs), we can bound its distance from the optimum.

▶ **Proposition 10.** *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* $\text{runs}(L) \leq r_{OPT} + 2 \cdot c_{\mathcal{M}}$, *where $c_{\mathcal{M}}$ is the number of interesting intervals.*

Bentley, Gibney, and Thankachan recently gave a linear-time algorithm for computing the order of the dollars which minimizes the number of runs [4], i.e. the optimal order for mdolBWT. The idea is, in effect, to start from the colex-order and then adjust, 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 4, we show the result on our toy example, where it reduces the number of runs by 2 w.r.t. colex order. We implemented an algorithm that computes the number of optimal runs according to the method of [4] and applied it to our datasets. In the next section, we compare the number of runs of each of the five BWT variants to the optimum.

5 Experimental results

We computed the five BWT variants for eight different genomic datasets, with different characteristics. Four of the datasets contain short reads: SARS-CoV-2 short [51], Simons Diversity reads [39], 16S rRNA short [57], Influenza A reads [56], and four contain long sequences: SARS-CoV-2 long [25], 16S rRNA long [15], Candida auris reads [58], one of which, SARS-CoV-2 genomes, whole viral genomes [6]. The main features of the datasets, including the number of sequences, sequence length, and the mean runlength of the optimal BWT are reported in Table 6. Details of the experiment setup are included in the full version.

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 obvious computational reasons), computing also the Hamming distance on the small set, for comparison. We generated some statistics on each of the data sets: the number of interesting intervals, the fraction of positions within interesting intervals (total length of interesting intervals divided by total length of the dataset), and the dataset's variability (Def. 9). To study the variation of the r-parameter, we implemented the algorithm by Bentley et al. [4] for the optimal input order and computed r_{OPT} for each data set, comparing it to the number of runs of all five BWT variants. In Table 8 and 9, we include a compact version of these results for the two datasets with the highest and the lowest variation between the BWT variants, the SARS-CoV-2 short sequences and the SARS-CoV-2 genomes, respectively. The full experimental results for all eight datasets are contained in the full version.

In Table 7 we give a brief summary of the results, reporting, for each dataset, the fraction of positions in interesting intervals, the dataset's variability, the average pairwise Hamming distance between separator-based BWT variants, and the maximum and minimum value, among the five BWT variants, of the average runlength of the BWT.

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 Figure 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 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 full version). To better understand how far the colexBWT is from the optimum, we plot in Figure 2 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 in the input collection. 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 Figure 3, 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, SARS-CoV-2 short dataset and the Simons Diversity reads, 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.

Table 6 Table summarizing the main parameters of the eight datasets. From left to right we report the dataset name, the number of sequences, the total length, the average, minimum and maximum sequence length and the optimum average runlength (n/r), according to [4].

dataset	no. seq	total length	avg	min	max	n/r (opt)
SARS-CoV-2 short	500,000	25,000,000	50	50	50	35.125
Simons Diversity reads	500,000	50,000,000	100	100	100	8.133
16S rRNA short	500,000	75,929,833	152	69	301	44.873
Influenza A reads	500,000	115,692,842	231	60	251	50.275
SARS-CoV-2 long	50,000	53,726,351	1,075	265	3,355	74.498
16S rRNA long	16,741	25,142,323	1,502	1,430	1,549	47.140
Candida auris reads	50,000	124,150,880	2,483	214	8,791	1.732
SARS-CoV-2 genomes	2,000	59,610,692	29,805	22,871	29,920	523.240

Table 7 Table summarizing the results on the eight datasets. From left to right we report dataset names followed by the ratio of positions in interesting intervals, the variability of the dataset (see Def. 9), the average normalized Hamming distance between any two separator-based BWT variants. In the last two columns we report the maximum and minimum average runlength (n/r) taken over all five BWT variants.

dataset	ratio pos.s in intr.int.s	varia-bility	avg. Hamming d. betw. \$-sep. BWTs	max n/r (avg. runlength)	min n/r (avg. runlength)
SARS-CoV-2 short	0.792	0.210	0.11754	31.524	7.494
Simons Diversity reads	0.107	0.976	0.07195	7.873	5.299
16S rRNA short	0.741	0.058	0.02982	44.253	18.836
Influenza A reads	0.103	0.363	0.02609	49.172	23.100
SARS-CoV-2 long	0.175	0.037	0.00464	73.204	57.568
16S rRNA long	0.047	0.104	0.00289	46.879	45.015
Candida auris reads	0.007	0.497	0.00246	1.732	1.726
SARS-CoV-2 genomes	0.001	0.148	0.00012	521.610	499.549

Table 8 Results for the SARS-CoV-2 short dataset. Top left: absolute and normalized pairwise Hamming distance between separator-based BWT variants. Top right: summary of the dataset properties. Bottom left: absolute and normalized pairwise edit distance between all BWT variants on a subset of the input collection. Bottom right: number of runs and average runlength (n/r) taken over all BWT variants.

SARS-CoV-2 short (500,000 short sequences)

Hamming d. / *norm. Hamming d.*	*Hamming distance on the big dataset*			
	dolEBWT	mdolBWT	concBWT	colexBWT
dolEBWT	0	3,014,183	2,926,602	2,912,860
mdolBWT	0.11820	0	3,013,908	3,102,887
concBWT	0.11477	0.11819	0	3,013,634
colexBWT	0.11423	0.12168	0.11818	0

dataset properties	
no. sequences	500,000
average length	50
total length	25,000,000
no. of interesting intervals	116,598
total length intr.int.s	20,187,840
fraction pos.s in intr.int.s	0.792
variability	0.210

edit d. / *norm. edit d.*	*edit distance on a subset of 5,000 sequences*				
	eBWT	dolEBWT	mdolBWT	concBWT	colexBWT
eBWT	0	28,702	43,903	43,828	46,936
dolEBWT	0.11256	0	17,000	16,921	20,104
mdolBWT	0.17217	0.06667	0	16,130	20,812
concBWT	0.17187	0.06636	0.06325	0	20,830
colexBWT	0.18406	0.07884	0.08162	0.08169	0

no. runs big dataset		
	r	n/r
eBWT	1,902,148	13.143
dolEBWT	1,868,581	13.647
mdolBWT	3,113,818	8.189
concBWT	3,402,513	7.494
colexBWT	808,906	31.524
optimum	725,979	35.125

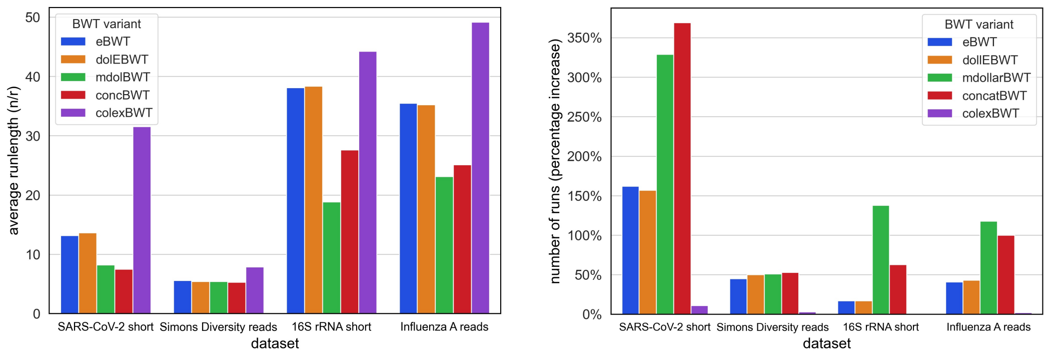

Figure 1 Results regarding r on short sequence datasets, of all BWT variants. Left: average runlength (n/r). Right: number of runs (percentage increase with respect to optimal BWT).

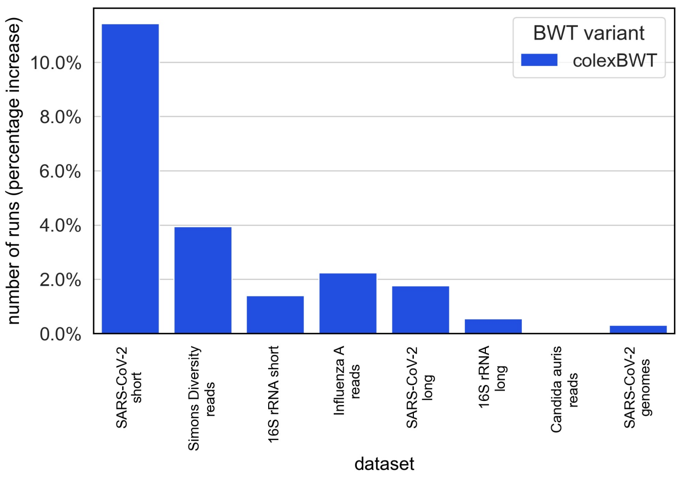

Figure 2 Number of runs of the colexBWT with respect to optimal BWT (percentage increase) on all eight datasets.

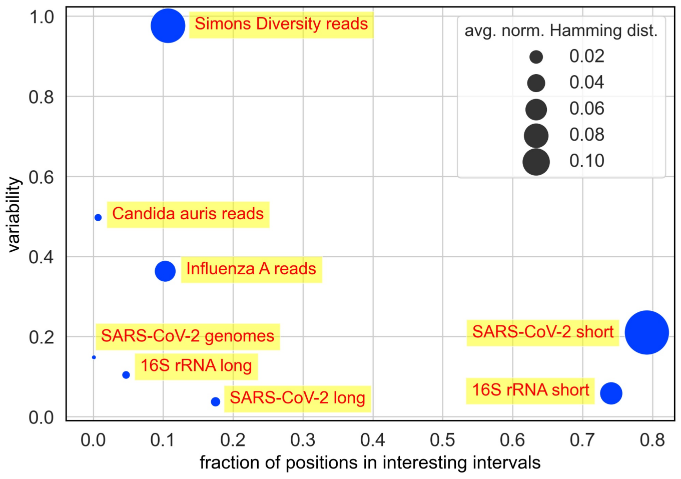

Figure 3 Average normalized Hamming distance variations with respect to variability and fraction of positions in interesting intervals on all datasets.

Table 9 Results for the SARS-CoV-2 genomes dataset. Top left: absolute and normalized pairwise Hamming distance between separator-based BWT variants. Top right: summary of the dataset properties. Bottom left: absolute and normalized pairwise edit distance between all BWT variants on a subset of the input collection. Bottom right: number of runs and average runlength (n/r) taken over all BWT variants.

SARS-CoV-2 genomes (2,000 long sequences)

Hamming d. / *norm. Hamming d.*	*Hamming distance on the big dataset*			
	dolEBWT	mdolBWT	concBWT	colexBWT
dolEBWT	0	7,958	7,900	7,263
mdolBWT	0.00013	0	7,958	7,957
concBWT	0.00013	0.00013	0	7,990
colexBWT	0.00012	0.00013	0.00013	0

dataset properties	
no. sequences	2,000
total length	59,612,692
average length	29,085
no. interesting intervals	1863
total length intr.int.s	80,486
fraction pos.s in intr.int.s	0.001
variability	0.148

edit d. / *norm. edit d.*	*edit distance on a subset of 50 sequences*				
	eBWT	dolEBWT	mdolBWT	concBWT	colexBWT
eBWT	0	786	795	801	791
dolEBWT	0.00053	0	98	107	86
mdolBWT	0.00053	0.00007	0	105	112
concBWT	0.00054	0.00007	0.00007	0	114
colexBWT	0.00053	0.00006	0.00008	0.00008	0

no. runs big dataset		
	r	n/r
eBWT	117,628	506.773
dolEBWT	117,410	507.731
mdolBWT	118,870	501.495
concBWT	119,334	499.549
colexBWT	114,287	521.605
optimum	113,930	523.240

6 Conclusion

We presented the first study of the different variants of the Burrows-Wheeler-Transform for string collections. We found that the data structures 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 BWT variants in use are input order dependent, so the same tool can produce different variants 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 or the optimal order of Bentley et al. [4].

References

1 Tooru Akagi, Mitsuru Funakoshi, and Shunsuke Inenaga. Sensitivity of string compressors and repetitiveness measures. *CoRR*, abs/2107.08615, 2021. arXiv:2107.08615.

2 Hideo Bannai, Travis Gagie, and Tomohiro I. Refining the r-index. *Theor. Comput. Sci.*, 812:96–108, 2020. doi:10.1016/j.tcs.2019.08.005.

3 Markus J. Bauer, Anthony J. Cox, and Giovanna Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theor. Comput. Sci.*, 483:134–148, 2013. doi:10.1016/j.tcs.2012.02.002.

4 Jason W. Bentley, Daniel Gibney, and Sharma V. Thankachan. On the complexity of BWT-runs minimization via alphabet reordering. In *Proc. of 28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *LIPIcs*, pages 15:1–15:13, 2020. doi:10.4230/LIPIcs.ESA.2020.15.

5 Paola Bonizzoni, Gianluca Della Vedova, Yuri Pirola, Marco Previtali, and Raffaella Rizzi. Multithread multistring Burrows-Wheeler Transform and Longest Common Prefix array. *J. Comput. Biol.*, 26(9):948–961, 2019. doi:10.1089/cmb.2018.0230.

6 Christina Boucher, Davide Cenzato, Zsuzsanna Lipták, Massimiliano Rossi, and Marinella Sciortino. Computing the original eBWT faster, simpler, and with less memory. In *Proc. of 28th International Symposium on String Processing and Information Retrieval (SPIRE 2021)*, volume 12944 of *LNCS*, pages 129–142, 2021. doi:10.1007/978-3-030-86692-1_11.

7 Christina Boucher, Ondrej Cvacho, Travis Gagie, Jan Holub, Giovanni Manzini, Gonzalo Navarro, and Massimiliano Rossi. PFP compressed suffix trees. In *Proc. of 23rd Symposium on Algorithm Engineering and Experiments (ALENEX 2021)*, pages 60–72. SIAM, 2021. doi:10.1137/1.9781611976472.5.

8 Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big BWTs. *Algorithms Mol. Biol.*, 14(1):13:1–13:15, 2019. doi:10.1186/s13015-019-0148-5.

9 Michael Burrows and David J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

10 Bastien Cazaux and Eric Rivals. Linking BWT and XBW via Aho-Corasick automaton: Applications to run-length encoding. In *Proc. of 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*, volume 128 of *LIPIcs*, pages 24:1–24:20, 2019. doi:10.4230/LIPIcs.CPM.2019.24.

11 Shubham Chandak, Kedar Tatwawadi, Idoia Ochoa, Mikel Hernaez, and Tsachy Weissman. SPRING: a next-generation compressor for FASTQ data. *Bioinform.*, 35(15):2674–2676, 2019. doi:10.1093/bioinformatics/bty1015.

12 Dustin Cobas, Travis Gagie, and Gonzalo Navarro. A fast and small subsampled r-index. In *Proc. of 32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*, volume 191 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CPM.2021.13.

13 Anthony J. Cox, Markus J. Bauer, Tobias Jakobi, and Giovanna Rosone. Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform. *Bioinform.*, 28(11):1415–1419, 2012. doi:10.1093/bioinformatics/bts173.

14 Diego Díaz-Domínguez and Gonzalo Navarro. Efficient construction of the extended BWT from grammar-compressed DNA sequencing reads. *CoRR*, abs/2102.03961, 2021. arXiv:2102.03961.

15 Robert C Edgar. Updating the 97% identity threshold for 16S ribosomal RNA OTUs. *Bioinf.*, 34(14):2371–2375, 2018. doi:10.1093/bioinformatics/bty113.

16 Lavinia Egidi, Felipe A. Louza, Giovanni Manzini, and Guilherme P. Telles. External memory BWT and LCP computation for sequence collections with applications. *Algorithms Mol. Biol.*, 14(1):6:1–6:15, 2019. doi:10.1186/s13015-019-0140-0.

17 Paolo Ferragina, Travis Gagie, and Giovanni Manzini. Lightweight data indexing and compression in external memory. *Algorithmica*, 63(3):707–730, 2012. doi:10.1007/s00453-011-9535-0.

18 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *Proc. of 46th IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pages 184–193, 2005. doi:10.1109/SFCS.2005.69.

19 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1):4:1–4:33, 2009. `doi:10.1145/1613676.1613680`.

20 Johannes Fischer and Florian Kurpicz. sais-lite-lcp. `https://github.com/kurpicz/sais-lite-lcp`. Accessed: 2022-02-05.

21 Travis Gagie, Garance Gourdel, and Giovanni Manzini. Compressing and indexing aligned readsets. In *Proc. of 21st International Workshop on Algorithms in Bioinformatics (WABI 2021)*, volume 201 of *LIPIcs*, pages 13:1–13:21, 2021. `doi:10.4230/LIPIcs.WABI.2021.13`.

22 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in BWT-runs bounded space. In *Proc. of 39th ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1459–1477, 2018. `doi:10.1137/1.9781611975031.96`.

23 Joseph Yossi Gil and David Allen Scott. A bijective string sorting transform. *CoRR*, abs/1201.3077, 2012. `arXiv:1201.3077`.

24 Sara Giuliani, Shunsuke Inenaga, Zsuzsanna Lipták, Nicola Prezza, Marinella Sciortino, and Anna Toffanello. Novel results on the number of runs of the Burrows-Wheeler-Transform. In *Proc. of 47th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2021)*, volume 12607 of *LNCS*, pages 249–262, 2021. `doi:10.1007/978-3-030-67731-2_18`.

25 Allison J. Greaney et al. A SARS-CoV-2 variant elicits an antibody response with a shifted immunodominance hierarchy. *PLOS Pathogens*, 18:1–27, February 2022. `doi:10.1101/2021.10.12.464114`.

26 Ilya Grebnov. libsais. `https://github.com/IlyaGrebnov/libsais`. Accessed: 2022-02-05.

27 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

28 James Holt and Leonard McMillan. Merging of multi-string BWTs with applications. *Bioinform.*, 30(24):3524–3531, 2014. `doi:10.1093/bioinformatics/btu584`.

29 Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler Transform conjecture. In *Proc. of 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS 2020)*, pages 1002–1013, 2020. `doi:10.1109/FOCS46700.2020.00097`.

30 Dominik Köppl, Daiki Hashimoto, Diptarama Hendrian, and Ayumi Shinohara. In-place Bijective Burrows-Wheeler Transforms. In *Proc. of 31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020)*, volume 161 of *LIPIcs*, pages 21:1–21:15, 2020. `doi:10.4230/LIPIcs.CPM.2020.21`.

31 Gregory Kucherov, Lilla Tóthmérész, and Stéphane Vialette. On the combinatorics of suffix arrays. *Inf Process Lett*, 113(22-24):915–920, 2013. `doi:10.1016/j.ipl.2013.09.009`.

32 Alan Kuhnle, Taher Mun, Christina Boucher, Travis Gagie, Ben Langmead, and Giovanni Manzini. Efficient construction of a complete index for pan-genomics read alignment. In *Proc. of 23rd Annual Conference in Computational Molecular Biology (RECOMB 2019)*, volume 11467 of *LNCS*, pages 158–173, 2019. `doi:10.1089/cmb.2019.0309`.

33 Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–359, 2012. `doi:10.1038/nmeth.1923`.

34 Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10:R25, 2009. `doi:10.1186/gb-2009-10-3-r25`.

35 Heng Li. Fast construction of FM-index for long sequence reads. *Bioinform.*, 30(22):3274–3275, 2014. `doi:10.1093/bioinformatics/btu541`.

36 Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010. `doi:10.1093/bioinformatics/btp698`.

37 Felipe A. Louza, Guilherme P. Telles, Simon Gog, Nicola Prezza, and Giovanna Rosone. gsufsort: constructing suffix arrays, LCP arrays and BWTs for string collections. *Algorithms Mol. Biol.*, 15(1):18, 2020. `doi:10.1186/s13015-020-00177-y`.

38 Felipe A. Louza, Guilherme P. Telles, Steve Hoffmann, and Cristina Dutra de Aguiar Ciferri. Generalized enhanced suffix array construction in external memory. *Algorithms Mol. Biol.*, 12(1):26:1–26:16, 2017. doi:10.1186/s13015-017-0117-9.

39 Swapan Mallick et al. The Simons Genome Diversity Project: 300 genomes from 142 diverse populations. *Nature*, 538(7624):201–206, 2016. doi:10.1038/nature18964.

40 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. An extension of the Burrows-Wheeler Transform. *Theor. Comput. Sci.*, 387(3):298–312, 2007. doi:10.1016/j.tcs.2007.07.014.

41 Giovanni Manzini. XBWT tricks. In *Proc. of 23rd International Symposium on String Processing and Information Retrieval (SPIRE 2016)*, volume 9954 of *LNCS*, pages 80–92, 2016. doi:10.1007/978-3-319-46049-9_8.

42 Yuta Mori. libdivsufsort. https://github.com/y-256/libdivsufsort. Accessed: 2022-02-05.

43 Gonzalo Navarro. Indexing highly repetitive string collections, part I: repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2021. doi:10.1145/3434399.

44 Genome 10K Community of Scientists. A proposal to obtain whole-genome sequence for 10,000 vertebrate species. *J Hered.*, 100:659-674, 2009. doi:10.1093/jhered/esp086.

45 Enno Ohlebusch. *Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction*. Oldenbusch Verlag, 2013.

46 Enno Ohlebusch, Stefan Stauß, and Uwe Baier. Trickier XBWT tricks. In *Proc. of 25th International Symposium in String Processing and Information Retrieval (SPIRE 2018)*, volume 11147 of *LNCS*, pages 325–333, 2018. doi:10.1007/978-3-030-00479-8_26.

47 Marco Oliva, Massimiliano Rossi, Jouni Sirén, Giovanni Manzini, Tamer Kahveci, Travis Gagie, and Christina Boucher. Efficiently merging r-indexes. In *Proc. of 31st Data Compression Conference (DCC 2021)*, pages 203–212, 2021. doi:10.1109/DCC50243.2021.00028.

48 Jacopo Pantaleoni. BWT of large string sets. *CoRR*, abs/1410.0562, 2014. arXiv:1410.0562.

49 Simon J. Puglisi and Bella Zhukova. Document retrieval hacks. In *Proc. of 19th International Symposium on Experimental Algorithms (SEA 2021)*, volume 190 of *LIPIcs*, pages 12:1–12:12, 2021. doi:10.4230/LIPIcs.SEA.2021.12.

50 Jouni Sirén. Burrows-Wheeler Transform for terabases. In *Proc. of 26th Data Compression Conference (DCC 2016)*, pages 211–220, 2016. doi:10.1109/DCC.2016.17.

51 Tyler N. Starr et al. Deep mutational scanning of SARS-CoV-2 receptor binding domain reveals constraints on folding and ACE2 binding. *Cell*, 182(5):1295–1310.e20, 2020. doi:10.1016/j.cell.2020.08.012.

52 C. Sun et al. RPAN: rice pan-genome browser for 3000 rice genomes. *Nucleic Acids Res*, 45(2):597–605, 2017. doi:10.1093/nar/gkw958.

53 The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–74, 2015. doi:10.1038/nature15393.

54 The 1001 Genomes Consortium. Epigenomic Diversity in a Global Collection of Arabidopsis thaliana Accessions. *Cell*, 166(2):492–505, 2016. doi:10.1016/j.cell.2016.06.044.

55 C. Turnbull et al. The 100,000 genomes project: bringing whole genome sequencing to the NHS. *Br Med J*, 361, 2018. doi:10.1136/bmj.k1687.

56 Silvie Van den Hoecke, Judith Verhelst, Marnik Vuylsteke, and Xavier Saelens. Analysis of the genetic diversity of influenza A viruses using next-generation DNA sequencing. *BMC Genomics*, 16(1):79, 2015. doi:10.1186/s12864-015-1284-z.

57 Raf Winand 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. *Int. J. of Mol. Sci.*, 21(1):298, 2019. doi:10.3390/ijms21010298.

58 Michael H. Woodworth 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*, 25(5):677–680, 2019. doi:10.1089/mdr.2018.0408.