Block sorting-based transformations on words: beyond the magic BWT*

Raffaele Giancarlo[1], Giovanni Manzini[2], Antonio Restivo[1], Giovanna Rosone[3], and Marinella Sciortino[1]

[1]University of Palermo, Palermo, Italy
[2]University of Eastern Piedmont and IIT-CNR Pisa, Italy
[3]University of Pisa, Pisa, Italy

Abstract

The Burrows-Wheeler Transform (BWT) is a word transformation introduced in 1994 for Data Compression and later results have contributed to make it a fundamental tool for the design of self-indexing compressed data structures. The Alternating Burrows-Wheeler Transform (ABWT) is a more recent transformation, studied in the context of Combinatorics on Words, that works in a similar way, using an alternating lexicographical order instead of the usual one. In this paper we study a more general class of block sorting-based transformations. The transformations in this new class prove to be interesting combinatorial tools that offer new research perspectives. In particular, we show that all the transformations in this class can be used as booster for memoryless compressors and we provide an upper bound on the number of equal-letter runs in their output. Moreover, we introduce the notion of rank-invertibility, a property related to the implementation of an efficient inversion procedure. We show that the BWT and the Alternating BWT are the only rank-invertible transformations in the class we have defined.

*The final authenticated publication is available online at `https://doi.org/10.1007/978-3-319-98654-8_1`. Please, cite the publisher version: Giancarlo R., Manzini G., Restivo A., Rosone G., Sciortino M. (2018) Block Sorting-Based Transformations on Words: Beyond the Magic BWT. In: Hoshi M., Seki S. (eds) Developments in Language Theory. DLT 2018. Lecture Notes in Computer Science, vol 11088. Springer, Cham DOI: `https://doi.org/10.1007/978-3-319-98654-8_1`

1 Introduction

Michael Burrows and David Wheeler introduced in 1994 a reversible word transformation [**?**], denoted by BWT, that turned out to have "myriad virtues". At the time of its introduction in the field of text compression, the Burrows-Wheeler Transform was perceived as a magic box: when used as a preprocessing step it would bring rather weak compressors to be competitive in terms of compression ratio with the best ones available [**?**]. In the years that followed, many studies have shown the effectiveness of BWT and its central role in the field of Data Compression due to the fact that it can be seen as a "booster" of the performance of memoryless compressors [**?**, **?**, **?**]. Moreover, it was shown [**?**] that the BWT can be used to efficiently search for occurrences of patterns inside the original text. Such capabilities of the BWT have originated the field of Compressed Full-text Self-indices [**?**, **?**].

More in detail, the BWT is defined via a sorting in lexicographic order of all the cyclic rotations of the input word. The BWT can be computed in linear time, it produces strings which are provably compressible in terms of the high order entropy of the input, and it can be inverted in linear time by just counting operations (such a property will be formalized in the follows as *rank-invertibility*). Despite its simplicity, the BWT presents some combinatorial properties that have aroused great interest both from the theoretical and applicative points of view [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**].

In the context of Combinatorics on Words, many studies have addressed the characterization of the words that become the most compressible after the application of the BWT [**?**, **?**, **?**, **?**, **?**, **?**]. Recent studies have focused on measuring the "clustering effect" of BWT. That is, a property related to its boosting role as preprocessing of a text compressor [**?**, **?**].

In [**?**], the authors characterize the BWT as the inverse of a known bijection between words and multisets of primitive necklaces [**?**]. From this result, in [**?**] the authors introduce and study the basic properties of the *Alternating* BWT, $ABWT$ from now on. It is a transformation on words analogous to the BWT but the cyclic rotations of the input word are sorted by using the *alternating* lexicographic order instead of the usual lexicographic order. The alternating lexicographic order is defined for infinite words as follows: the first letters are compared with the given alphabetic order, in case of equality the second letters are compared with the opposite order, and so on alternating the two orders for even/odd positions.

We show that the $ABWT$ satisfies most of the properties that make the BWT such a useful transformation. Indeed, the $ABWT$ can be computed in linear time and inverted, still in linear time, by simple counting

operations. However, although these two transformations have some similarities, they represent two very different combinatorial tools. We show that some combinatorial properties or characterizations proved for BWT, change considerably when $ABWT$ is considered.

The existence of the $ABWT$ shows that the classical lexicographic order is not the only order relation that one can use to obtain a reversible transformation. Indeed, lexicographic and alternating lexicographic order are two particular cases of a more general class of order relations considered in [**?**, **?**]. In this paper we explore the class of transformations that use the previously mentioned orders to sort the cyclic rotations of the input word. We prove that, as for the BWT, each transformation of the class can be used as a booster of the performance of a memoryless compressor. Furthermore, for all transformations we show that the number of runs of consecutive equal symbols is at most twice the number of runs of consecutive equal symbols the input word.

Finally, we prove that such word transformations are invertible and we try to establish under which conditions they can be efficiently inverted by using counting and rank operations as for the BWT and $ABWT$. To this end, we introduce the notion of *rank-invertibility* and we prove that BWT and $ABWT$ are the only transformations within this class that are rank-invertible.

2 Preliminaries

Let $\Sigma = \{c_0, c_1, \ldots, c_{\sigma-1}\}$ be a finite ordered alphabet with $c_0 < c_1 < \ldots < c_{\sigma-1}$, where $<$ denotes the standard lexicographic order. We denote by Σ^* the set of words over Σ. Given a finite word $w = w_0 w_1 \cdots w_{n-1} \in \Sigma^*$ with each $w_i \in \Sigma$, the length of w, denoted $|w|$, is equal to n. We use ϵ to denote the empty word. We denote by $|w|_c$ the number of occurrences of a letter c in w. The Parikh vector P_w of a word w is a σ-length array of integers such that for each $c \in \Sigma$, $P_w[c] = |w|_c$. Given a word x and $c \in \Sigma$, we write $\mathsf{rank}_c(x, i)$ to denote the number of occurrences of c in $x[0, i]$.

Given a finite word w, a *factor* of w is written as $w[i, j] = w_i \cdots w_j$ with $0 \leq i \leq j \leq n-1$. A factor of type $w[0, j]$ is called a *prefix*, while a factor of type $w[i, n-1]$ is called a *suffix*. The i-th symbol in w is denoted by $w[i]$. Two words $x, y \in \Sigma^*$ are *conjugate*, if $x = uv$ and $y = vu$, where $u, v \in \Sigma^*$. We also say that x is a *cyclic rotation* of y. A word x is *primitive* if all its cyclic rotations are distinct. Conjugacy between words is an equivalence relation over Σ^*. A word z is called a *circular factor* of x if it is a factor of

some conjugate of x.

Given two words of the same length $x = x_0x_1 \dots x_{s-1}$ and $y = y_0y_1 \dots y_{s-1}$, we write $x \leq_{lex} y$ if and only if $x = y$ or $x_i < y_i$, where i is the smallest index in which the corresponding characters of the two words differ. Analogously, and with the same notation as before, we write $x \leq_{alt} y$ if and only if $x = y$ or (a) i is even and $x_i < y_i$ or (b) i is odd and $x_i > y_i$. Notice that $\leq_{lex}$ is the standard lexicographic order relation on words while $\leq_{alt}$ is the *alternating* lexicographic order relation. Such orders are used in Section 3 to define two different transformations on words.

The *run-length encoding* of a word w, denoted by $\texttt{rle}(w)$, is a sequence of pairs (w_i, l_i) such that $w_iw_{i+1} \cdots w_{i+l_i-1}$ is a maximal run of a letter w_i (i.e., $w_i = w_{i+1} = \cdots = w_{i+l_i-1}$, $w_{i-1} \neq w_i$ and $w_{i+l_i} \neq w_i$), and all such maximal runs are listed in $\texttt{rle}(w)$ in the order they appear in w. We denote by $\rho(w) = |\texttt{rle}(w)|$ i.e., is the number of pairs in w, or equivalently the number of equal-letter runs in w. Moreover we denote by $\rho(w)_{c_i}$ the number of pairs (w_j, l_j) in $\texttt{rle}(w)$ where $w_j = c_i$. Notice that $\rho(w) \leq \rho(w_1) + \rho(w_2) + \cdots + \rho(w_p)$, where $w_1w_2 \cdots w_p = w$ is any partition of w.

The zero-th order empirical entropy of the word w is defined as

$$H_0(w) = -\sum_{i=0}^{\sigma-1} \frac{|w|_{c_i}}{|w|} \log \frac{|w|_{c_i}}{|w|}$$

(all logarithms are taken to the base 2 and we assume $0 \log 0 = 0$). The value $|w|H_0(w)$ is the output size of an ideal compressor that uses $-\log(|w|_{c_i}/|w|)$ bits to encode each occurrence of symbol c_i. This is the minimum size we can achieve using a uniquely decodable code in which a fixed codeword is assigned to each symbol.

For any length-k factor x of w, we denote by x_w the sequence of characters preceding the occurrences of x in w, taken from left to right. If x is not a factor of w the word x_w is empty. The k-th order empirical entropy of w is defined as

$$H_k(w) = \frac{1}{|w|} \sum_{x \in \Sigma^k} |x_w| H_0(x_w).$$

The value $|w|H_k(w)$ is a lower bound to the output size of any compressor that encodes each symbol with a code that only depends on the symbol itself and on the k preceding symbols. Since the use of a longer context helps compression, it is not surprising that for any $k \geq 0$ it is $H_{k+1}(w) \leq H_k(w)$.

3 BWT and Alternating BWT

In this section we describe two different invertible transformations on words based on the lexicographic and alternating lexicographic order respectively. Given a primitive word w of length n in Σ^*, *the Burrows-Wheeler transform* denoted by BWT [**?**] (resp. *the Alternating Burrows-Wheeler transform* denoted by $ABWT$ [**?**]) for w is defined constructively as follows:

1. Create the matrix $M(w)$ of the cyclic rotations of w;
2. Create the matrix $M_{lex}(w)$ by sorting the rows of $M(w)$ according to $\leq_{lex}$ (resp. the matrix $M_{alt}(w)$ by sorting the rows of $M(w)$ according to $\leq_{alt}$);
3. Return as output $bwt(w)$ (resp. $abwt(w)$) the last column L in the matrix $M_{lex}(w)$ (resp. $M_{alt}(w)$) and the integer I giving the position of w in that matrix.

The output of BWT (resp. $ABWT$) is the pair $(bwt(w), I)$ (resp. $(abwt(w), I)$). An example of the above process, together with the corresponding output, is provided in Fig. 1.

$$
\begin{array}{cccccc}
a & b & r & a & c & a \\
b & r & a & c & a & a \\
r & a & c & a & a & b \\
a & c & a & a & b & r \\
c & a & a & b & r & a \\
a & a & b & r & a & c
\end{array}
\qquad M(w)
$$

$$
\begin{array}{cccccc}
 & F & & & & L \\
 & \downarrow & & & & \downarrow \\
 & a & a & b & r & a & c \\
 & a & b & r & a & c & a \\
I \rightarrow & a & c & a & a & b & r \\
 & b & r & a & c & a & a \\
 & c & a & a & b & r & a \\
 & r & a & c & a & a & b
\end{array}
\qquad M_{lex}(w)
$$

$$
\begin{array}{cccccc}
 & F & & & & L \\
 & \downarrow & & & & \downarrow \\
I \rightarrow & a & c & a & a & b & r \\
 & a & b & r & a & c & a \\
 & a & a & b & r & a & c \\
 & b & r & a & c & a & a \\
 & c & a & a & b & r & a \\
 & r & a & c & a & a & b
\end{array}
\qquad M_{alt}(w)
$$

Figure 1: Left: the matrix $M(w)$ of all cyclic rotations of the word $w = acaabr$. Center: the matrix $M_{lex}(w)$; the pair $(caraab, 2)$ is the output $bwt(w)$. Right: the matrix $M_{alt}(w)$; the pair $(racaab, 0)$ is the output of $ABWT(w)$.

Notice that, if two words are conjugate the BWT (resp. $ABWT$) will have the same column L and differ only in I, whose purpose is only to distinguish between the different members of the conjugacy class. However,

I is not necessary in order to recover the matrix M from the last column L. The following proposition states that three well known properties of the BWT holds, in a slightly modified form, for the $ABWT$ as well.

Proposition 1. *Let w be a word and let (L, I) be the output of BWT or $ABWT$ applied to w. The following properties hold:*

1. *Let F denote the first column of $M_{lex}(w)$ (resp. $M_{alt}(w)$), then F is obtained by lexicographically sorting the symbols of L.*
2. *For every i, $0 \leq i < n$, $L[i]$ circularly precedes $F[i]$ in the original word, for both BWT and $ABWT$.*
3. *For each symbol a, and $1 \leq j \leq |w|_a$, the j-th occurrence of a in F corresponds*
   (a) *for BWT, to its j-th occurrence in L*
   (b) *for $ABWT$, to its $(|w|_a - j + 1)$-th occurrence in L.*

Proof. Properties 1, 2 and 3a for the BWT have been established in [**?**]. Properties 1 and 2 for the $ABWT$ are straightforward. To prove property 3b, consider two rows i and j in $M_{alt}(w)$ with $i < j$ starting with the symbol a. Let w_i and w_j are the two conjugates of w in rows i and j of $M_{alt}(w)$. By construction we have $w_i = au$, $w_j = av$ and $w_i \leq_{alt} w_j$. To establish Property 3b we need to show that row w_j cyclically rotated precedes in the $\leq_{alt}$ order row w_i cyclically rotated, in other words we need to show that

$$au \leq_{alt} av \implies va \leq_{alt} ua.$$

To prove the above implication, we notice that if the first position in which au and av differ is odd (resp even) then the first position in which va and ua differ will be in an even (resp. odd) position. The thesis follow by the alternate use of the standard and reverse order in $\leq_{alt}$ (see [**?**] for a different proof of the same property). □ □

It is well known that in the BWT the occurrences of the same symbol appear in columns F and L in the **same** relative order; according to Property 3b, in the $ABWT$ the occurrences in L appear in the **reverse** order than in F. For example, in Fig. 1 (right) we see that the a's of $acaabr$ in the columns F appear in the order 1st, 3rd, and 2nd, while in column L they are in the reverse order: 2nd, 3rd, and 1st.

Note that, although BWT and $ABWT$ are very similarly defined, they are very different combinatorial tools. For instance, BWT allows to characterize a family of words very well known in the field of Combinatorics in Words, the standard Sturmian words [**?**]. These words have several characterizations as, for instance, a special decomposition into palindrome words and an extremal property on the periods of the word that is closely related to Fine and Wilf's theorem [**?**, **?**]. Moreover they also appear as extremal case in the Knuth-Morris-Pratt pattern matching algorithm (see [**?**]). It has been proved [**?**] that, for binary alphabets, standard Sturmian words represent the extremal case of BWT in the sense that the transformation produces a total clustering of all the instances of any character. Thus, in terms of number of runs, $\rho(bwt(w)) = 2$ if and only if w is a conjugate of standard Sturmian words. The same property does not hold for the $ABWT$. For example, for $w = abaababa$ it is $bwt(w) = bbbaaaaa$ and $abwt(w) = ababbaaa$. More in general, one can prove that for every not unary word w having length greater that 2, it is $\rho(abwt(w)) > 2$. More details on the combinatorial study of the equal-letter runs in the output of $ABWT$ will be given in the full paper.

Other combinatorial aspects that distinguish $ABWT$ and BWT have been studied in [**?**]. In the next section we introduce a generalization of the BWT that includes the $ABWT$ as a special case. Hence, all properties established there will held *a fortiori* for the $ABWT$.

4 Generalized BWTs

Given the alphabet Σ of size σ, in the following we denote by Π_Σ the set of $\sigma!$ permutations of the alphabet symbols. Inside Π_Σ we distinguish two important permutations: the identity permutation Id corresponding to the lexicographic order, and the reverse permutation Rev corresponding to the reverse lexicographic order. We consider generalized lexicographic orders introduced in [**?**] (cf. also [**?**]) that, for the purposes of this paper, can be formalized as follows.

Definition 1. *Given a k-tuple $K = (\pi_0, \pi_1, \ldots, \pi_{k-1})$ of elements of Π_Σ we denote by $\preceq_K$ the lexicographic order such that given two words of the same length $x = x_0 x_1 \cdots x_{s-1}$ and $y = y_0 y_1 \cdots y_{s-1}$ it is $x \preceq_K y$ if and only if $x = y$ or $x_i <_i y_i$ where i is the smallest index such that $x_i \neq y_i$, and $<_i$ is the lexicographic order induced by the permutation $\pi_{i \bmod k}$. Without loss of generality, we can assume $\pi_0 = Id$.*

Using the above definition we can define a class of generalized BWTs as follows:

Definition 2. *Given a k-tuple $K = (Id, \pi_1, \ldots, \pi_{k-1})$ of elements of Π_Σ we denote by BWT_K the transformation mapping a primitive word w to the last column L of the matrix $M_K(w)$ containing the cyclic rotations of w sorted according to the lexicographic order $\preceq_K$. The output of BWT_K applied to w is the pair $(bwt_K(w), I)$, where $bwt_K(w)$ is the last column L of the matrix and I is the row of $M_K(w)$ containing the word w.*

Note that for $K = (Id)$, BWT_K is the usual BWT, while for $K = (Id, Rev)$, BWT_K coincides with the $ABWT$ defined in Section 3.

Remark 1. *For most applications it is assumed that the last symbol of the word is a unique end-of-string marker. Under this assumption, lexicographically sorting the suffixes is equivalent to building the word suffix tree*[1]*, which can be done in linear time. In this setting we can also compute $bwt_K(w)$ in linear time: we build w's suffix tree and then we do a depth-first visit in which the children of each node are visited in the order induced by K. In other words, the children of each node v are visited according to the order $\pi_{|v| \bmod k}$ where $|v|$ is the string-depth*[2] *of node v. Since the suffix tree has $O(|w|)$ nodes, for constant alphabet the whole procedure takes linear time.*

To make the transformations BWT_K interesting it is essential they are invertible, which is guaranteed by the following theorem.

Theorem 1. *For every k-tuple $K = (Id, \pi_1, \ldots, \pi_{k-1})$ the transformation BWT_K is invertible in $O(n^3)$ time and $O(n^2)$ space, where $n = |w|$.*

Proof. Let w be a primitive word of length n and let $bwt_K(w) = (L, I)$ be the output of BWT_K applied to w. We first compute $M_K(w)$ and then we obtain w from it. By definition, L is the last column of the matrix $M_K(w)$. Assume that $K_0 = (Id), K_1 = (Id, \pi_1), \ldots, K_{k-1} = (Id, \pi_1, \ldots, \pi_{k-1})$. The first column F of the matrix $M_K(w)$ can be recovered from L by sorting it according to $\preceq_{K_0}$ and, for each $0 \leq i \leq n-1$, we know that $L[i]$ circularly precedes $F[i]$ in w. As in the construction of the matrix $M_{lex}(w)$ for the usual BWT, at each step j we can build the list L_{K_j} of circular factors of w of length $j+1$ sorted by using the $(j+1)$-tuple K_j. The sorted list L_{K_0} is equal to F, so we concatenate, for each $0 \leq i \leq n-1$, $L[i]$ and $L_{K_0}[i]$ and obtain all pairs of consecutive symbols $L[i]F[i]$ in w. Now, by sorting

[1] It is a tree that represents the suffixes of a word in which any path of non-branching nodes is merged into a single edge. The labels of the edges are words of positive variable length [**?**, **?**].

[2] The number of letters in the word obtained by concatenating the labels of the edges in the path from the root of the suffix tree to the node v

this list of pairs using $\preceq_{K_1}$ we obtain the sorted list L_{K_1} of all circular factors of length 2, i.e. the first two columns of $M_K(w)$. In the same way, we concatenate L to each element of L_{K_1} and sort the new list using $\preceq_{K_2}$ obtaining the sorted list L_{K_2} of the circular factors of length 3, i.e. the first three columns of $M_K(w)$. In general, for each $1 \leq j \leq n-1$ we concatenate each symbol $L[i]$ in the last column to each circular factor $L_{K_{j-1}}[i]$ of length j, i.e. we obtain the circular factor $L[i]L_{K_{j-1}}[i]$ of length $j+1$. Then, we sort this list by using $\preceq_{K_j}$ and obtain the new list L_{K_j} of all circular factors of length j that constitute the first j columns of $M_K(w)$. When $j = n-1$, the sorted list $L_{K_{n-1}}$ contains the circular factors of length n that are exactly all the cyclic rotations of w in $M_K(w)$. By construction, the input word w is the row at the position I of $M_K(w)$. The space and time complexities follow from the observation that at each step $0 \leq j \leq n-1$ the list of n words of length $j+1$ is sorted in $O(jn)$ time by iterating a variant of counting sort. □ □

Example 1. *Let $K = ((a,b),(b,a),(b,a),(a,b),(a,b),(b,a))$, $bwt_K(w) = L = babaab$ and $I = 3$. Note that $L_{K_0} = F$. The steps for constructing $M_K(w)$ are the following.*

L	L_{K_0}
b	a
a	a
b	a
a	b
a	b
b	b

⇒

L	L_{K_1}	(a,b)	$\boldsymbol{(b,a)}$
b	a	a	$\boldsymbol{b}$
a	a	a	$\boldsymbol{b}$
b	a	a	$\boldsymbol{a}$
a	b	b	$\boldsymbol{b}$
a	b	b	$\boldsymbol{a}$
b	b	b	$\boldsymbol{a}$

⇒

L	L_{K_1}	(a,b)	$\boldsymbol{(b,a)}$	$\boldsymbol{(b,a)}$
b	$a\ b$	a	$\boldsymbol{b}$	$\boldsymbol{b}$
a	$a\ b$	a	$\boldsymbol{b}$	$\boldsymbol{a}$
b	$a\ a$	a	$\boldsymbol{a}$	$\boldsymbol{b}$
a	$b\ b$	b	$\boldsymbol{b}$	$\boldsymbol{a}$
a	$b\ a$	b	$\boldsymbol{a}$	$\boldsymbol{b}$
b	$b\ a$	b	$\boldsymbol{a}$	$\boldsymbol{a}$

⇒

L	L_{K_2}	(a,b)	$\boldsymbol{(b,a)}$	$\boldsymbol{(b,a)}$	(a,b)
b	$ab\ b$	a	$\boldsymbol{b}$	$\boldsymbol{b}$	a
a	$ab\ a$	a	$\boldsymbol{b}$	$\boldsymbol{a}$	b
b	$aa\ b$	a	$\boldsymbol{a}$	$\boldsymbol{b}$	a
a	$bb\ a$	b	$\boldsymbol{b}$	$\boldsymbol{a}$	a
a	$ba\ b$	b	$\boldsymbol{a}$	$\boldsymbol{b}$	b
b	$ba\ a$	b	$\boldsymbol{a}$	$\boldsymbol{a}$	b

⇒

L	L_{K_3}	(a,b)	$\boldsymbol{(b,a)}$	$\boldsymbol{(b,a)}$	(a,b)	(a,b)
b	$abba$	a	$\boldsymbol{b}$	$\boldsymbol{b}$	a	a
a	$abab$	a	$\boldsymbol{b}$	$\boldsymbol{a}$	b	b
b	$aaba$	a	$\boldsymbol{a}$	$\boldsymbol{b}$	a	b
a	$bbaa$	b	$\boldsymbol{b}$	$\boldsymbol{a}$	a	b
a	$babb$	b	$\boldsymbol{a}$	$\boldsymbol{b}$	b	a
b	$baab$	b	$\boldsymbol{a}$	$\boldsymbol{a}$	b	a

⇒

L	L_{K_4}	(a,b)	$\boldsymbol{(b,a)}$	$\boldsymbol{(b,a)}$	(a,b)	(a,b)	$\boldsymbol{(b,a)}$
b	$abbaa$	a	$\boldsymbol{b}$	$\boldsymbol{b}$	a	a	$\boldsymbol{b}$
a	$ababb$	a	$\boldsymbol{b}$	$\boldsymbol{a}$	b	b	$\boldsymbol{a}$
b	$aabab$	a	$\boldsymbol{a}$	$\boldsymbol{b}$	a	b	$\boldsymbol{b}$
a	$bbaab$	b	$\boldsymbol{b}$	$\boldsymbol{a}$	a	b	$\boldsymbol{a}$
a	$babba$	b	$\boldsymbol{a}$	$\boldsymbol{b}$	b	a	$\boldsymbol{a}$
b	$baaba$	b	$\boldsymbol{a}$	$\boldsymbol{a}$	b	a	$\boldsymbol{b}$

Once we have reconstructed $M_K(w)$, since $I = 3$ we conclude that the original word is $w = bbaaba$.

The following theorem shows that each transformation BWT_K produces a number of equal-letter runs that is at most the double of the number of equal-letter runs of the input word, as analogously proved for BWT [**?**].

Theorem 2. *Given a k-tuple $K = (Id, \pi_1, \ldots, \pi_{k-1})$ and a word w over a finite alphabet Σ, then*

$$\rho(bwt_K(w)) \leq 2\rho(w).$$

Proof. Let $\Sigma = \{c_0, c_1, \ldots, c_{\sigma-1}\}$ with $c_0 < c_1 < \cdots < c_{\sigma-1}$ and let $\texttt{rle}(w) = (a_1, l_1), (a_2, l_2), \ldots, (a_k, l_k)$, where $a_1, a_2, \ldots a_k \in \Sigma$.

When we compute $bwt_K(w)$, the matrix M_K can be split into groups of rows according to their first letter c_i ($i = 0, 1, \ldots, \sigma - 1$). This splitting induces a parsing on $bwt_K(w)$. We denote by u_{c_i} the factor in $bwt_K(w)$ associated to the letter c_i, i.e., all the letters that in the input word precede an occurrence of the letter c_i. Such words u_{c_i}, for $i = 0, \ldots, \sigma - 1$, define a partition of $bwt_K(w)$, i.e. $bwt_K(w) = u_{c_0} u_{c_1} \cdots u_{c_{\sigma-1}}$.

Each factor u_{c_j} contains at most as many letters different from c_j as the number of different equal-letter runs of c_j in w. So, the number of runs contained in u_{a_j} is at most equal to $2\,\rho(w)_{c_j}$. Then,

$$\rho(bwt_K(w)) \leq \sum_{i=0}^{\sigma-1} \rho(u_{c_i}) \leq \sum_{i=0}^{\sigma-1} 2\,\rho(w)_{c_i} = 2 \sum_{i=0}^{\sigma-1} \rho(w)_{c_i} = 2\,\rho(w).$$

□ □

Finally, we study properties of the BWT_K transformations in terms of data compression. Let w^R denote the word w reversed. It is well known [**?**, **?**] that the BWT has the property of reducing the problem of compressing a string w up to its r-th order entropy to the problem of compressing a collection of factors of $bwt(w^R)$ up to their 0-th order entropy. Thus, combining the BWT with a zero order (memoryless) compressor, we can achieve the same high order compression typical of more complex tools such as Lempel-Ziv encoders. In this sense, the BWT can be seen as a *booster* of the performance of simple compressors. The following theorem shows that the same property holds for the transformation BWT_K as well.

Theorem 3. *Let K be a k-tuple and $u = bwt_K(w^R)$, where w^R is the reverse of the word w. For each positive integer r, there exists a factorization of $u = u_1 u_2 \ldots u_m$ such that*

$$H_r(w) = \frac{1}{|u|} \sum_{i=1}^{m} |u_i| H_0(u_i).$$

Proof. For each factor x of w of length $r > 0$, the characters following x in w are the characters preceding x^R in w^R. They are grouped together inside

		BWT		$ABWT$	
	input size	output size	saving %	output size	savings %
chr22.dna	34.553.758	7.927.682	77,06	7.929.910	77,05
etext99	105.277.340	26.559.052	74,77	26.558.378	74,77
howto	39.422.105	9.468.681	75,98	9.470.212	75,98
jdk13c	69.728.899	3.945.465	94,34	3.931.722	94,36
sprot34.dat	109.617.186	21.853.565	80,06	21.821.314	80,09
rctail96	114.711.151	12.644.252	88,96	12.651.821	88,97
rfc	116.421.901	19.084.881	83,61	19.100.971	83,59
w3c2	104.201.579	8.219.970	92,11	8.203.311	92,13

Table 1: Output size and space saving achieved by BWT and $ABWT$ when used within the compression booster paradigm.

$bwt_K(w^R)$ since all the cyclic rotations starting with x^R are consecutive in the matrix $M_K(w^R)$. This means that $bwt_K(w^R)$ contains, as a factor, a permutation of x_w. So, all the factors x of length r define a factorization of u in factors u_i, each of them is a permutation of x_w for some x. The thesis follows from the fact that permuting a word does not change its zeroth order entropy. □ □

We experimentally tested the above theorem by comparing BWT and $ABWT$ as compression tools. To compute the $ABWT$ we have adapted the code of the BCR algorithm [3] [**?**] originally designed to compute the BWT (note that BCR computes the two transformations by sorting the suffixes of the input text and do so appending a unique symbol at the end of the input, as we mentioned in Remark 1). Both BWT and the $ABWT$ have been used within the compression booster framework [**?**] which computes, in linear time, the partition of the BWT (or $ABWT$) that maximizes the compression. To compress the single elements of the partition we use the standard combination of move-to-front followed by arithmetic coding using the tools in the compression boosting library [**?**]. Table 1 reports the output size and the space saving achieved by BWT and $ABWT$ on a corpus of files with different kind of data[4] (see [**?**] for a description of the files content). The results show that the behavior of the two transformations is essentially equivalent in terms of compression.

[3] `https://github.com/giovannarosone/BCR_LCP_GSA`

[4] `https://people.unipmn.it/~manzini/lightweight/corpus/`

5 Rank-invertible transformations

It is well known that the key to efficiently reverse the original BWT is the existence of a easy-to-compute permutation mapping, in the matrix $M_{lex}(w)$, a row index i to the row index $LF(i)$ containing row i right-shifted by one position. This permutation is called LF-mapping since, by Proposition 1, $LF(i)$ is the position in the first column F of $M_{lex}(w)$ corresponding to the i-th entry in column L: in other words, $F[LF(i)]$ is the same symbol in w of $L[i]$. Again, by Proposition 1 we have that $L[LF(i)]$ is the symbol preceding $L[i]$ inside the input word w. Define $LF^0(x) = x$ and $LF^{j+1}(x) = LF(LF^j(x))$. If $bwt(w) = (L, I)$ with $|w| = n$, by construction $L[I] = w_{n-1}$ we can recover w with the formula:

$$w_{n-1-j} = L[LF^j(I)] \tag{1}$$

Note that the inversion formula (1) only depends on Properties 1 and 2 of Proposition (1). Since such properties hold for every generalized transformation BWT_K, (1) provides an inversion formula for every transformation in that class. In other words, inverting a generalized BWT amounts to computing n iterations of the LF-mapping.

By Property 3a in Proposition 1 the LF-mapping for the original BWT can be expressed using the Parikh vector P_L of L and a rank operation over L:

$$LF(i) = \sum_{c \in \Sigma}^{c < L(i)} P_L[c] \; + \; \mathsf{rank}_{L[i]}(L, i) \tag{2}$$

Note that $\sum_{c\in\Sigma}^{c<L(i)} P_L[c]$ is simply the total number of occurrences of symbols smaller than $L[i]$ in L. By Property 3b in Proposition 1, for the $ABWT$ the corresponding formula is:

$$LF(i) = \sum_{c \in \Sigma}^{c \leq L(i)} P_L[c] \; - \; \mathsf{rank}_{L[i]}(L, i) + 1 \tag{3}$$

Using known data structures [**?**], both (2) and (3) can be computed in constant time. Hence, we have established that, thanks to the simple structure of its LF-mapping, also the $ABWT$ can be inverted in linear time. In the full paper, we will prove that, using the LF-map (3), for any pattern p it is possible to compute in $O(|p|)$ time the range of rows of the matrix $M_{alt}(w)$ which are prefixed by p. This implies that the $ABWT$ can be used as a compressed index in the same way as the BWT [**?**].

The above observations suggest us to define the notion of rank-invertibility for the class of BWT_K transformations.

Definition 3. *The transformation BWT_K is rank-invertible, if there exists a function f_K such that for any word w setting $L = bwt_K(w)$ we have*

$$LF(i) = f_K(P_L, L[i], \mathsf{rank}_{L[i]}(L, i))$$

in other words, $LF(i)$ only depends on the Parikh vector P_L of L, the symbol $L[i]$, and the number of occurrences of $L[i]$ in L up to position i.

Note that we pose no limit to the complexity of the function f_K, we only ask that it can be computed using only P_L and the number of occurrences of $L[i]$ inside $L[0, i]$.

We observed that, for $K = (Id, Rev)$, BWT_K coincides with $ABWT$ and it is therefore rank-invertible. The main result of this section is to show that BWT and $ABWT$ are the only rank-invertible transformations in the class BWT_K.

In the proofs of the following statements we distinguish the case of the words on binary alphabets and the words on alphabets with cardinality greater than 2. This depends on the fact that in the binary case the only possible permutations on binary alphabet are the identity and reverse permutation. We first consider the case in which $|K| = 2$. Lemma 1 provides a necessary condition for BWT_K to be rank-invertible for ternary alphabets.

Lemma 1. *Let $\Sigma = \{a, b, c\}$, and $K = (Id, \pi)$ where π a permutation of Σ. If there exist two pairs $t_1 = (x, y)$ and $t_2 = (z, w)$ of symbols of Σ such that*

$$x <_{Id} y, \qquad z <_{Id} w \qquad \text{and} \qquad x <_\pi y, \qquad z >_\pi w,$$

then BWT_K is not rank-invertible.

Proof. Consider for example the case $\pi = (c, a, b)$. Two pairs satisfying the hypothesis are $t_1 = (a, b)$ and $t_2 = (b, c)$ since according to the ordering $<_\pi$ it is

$$a <_\pi b \qquad \text{and} \qquad b >_\pi c.$$

Consider now the two words $s_1 = aabcc$ and $s_2 = abacc$. Both words contain two a's. In the first word the a's are followed respectively by a, b (the symbols in t_1), and in s_2 the a's are followed by b, c (the symbols in t_2).

Let F_1, L_1 (resp. F_2, L_2) denote the first and last column of the matrix M_K associated to $bwt_K(s_1)$ (resp. $bwt_K(s_2)$). By definition, each matrix

is obtained sorting the cyclic rotations of s_1 and s_2 according to the lexicographic order $\prec_K$ where symbols in odd positions are sorted according to the usual alphabetic order, while symbols in even positions are sorted according to the ordering π. We show the two matrices in Fig. 2, where we use subscripts to distinguish the two a's occurrences in s_1 and s_2.

The relative position of the two a's in L_1 is determined by the symbols following them in s_1, namely those in $t_1 = (a, b)$. Since these symbols are in the first column of the cyclic rotations matrix, which is sorted according to the usual alphabetic order, the two a's appear in L_1 in the order a_1, a_2. The same is true for L_2: since the pair t_2 is also sorted, the two a's appear in L_2 in the order a_1, a_2.

The position of the two a's in F_1 is also determined by the symbols following them in s_1; but since these symbols are now in the second column, their relative order is determined by the ordering π. Hence the two a's appear in F_1 in the order a_1, a_2. In F_2 the ordering of the a's is a_2, a_1 since it depends from the π-ordering of t_2's symbols which *by construction* is different than their Id-ordering.

Note that s_1 and s_2 have the same Parikh vector $\langle 2, 1, 2\rangle$. If, by contradiction, BWT_K were rank invertible, the function f_K should give the correct LF-mapping for both s_1 and s_2. This is impossible since for s_1 we should have

$$f_K(\langle 2, 1, 2\rangle, a, 1) = 1, \qquad f_K(\langle 2, 1, 2\rangle, a, 2) = 2,$$

while for s_2 we should have

$$f_K(\langle 2, 1, 2\rangle, a, 1) = 2, \qquad f_K(\langle 2, 1, 2\rangle, a, 2) = 1.$$

In the general case of an arbitrary permutation π satisfying the hypothesis of the lemma the reasoning is the same. Note that such permutations are (a, c, b), (b, a, c), (b, c, a) and (c, a, b). Given the two pairs t_1 and t_2 we build two words s_1 and s_2 with Parikh vector $\langle 2, 1, 2\rangle$ such that in s_1 (resp. s_2) the two occurrences of a are followed by the symbols in t_1 (resp. t_2). We then build the rotation matrices as before, and we find that in both L_1 and L_2 the two a's are in the order a_1, a_2. However, in columns F_1 and F_2 the two a's are not in the same relative order since it depends on the ordering π, and, by construction, such an order is not the same. Reasoning as before, we get that there cannot exist a function f_K giving the correct LF-mapping for both s_1 and s_2. □ □

Theorem 4. *Let $|\Sigma| \geq 2$ and $K = (Id, \pi)$. Then BWT_K is rank-invertible if and only if $\pi = Id$ or $\pi = Rev$.*

$$
\begin{array}{cccccc}
 & F_1 & & & & L_1 \\
 & \downarrow & & & & \downarrow \\
s_1 \rightarrow & a_1 & a_2 & b & c & c \\
 & a_2 & b & c & c & a_1 \\
 & b & c & c & a_1 & a_2 \\
 & c & c & a_1 & a_2 & b \\
 & c & a_1 & a_2 & b & c
\end{array}
\qquad
\begin{array}{cccccc}
 & F_2 & & & & L_2 \\
 & \downarrow & & & & \downarrow \\
 & a_2 & c & c & a_1 & b \\
s_2 \rightarrow & a_1 & b & a_2 & c & c \\
 & b & a_2 & c & c & a_1 \\
 & c & c & a_1 & b & a_2 \\
 & c & a_1 & b & a_2 & c
\end{array}
$$

Figure 2: Cyclic rotation matrices for the words s_1 and s_2. We use subscripts to distinguish the two occurrences of a in each word.

Proof. If $|\Sigma| = 2$ the result is trivial. Let us assume $|\Sigma| \geq 3$. We need to prove that if $\pi \neq Id$ and $\pi \neq Rev$ then BWT_K is not rank-invertible.

Note that any permutation π over the alphabet Σ induces a new ordering on any triplets of symbols in Σ. For example, if $\Sigma = \{a, b, c, d, e, f\}$ the permutation $\pi = (d, e, c, f, a, b)$ induces on the triplet $\{a, b, c\}$ the ordering $\pi_{abc} = (c, a, b)$.

It is easy to prove on induction on the alphabet size that, if $\pi \neq Id$ and $\pi \neq Rev$, then there exists a triplet $\{x, y, z\}$, with $x < y < z$, such that $\pi_{xyz} \neq (x, y, z)$ and $\pi_{xyz} \neq (z, y, x)$. That is, π restricted to $\{x, y, z\}$ is different from the identity and reverse permutation. Without loss of generality we can assume that the triplet is $\{a, b, c\}$.

It is easy to see that for any permutation π_{abc} different from (a, b, c) and (c, b, a) there exist two pairs satisfying the hypothesis of Lemma 1. Hence, we can build two words s_1 and s_2 which show that BWT_K is not rank-invertible. Note that the argument in the proof of Lemma 1 is still valid if we add to s_1 and s_2 the same number of occurrences of symbols in Σ different from a, b, c so that s_1 and s_2 are effectively over an alphabet of size $|\Sigma|$. □ □

Theorem 4 establishes which BWT_K transformations are rank-invertible when $|K| = 2$. To study the general case $|K| > 2$, we start by establishing a simple corollary.

Corollary 1. *Let $|\Sigma| \geq 3$ and $K = (Id, \pi, \pi_2, \ldots, \pi_{k-1})$. If $\pi \neq Id$ and $\pi \neq Rev$ then BWT_K is not rank-invertible.*

Proof. We reason as in the proof of Theorem 4, observing that the presence of the permutations $\pi_2, \ldots, \pi_{k-1}$ has no influence on the proof since the row ordering is determined by the first two symbols of each rotation. □ □

The following three lemmas establish necessary conditions on the structure of the tuple K for BWT_K to be rank-invertible. In particular, the following lemma shows that BWT_K is not rank-invertible if K contains anywhere a triplet (Id, Id, π) with $\pi \neq Id$.

Lemma 2. *Let $|\Sigma| \geq 2$ and $K = (Id, \pi_1, \ldots, \pi_{i-1}, Id, Id, \pi, \pi_{i+3}, \ldots, \pi_{k-1})$, $i \geq 0$, with $\pi \neq Id$. Then BWT_K is not rank-invertible.*

Proof. Note that when $i = 0$ the k-tuple K starts with the triplet (Id, Id, π).

We first analyze the case $|\Sigma| = 2$, this means that $\pi = Rev$. Let us consider the words $s_1 = a_1 b^i a_2 b^{i+1} bb$ and $s_2 = a_1 b^{i+1} a_2 b^{i+1} b$ where we use subscripts to distinguish the two different occurrences of the symbol a. It is easy to see that, in the cyclic rotations matrix for s_1, a_1 precedes a_2 in both the first and the last column. Hence if BWT_K were rank-invertible we should have

$$f_K(\langle 2, 2i+3 \rangle, a, 1) = 1, \qquad f_K(\langle 2, 2i+3 \rangle, a, 2) = 2.$$

At the same time, in the cyclic rotations matrix for s_2, a_1 precedes a_2 in the last columns, but in the first column a_2 precedes a_1 since the two rotations prefixed by a differ in the third column and $b <_{Rev} a$. Hence we should have

$$f_K(\langle 2, 2i+3 \rangle, a, 1) = 2, \qquad f_K(\langle 2, 2i+3 \rangle, a, 2) = 1$$

hence BWT_K cannot be rank-invertible.

Let us consider the case $|\Sigma| \geq 3$. Since $\pi \neq Id$ there are two symbols, say b and c, such that their relative order according to π is reversed, that is, $b < c$ and $c <_\pi b$. Consider now the words $s_1 = a_1 c^i b a_2 c^i ccc$ and $s_2 = a_1 c^{i+1} b a_2 c^{i+1} c$ where we use subscripts to distinguish the two different occurrences of the symbol a. It is immediate to see that, in the cyclic rotations matrix for s_1, a_1 precedes a_2 in both the first and the last column. Hence if BWT_K were rank-invertible we should have

$$f_K(\langle 2, 1, 2i+3 \rangle, a, 1) = 1, \qquad f_K(\langle 2, 1, 2i+3 \rangle, a, 2) = 2.$$

At the same time, in the cyclic rotations matrix for s_2, a_1 precedes a_2 in the last columns, but in the first column a_2 precedes a_1 since the two rotations prefixed by a differ in the $(i+3)$-th column and $c <_\pi b$. Hence we should have

$$f_K(\langle 2, 1, 2i+3 \rangle, a, 1) = 2, \qquad f_K(\langle 2, 1, 2i+3 \rangle, a, 2) = 1$$

hence BWT_K cannot be rank-invertible. □ □

The following lemma shows that BWT_K is not rank-invertible if K contains anywhere a triplet (Id, Rev, π), with $\pi \neq Id$.

Lemma 3. *Let $|\Sigma| \geq 2$ and $K = (Id, \pi_1, \ldots, \pi_{i-1}, Id, Rev, \pi, \pi_{i+3}, \ldots, \pi_{k-1})$, $i \geq 0$, with $\pi \neq Id$. Then BWT_K is not rank-invertible.*

Proof. As in the proof of Lemma 2, we can consider the words $s_1 = a_1 b^i a_2 b^{i+1} bb$ and $s_2 = a_1 b^{i+1} a_2 b^{i+1} b$ in case of binary alphabet, and the words $s_1 = a_1 c^i b a_2 c^i ccc$ and $s_2 = a_1 c^{i+1} b a_2 c^{i+1} c$ in the general case by assuming that there are two symbols, say b and c, such that their relative order according to π is reversed, that is, $b < c$ and $c <_\pi b$. Recall that we use subscripts to distinguish the two different occurrences of the symbol a. In the cyclic rotations matrix for s_1, in the first column a_2 precedes a_1 while in the last column a_1 precedes a_2. At the same time, in both the first and the last column of the cyclic rotations matrix for s_2, a_2 precedes a_1. Reasoning as in the proof of Lemma 2 we get that BWT_K cannot be rank-invertible. □ □

The following lemma shows that BWT_K is not rank-invertible if K contains anywhere a triplet (Rev, Id, π), with $\pi \neq Rev$.

Lemma 4. *Let $|\Sigma| \geq 2$ and $K = (Id, \pi_1, \ldots, \pi_{i-1}, Rev, Id, \pi, \pi_{i+3}, \ldots, \pi_{k-1})$, $i \geq 0$, with $\pi \neq Rev$. Then BWT_K is not rank-invertible.*

Proof. We reason as in the proof of Lemma 3 considering again the words $s_1 = ab^i ab^{i+1} bb$ and $s_2 = ab^{i+1} ab^{i+1} b$ in case of binary alphabet and the words $s_1 = ac^i bac^i ccc$ and $s_2 = ac^{i+1} bac^{i+1} c$ in the general case. □ □

We are now ready to establish the main result of this section.

Theorem 5. *If $|\Sigma| \geq 2$, BWT and ABWT are the only transformations BWT_K which are rank invertible.*

Proof. For $|K| = 2$, the result follows by Theorem 4. Consider now $K = (Id, \pi_1, \ldots, \pi_{k-1})$ with $k > 2$ and suppose that BWT_K is rank invertible. Both in case of binary alphabet and in the general case, by using Corollary 1, we must have $\pi_1 = Id$ or $\pi_1 = Rev$. Let us consider the case $\pi_1 = Id$. If $BWT_K \neq BWT$ then the k-tuple K must contain the triplet (Id, Id, π) with $\pi \neq Id$. This fact contradicts Lemma 2. Let us consider now that $\pi_1 = Rev$. By Lemma 3 $\pi_2 = Id$. We have therefore established that K has the form $K = (Id, Rev, Id, \pi_3, \ldots, \pi_{k-1})$. By using Lemma 4 $\pi_3 = Rev$. By iterating the same reasoning we can conclude that BWT_k coincides with $ABWT$ and the theorem is proved. □ □

6 Conclusions

In this paper we have considered a class of word transformations BWT_K defined in terms of a k-tuple K of alphabet orderings. This class includes both the original BWT and the Alternating BWT proposed in [**?**]. We have proved that each transformation has combinatorial properties useful to perform a role of a booster of a memoryless compressor in the same way as the BWT. We have also introduced the notion of rank-invertibility to explore which transformations can be efficiently inverted. We have proved that $ABWT$ and BWT are the only transformations in the class that are rank-invertible. Our conclusion is that the $ABWT$, although it is a combinatorial tool with very peculiar properties, can be considered a good candidate to replace the BWT in several contexts. We are therefore interested in further studying the combinatorial properties of $ABWT$, with the main purpose of finding new characterizations of word families for which $ABWT$ assumes a significant behavior, for instance in terms of the number of consecutive equal-letter runs produced. More in general, since the compressibility of a text is related with the number of equal-letter runs, it would be interesting to study how the bounds on the number of equal-letter runs varies according to the order taken into consideration.

From an algorithmic point of view, we are interested to explore new block sorting-based transformations in order to investigate the combinatorial properties that not only guarantee good compression performance, but also support efficient search operations in compressed indexing data structures.

Acknowledgements

Thanks to Published source. R.G. is partially supported by INdAM - GNCS Project 2018 "Analysis and Processing of Big Data based on Graph Models". G.R. and M.S. are partially supported supported by the project Italian MIUR-SIR CMACBioSeq ("Combinatorial methods for analysis and compression of biological sequences") grant n. RBSI146R5L.

References

[1] M. J. Bauer, A. J. Cox, and G. Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theor. Comput. Sci.*, 483(0):134 – 148, 2013.

[2] M. Burrows and D. J. Wheeler. A block sorting data compression algorithm. Technical report, DIGITAL System Research Center, 1994.

[3] A. J. Cox, T. Jakobi, G. Rosone, and O. B. Schulz-Trieglaff. Comparing DNA sequence collections by direct comparison of compressed text indexes. In *12th International Workshop on Algorithms in Bioinformatics, WABI 2012*, volume 7534 LNBI of *LNCS*, pages 214–224. Springer, 2012.

[4] M. Crochemore, J. Désarménien, and D. Perrin. A note on the Burrows-Wheeler transformation. *Theor. Comput. Sci.*, 332:567–572, 2005.

[5] A. de Luca. Combinatorics of standard sturmian words. In *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht*, volume 1261 of *LNCS*, pages 249–267. Springer, 1997.

[6] A. de Luca and F. Mignosi. Some combinatorial properties of sturmian words. *Theor. Comput. Sci.*, 136(2):361–385, 1994.

[7] F. Dolce, A. Restivo, and C. Reutenauer. On generalized Lyndon words. Submitted.

[8] P. Fenwick. The Burrows-Wheeler transform for block sorting text compression: Principles and improvements. *Comput. J.*, 39(9):731–740, 1996.

[9] S. Ferenczi and L. Q. Zamboni. Clustering Words and Interval Exchanges. *Journal of Integer Sequences*, 16(2):Article 13.2.1, 2013.

[10] P. Ferragina, T. Gagie, and G. Manzini. Lightweight data indexing and compression in external memory. In *LATIN 2010: Theoretical Informatics, 9th Latin American Symposium, 2010. Proceedings*, volume 6034 of *LNCS*, pages 697–710. Springer, 2010.

[11] P. Ferragina, R. Giancarlo, and G. Manzini. The Engineering of a Compression Boosting Library: Theory vs Practice in BWT Compression. In *Algorithms - ESA 2006, 14th Annual European Symposium, 2006, Proceedings*, volume 4168 of *LNCS*, pages 756–767. Springer, 2006.

[12] P. Ferragina, R. Giancarlo, G. Manzini, and M. Sciortino. Boosting textual compression in optimal linear time. *J. ACM*, 52(4):688–713, 2005.

[13] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *FOCS 2000*, pages 390–398. IEEE Computer Society, 2000.

[14] P. Ferragina and G. Manzini. Indexing compressed text. *J. ACM*, 52:552–581, 2005.

[15] T. Gagie, G. Manzini, and J. Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theor. Comput. Sci.*, 698:67–78, 2017.

[16] I. M. Gessel, A. Restivo, and C. Reutenauer. A bijection between words and multisets of necklaces. *Eur. J. Combin.*, 33(7):1537 – 1546, 2012.

[17] I. M. Gessel and C. Reutenauer. Counting permutations with given cycle structure and descent set. *J. Comb. Theory A*, 64(2):189–215, 1993.

[18] R. Giancarlo, A. Restivo, and M. Sciortino. From first principles to the Burrows and Wheeler transform and beyond, via combinatorial optimization. *Theor. Comput. Sci.*, 387:236 – 248, 2007.

[19] D. Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

[20] K. Kimura and A. Koike. Ultrafast SNP analysis using the Burrows-Wheeler transform of short-read data. *Bioinformatics*, 31(10):1577–1583, 2015.

[21] D. Knuth, J. Morris, and V. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[22] H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[23] M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002.

[24] M. Lothaire. *Applied Combinatorics on Words (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, New York, NY, USA, 2005.

[25] Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, 2015.

[26] S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. An extension of the Burrows-Wheeler Transform. *Theor. Comput. Sci.*, 387(3):298–312, 2007.

[27] S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. A new combinatorial approach to sequence comparison. *Theory Comput. Syst.*, 42(3):411–429, 2008.

[28] S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. Burrows-Wheeler Transform and Run-Length Enconding. In *Combinatorics on Words - 11th International Conference, WORDS 2017. Proceedings*, volume 10432 of *LNCS*, pages 228–239. Springer, 2017.

[29] S. Mantaci, A. Restivo, G. Rosone, M. Sciortino, and L. Versari. Measuring the clustering effect of BWT via RLE. *Theor. Comput. Sci.*, 698:79–87, 2017.

[30] S. Mantaci, A. Restivo, and M. Sciortino. Burrows-Wheeler transform and Sturmian words. *Information Processing Letters*, 86:241–246, 2003.

[31] S. Mantaci, A. Restivo, and M. Sciortino. Distance measures for biological sequences: Some recent approaches. *Int. J. Approx. Reasoning*, 47(1):109–124, 2008.

[32] G. Manzini. An analysis of the Burrows-Wheeler transform. *J. ACM*, 48(3):407–430, 2001.

[33] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007.

[34] Gonzalo Navarro. *Compact Data Structures - A Practical Approach.* Cambridge University Press, 2016.

[35] I. Pak and A. Redlich. Long cycles in abc-permutations. *Functional Analysis and Other Mathematics*, 2:87–92, 2008.

[36] N. Prezza, N. Pisanti, M. Sciortino, and G. Rosone. Detecting mutations by eBWT. In *18th International Workshop on Algorithms in Bioinformatics, WABI 2018*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. To appear.

[37] A. Restivo and G. Rosone. Burrows-Wheeler transform and palindromic richness. *Theor. Comput. Sci.*, 410(30-32):3018 – 3026, 2009.

[38] A. Restivo and G. Rosone. Balancing and clustering of words in the Burrows-Wheeler transform. *Theor. Comput. Sci.*, 412(27):3019 – 3032, 2011.

[39] C. Reutenauer. Mots de Lyndon généralisés 54. *Sém. Lothar. Combin.*, pages 16, B54h, 2006.

[40] G. Rosone and M. Sciortino. The Burrows-Wheeler Transform between Data Compression and Combinatorics on Words. In *The Nature of Computation. Logic, Algorithms, Applications - 9th Conference on Computability in Europe, CiE 2013. Proceedings*, volume 7921 of *LNCS*, pages 353–364. Springer, 2013.

[41] J. Simpson and S. J. Puglisi. Words with simple Burrows-Wheeler transforms. *Electronic Journal of Combinatorics*, 15, article R83, 2008.

[42] L. Yang, X. Zhang, and T. Wang. The Burrows-Wheeler similarity distribution between biological sequences based on Burrows-Wheeler transform. *Journal of Theoretical Biology*, 262(4):742–749, 2010.