BWT and Combinatorics on Words

Gabriele Fici
Department of Mathematics and Informatics, University of Palermo, Italy

Sabrina Mantaci
Department of Mathematics and Informatics, University of Palermo, Italy

Antonio Restivo
Department of Mathematics and Informatics, University of Palermo, Italy

Giuseppe Romana[1]
Department of Mathematics and Informatics, University of Palermo, Italy

Giovanna Rosone
Department of Computer Science, University of Pisa, Italy

Marinella Sciortino
Department of Mathematics and Informatics, University of Palermo, Italy

Abstract

The Burrows-Wheeler Transform (BWT) is a reversible transformation on words (strings) introduced in 1994 in the context of data compression, which is a permutation of the characters in the word. Its clustering effect, i.e., the remarkable property of grouping identical characters (BWT runs) when they share common contexts, has made it a powerful tool for boosting compression performances and enabling efficient pattern searching in highly repetitive string collections.

In this chapter, we analyze the Burrows–Wheeler transform under the combinatorial point of view, and we survey known properties and connections with different aspects of combinatorics on words. In particular, we focus on the properties of words in relation to the number of their BWT runs. The value r, which counts the number of BWT runs, impacts both compression performance and indexing efficiency, and is considered a measure to evaluate the above-mentioned clustering effect and, consequently, the repetitiveness of a word. We give an overview of the results relating r to other combinatorial repetitiveness measures related to the factor complexity. The chapter also explores extremal cases of the clustering effect. Finally, some results on the sensitivity of the measure r are considered, where the effects of combinatorial operations are studied, such as reversal, edits, and the application of morphisms.

2012 ACM Subject Classification Theory of computation → Data compression; Mathematics of computing → Combinatorics; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Burrows–Wheeler Transform, Combinatorics on Words, Clustering Effect, BWT Runs

Digital Object Identifier 10.4230/OASIcs.Manzini.2025.1

Funding *Gabriele Fici*: Supported by MUR project PRIN 2022 APML – 20229BCXNW, funded by the European Union – Mission 4 "Education and Research" C2 - Investment 1.1. CUP Master_B53D23012910006.
Sabrina Mantaci: Supported by Project "ACoMPA" (CUP B73C24001050001) funded by the NextGeneration EU programme PNRR MUR M4 C2 Inv. 1.5 – Project ECS00000017 Tuscany Health Ecosystem (Spoke 6), CUP Master B63C22000680007.
Giuseppe Romana: Supported by the MUR PRIN Project "PINC, Pangenome INformatiCs: from Theory to Applications" (Grant No. 2022YRB97K), funded by Next Generation EU PNRR M4 C2, Inv. 1.1. and by the INdAM - GNCS Project CUP_E53C24001950001.

[1] Corresponding author.

Giovanna Rosone: Partially supported by PNRR - M4C2 - Investimento 1.5, Ecosistema dell'Innovazione ECS00000017 - "THE - Tuscany Health Ecosystem" - Spoke 6 "Precision medicine & personalized healthcare", funded by the European Commission under the NextGeneration EU program, by MUR PRIN 2022 YRB97K "PINC" funded by Next Generation EU PNRR M4 C2 Inv. 1.1, and by the INdAM - GNCS Project CUP_E53C24001950001.
Marinella Sciortino: Supported by Project "ACoMPA" (CUP B73C24001050001) funded by the NextGeneration EU programme PNRR MUR M4 C2 Inv. 1.5 – Project ECS00000017 Tuscany Health Ecosystem (Spoke 6), CUP Master B63C22000680007, by MUR PRIN 2022 YRB97K "PINC" funded by Next Generation EU PNRR M4 C2 Inv. 1.1, and by the INdAM - GNCS Project CUP_E53C24001950001.

1 Introduction

The Burrows-Wheeler transform (BWT) is a fundamental algorithm introduced in 1994 by Michael Burrows and David Wheeler in the field of data compression [12]. Although more than 30 years have passed since its introduction, the BWT remains at the core of many tools used in bioinformatics for analyzing biological sequences (see [40, 45, 41]) and, in particular, constitutes the backbone of popular efficient compressed indexing schemes [21, 25].

The *clustering effect* of the BWT, i.e., the remarkable property of grouping identical characters when they share common contexts, has made it a powerful tool for boosting compression performances [54, 20] and enabling very efficient pattern searching in highly repetitive sequence collections [25, 38].

The BWT is a reversible transformation that lexicographically sorts all cyclic rotations of the input text w and produces a permutation of the characters of w, denoted by $bwt(w)$, by taking the last character of these sorted rotations. From a combinatorial point of view, its simplicity combined with its effectiveness has made the BWT a fascinating study object [66]. The study of the combinatorial properties of the BWT not only helps in understanding its theoretical foundations but also in exploring its potential as a new combinatorial tool for analyzing the structural properties of words. A pioneering work in this area was done by Mantaci et al. [53], who showed that the BWT can be used to characterize standard Sturmian words, an infinite family of binary words widely studied in combinatorics on words [48].

This chapter aims to highlight strong connections between the BWT and combinatorics on words. Most combinatorial studies on the BWT have focused on the properties of words produced by the transformation and their relation to its clustering effect. Here, we provide an overview of these results, showing the role of the BWT as an effective new tool for assessing the complexity of combinatorial objects.

The effect of the BWT clustering on a word w can be evaluated using the measure $r(w)$, which denotes the number of clusters (also called BWT runs) produced by the BWT applied to w, i.e., the number of runs of consecutive identical characters in $bwt(w)$. The value $r(w)$ is also used as a measure of the repetitiveness of the word w. In fact, the more repetitive a word is, the more likely it is that $bwt(w)$ contains long runs of consecutive identical characters. This property makes w highly compressible using BWT-based compression algorithms. Furthermore, the performance in terms of both space and time of text compressors and compressed indexing data structures applied on a text w can be evaluated by using $r(w)$ [24, 25]. Exploring the relationships between r and the combinatorial properties of the words, as well as its sensitivity to some combinatorial transformations on words, can also have implications for the analysis and definition of BWT-based compressed indexing schemes.

In Section 3, we provide an overview of the main combinatorial properties of the BWT and relate them to some well-known permutations that can be associated with the words produced by the BWT as output (also called BWT images) [46].

Section 4 is focused on results that relate the number of BWT runs to the number of maximal runs in the input word [52, 51]. On the one hand, this provides a measure of the effectiveness of the BWT; on the other hand, it allows a deeper investigation into the combinatorial properties of words for which the BWT is more effective. In particular, in [11, 23] this relationship has been studied for morphic words, which are infinite families of binary words obtained by iteratively applying a morphism starting from a single character. Morphisms are a well-known combinatorial tool for constructing highly repetitive families of words. Other interesting researches connect the measure r with other repetitiveness measures used in combinatorics on words, such as *factor complexity*, which counts the number of distinct factors that appear in a word for each length. Periodic words, for example, which are obtained by iterating the same factor, have a constant factor complexity. It is possible to define upper and lower bounds for the measure r as a function of the measure δ, the supremum of the normalized factor complexity [33].

Much attention has been given to the combinatorial characterization of the so-called *clustering words* under the BWT. This is the family of words for which r assumes its minimum value. These words are the most compressible using BWT-based compressors. This topic is the focus of Section 5. The case of binary words has been studied in [53]. For larger alphabets, the relationship between clustering words and the trajectories of interval exchange transformations has been investigated in [19]. The combinatorial properties of a subclass of these words, called perfectly clustering words, for which the BWT produces sequences of lexicographically decreasing maximal runs, have been extensively studied. Only recently, the first combinatorial characterization of perfectly clustering words has been established [43]. Similarly, it can be interesting to explore the combinatorial properties of words for which the number of BWT runs is maximal. However, no known combinatorial characterization of such words exists. De Brujin words, on the other hand, are examples of infinite families of words for which the number of BWT runs is close to the maximum [32]. This result has been used in [47] to define a BWT-based algorithm to generate random de Brujin words.

In Section 6, we present results showing how the number of BWT runs is affected by some combinatorial operations on words. Sensitivity analysis is crucial for understanding the stability of compression-based repetitiveness measures [1]. It has been shown that, unlike other repetitiveness measures, r is not invariant under reversal, with known upper and lower bounds on the ratio between r of a word and of its reverse [29]. The impact of edit operations, such as insertion, deletion, and substitution, is also analyzed, establishing both upper and lower bounds for the growth in the number of BWT runs after edits [1, 30, 31]. Additionally, the section explores results on the behavior of r under morphisms, proving that injective binary morphisms always increase or preserve r, with the well-known family of *Sturmian morphisms* being the only ones that preserve it [22].

2 Preliminaries

Let $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$, with $a_1 < a_2 < \ldots < a_\sigma$, be a finite ordered alphabet. The order on Σ induces the standard lexicographic order $\leq_{lex}$ on the set Σ^* of words over Σ.

Given a word $w = w_1 w_2 \cdots w_n \in \Sigma^*$ with each $w_i \in \Sigma$, the length n of w is denoted by $|w|$. We let ε denote the empty word, the only word of length 0. The set of words of length n over Σ is denoted by Σ^n. With $alph(w)$ we denote the subset of Σ of letters occurring in w.

We let $|w|_{a_i}$ denote the number of occurrences in w of the letter a_i. The *Parikh vector* P_w of a word w is the vector $[n_1, \ldots, n_\sigma]$, where n_i is the number of occurrences of a_i in w, i.e, for each $a_i \in \Sigma$, $P_w[i] = |w|_{a_i}$.

A word v is said to be a *factor* of a word w if it occurs as a subword in w, i.e., $v = w[i, j] = w_i \cdots w_j$, for some $1 \leq i \leq j \leq n$. The set of all factors of a word w is denoted by $Fact(w)$. A factor of type $w[1, j]$ is called a *prefix*, while a factor of type $w[i, n]$ is called a *suffix*. The longest proper factor of w that is both a prefix and a suffix is called the *longest border* of w. Let $Fact(w)$ be the set of factors of w. A factor u of w is *right* (resp. *left*) *special* in w if there exist $a, b \in \Sigma$, $a \neq b$, such that $ua, ub \in Fact(w)$ (resp. $au, bu \in Fact(w)$). A factor u of w is *bispecial* if it is both right and left special.

Two words $x, y \in \Sigma^*$ are *conjugates* (or x is a *cyclic rotation* of y), if $x = uv$ and $y = vu$, for some $u, v \in \Sigma^*$. Conjugacy between words is an equivalence relation over Σ^*. A word z is called a *circular factor* of w if it is a factor of some conjugate of w. A word w is *primitive* if $w \neq u^h$ for every nonempty word u and integer $h > 1$. The cyclic rotations of a primitive word are all distinct. The *(primitive) root* of a word w, denoted by $\sqrt{w}$, is the shortest word such that $w = (\sqrt{w})^h$, for some integer $h \geq 1$. So, a word is primitive if and only if it coincides with its root. A *Lyndon word* is a primitive word that is smaller than all of its proper conjugates (or, equivalently, suffixes).

The *reversal* of the word $w = w_1 \cdots w_n$ is the word $\tilde{w} = w_n \cdots w_1$. If $w = \tilde{w}$, then w is called a *palindrome*. The empty word is also assumed to be a palindrome. A word w has the *two-palindrome property* if it can be written as uv where u and v are palindromes. This is equivalent to the fact that w is *symmetric*, i.e., w is a conjugate of its reversal $\tilde{w}$.

A *maximal run* (or *cluster*) u in a word w is the factor $u = w[i, j]$ (with $i < j$) where $w_i = w_{i+1} = \cdots = w_j$, and $w_{i-1} \neq w_i$, $w_j \neq w_{j+1}$ if w_{i-1} and/or w_{j+1} exist. Informally, a maximal run is a maximal factor of consecutive occurrences of the same letter.

Morphisms

Given two finite alphabets Σ and Γ, a *morphism* μ is a map from Σ^* to Γ^* such that $\mu(uv) = \mu(u)\mu(v)$ for all words $u, v \in \Sigma^*$. Therefore, a morphism μ can be defined by specifying its action on the letters of Σ. A morphism μ is *prolongable* on a letter $a \in \Sigma$ if a is a prefix of $\mu(a)$. Very well-known examples of morphisms on the binary alphabet $\{a, b\}$ are the *Fibonacci morphism* φ, which maps a to ab and b to a, and the *Thue–Morse morphism* τ, which maps a to ab and b to ba. Fibonacci and Thue–Morse morphisms can be used to define two well-known infinite sequences, obtained by recursively applying the morphisms starting from a: the *Fibonacci infinite word* $f = abaababaabaab\cdots$ and the *Thue–Morse word* $t = abbabaabbaababba\cdots$, respectively.

Sturmian and Episturmian Words

An infinite word over an alphabet Σ is an infinite sequence of letters from Σ. An infinite word w is *ultimately periodic* if there exist words u and v such that $w = uvvvv\cdots$; otherwise it is *aperiodic*.

An important class of aperiodic infinite words is the class of *Sturmian words*. Sturmian words appear in many contexts and have many equivalent definitions, witnessing the richness of their combinatorial properties.

A (finite or infinite) word is *balanced* if, for any pair of its factors u, v with $|u| = |v|$, and for every $a \in \Sigma$, $|u|_a - |v|_a \leq 1$. A finite word is *circularly balanced* if all its conjugates are balanced. *Sturmian words* are infinite words over a two-letter alphabet that are balanced and aperiodic. Note that, for instance, the Fibonacci infinite word f is balanced and aperiodic,

therefore, it is Sturmian, whereas the Thue–Morse word t is not, since it is aperiodic but not balanced (it contains the factors aa and bb). Equivalently, an infinite binary word is Sturmian if it has exactly one left special factor of length n for each $n \geq 0$ [48].

Within the family of Sturmian words, the *standard* (or *characteristic*) Sturmian words can be considered as representative, in the sense that for every Sturmian word there is exactly one standard Sturmian word with the same set of factors. A Sturmian word is called standard if every prefix is left special (and is therefore the only left special factor of that length). The following equivalent definition of standard Sturmian word highlights its structure. Let $d_1, d_2, \ldots, d_n, \ldots$ be a sequence of integers, with $d_1 \geq 0$ and $d_i > 0$ for $i > 1$. Consider the following sequence of words $\{s_n\}_{n\geq 0} : s_0 = b, s_1 = a$, and $s_{n+1} = s_n^{d_n} s_{n-1}$ for $n \geq 1$. The sequence $\{s_n\}_{n\geq 0}$ converges to a standard Sturmian word. The sequence $\{s_n\}_{n\geq 0}$ is called the *standard sequence* of s and $(d_1, d_2, \ldots, d_n, \ldots)$ is the *directive sequence* of s. Each finite binary word s_n appearing in some standard sequence is called *standard word* of *order* n. For example, the Fibonacci infinite word is a standard Sturmian word, with directive sequence $d_i = 1$ for every i, and the standard sequence of the Fibonacci infinite word is the sequence of *Fibonacci words*, which can also be defined by: $f_0 = b$, $f_1 = a$, and $f_n = f_{n-1}f_{n-2}$ for every $n \geq 2$.

A binary word is a *Christoffel word* if it is the lexicographically minimal (Lyndon) conjugate of a standard word (note that standard words are always primitive).

Infinite Sturmian words have many combinatorial characterizations. There exist several generalizations of Sturmian words to alphabets of cardinality larger than 2. Unfortunately, these generalizations are no longer equivalent, leading to distinct families of infinite words. Here, we consider some of these definitions, which generalize, from a specific perspective, the notion of the Sturmian word.

An infinite word w over an alphabet Σ of cardinality at least 2 is *episturmian* [18] if $Fact(w)$ is closed under reversal and w has at most one left special factor (or, equivalently, at most one right special factor) of each length. Sturmian words are binary episturmian words. However, episturmian words may not be aperiodic, and may not be balanced.

An infinite word w is *standard episturmian*, or *epistandard*, if all of its left special factors appear as prefixes in w. The *Rauzy rules*, described below, allow one to recursively construct, similarly to standard Sturmian words, any epistandard word.

Let $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$ be an alphabet of cardinality $\sigma \geq 2$. The sequence $(R_n)_{n\geq 0}$ of Rauzy tuples $R_n = (A_n^{(1)}, A_n^{(2)}, \ldots A_n^{(\sigma)})$ is recursively defined by:

- $R_0 = (a_1, a_2, \ldots, a_\sigma)$;
- R_{n+1} is obtained from R_n by applying one of the Rauzy rules, labeled with $1, 2, \ldots k$, with the rule $i \in \{1, 2, \ldots, \sigma\}$ being defined by:
  - $A_{n+1}^{(i)} = A_n^{(i)}$,
  - $A_{n+1}^{(j)} = A_n^{(i)} A_n^{(j)}$ for $j \neq i$.

A word $w \in \Sigma^*$ is a *finite epistandard word* if it is an element of a tuple R_n, for some $n \geq 1$. Note that if $\sigma = 2$, this definition coincides with the one defining standard words. In this sense, epistandard words are a generalization of standard words.

3 BWT Matrix and Permutations

The *Burrows–Wheeler matrix* (BWT matrix) of a word w is the matrix $\mathcal{B}_w$ whose rows are the conjugates of w in ascending lexicographic order. Let us denote by F and L, respectively, the first and the last column of the BWT matrix of a word w. We have that F is the sequence of sorted letters of w, i.e., $F = a_1^{n_1} a_2^{n_2} \cdots a_\sigma^{n_\sigma}$, where $(n_1, n_2, \ldots, n_\sigma)$ is the Parikh vector of w; L, instead, is the *Burrows–Wheeler Transform* of w and is denoted by $bwt(w)$.

▶ Remark 1. By definition, the Burrows–Wheeler transform is the same for all words in the same conjugacy class. Moreover, the Burrows–Wheeler transform is an injective map on the set of conjugacy classes of words: two words u and v are conjugates if and only if $bwt(u) = bwt(v)$.

A word is a *BWT image* if it is the Burrows–Wheeler Transform of some word. Not every word is a BWT image. The Burrows–Wheeler Transform is therefore a (non-surjective) map from the set of conjugacy classes of words to the set of words.

▶ **Proposition 2** ([53, Proposition 2]). *Let η_n be the morphism defined by $\eta_n(a_i) = a_i^n$ for every $a_i \in \Sigma$. If $w = v^n$, then $bwt(w) = \eta_n(bwt(v))$.*

The *standard permutation* of a word $u = u_1u_2\cdots u_n$ over Σ is the permutation π_u of $\{1, 2, \ldots, n\}$ such that $\pi_u(i) < \pi_u(j)$ if and only if $u_i < u_j$ or $u_i = u_j$ and $i < j$. In other words, π_u orders distinct letters of u lexicographically, and equal letters by occurrence order.

A word u is a BWT image of a primitive word if and only if π_u is a length-n cycle, in which case the standard permutation of u is also called *LF-mapping*. More generally, the problem of characterizing BWT images has been faced in [46], where the authors prove two necessary and sufficient conditions. The first one describes the words that are BWT images, while the second one explains which words can be converted to BWT images using a natural "pumping" procedure. Both conditions can be checked in linear time. In particular, the following theorem is proved.

▶ **Theorem 3** ([46, Theorem 1]). *Given a word $u \in \Sigma^*$, $u = bwt(w)$ for some $w \in \Sigma^*$ if and only if the number of cycles of the standard permutation π_u equals the greatest common divisor of the lengths of maximal runs in u.*

By using the theorem one can deduce that the word $u = bccaaab = b^1c^2a^3b^1$ is not a BWT image because the π_u decomposes into the 2 cycles (1 4) and (2 6 3 7 5), but the lengths $1, 2, 3, 1$ of the maximal runs in u are coprime.

In the paper [46], the authors also characterize the words $v = c_1\cdots c_h$ that can be transformed to BWT images by a sort of "pumping": each letter c_i can be replaced by $c_i^{p_i}$ for an arbitrary positive number p_i. In particular, they prove the following theorem that uses the notion of global ascent[2] of a word.

▶ **Theorem 4** ([46, Theorem 2]). *Let $v = c_1\cdots c_h$ be an arbitrary word with at least two different letters. A BWT image of the form $c_1^{p_1}c_2^{p_2}\cdots c_h^{p_h}$, where $p_i > 0$ for all i, exists if and only if v has no global ascents.*

Using the algorithm provided in their paper, starting from the word $v = dacba$, one can construct the word $w = daccbaa$ that is the *bwt* image of *aacbcad*. An interesting open question is that of constructing the shortest possible *bwt* image with a given order of letters.

The *inverse standard permutation* π_u^{-1} of a word u, i.e., the inverse permutation of the standard permutation of u, can be obtained by listing in left-to-right order the positions of a_1 in u, then the positions of a_2, and so on. The inverse standard permutation of a BWT image is also called *FL-mapping*. If w is a Lyndon word, the inverse standard permutation of $bwt(w)$, in cycle form, can also be obtained by reading the Inverse Suffix Array (ISA) of w as a cycle [16, 59]. Recall that the ISA of a word w is the array whose ith entry is the ranking of the suffix of w starting at position i.

In the rest of this chapter, unless otherwise specified, we will deal with primitive words only.

[2] A word v has a *global ascent* if $v = xy$, where x, y are nonempty and the biggest letter of x is less than or equal to the smallest letter of y.

If w is a Lyndon word and u is its BWT, then w can be uniquely reconstructed from u. In fact, we have: $w_n = u_1$ and, for every $j = 1, \ldots, n-1$, $w_{n-j} = u_{\pi_u^j(1)}$. In other words, if $\pi_u = (j_1\ j_2\ \cdots\ j_n)$ is the cycle corresponding to the standard permutation of u, with $j_1 = 1$, then $w = u_{j_n} u_{j_{n-1}} \cdots u_{j_1}$. If w is a primitive word that is not Lyndon, it is sufficient to memorize its index in the BWT matrix to reconstruct it uniquely from its BWT.

▶ Remark 5. The reversibility of $bwt(w)$ is achieved thanks to the two known fundamental properties:

- for all $i = 1, \ldots, n$, the letter $F[i]$ circularly follows the letter $L[i]$ in the word w;
- for each $a \in \Sigma$ and for each k, the k-th occurrence of a in L corresponds to the k-th occurrence of a in F.

▶ **Example 6.** The BWT matrix $\mathcal{B}_w$ of the word $w = aabacacb$ is shown in Figure 1. The word $u = babccaaa$ is a BWT image since $u = bwt(w)$. The standard permutation of u is $\pi_u = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 1 & 6 & 7 & 8 & 2 & 3 & 4 \end{pmatrix}$, or, in cycle form, $\pi_u = (1\ 5\ 8\ 4\ 7\ 3\ 6\ 2)$.

The inverse standard permutation of u is $\pi_u^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 6 & 7 & 8 & 1 & 3 & 4 & 5 \end{pmatrix} = (1\ 2\ 6\ 3\ 7\ 4\ 8\ 5)$ (notice that, as cycle, it is the reversal of π_u), and $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 6 & 3 & 7 & 4 & 8 & 5 \end{pmatrix} = ISA(aabacacb)$.

$$\begin{pmatrix} a & a & b & a & c & a & c & b \\ a & b & a & c & a & c & b & a \\ a & c & a & c & b & a & a & b \\ a & c & b & a & a & b & a & c \\ b & a & a & b & a & c & a & c \\ b & a & c & a & c & b & a & a \\ c & a & c & b & a & a & b & a \\ c & b & a & a & b & a & c & a \end{pmatrix}$$

Figure 1 BWT-matrix $\mathcal{B}_w$ of the word $w = aabacacb$.

4 Combinatorics of BWT Runs

One of the key properties of the BWT is that it produces a word that is typically more compressible than the word given as input. Such compressibility arises from the fact that sorting the conjugates of a given word w places those with equal or similar prefixes consecutively in the matrix $\mathcal{B}_w$, so the characters preceding those similar contexts are grouped in $bwt(w)$. As a result, repetitions in the input word w tend to produce long runs of equal characters in $bwt(w)$ (so-called *clustering effect*). For this reason, a commonly used measure to evaluate the repetitiveness of a text, and consequently its compressibility via the BWT, is the number of BWT runs, i.e., the number of maximal runs the BWT applied produces, after being applied to the text. More formally, given a word w, the measure $r = r(w)$ denotes the number of the *BWT runs* and it is defined as

$$r(w) = runs(bwt(w)),$$

where $runs(v)$ denotes the number of maximal runs of a word v. The more repetitive a word w is, the lower the number r of its BWT runs.

From a combinatorial point of view, the parameter r has been studied to obtain more information about the combinatorial complexity and repetitiveness of a word from the number of BWT runs it produces. The following proposition, which immediately follows from Proposition 2, shows that the number of BWT runs of a periodic word is exactly the number of BWT runs of its primitive root.

▶ **Proposition 7.** *Let w be a word and $n > 0$ an integer. Then, $r(w) = r(w^n)$.*

4.1 BWT Runs and Clustering Effect

A first combinatorial approach to the BWT clustering effect has been investigated in [51, 52], where the so-called *BWT-clustering ratio* $\rho(w)$ of a word w has been studied, defined as

$$\rho(w) = \frac{r(w)}{runs(w)}.$$

If two words u and v are conjugates, then $|runs(u) - runs(v)| \leq 1$, and, within a conjugacy class, a power of a Lyndon word is one of the conjugates with the fewest maximal runs. Analyzing the clustering effect of the BWT from a combinatorial perspective is equivalent to studying how the number of BWT runs can grow with respect to the number of maximal runs in the input word. Based on Remark 1 and Proposition 7, we restrict our attention to Lyndon words, and the following theorem shows that the number of BWT runs can be at most twice the number of maximal runs in the input word.

▶ **Theorem 8** ([52, Theorem 3.3])**.** *Let w be a Lyndon word over a finite alphabet Σ. Then:*

$$|alph(w)| \leq r(w) \leq 2 \cdot runs(w).$$

This is equivalent to stating that, for a Lyndon word w, $0 < \rho(w) \leq 2$. The following result shows that all positive rational numbers less than 2 are possible BWT-clustering ratios. Note that the proof in [51] explicitly shows how to construct a Lyndon word with a given BWT-clustering ratio.

▶ **Theorem 9** ([51, Theorem 2])**.** *For every $q \in \mathbb{Q} \cap (0, 2]$, there exists a Lyndon word $w \in \{a, b\}^*$ having $\rho(w) = q$.*

The BWT-clustering ratio has also been analyzed for infinite families of highly repetitive words, specifically those generated by a morphism. In particular, in [11], the value of the measure r and the combinatorial properties of the BWT on finite words generated by compositions of the Thue–Morse morphism and some Sturmian morphisms have been studied. It has been proved that for such words the BWT-clustering ratio is significantly less than 1. This analysis has been generalized in [23], where all binary prolongable morphisms have been considered. The following result shows that all finite words generated by iterating the application of a morphism an arbitrary number of times, starting from a letter of the alphabet, are highly compressible after applying the BWT.

▶ **Theorem 10** ([23, Theorem 28])**.** *Let μ be a binary morphism prolongable on a letter a, such that $\mu(a) \neq ab^m$ and $\mu(b) \neq b^n$, for every $m \geq 1$, $n \geq 1$. Then*

$$\lim_{i \to \infty} \rho(\mu^i(a)) = 0.$$

4.2 BWT Runs and Factor Complexity

Great attention has been given to the relationship between the measure r and other complexity measures used in combinatorics on words, such as the factor complexity. Understanding these connections can provide deeper insights into compressibility and the structural properties of words. In the next sections, some results on the number of BWT runs for words with minimal and maximal factor complexity are reported.

4.2.1 Insights from Special Factors

The factor complexity of a word is closely related to the number of its special factors [14, Proposition 4.5.3][3], and provides important information about the structural complexity of the word itself. In this subsection, we present some results on the relationship between BWT runs and some measures related to special factors.

Given an alphabet Σ, a word $w \in \Sigma^*$ and a factor $u \in Fact(w)$, the *right* (resp. *left*) *extensions of* u *in* w are defined as $e_r(u) = |\{ua \mid a \in \Sigma, ua \in Fact(w)\}| - 1$ (resp. $e_\ell(u) = |\{au \mid a \in \Sigma, au \in Fact(w)\}| - 1$).

So, u is right (resp. left) special if $e_r(u) \geq 1$ (resp. $e_\ell(u) \geq 1$), while it is bispecial if and only if $e_r(u), e_\ell(u) \geq 1$. A bispecial factor u of w is called *weakly bispecial* if $|Fact(w) \cap \Sigma u \Sigma| = \max\{e_\ell(u), e_r(u)\} + 1$. We denote the set of (resp. *weakly*) *bispecial factors* of w by $BS(w)$ (resp. $WBS(w)$).

With a slight abuse of notation, if $[w]$ is the conjugacy class of w, $Fact([w])$ denotes the set of circular factors of w, i.e., the factors of all words in $[w]$, while $e_r(u)$ (resp. $e_\ell(u)$) are the right (resp. left) extensions of u in all words of the conjugacy class $[w]$. Analogously, $BS([w])$ (resp. $WBS([w])$) denotes the set of bispecial (resp. weakly bispecial) factors in $Fact([w])$.

The following lemma shows the relationship between the number of (weakly) bispecial factors in $[w]$ and the number of BWT runs of any word in $[w]$.

▶ **Lemma 11** ([23, Lemma 13], [65, Lemma 19]). *Let $w \in \Sigma^*$ be a finite word. Then,*

$$\left(\sum_{u \in WBS([w])} \min\{e_\ell(u), e_r(u)\} \right) + 1 \leq r(w) \leq \left(\sum_{u \in BS([w])} e_r(u) \right) + 1.$$

4.2.2 BWT Runs and δ Measure

In the context of data compression, recent research has led to the design of runlength-compressed BWT-based indexing data structures whose space complexity is bounded by a function of r. The most relevant one is the r-index [24, 25], a fully compressed index of size $O(r)$ that allows one to locate the occ occurrences of a factor in a text of length n in $O(occ \log\log n)$ time. These structures are very effective for indexing collections of highly repetitive texts. Other repetitiveness measures have been used to evaluate the performance of compressed indexing data structures for highly repetitive string collections. A survey of these measures can be found in [56]. Examples of such measures include, but are not limited to, the number z of factors in LZ77 factorization [44, 36], the number g of rules in the smallest context-free grammar generating the word [35, 67], the size b of the smallest bidirectional

[3] In [14] the study of the factor complexity of infinite words is considered; however, we can extend their results by observing that the circular factor complexity of a finite word w is equivalent to the factor complexity of the infinite periodic word $w^\omega = www\cdots$.

macroscheme [71], the size (number of nodes plus number of edges) e of the Compact Directed Acyclic Word Graph [9] and the size γ of the smallest string attractor [34, 49]; some of these measures have been also investigated for well-known families of words [15, 39, 2, 68, 62, 55, 13]. All these measures are lower-bounded by the measure δ, which is defined by using the factor complexity:

$$\delta(w) = \max_{1 \le i \le |w|} \left\{ \frac{|Fact(w) \cap \Sigma^i|}{i} \right\}.$$

Here, we are interested in deriving an upper bound for r in terms of δ. In [33], an upper bound has been proved when the measure r is computed for words with an end-of-string symbol appended. Note that, as shown in [31], the presence of the end-of-word symbol can significantly affect the value of r[4]. However, in the next theorem, we show that an analogous bound established in [33] also holds for r when a generic word is considered.

▶ **Theorem 12.** *Every word $w \in \Sigma^n$ satisfies*

$$r(w) = O\left(\delta(\sqrt{w}) \log \delta(\sqrt{w}) \cdot \max\left\{1, \log \frac{|\sqrt{w}|}{\delta(\sqrt{w}) \log \delta(\sqrt{w})}\right\}\right)$$

Proof. By [33, Theorem 3.7] it follows that

$$r(w\$) = O\left(\delta(w) \log \delta(w) \cdot \max\left\{1, \log \frac{n}{\delta(w) \log \delta(w)}\right\}\right),$$

where $\$ \notin \Sigma$ is an end-of-word symbol smaller than any symbol from Σ. Let us suppose w is primitive. By [65, Lemma 16], for every word $w = uv$ such that vu is Lyndon it holds that $r(vu\$) - 2 \le r(w) \le r(vu\$) - 1$, while by [65, Lemma 17] we have that $|\delta(uv) - \delta(vu)| < 1$, and clearly $\delta(vu\$) - 1 \le \delta(vu) < \delta(vu\$)$. Hence, for every primitive word $w = uv$ such that vu is Lyndon, it holds

$$\begin{aligned} r(vu) = r(uv) &= \Theta(r(uv\$)) \\ &= O\left(\delta(uv\$) \log \delta(uv\$) \cdot \max\left\{1, \log \frac{|uv\$|}{\delta(uv\$) \log \delta(uv\$)}\right\}\right) \\ &= O\left(\delta(vu) \log \delta(vu) \cdot \max\left\{1, \log \frac{n}{\delta(vu) \log \delta(vu)}\right\}\right). \end{aligned}$$

On the other hand, if $w = (uv)^h$ for some $h > 1$ such that vu is Lyndon, by Proposition 7 it holds that $r(w) = r(\sqrt{w}) = r(vu)$, and the thesis follows. ◀

5 Extremal Cases in the Number of BWT Runs

In the previous section, we have shown that $r(w)$, the number of maximal runs of the BWT of a word w, can range from the size of the alphabet to, at most, the double of the number of runs of the original word w. In this section, we focus on these special extreme cases, namely, the one where $r(w) = |\Sigma|$ and the one where $r(w) = 2 \cdot runs(w)$.

These extreme cases are important because they could have implications for the compressibility of the input text. Indeed, when the number of maximal runs is minimized, it is possible to achieve the optimal performance of the BWT-based compressor by using techniques such

[4] There can be a multiplicative $\Theta(\log n)$ [31, Propositions 43], or an additive $\Theta(\sqrt{n})$ [31, Propositions 44] factor difference between $r(w)$ and $r(w\$)$, where $n = |w|$.

as run-length encoding (RLE) and move-to-front (MTF) [7]. In the opposite case, using a BWT-based compressor may not be effective. To understand the "clustering effect" from a combinatorial perspective, several studies have analyzed the structural properties of words that produce the maximum or minimum number of maximal runs when a text is transformed using the BWT.

5.1 Minimum Number of BWT Runs

We first consider the case of the minimum number of BWT runs, i.e., the size of the alphabet. To introduce the results, we first need to recall some definitions and give some notation.

A primitive word w over a totally ordered alphabet $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$, with $a_1 < a_2 < \ldots < a_\sigma$ is a *clustering word* if in $bwt(w)$ all equal letters appear in the same cluster. This means that $bwt(w)$ has exactly $|\Sigma|$ clusters. Up to now, no combinatorial or structural characterization for all clustering words has been established. However, in [19] it has been proved that clustering words are exactly those words w such that ww occurs in a trajectory of an interval exchange transformation.

▶ **Example 13.** The word $w = bcabcbcabcbca$ is a clustering word. Indeed, $bwt(w) = c^5a^3b^5$.

Perfectly clustering words, widely studied in the literature, are a special case of clustering words.

A *perfectly clustering word* is a word over a totally ordered alphabet $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$, with $a_1 < a_2 < \ldots < a_\sigma$, such that $bwt(w) = a_\sigma^{n_\sigma} \cdots a_2^{n_2} a_1^{n_1}$ for some non-negative integers $n_1, n_2, \ldots, n_\sigma$.

▶ **Example 14.** The word $v = cacbcac$ is perfectly clustering, since $bwt(v) = c^4ba^2$. Note also that the word $w = bcabcbcabcbca$ in Example 13, although clustering, is not perfectly clustering.

Perfectly clustering words have first been studied and characterized for binary alphabets in [53]. The main result is formalized as follows:

▶ **Theorem 15** ([53, Corollary 11])**.** *Given a word $w \in \{a, b\}^*$, the following conditions are equivalent:*

1. *$bwt(w) = b^p a^q$, with $p, q \geq 0$, i.e., w is perfectly clustering;*
2. *w is a circularly balanced word;*
3. *w is a conjugate of a power of a standard word.*

▶ **Example 16.** Let $f_6 = abaababaabaab$ be the Fibonacci word of order 6. One can easily verify that $bwt(f_6) = b^5a^8$. Recall that Fibonacci words are special cases of standard words.

It is an easy remark that, when $|alph(w)| > 1$, the first letter of $bwt(w)$ cannot be the least letter in the alphabet, so the only clustering words over a binary alphabet are perfectly clustering. Then the powers of standard words and their conjugates are the only possible clustering words in the binary case.

Having established a characterization of perfectly clustering words for the two-letter alphabet, the natural next step is to explore larger alphabets. This extension, however, is not straightforward. Unlike the binary case, the defining properties of standard words for larger alphabets no longer coincide, leading to different word sets. Consequently, finding a complete characterization of perfectly clustering words that extends Theorem 15 to larger alphabets has been complex and lasted nearly twenty years, marked by a series of insightful intermediate achievements.

The first of these results, due to Simpson and Puglisi [69], is stated in the next theorem and provides a strategy to construct primitive perfect clustering words over the ternary alphabet $\{a, b, c\}$. It starts from the set of all standard words over the binary alphabets $\{b, c\}$ and $\{a, c\}$, and considers the minimal set T including such words that is closed under the three ternary morphisms μ_1, μ_2 and μ_3, where μ_1 maps a to a, b to ac, c to ac; μ_2 maps a to c, b to b, c to a, and $\mu_3(x)$ is obtained from x by inserting a b in the middle of each occurrence of the factors aa, cc, ba and bc, by considering x as a circular word.

▶ **Theorem 17** ([69, Theorem 3.11]). *All perfectly clustering ternary words are precisely the conjugates of the words in the set T and the powers of these conjugates.*

Note that no extension of such construction for larger alphabets is known. In the same paper [69], the following characterization of the Parikh vectors of clustering words is provided. This result has been independently given in [58].

▶ **Theorem 18** ([58, Lemma 1], [69, Theorems 3.9 and 3.12]). *Let w be a primitive ternary word with Parikh vector $P_w = [\alpha, \beta, \gamma]$. The word w is perfectly clustering if and only if* $\gcd(\alpha, \beta, \gamma) = \gcd(\alpha + \beta, \beta + \gamma) = 1$.

▶ **Example 19.** The word $w = cacacacbbbbbb$ is perfectly clustering, since $bwt(w) = c^4b^6a^3$. Indeed its Parikh vector $P_w = [3, 6, 4]$ is such that $gcd(3, 6, 4) = 1$ and $gcd(3 + 6, 6 + 4) = 1$.

Extending Theorem 18 to the case of larger alphabets is still an open problem.

Another characterization of perfectly clustering ternary words in terms of the Three-distance theorem has been recently given in [8].

Perfectly clustering words have also been characterized by looking at the corresponding BWT matrix. The following result was obtained independently in [69, 63].

▶ **Theorem 20** ([69, Theorem 4.3], [63, Theorem 4.2]). *A word w is perfectly clustering if and only if $\mathcal{B}_w = R$, where R is the matrix obtained from the BWT-matrix $\mathcal{B}_w$ by a rotation of* $180°$.

The previous theorem, although it is an immediate consequence of the BWT properties described in Remark 5, allows one to observe that the rows of R correspond to the conjugates of $\tilde{w}$ and for each row i, the reverse of the i-th conjugate of w in $\mathcal{B}_w$ coincides with the $(n + 1 - i)$-th conjugate of w in $\mathcal{B}_w$. Using this fact, the following corollary can be derived:

▶ **Corollary 21** ([69, Corollary 4.4]). *Every conjugate of a perfectly clustering word has the two-palindrome property.*

▶ **Example 22.** The word $w = cacacacbbbbbb$ in Example 19 has the two-palindrome property, since it is the concatenation of *cacacac* and *bbbbbb*.

The connection between palindromic factors and the property of being perfectly clustering is deepened in the following results. Recall that a word w is *circularly rich* if $w^2 = ww$ contains the maximum number $(|w^2| + 1)$ of distinct palindromic factors. In [63], the authors show a relationship between perfectly clustering words and circularly rich words.

▶ **Theorem 23** ([63, Theorem 6.2]). *If a word is perfectly clustering, then it is circularly rich.*

▶ **Example 24.** The word $w = cacacacbbbbbb$ of length 13 in Example 19 is circularly rich since w^2 has 27 distinct palindromic factors.

As mentioned before, extending the results of Theorem 15 to the case of larger alphabets is not easy. In fact, the following sets:

- circularly balanced words;
- perfectly clustering words;
- conjugates of powers of finite epistandard words,

which are equivalent for a two-letter alphabet, do not coincide when the alphabet has size ≥ 3.

Also observe that, while the notions of epistandard, circularly balanced, and circularly rich word remain invariant under character permutation, the property of being perfectly clustering depends on the order of the alphabet. Hence, the equivalence among these three notions, as stated in the following theorem, holds for some permutation of the characters in the alphabet, but under the restrictive assumption of being a circularly balanced word.

▶ **Theorem 25** ([64, Theorem 8.16]). *Let Σ be an alphabet, and let $w \in \Sigma^*$ be a circularly balanced word over Σ. The following statements are equivalent, for some character permutation:*

i) *w is a perfectly clustering word;*
ii) *w is a circularly rich word;*
iii) *w is a conjugate of a power of a finite epistandard word.*

However, Theorem 25 does not provide a complete characterization of perfectly clustering words. In fact, there exist circularly rich words that, although not circularly balanced, are perfectly clustering: an example is $w = bbbbbacaca$.

Only recently, a complete characterization of the perfectly clustering words has been given by Lapointe and Reutenauer in [43]. In order to introduce this characterization, we first need some definitions.

Let w be a word over a totally ordered alphabet Σ. A *special factorization of w* is a factorization of the form

$$w = a_1 w_1 a_2 w_2 \cdots a_{k-1} w_{k-1} a_k,$$

where $a_1 < a_2 < \cdots < a_k$ and $w_1, \cdots, w_{k-1} \in \Sigma^*$. If w satisfies this equality, we set $W = a_k w_{k-1} \cdots w_2 a_2 w_1 a_1$. We call this special factorization *palindromic* if each word w_i is a palindrome; in this case, $W = \tilde{w}$.

The characterization of perfectly clustering words is stated in the following theorem:

▶ **Theorem 26** ([43, Theorem 3.1]). *The following conditions are equivalent, for a primitive word w over a totally ordered finite alphabet Σ:*

1. *w is a perfectly clustering Lyndon word;*
2. *w has the two-palindrome property and w has a palindromic special factorization;*
3. *w has a special factorization such that w is a conjugate of W.*

Interestingly, Theorem 26 generalizes two important results about Christoffel words (that are clustering Lyndon words in the binary case). In particular, the equivalence of 1 and 2 is a generalization of a result of de Luca and Mignosi [17], and the equivalence of 1 and 3 is a generalization of a result of Pirillo [60].

Note that in [43, Corollary 6.1], the authors proved that perfectly clustering Lyndon words have a unique special factorization. Furthermore, the following theorem shows another interesting combinatorial property of the BWT matrix of a perfect clustering word. In Example 28, several properties of perfectly clustering words described in this section are illustrated.

▶ **Theorem 27** ([43, Theorem 6.1]). *Let w be a perfectly clustering word. Then, for any two consecutive rows of its Burrows-Wheeler matrix $\mathcal{B}_w$, viewed as two words w' and w'', one has*

$$w' = ymx, \quad w'' = ym\tilde{x},$$

for some words x, y, m such that xy is one of the palindromes in the special palindromic factorization of w.

Providing a combinatorial characterization of the palindromic words that appear in the special palindromic factorization of perfectly clustering words remains an open problem. The case of the ternary alphabet has been studied in [42].

$$\begin{pmatrix}
a\ c\ b\ c\ b\ c\ a\ d\ a\ d\ a\ d\ a\ \mathbf{d}\\
a\ d\ a\ c\ b\ c\ b\ c\ a\ d\ a\ d\ a\ \mathbf{d}\\
a\ d\ a\ d\ a\ c\ b\ c\ b\ c\ a\ d\ a\ \mathbf{d}\\
a\ d\ a\ d\ a\ d\ a\ c\ b\ c\ b\ c\ a\ \mathbf{d}\\
a\ d\ a\ d\ a\ d\ a\ d\ a\ c\ b\ c\ b\ \mathbf{c}\\
b\ c\ a\ d\ a\ d\ a\ d\ a\ d\ a\ c\ b\ \mathbf{c}\\
b\ c\ b\ c\ a\ d\ a\ d\ a\ d\ a\ d\ a\ \mathbf{c}\\
c\ a\ d\ a\ d\ a\ d\ a\ d\ a\ c\ b\ c\ \mathbf{b}\\
c\ b\ c\ a\ d\ a\ d\ a\ d\ a\ d\ a\ c\ \mathbf{b}\\
c\ b\ c\ b\ c\ a\ d\ a\ d\ a\ d\ a\ d\ \mathbf{a}\\
d\ a\ c\ b\ c\ b\ c\ a\ d\ a\ d\ a\ d\ \mathbf{a}\\
d\ a\ d\ a\ c\ b\ c\ b\ c\ a\ d\ a\ d\ \mathbf{a}\\
d\ a\ d\ a\ d\ a\ c\ b\ c\ b\ c\ a\ d\ \mathbf{a}\\
d\ a\ d\ a\ d\ a\ d\ a\ c\ b\ c\ b\ c\ \mathbf{a}
\end{pmatrix}$$

Figure 2 BWT-matrix $\mathcal{B}_w$ of $w = acbcbcadadadad$, $bwt(w)$ is highlighted in bold.

▶ **Example 28.** Consider the word $w = acbcbcadadadad$ over the 4-letter alphabet $\{a, b, c, d\}$, with $a < b < c < d$. Figure 2 shows the BWT matrix $\mathcal{B}_w$. According to Theorem 26, the following properties can be verified: 1) w is perfectly clustering, in fact $bwt(w) = d^4c^3b^2a^5$; 2) $w = (acbcbca)(dadadad)$ has the two-palindrome property and $w = a(cbc)bc(adadada)d$ has a palindromic special factorization; and 3) from the special factorization $w = a(cbc)bc(adadada)d$ one can get $W = d(adadada)cb(cbc)a$, which is the last row in the $\mathcal{B}_w$, i.e., it is a conjugate of w. Moreover, let us consider the consecutive rows $w' = bcadadadadacbc$ and $w'' = bcbcadadadadac$ corresponding to rows 6 and 7 of the BWT-matrix $\mathcal{B}_w$. As stated in Theorem 27, the following factorizations $w' = (bc)(adadadadacb)(c)$ and $w'' = (bc)(bcadadadada)(c)$ can be obtained. Finally, the matrix $\mathcal{B}_w$ is invariant under a 180°-rotation (cf. Theorem 20).

5.2 Maximum Number of BWT Runs

In this section, we restrict our attention to the binary alphabet. For a word w of length n, it is possible in principle that the BWT of w has n clusters. For example, the BWT of $w = aaababbbab$ is $bababababa = (ba)^5$. Of course, if this happens, then n must be even, i.e., $n = 2m$. Moreover, this can only happen under specific restrictions on the length, as it is shown below in this section. In particular, it is not known if this can happen for infinitely many lengths.

Recall that a prime number q is called 2-*rooted* if 2 is a generator of the cyclic group $\mathbb{Z}_q^*$. For example, 5 is 2-rooted since $\{2^i \mod 5 \mid i = 1, \ldots, 4\} = \{2, 4, 3, 1\} = \mathbb{Z}_5^*$, while 7 is not 2-rooted since $\{2^i \mod 7 \mid i = 1, \ldots, 6\} = \{1, 2, 4\}$. The sequence of 2-rooted primes is A001122 in OEIS [70]: 3, 5, 11, 13, 19, 29, 37, 53, 59, 61, 67, etc.

$$\begin{pmatrix} a\ a\ a\ b\ a\ b\ b\ \mathbf{b} \\ a\ a\ b\ a\ b\ b\ b\ \mathbf{a} \\ a\ b\ a\ b\ b\ b\ a\ \mathbf{a} \\ a\ b\ b\ b\ a\ a\ a\ \mathbf{b} \\ b\ a\ a\ a\ b\ a\ b\ \mathbf{b} \\ b\ a\ b\ b\ b\ a\ a\ \mathbf{a} \\ b\ b\ a\ a\ a\ b\ a\ \mathbf{b} \\ b\ b\ b\ a\ a\ a\ b\ \mathbf{a} \end{pmatrix}$$

Figure 3 BWT-matrix of the de Bruijn word $w = aaababbb$, $bwt(w)$ is highlighted in bold.

▶ **Proposition 29** ([52, Proposition 4.3]). *We have that $(ba)^m$ is a BWT image if and only if $2m+1$ is a 2-rooted prime.*

This follows from the fact that the inverse standard permutation of $(ba)^m$, in cycle form, is equal to $(1\ 2\ 2^2\ \dots\ 2^{2m}) \mod (2m+1)$; the fact that it is a permutation means that its elements are all distinct, and therefore coincides with the cyclic group $\mathbb{Z}^*_{2m+1}$.

▶ **Conjecture 30.** *There are infinitely many values of m for which $(ba)^m$ is a BWT image.*

The previous conjecture is related to the famous conjecture of Emil Artin on primitively rooted primes.

However, one can show that there exist infinitely many lengths n for which the BWT has *almost* n runs. This is the case of binary de Bruijn words.

Recall that a binary word w is a de Bruijn word of order $d > 0$ if all the 2^d possible words of length d occur exactly once in it circularly. For example, $aaababbb$ is a de Bruijn word of order 3. A binary de Bruijn word of order d has length 2^d.

The BWT of a binary de Bruijn word of order d is always a word in the set $\{ab, ba\}^{2^{d-1}}$. Indeed, all the factors of length 2^{d-1} must occur twice in it, once preceded by a and once preceded by b, and in consecutive rows of the BWT matrix. This property was used in [47] for designing an efficient algorithm that generates every de Bruijn word with positive probability.

▶ **Example 31.** The BWT matrix of the de Bruijn word $aaababbb$ is depicted in Figure 3. So that $bwt(aaababbb) = baabbaba \in \{ab, ba\}^4$.

As a consequence, we have (see also [52]):

▶ **Proposition 32.** *The BWT of any binary de Bruijn word of order $d > 2$ has at least $2^{d-1} + 2$ and at most $2^d - 2$ clusters.*

A binary de Bruijn word of order $d > 2$ cannot have 2^d clusters, i.e., no binary de Bruijn word of order $d > 2$ can have BWT equal to $(ba)^{2^{d-1}}$. This follows from Proposition 29 and the fact that 3 and 5 are the only 2-rooted primes of the form $2^d + 1$; in the cases $d = 1, 2$ we have, respectively, the de Bruijn words ab and $aabb$, whose BWT have 2 and 4 clusters, respectively.

6 Sensitivity of r to Modifications of the Input Word

In this section, we present some recent results concerning the sensitivity of r with respect to some combinatorial operations on words. Sensitivity analysis is a fundamental aspect of understanding the stability of compression-based repetitiveness measures.

6.1 Reverse Operation

Several repetitiveness measures, such as b, g, γ, and δ, are invariant under the reverse operation. This is not the case for the measure r, despite experimental results showing that, on real text collections, it exhibits a behavior that brings it closer to the other above measures [5, 61]. A first upper bound on the ratio between r and the number of runs in the BWT of the reverse of the word was provided in [33], where it was proven that $\frac{r(\tilde{v})}{r(v)} = \mathcal{O}(\log^2 n)$, where n is the length of the word, leaving open the question of whether this bound is tight.

An answer to this problem was given in [29], where the impact of the reverse operation on the number of BWT runs was studied for an infinite family of words, the so-called Fibonacci-plus words.

A word v is *Fibonacci-plus* if and only if $v = sb$ for some Fibonacci word s of order $2k$, $k \geq 2$, or $v = sa$ for some Fibonacci word s of order $2k+1$, $k \geq 2$. The Fibonacci-plus word v has *even order* in the first case and *odd order* otherwise.

▶ **Example 33.** The words $v_1 = abaababaabaabb = f_6 b$ and $v_2 = abaababaabaababaababaa = f_7 a$ are Fibonacci-plus words. It can be easily verified that $r(v_1) = r(v_2) = 4$ and $r(\tilde{v_1}) = r(\tilde{v_2}) = 6$. This shows that r is not invariant under the reverse operation.

More specifically, if v is a Fibonacci-plus word, then $r(v) = \Theta(1)$, while $r(\tilde{v}) = \Theta(\log(|v|))$, as synthesized in the following theorem.

▶ **Theorem 34** ([29, Theorem 1])**.** *Let v be a Fibonacci-plus word, and let $|v| = n$. Then $\frac{r(\tilde{v})}{r(v)} = \Theta(\log n)$.*

Closing the gap between the $\Omega(\log n)$ lower bound (established in [29]) and the $O(\log^2 n)$ upper bound (given in [33]) remains an open problem.

6.2 Edit Operations

In [1], the sensitivity of the measure r with respect to a single edit operation on a word (deletion, substitution, insertion of a character) has been studied. More precisely, the *multiplicative sensitivity* MS of r is defined as the maximum multiplicative factor by which a single-character edit operation on a word w can alter the value $r(w)$. Furthermore, the *additive sensitivity* AS, i.e., the maximum additive factor that an edit operation on a word w may introduce in $r(w)$. Formally, if we denote by $D(w)$, $S(w)$, and $I(w)$ the set of words obtained after a single deletion, substitution, and insertion on a word w respectively, the multiplicative and additive sensitivities for each of the edit operations are defined as follows:

$$MS_{del}(n) = \max_{\substack{w\in\Sigma^n \\ w'\in D(w)}} \left\{ \frac{r(w')}{r(w)} \right\}; \qquad AS_{del}(n) = \max_{\substack{w\in\Sigma^n \\ w'\in D(w)}} \{r(w') - r(w)\};$$

$$MS_{sub}(n) = \max_{\substack{w\in\Sigma^n \\ w'\in S(w)}} \left\{ \frac{r(w')}{r(w)} \right\}; \qquad AS_{sub}(n) = \max_{\substack{w\in\Sigma^n \\ w'\in S(w)}} \{r(w') - r(w)\};$$

$$MS_{ins}(n) = \max_{\substack{w\in\Sigma^n \\ w'\in I(w)}} \left\{ \frac{r(w')}{r(w)} \right\}; \qquad AS_{ins}(n) = \max_{\substack{w\in\Sigma^n \\ w'\in I(w)}} \{r(w') - r(w)\}.$$

By combining the results on the sensitivity of the measure δ [1, Theorem 2] and on the relationship between the measure r and δ [33, Theorem 3.7], the following upper bounds on the sensitivities of r hold.

▶ **Proposition 35** ([1, Corollary 3]). *The following upper bounds on the sensitivity of the measure r hold:*

- $MS_{del}(n) = MS_{sub}(n) = MS_{ins}(n) = O(\log n \log r)$;
- $AS_{del}(n) = AS_{sub}(n) = AS_{ins}(n) = O(r \log n \log r)$;

On the other hand, the lower bounds for the multiplicative and the additive sensitivities have been proved by showing the effect of single edit operations on specific families of words [29, 30].

▶ **Proposition 36** ([29, Theorem 1], [30, Propositions 5, 6, and 9]). *The following lower bounds on the sensitivity of the measure r hold:*

- $MS_{del}(n) = MS_{sub}(n) = MS_{ins}(n) = \Omega(\log n)$;
- $AS_{del}(n) = AS_{sub}(n) = AS_{ins}(n) = \Omega(\sqrt{n})$;

The lower bounds on the multiplicative sensitivity for deletion, substitution, and insertion have been proved via Fibonacci words, where an increase by a multiplicative $\Theta(\log n)$ factor is attained after each edit operation is applied. Since the measure r for Fibonacci words is $\Theta(1)$ [53, Theorem 9], this proves that the upper bound on MS given in Proposition 35 is tight.

The $\Omega(\sqrt{n})$ lower bound on the additive sensitivity is rather obtained with a novel family of words defined in [30]. Note, however, that such a family is already loosely compressible via BWT, as the measure r is $\Theta(\sqrt{n})$ on the words from this family before and after the edit operations are applied. The problem of proving the tightness of the upper bound $O(r \log n \log r)$ given in Proposition 35 for the additive sensitivity remains open.

6.3 Morphisms

Another interesting question is how the measure r changes under the application of a morphism on a word w. This problem has been explored by Brlek et al. in [11], where they studied the BWT runs for words obtained by applying a composition of the Thue–Morse morphism with some *Sturmian morphisms*. Recall that a morphism is Sturmian if it maps Sturmian words to Sturmian words. Recall also that every Sturmian morphism is a composition of morphisms taken from the set $\{E, \varphi, \tilde{\varphi}\}$, where:

$$E : \begin{cases} a \mapsto b \\ b \mapsto a \end{cases} \qquad \varphi : \begin{cases} a \mapsto ab \\ b \mapsto a \end{cases} \qquad \tilde{\varphi} : \begin{cases} a \mapsto ba \\ b \mapsto a \end{cases}.$$

Specifically, Brlek et al. proved that $r(\tau^{n+1}(a)) = r((\tau \circ \varphi)^n(a))$, for all $n \geq 1$ [11, Theorems 2 and 5], and the same has been experimentally observed for compositions of the Thue–Morse morphism with other Sturmian morphisms. In fact, this is a consequence of the next theorem, which provides a characterization of Sturmian morphisms in terms of their effect on the measure r.

▶ **Theorem 37** ([22, Theorem 21]). *Let μ be a binary morphism. Then $r(w) = r(\mu(w))$ for every word w with $|alph(w)| = 2$ if and only if μ is a Sturmian morphism.*

▶ **Example 38.** Let $w = abbaba$. In Figure 4 we show the BWT-matrices for the words w and $\varphi(w)$. Since $\tilde{\varphi} \in [\varphi(w)]$, one has $\mathcal{B}_{\varphi(w)} = \mathcal{B}_{\tilde{\varphi}(w)}$. Moreover, one can see that $bwt(\varphi(w)) = bwt(\tilde{\varphi}(w)) = E(bwt(\tilde{w}))a^{|w|_a}$.

$$\begin{pmatrix} a\ a\ b\ b\ a\ \mathbf{b} \\ a\ b\ a\ a\ b\ \mathbf{b} \\ a\ b\ b\ a\ b\ \mathbf{a} \\ b\ a\ a\ b\ b\ \mathbf{a} \\ b\ a\ b\ a\ a\ \mathbf{b} \\ b\ b\ a\ b\ a\ \mathbf{a} \end{pmatrix}$$

(a) $\mathcal{B}_w$.

$$\begin{pmatrix} a\ a\ a\ b\ a\ a\ b\ a\ \mathbf{b} \\ a\ a\ b\ a\ a\ b\ a\ b\ \mathbf{a} \\ a\ a\ b\ a\ b\ a\ a\ a\ \mathbf{b} \\ a\ b\ a\ a\ a\ b\ a\ a\ \mathbf{b} \\ a\ b\ a\ a\ b\ a\ b\ a\ \mathbf{a} \\ a\ b\ a\ b\ a\ a\ a\ b\ \mathbf{a} \\ b\ a\ a\ a\ b\ a\ a\ b\ \mathbf{a} \\ b\ a\ a\ b\ a\ b\ a\ a\ \mathbf{a} \\ b\ a\ b\ a\ a\ a\ b\ a\ \mathbf{a} \end{pmatrix}$$

(b) $\mathcal{B}_{\varphi(w)} = \mathcal{B}_{\tilde{\varphi}(w)}$.

Figure 4 In Subfigure 4a is shown the BWT-matrix for the word $w = abbaba$, while in Subfigure 4b is shown the BWT-matrix for the word $\varphi(w) = abaaabaab$, where φ is the Fibonacci morphism.

In [22], the more general problem of the sensitivity of the measure r to the application of morphisms has been tackled. In particular, they showed that for every binary word w and every binary injective morphism μ, $r(\mu(w)) \geq r(w)$ holds, so the Sturmian morphisms are the only binary morphisms achieving such a lower bound.

For the family of *Thue–Morse-like morphisms* defined as $\bigcup_{p,q\geq 1}\{\tau_{p,q} : \{a,b\}^* \to \{a,b\}^* \mid \tau_{p,q}(a) = ab^p, \tau_{p,q}(b) = ba^q)\}$, the inequality $r(\tau_{p,q}(w)) \leq r(w) + 2$ holds [22, Proposition 23]. It is natural to ask whether or not the sensitivity of r to the application of a binary morphism is bounded, i.e., if for each binary injective morphism μ there exists a constant k that solely depends on μ such that $r(\mu(w)) \leq r(w) + k$, for every binary word w. However, the *period-doubling morphism* $\theta : \{a,b\}^* \to \{a,b\}^*$, with $\theta(a) = ab$ and $\theta(b) = aa$, negatively answers this question.

▶ **Lemma 39** ([22, Lemma 29]). *Let θ be the period-doubling morphism. For any positive integer k there exists a word w such that $r(\theta(w)) > r(w) + k$.*

The problem of characterizing binary morphisms based on their effect on the measure r is still open.

7 Conclusions

The study of combinatorial aspects of the BWT has attracted significant interest over the years. On the one hand, it has led to a deeper understanding of the properties that make the BWT so distinctive. On the other hand, it has opened the way for different types of extensions or generalizations that have not been covered in this chapter. One such extension concerns the input. As described in Section 3, the BWT can be viewed as a mapping from the set of conjugacy classes (also called necklaces) of words in Σ^* to Σ^*, where Σ is a finite ordered alphabet. However, this map is not surjective. In 2007, Mantaci et al. introduced in [50] the *Extended Burrows Wheeler Transform* (EBWT), which is instead a bijective map between the multisets of conjugacy classes of words in Σ^* and Σ^*. Efficient computations of this transformation have been recently provided [10, 4, 57]. The *EBWT* also allows to define a bijective transformation on Σ^*, known as the Bijective BWT (BBWT), which uses the unique factorization of a word into non-increasing Lyndon factors [37]. The relationship between the number of runs produced by the *BBWT* and the number of BWT runs is studied in [3]. Transformations *EBWT* and *BBWT* are discussed in other chapters of this Festschrift.

Another generalization of the BWT is based on the order used to sort the cyclic rotations of a word. In [26], the authors introduced and studied the basic properties of the Alternating BWT (ABWT). It works similarly to the BWT, but instead of using the standard lexicographic order, the cyclic rotations of the input word are sorted according to the alternating lexicographic order. The alternating lexicographic order is defined as follows: the first characters are compared using the given alphabetic order, in case of equality, the second characters are compared using the opposite order, and so on, alternating between the two orders for even and odd positions. In [27], the authors showed that the $ABWT$ retains most of the key properties that make the BWT a powerful transformation. In particular, the $ABWT$ can be computed and inverted in linear time and can also be used to construct an efficient compressed full-text index for the transformed word. In [28], a new class of BWT variants has been introduced based on local orderings, where the alphabet order depends only on a fixed portion of the preceding context of each character. These transformations maintain the key properties of both the BWT and $ABWT$, by showing that an r-index can always be built on top of a BWT defined by a local ordering. This makes such transformations particularly well-suited for indexing highly repetitive text collections. While it is possible to find in linear time, for a given word w, a BWT variant based on context-adaptive alphabet orderings that minimizes the number of BWT runs [28], it has been proven that the problem of finding an alphabet reordering that minimizes the number of BWT runs is NP-complete [6].

References

1 Tooru Akagi, Mitsuru Funakoshi, and Shunsuke Inenaga. Sensitivity of string compressors and repetitiveness measures. *Inf. Comput.*, 291:104999, 2023. doi:10.1016/J.IC.2022.104999.

2 Hideo Bannai, Mitsuru Funakoshi, Tomohiro I, Dominik Köppl, Takuya Mieno, and Takaaki Nishimoto. A separation of γ and b via Thue-Morse words. In *SPIRE*, volume 12944 of *Lecture Notes in Computer Science*, pages 167–178. Springer, 2021. doi:10.1007/978-3-030-86692-1_14.

3 Hideo Bannai, Tomohiro I, and Yuto Nakashima. On the Compressiveness of the Burrows-Wheeler Transform. In *CPM*, volume 331 of *LIPIcs*, pages 17:1–17:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.CPM.2025.17.

4 Hideo Bannai, Juha Kärkkäinen, Dominik Köppl, and Marcin Piatkowski. Constructing and indexing the bijective and extended Burrows-Wheeler transform. *Inf. Comput.*, 297:105153, 2024. doi:10.1016/J.IC.2024.105153.

5 Djamal Belazzougui, Fabio Cunial, Travis Gagie, Nicola Prezza, and Mathieu Raffinot. Composite Repetition-Aware Data Structures. In *26th Annual Symposium on Combinatorial Pattern Matching (CPM 2015)*, volume 9133 of *Lecture Notes in Computer Science*, pages 26–39. Springer, 2015. doi:10.1007/978-3-319-19929-0_3.

6 Jason W. Bentley, Daniel Gibney, and Sharma V. Thankachan. On the Complexity of BWT-Runs Minimization via Alphabet Reordering. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *LIPIcs*, pages 15:1–15:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.15.

7 Jon Louis Bentley, Daniel D. Sleator, Robert E. Tarjan, and Victor K. Wei. A Locally Adaptive Data Compression Scheme. *Commun. ACM*, 29(4):320–330, 1986. doi:10.1145/5684.5688.

8 Valérie Berthé and Christophe Reutenauer. On the Three-Distance Theorem. *The Mathematical Intelligencer*, 46(2):183–188, 2024. doi:10.1007/s00283-023-10316-z.

9 Anselm Blumer, J. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987. doi:10.1145/28869.28873.

10 Christina Boucher, Davide Cenzato, Zsuzsanna Lipták, Massimiliano Rossi, and Marinella Sciortino. Computing the Original eBWT Faster, Simpler, and with Less Memory. In

28th International Symposium on String Processing and Information Retrieval (SPIRE), volume 12944 of *Lecture Notes in Computer Science*, pages 129–142. Springer, 2021. doi:10.1007/978-3-030-86692-1_11.

11 Srecko Brlek, Andrea Frosini, Ilaria Mancini, Elisa Pergola, and Simone Rinaldi. Burrows-Wheeler Transform of Words Defined by Morphisms. In *IWOCA*, volume 11638 of *Lecture Notes in Computer Science*, pages 393–404. Springer, 2019. doi:10.1007/978-3-030-25005-8_32.

12 Michael Burrows and David J. Wheeler. A Block Sorting data Compression Algorithm. Technical report, DIGITAL System Research Center, 1994.

13 Julien Cassaigne, France Gheeraert, Antonio Restivo, Giuseppe Romana, Marinella Sciortino, and Manon Stipulanti. New string attractor-based complexities for infinite words. *Journal of Combinatorial Theory, Series A*, 208:105936, 2024. doi:10.1016/J.JCTA.2024.105936.

14 Julien Cassaigne and François Nicolas. Factor complexity. In Valérie Berthé and Michel Rigo, editors, *Combinatorics, Automata and Number Theory*, Encyclopedia of Mathematics and its Applications, pages 163–247. Cambridge University Press, 2010. doi:10.1017/CBO9780511777653.

15 Sorin Constantinescu and Lucian Ilie. The Lempel–Ziv complexity of fixed points of morphisms. *SIAM J. Discret. Math.*, 21(2):466–481, 2007. doi:10.1137/050646846.

16 Maxime Crochemore, Jacques Désarménien, and Dominique Perrin. A note on the Burrows-Wheeler transformation. *Theoret. Comput. Sci.*, 332(1-3):567–572, 2005. doi:10.1016/j.tcs.2004.11.014.

17 Aldo de Luca and Filippo Mignosi. Some Combinatorial Properties of Sturmian Words. *Theor. Comput. Sci.*, 136(2):361–285, 1994. doi:10.1016/0304-3975(94)00035-H.

18 Xavier Droubay, Jacques Justin, and Giuseppe Pirillo. Episturmian words and some constructions of de Luca and Rauzy. *Theoret. Comput. Sci.*, 255(1-2):539–553, 2001. doi:10.1016/S0304-3975(99)00320-5.

19 Sébastien Ferenczi and Luca Q. Zamboni. Clustering Words and Interval Exchanges. *Journal of Integer Sequences*, 16(2):Article 13.2.1, 2013. URL: https://cs.uwaterloo.ca/journals/JIS/VOL16/Zamboni/zamboni2.pdf.

20 Paolo Ferragina, Raffaele Giancarlo, Giovanni Manzini, and Marinella Sciortino. Boosting textual compression in optimal linear time. *J. ACM*, 52(4):688–713, 2005. doi:10.1145/1082036.1082043.

21 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

22 Gabriele Fici, Giuseppe Romana, Marinella Sciortino, and Cristian Urbina. On the Impact of Morphisms on BWT-Runs. In *CPM 2023*, volume 259 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CPM.2023.10.

23 Andrea Frosini, Ilaria Mancini, Simone Rinaldi, Giuseppe Romana, and Marinella Sciortino. Logarithmic equal-letter runs for BWT of purely morphic words. In *DLT 2022*, volume 13257 of *Lecture Notes in Computer Science*, pages 139–151. Springer, 2022. doi:10.1007/978-3-031-05578-2_11.

24 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in BWT-runs bounded space. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1459–1477, Philadelphia, PA, 2018. SIAM. doi:10.1137/1.9781611975031.96.

25 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020. doi:10.1145/3375890.

26 Ira M. Gessel, Antonio Restivo, and Cristophe Reutenauer. A bijection between words and multisets of necklaces. *European Journal of Combinatorics*, 33(7):1537–1546, 2012. Groups, Graphs, and Languages. doi:10.1016/J.EJC.2012.03.016.

27 Raffaele Giancarlo, Giovanni Manzini, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. The alternating BWT: An algorithmic perspective. *Theor. Comput. Sci.*, 812:230–243, 2020. doi:10.1016/j.tcs.2019.11.002.

28 Raffaele Giancarlo, Giovanni Manzini, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. A new class of string transformations for compressed text indexing. *Inf. Comput.*, 294:105068, 2023. doi:10.1016/J.IC.2023.105068.

29 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 *SOFSEM*, volume 12607 of *Lecture Notes in Computer Science*, pages 249–262. Springer, 2021. doi:10.1007/978-3-030-67731-2_18.

30 Sara Giuliani, Shunsuke Inenaga, Zsuzsanna Lipták, Giuseppe Romana, Marinella Sciortino, and Cristian Urbina. Bit Catastrophes for the Burrows-Wheeler Transform. In *DLT 2023*, volume 13911 of *Lecture Notes in Computer Science*, pages 86–99. Springer, 2023. doi:10.1007/978-3-031-33264-7_8.

31 Sara Giuliani, Shunsuke Inenaga, Zsuzsanna Lipták, Giuseppe Romana, Marinella Sciortino, and Cristian Urbina. Bit Catastrophes for the Burrows-Wheeler Transform. *Theory Comput. Syst.*, 69(2):19, 2025. doi:10.1007/s00224-024-10212-9.

32 Peter M. Higgins. Burrows-Wheeler transformations and de Bruijn words. *Theor. Comput. Sci.*, 457:128–136, 2012. doi:10.1016/j.tcs.2012.07.019.

33 Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler transform conjecture. *Commun. ACM*, 65(6):91–98, 2022. doi:10.1145/3531445.

34 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *STOC*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

35 John C. Kieffer and En hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000. doi:10.1109/18.841172.

36 Dmitry Kosolobov and Arseny M. Shur. Comparison of LZ77-type parsings. *Inf. Process. Lett.*, 141:25–29, 2019. doi:10.1016/J.IPL.2018.09.005.

37 Manfred Kufleitner. On bijective variants of the Burrows-Wheeler transform. In *Proceedings of PSC*, pages 65–79, 2009. URL: http://www.stringology.org/event/2009/p07.html.

38 Alan Kuhnle, Taher Mun, Christina Boucher, Travis Gagie, Ben Langmead, and Giovanni Manzini. Efficient construction of a complete index for pan-genomics read alignment. *J. Comput. Biol.*, 27(4):500–513, 2020. doi:10.1089/CMB.2019.0309.

39 Kanaru Kutsukake, Takuya Matsumoto, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. On repetitiveness measures of Thue-Morse words. In *SPIRE*, volume 12303 of *Lecture Notes in Computer Science*, pages 213–220. Springer, 2020. doi:10.1007/978-3-030-59212-7_15.

40 Tak Wah Lam, Ruiqiang Li, Alan Tam, Simon C. K. Wong, Edward Wu, and Siu-Ming Yiu. High throughput short read alignment via bi-directional BWT. In *2009 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2009, Washington, DC, USA, November 1-4, 2009, Proceedings*, pages 31–36, Los Alamitos, CA, 2009. IEEE Computer Society. doi:10.1109/BIBM.2009.42.

41 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.

42 Mélodie Lapointe and Nathan Plourde-Hébert. Perfectly Clustering Words and Iterated Palindromes over a Ternary Alphabet. In *International Conference on Random and Exhaustive Generation of Combinatorial Structures*, 2024. URL: https://cgi.cse.unsw.edu.au/~eptcs/paper.cgi?GASCom2024.28.

43 Mélodie Lapointe and Christophe Reutenauer. Characterizations of Perfectly Clustering Words, 2024. arXiv:2407.19140.

44 Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976. doi:10.1109/TIT.1976.1055501.

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

46 Konstantin M. Likhomanov and Arseny M. Shur. Two combinatorial criteria for BWT images. In *Computer Science - Theory and Applications - 6th International Computer Science Symposium in Russia, CSR 2011, St. Petersburg, Russia, June 14-18, 2011. Proceedings*, volume 6651 of *Lecture Notes in Computer Science*, pages 385–396. Springer, 2011. doi:10.1007/978-3-642-20712-9_30.

47 Zsuzsanna Lipták and Luca Parmigiani. A BWT-Based Algorithm for Random de Bruijn Sequence Construction. In *LATIN (1)*, volume 14578 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2024. doi:10.1007/978-3-031-55598-5_9.

48 M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002. doi:10.1017/CBO9781107326019.

49 Sabrina Mantaci, Antonio Restivo, Giuseppe Romana, Giovanna Rosone, and Marinella Sciortino. A combinatorial view on string attractors. *Theor. Comput. Sci.*, 850:236–248, 2021. doi:10.1016/J.TCS.2020.11.006.

50 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.

51 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Burrows-Wheeler Transform and Run-Length Enconding. In *WORDS 2017*, volume 10432 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2017. doi:10.1007/978-3-319-66396-8_21.

52 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, Marinella Sciortino, and Luca Versari. Measuring the clustering effect of BWT via RLE. *Theor. Comput. Sci.*, 698:79–87, 2017. doi:10.1016/j.tcs.2017.07.015.

53 Sabrina Mantaci, Antonio Restivo, and Marinella Sciortino. Burrows-Wheeler transform and Sturmian words. *Information Processing Letters*, 86:241–246, 2003. doi:10.1016/S0020-0190(02)00512-4.

54 Giovanni Manzini. An analysis of the Burrows-Wheeler transform. *J. ACM*, 48(3):407–430, 2001. doi:10.1145/382780.382782.

55 Takuya Mieno, Shunsuke Inenaga, and Takashi Horiyama. RePair grammars are the smallest grammars for Fibonacci words. In *CPM*, volume 223 of *LIPIcs*, pages 26:1–26:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CPM.2022.26.

56 Gonzalo Navarro. Indexing highly repetitive string collections, part I: repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2022. doi:10.1145/3434399.

57 Jannik Olbrich, Enno Ohlebusch, and Thomas Büchler. Generic non-recursive suffix array construction. *ACM Trans. Algorithms*, 20(2), 2024. doi:10.1145/3641854.

58 Igor Pak and Amanda Redlich. Long cycles in abc-permutations. *Functional Analysis and Other Mathematics*, 2:87–92, 2008. doi:10.1007/s11853-008-0017-0.

59 Dominique Perrin and Antonio Restivo. Words. In Miklós Bóna, editor, *Handbook of Enumerative Combinatorics*, Discrete Mathematics and its Applications, pages 485–539. Chapman & Hall/CRC, 2015. doi:10.1201/b18255.

60 Giuseppe Pirillo. A new characteristic property of the palindrome prefixes of a standard Sturmian word. In *Séminaire Lotharingien de Combinatoire 43*, 1999. URL: http://eudml.org/doc/120436.

61 Alberto Policriti and Nicola Prezza. From LZ77 to the Run-Length Encoded Burrows-Wheeler Transform, and Back. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*, volume 78 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.17.

62 Antonio Restivo, Giuseppe Romana, and Marinella Sciortino. String attractors and infinite words. In *LATIN*, volume 13568 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2022. doi:10.1007/978-3-031-20624-5_26.

63 Antonio Restivo and Giovanna Rosone. Burrows–Wheeler transform and palindromic richness. *Theoretical Computer Science*, 410(30):3018–3026, 2009. doi:10.1016/j.tcs.2009.03.008.

64 Antonio Restivo and Giovanna Rosone. Balancing and clustering of words in the Burrows–Wheeler transform. *Theoretical Computer Science*, 412(27):3019–3032, 2011. doi:10.1016/j.tcs.2010.11.040.

65 Giuseppe Romana. *Repetitiveness Measures based on String Attractors and Burrows-Wheeler Transform: Properties and Applications*. PhD thesis, University of Palermo, 2023. URL: https://hdl.handle.net/10447/595273.

66 Giovanna Rosone and Marinella Sciortino. The Burrows-Wheeler Transform between Data Compression and Combinatorics on Words. In *CiE 2013*, volume 7921 of *Lecture Notes in Computer Science*, pages 353–364. Springer, 2013. doi:10.1007/978-3-642-39053-1_42.

67 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.

68 Luke Schaeffer and Jeffrey O. Shallit. String attractors for automatic sequences, 2021. doi:10.48550/arXiv.2012.06840.

69 Jamie Simpson and Simon J. Puglisi. Words with Simple Burrows-Wheeler Transforms. *Electron. J. Comb.*, 15(1), 2008. doi:10.37236/807.

70 Neil J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. Available electronically at http://oeis.org.

71 James A. Storer and Thomas G. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982. doi:10.1145/322344.322346.