Theory of Computing Systems (2026) 70:42
https://doi.org/10.1007/s00224-025-10235-w

Bijective BWT Based Compression Schemes

Golnaz Badkobeh[1] · Hideo Bannai[2] · Tomohiro I[3] · Dominik Köppl[4]

Accepted: 2 August 2025

Abstract

We investigate properties of the bijective Burrows-Wheeler transform (BBWT). We show that for any string w, a bidirectional macro scheme of size $O(r_B)$ can be induced from the BBWT of w, where r_B is the number of maximal same-symbol runs in the BBWT. We also show that $r_B = O(z \log^2 n)$, where n is the length of w and z is the number of Lempel-Ziv 77 factors of w. Then, we show a separation between BBWT and BWT by a family of strings with $r_B = \Omega(\log n)$ but having only $r = 2$, where r is the maximal same-symbol runs in the standard Burrows–Wheeler transform (BWT). However, we observe that the smallest r_B among all cyclic rotations of w is always at most r. While computing an optimal rotation yielding the smallest r_B in $o(n^2)$ time remains an open problem, we show how to compute the Lyndon factorizations – a component for computing BBWT – of all cyclic rotations in $O(n)$ time using right and left Lyndon trees. We also show that the optimal rotations can be computed in $\tilde{O}(nh)$ time, where h is the (maximum) height of the two Lyndon trees which can vary from $\log n$ to n. Furthermore, we conjecture that we can transform two strings having the same Parikh vector to each other by BBWT and rotation operations, and prove this conjecture for the case of binary alphabets and permutations.

Keywords Data compression · Repetitiveness measures · Bijective Burrows-Wheeler transform · Lyndon factorization

✉ Hideo Bannai
hdbn.dsc@tmd.ac.jp

Golnaz Badkobeh
Golnaz.Badkobeh@citystgeorges.ac.uk

Tomohiro I
tomohiro@ai.kyutech.ac.jp

Dominik Köppl
dkppl@yamanashi.ac.jp

[1] Department of Computer Science, City St George's, University of London, London, UK

[2] M&D Data Science Center, Institute of Integrated Research, Institute of Science Tokyo, Tokyo, Japan

[3] Department of Artificial Intelligence, Kyushu Institute of Technology, Iizuka, Japan

[4] Department of Computer Science and Engineering, University of Yamanashi, Kofu, Japan

Published online: 03 July 2026

1 Introduction

The Burrows–Wheeler transform (BWT) [14] has seen numerous applications in data compression and text indexing, and is used heavily by various tools in the field of bioinformatics. For any string w, $\mathsf{BWT}(w)$ is defined as the string obtained by concatenating the last symbols of all cyclic rotations of w, in the lexicographic order of the cyclic rotations. BWT is not injective, as two strings are transformed to the same string if they are cyclic rotations of each other. Also, BWT is not surjective since BWT preserves the string length and by the pigeonhole principle, there exist strings that are not in the image of BWT.

For a string x, the inverse BWT transform is induced from the LF-mapping function $\psi_x(i)$ which maps position i in x to its rank among all positions ordered by $(x[i], i)$, i.e. $\psi_x(i) = |\{j \in [1, |x|] \mid x[j] < x[i]\}| + |\{j \in [1, i] \mid x[j] = x[i]\}|$. For a primitive string w, the standard BWT always constructs a string $x = \mathsf{BWT}(w)$ such that ψ_x forms a single cycle (i.e., for all $i, j \in [1, |x|]$ there exists a $k \geq 0$ such that $\psi_x^k(i) = j$), and $x[\psi_x^{|w|-1}(i)] \cdots x[\psi_x^0(i)]$ is a cyclic rotation of w. As an example, for $w[1..6] = \texttt{banana}$ and $x = \mathsf{BWT}(w) = \texttt{nnbaaa}$, $\psi_x(1) = 5$, $\psi_x(2) = 6$, $\psi_x(3) = 4$, $\psi_x(4) = 1$, $\psi_x(5) = 2$, $\psi_x(6) = 3$, and thus $\psi_x^6(i) = i$ for all $i \in [1, 6]$.

In general, ψ_x can form several cycles (e.g., when x is not in the image of BWT), and it is more natural to view the inverse BWT as a bijection from a string to a multiset of primitive cyclic strings [16]. For example, for $x = \texttt{annbaa}$, $\psi_x(1) = 1$, $\psi_x(2) = 5$, $\psi_x(3) = 6$, $\psi_x(4) = 4$, $\psi_x(5) = 2$, $\psi_x(6) = 3$, thus ψ_x forms the cycles (1), (2, 5), (3, 6), (4), respectively corresponding to the cyclic strings a, an, an, b. The extended BWT (eBWT) [23] can be viewed as the corresponding forward bijective transform (i.e., the inverse of the inverse BWT) from a multiset of cyclic primitive strings to a string. The bijective BWT (BBWT) [17, 20] exploits this bijection in order to define a bijection between strings; By selecting the lexicographically smallest rotation of each cyclic string and concatenating them in non-increasing lexicographic order, this mapping becomes a bijection that maps a string to another string. In the above example for $x = \texttt{annbaa}$, the strings b, an, an, a, are concatenated in this order to form banana. The forward transform $\mathsf{BBWT}(w)$ can then be defined as a transformation that first computes the Lyndon factorization [15] of w, and then computing the eBWT of the Lyndon factors, i.e., taking the last symbol of all cyclic rotations of all the Lyndon factors, sorted in ω-order $\prec_\omega$ – essentially the lexicographic order for cyclic strings – which can be defined, when x, y are primitive, as $x \prec_\omega y \iff x^\infty \prec y^\infty$, and $\prec$ denotes the standard lexicographic order.

It is known that the BBWT can be computed in linear time [8, 12]. It can also be used as an index similar to the BWT [7, 8], or as an index for a set of circular strings [13]. While the size r of the run-length compressed BWT (RLBWT) has been a focus of study in various contexts and is known to be small for highly-repetitive texts [27], the size r_B of the run-length compressed BBWT (RLBBWT) has not yet been studied rigorously. Biagi et al. [10, 11] study the sensitivity [1] of r_B with respect to the *reverse* operation, and present an infinite family of strings such that r_B of a string and its reverse can differ by a factor of $\Omega(\log n)$, where n is the length of the string.

In this paper, we investigate properties of BBWT and r_B. In detail, we show that we can induce a bidirectional macro scheme (BMS) [30] of size $O(r_B)$ for w, from the RLBBWT of w (Lemma 3). We further show that $r_B = O(z \log^2 n)$ where z is the number of Lempel-Ziv 77 (LZ77) factors of w (Theorem 1). Then, we show a separation between r_B and r, by a family of strings with $r_B = \Omega(\log n)$ but $r = 2$ (Theorem 2).

Noticing that $r_B = r$ for Lyndon words, the smallest r_B among all cyclic rotations of w is always at most r. While we do not yet know how to compute, in subquadratic time, such an optimal rotation that gives the smallest r_B, we show that we can compute the Lyndon factorizations of all rotations of w in linear time (Theorem 3) using right and left Lyndon trees [3, 5]. We further show that computing the optimal rotation can be done in $O(nh \log n)$ time using $O(n)$ space, or $O(nh \log \log n)$ time using $O(nh)$ space, where h is the (maximum) height of the Lyndon trees, which can vary from $\log n$ to n.

Finally, we conjecture that two strings having the same Parikh vector can be transformed into each other by BBWT and rotation operations (Conjecture 1), and prove this for special cases (Theorem 5), namely, for binary strings and permutations, i.e., each position of the string is a distinct symbol.

This paper is an extended version of a (short) paper presented at the 31st International Symposium on String Processing and Information Retrieval (SPIRE 2024) [2], and includes more detailed proofs, examples, and a new result concerning the computation of the optimal rotation and their relation to the height of Lyndon trees.

2 Preliminaries

Let Σ be a set of symbols referred to as the *alphabet*, and Σ^* the set of strings over Σ. For a string $x \in \Sigma^*$, $|x|$ denotes x's length. The empty string (the string of length 0) is denoted by ε. For integer $i \in [1, |x|]$, $x[i]$ is the i-th symbol of x, and for integer $j \in [i, |x|]$, $x[i..j] = x[i] \cdots x[j]$. For convenience, let $x[i..j] = \varepsilon$ if $i > j$. Let $x = x^1$, and for integer $k \geq 2$, $x^k = xx^{k-1}$. A string w is *primitive* if there is no string x and no integer $k \geq 2$ such that $w = x^k$. We will denote by x^∞ an infinite string obtained by concatenating x infinite times to itself. A *cyclic string* is a string whose last position is followed by the first position. The infinite string x^∞ can be viewed as a cyclic string in that for any integer i, the position i can be mapped to an appropriate position in $[1, |x|]$, i.e., $x^\infty[i] = x[((i-1) \bmod |x|) + 1]$.

Let $\mathsf{rot}(x) = x[|x|]x[1..|x|-1]$. A string y is a *cyclic rotation* (or simply a *rotation*) of x if there exists i such that $y = \mathsf{rot}^i(x)$, where $\mathsf{rot}^0(x) = x$, and for integer $k \geq 1$, $\mathsf{rot}^k(x) = \mathsf{rot}^{k-1}(\mathsf{rot}(x))$ and $\mathsf{rot}^{-k}(x) = \mathsf{rot}^{|x|-k}(x)$. A *conjugacy class* is any maximal set of strings that are cyclic rotations of each other.

Given a total order $\prec$ on Σ, the lexicographic order (also denoted by $\prec$) induced by $\prec$ is a total order on Σ^* such that $x \prec y$ if and only if x is a prefix of y, or, $x[i] \prec y[i]$ where $i = \min\{k \geq 1 \mid x[k] \neq y[k]\}$. A string w is a *Lyndon word*, if it is lexicographically smaller than all of its proper suffixes [22]. Lyndon words must therefore be primitive. Also, any string w can be partitioned into a unique sequence of lexicographically non-increasing Lyndon words, called the *Lyndon factorization* [15]

of w, i.e., $w = f_1^{k_1} \cdots f_{\ell(w)}^{k_{\ell(w)}}$ where each f_i $(1 \le i \le \ell(w))$ is a Lyndon word, and $f_i \succ f_{i+1}$ for all $1 \le i < \ell(w)$. We call $f_i^{k_i}$ the i-th *Lyndon necklace* of w. The ω-order $\prec_\omega$ is a total order over primitive strings, defined as: $x \prec_\omega y$ if and only if $x^\infty \prec y^\infty$. [1] For any string x, we denote the lexicographically smallest rotation of x as $\mathsf{rot}^*(x)$. Note that $\mathsf{rot}^*(x)$ is a Lyndon word if and only if x is primitive.

Given a string w, the Burrows–Wheeler transform $\mathsf{BWT}(w)$ is a string obtained by concatenating the last symbol of all cyclic rotations of w, in lexicographic order. The bijective BWT $\mathsf{BBWT}(w)$ is a string obtained by concatenating the last symbol of all cyclic rotations of the multiset of all Lyndon factors in the Lyndon factorization of w, in ω-order.

Since 1) $\mathsf{BWT}(w)$ is, by definition, invariant to rotations on w, 2) the Lyndon factorization of a Lyndon word w (or an integer power w^k of w) is w itself (or k copies of w), and 3) the ω-order is identical to the lexicographic order for strings of the same length, the following observation holds:

Observation 1 (Proposition 1 in [11]) *For any string w,* $\mathsf{BWT}(w) = \mathsf{BBWT}(\mathsf{rot}^*(w))$.

The number of maximal same-symbol runs in $\mathsf{BWT}(w)$ and $\mathsf{BBWT}(w)$ will be denoted by $r(w)$, and $r_B(w)$ respectively. Although $r(w)$, $r_B(w)$, and $\ell(w)$ are functions on strings to non-negative integers, we will omit writing the considered string and just write r, r_B, and ℓ, if the context is clear.

The LZ77 factorization of w [21, 33] is a factorization $w = s_1 \cdots s_z$ of the string where each s_i is the longest prefix of $s_i \cdots s_z$ that has at least two occurrences in $s_1 \cdots s_i$. We denote by $z(w)$, or simply z if the context is clear, the size of the LZ77 factorization of w. The following relation is known between the size $z(w)$ of the LZ77 factorization and the number $\ell(w)$ of distinct Lyndon factors in the Lyndon factorization:

Lemma 1 (Theorem 1 in [31]) *For any string w, $\ell(w) < 4z(w)$.*

A *bidirectional macro scheme (BMS)* [30], the most expressive form of dictionary compression, consists of a factorization of w into phrases, such that each phrase of length at least 2 can be represented as a reference to another substring of w. The referencing of the phrases induces a referencing forest over the positions: any position in a phrase of length at least 2 references another position in w, such that all positions in the same phrase have the same offset and thus adjacent positions point to adjacent positions, and there are no cycles. The number of phrases of a BMS is referred to as its size. The size of the smallest BMS that represents w is denoted by $b(w)$, or simply b if the context is clear.

3 Properties of r_B

We here analyze r_B as a repetitiveness measure for a string w. We first confirm that the RLBBWT of size r_B induces a BMS of size $O(r_B)$, implying $b = O(r_B)$, and thus

[1] Mantaci et al. [23] define the ω-order as a total order over arbitrary strings (including non-primitive strings), but as it is not relevant in our presentation, we omit this for simplicity.

can be interpreted as a repetitiveness measure in the class of dictionary compression, as is r.

We first show that the number of distinct Lyndon factors $\ell(w)$ is a lower bound for $r_B(w)$.

Lemma 2 *For any string w, $\ell(w) \leq r_B(w)$.*

Proof Corollary 4. of [13] states that for a set of m conjugate-free set of primitive strings, the number of runs r of its extended BWT (eBWT) satisfies $m \leq r$. The BBWT of w is equivalent to the eBWT of the Lyndon factors of the Lyndon factorization of w.

Recall that the multiplicity k_i of a Lyndon factor f_i of the Lyndon factorization $w = f_1^{k_1} \cdots f_{\ell(w)}^{k_{\ell(w)}}$ has no effect on the number of maximal same-symbol runs of the BBWT or the eBWT[2] because these copies are all adjacent in ω-order — and more generally: the i-th cyclic rotations of these copies are adjacent in ω-order, for every i. Since the set of $\ell(w)$ distinct factors of the Lyndon factorization of w are conjugate-free and primitive, and the eBWT of these strings have the same number of maximal same-symbol runs as the BBWT of w, the lemma follows. □

Lemma 3 *For any string w, there exists a BMS of size $O(r_B(w))$ that represents w.*

Proof We follow the existence proof for a BMS of size at most $2r(w)$ by Navarro et al. [28]. We consider a BMS such that each text position that does not correspond to a beginning of a same-symbol run in $\mathsf{BBWT}(w)$ will reference the text position corresponding to the preceding symbol in the $\mathsf{BBWT}(w)$. It is clear that there are no cycles in such a referencing, and we claim that this allows the string to be partitioned into $O(r_B(w))$ phrases, such that references of positions in the same phrase have the same offset.

Focus on the i-th Lyndon necklace $f_i^{k_i}$ of w. For any cyclic rotation of f_i, its k_i copies occur adjacently when ordering all cyclic conjugates of all Lyndon factors in the ω-order, and thus correspond to adjacent symbols in a same-symbol run in $\mathsf{BBWT}(w)$. It follows that for any position in the last $k-1$ copies of f_i, the reference points to the corresponding position in the preceding copy. Thus, the offset of references is the same for these positions, and they can be contained in the same phrase.

Next, consider a position in the first copy of f_i that does not correspond to a beginning of a same-symbol run in $\mathsf{BBWT}(w)$. Then, since the symbol at this position and the preceding (in ω-order) cyclic string is the same, their preceding positions must also correspond to adjacent positions in $\mathsf{BBWT}(w)$. This implies that, as long as the corresponding position is again not a beginning of a same-symbol run, the reference of the previous position will have the same offset unless the previous (in the cyclic sense) referenced position wraps around and is the last symbol of the referenced Lyndon factor, in which case the referenced position and previous position will not correspond to adjacent text positions. Being the beginning of a same-symbol run in $\mathsf{BBWT}(w)$ can happen at most $r_B(w)$ times. Being adjacent referenced positions in the cyclic sense, but not being adjacent referenced text positions can happen at most once

[2] For the eBWT, we consider the multiplicities of the input strings.

per referenced Lyndon necklace. Thus, the number of times adjacent text positions must be in a different phrase is bounded by $O(r_B(w) + \ell(w))$. From Lemma 2, we have that the number of phrases is thus $O(r_B(w))$ which concludes the proof. □

Example 1 For the string $w = \texttt{abbbabbababab}$, we give below an example for the BMS computed from its BBWT. The Lyndon factorization of w is abbb, abb, ab, ab, ab. The number of runs r_B is 6. BBWT positions belonging to a referencing phrase are marked with ✓ in the last row of Table 1. We therefore have 6 non-referencing phrases.

The referencing phrases are computed as follows: In Table 1, CSA is the circular suffix array whose entry $\mathsf{CSA}[i]$ denotes the corresponding text position of the *first* symbol of the i-th string in ω-order among all rotations of all Lyndon factors. The row $\mathsf{CSA}[i] - 1$ denotes the text position corresponding to $\mathsf{BBWT}[i]$, or equivalently, the corresponding text position of the *last* symbol of the i-th string in ω-order among all rotations of all Lyndon factors. By construction of our BMS, text positions corresponding to BBWT positions 2-5, 7-8, and 10 correspond to referencing phrases, i.e., the text positions 11, 13, 7, 4, 10, 12, 3. These positions refer to 9, 11, 13, 7, 8, 10, 6, respectively. If we group together neighboring text positions that have the same offset to their references, we obtain 9 phrases, which are visualized in Fig. 1.

Theorem 1 *For any string,* $r_B = O(z \log^2 n)$.

Proof We basically follow the proof of $r = O(z \log^2 n)$ by Kempa and Kociumaka [19], with changes to deal with the difference between BWT and BBWT.

The LCP array of a string w is an array of integers such that its i-th entry is 0 for $i = 1$, and for $i \geq 2$, the length of the longest common prefix (lcp) between the lexicographically $(i-1)$-th and i-th cyclic rotation of w. An *irreducible LCP position* is a position i such that $i = 1$ or $\mathsf{BWT}(w)[i-1] \neq \mathsf{BWT}(w)[i]$, and thus the number r of BWT runs is the number of irreducible LCP positions. For a multiset of primitive cyclic strings, we can analogously define the ω-LCP array [18]. Let x_i denote the i-th string, in ω-order, among all cyclic rotations of all Lyndon factors of the Lyndon factorization of w. Then, the i-th entry of the ω-LCP array is 0 for $i = 1$, and for $i \geq 2$, the lcp between x_{i-1}^{∞} and x_i^{∞}. By construction, the number r_B of BBWT runs is the number of irreducible ω-LCP positions, i.e., $i = 1$ or $\mathsf{BBWT}(w)[i-1] = x_{i-1}[0] \neq \mathsf{BBWT}(w)[i] = x_i[0]$. Note that ω-LCP values can be infinite when there are Lyndon necklaces in the Lyndon factorization with exponent at least 2. However, they can be

Table 1 BBWT and related arrays of $w = \texttt{abbbabbababab}$ of Example 1

i	1	2	3	4	5	6	7	8	9	10	11	12	13
$w[i]$	a	b	b	b	a	b	b	a	b	a	b	a	b
BBWT$[i]$	b	b	b	b	b	a	a	a	b	b	a	b	a
CSA$[i]$	8	10	12	5	1	9	11	13	7	4	6	3	2
CSA$[i]-1$	9	11	13	7	4	8	10	12	6	3	5	2	1
ref?		✓	✓	✓	✓		✓	✓		✓			

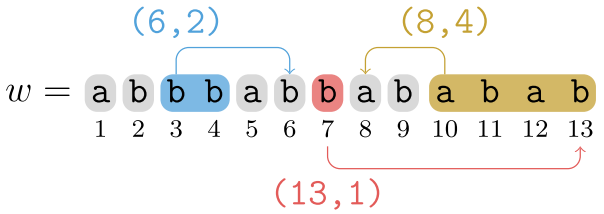

Fig. 1 The BMS factorization of Example 1

safely disregarded since they are not irreducible. The number of ω-LCP positions with ω-LCP value 0 is equal to the alphabet size which is upper-bounded by z. The theorem follows if we can show that for any value k, the number of irreducible ω-LCP values in $[k, 2k)$ is $O(z \log n)$, and considering $k = 2^0, \ldots, 2^{\lfloor \log n \rfloor}$.

If we assign cost k to each irreducible ω-LCP position $i \geq 2$, i.e. $\mathsf{BBWT}(w)[i-1] \neq \mathsf{BBWT}(w)[i]$ with ω-LCP value in $[k, 2k)$, the number of such positions is $(1/k)$-th of the total cost. More formally, $\mathsf{cost}_k(i) = k$ for $i \in \mathcal{I}_k$, where $\mathcal{I}_k = \{i \geq 2 \mid \mathsf{BBWT}(w)[i-1] \neq \mathsf{BBWT}(w)[i]$ and $\omega\text{-LCP}[i] \in [k, 2k)\}$, and $\mathsf{cost}_k(i) = 0$ for $i \notin \mathcal{I}_k$. Then, $\sum_{i \in \mathcal{I}_k} \mathsf{cost}_k(i)/k = |\mathcal{I}_k|$. Let S_{3k} (resp. S_k) denote the set of all distinct length $3k$ (resp. k) substrings of strings in the set $\{f_1^\infty, \ldots, f_{\ell(w)}^\infty\}$, i.e., the infinite repetition of the Lyndon factors of the Lyndon factorization of w. The size $|S_{3k}|$ (or $|S_k|$) of the set is upper-bounded by n since any position in f_p for any p, which corresponds to (at most) one distinct element of S_{3k} (or S_k), can be mapped to a distinct position (the corresponding position in the occurrence of f_p as a factor of the Lyndon factorization of w) in w. $|S_{3k}| = O(kz)$ also holds; Any string is either a substring of f_p (and therefore of w), or a string that is not a substring of f_p but of f_p^∞, for some $p \in [1, \ell(w)]$. The former is bounded by $3kz$ since any string must have an occurrence that contains the ending position of an LZ77 factor and there can be at most $3k$ different strings for each of the z ending positions. The latter is also $O(kz)$, since there can be at most $3k$ substrings for each f_p, and $\ell(w) < 4z$ by Lemma 1.

Next, we show how to charge the cost $k = \mathsf{cost}_k(i)$ of an irreducible ω-LCP position i, to positions of strings in S_{3k}. For each $k' = 1, \ldots, k$, a unit cost is charged to the k'-th position of the length $3k$ string $y = x_{i-j}^\infty[1-k'..3k-k']$, i.e., at $y[k'] = x_{i-j}^\infty[0]$, where $j \in \{0, 1\}$ is chosen so that the number of strings in S_k with suffix $x_{i-j}^\infty[0..k-k']$ is less than or equal to those with suffix $x_{i-(1-j)}^\infty[0..k-k']$. Then, since the number of strings in S_k with suffix $x_{i-j}^\infty[1..k-k'] = y[k'+1..k]$ is at least twice the number of strings in S_k with suffix $x_{i-j}^\infty[0..k-k'] = y[k'..k]$, it follows that a given string $y \in S_{3k}$ can be charged at most at $\log |S_k| \leq \log n$ different positions.

Furthermore, a given position k' of a given string $y \in S_{3k}$ can be charged at most twice: Let $[b, e]$ be the maximal range of cyclic strings in ω-order (which we denote by $x_b^\infty, \ldots, x_e^\infty$), prefixed by $y[k'+1..3k]$. Since $|y[k'+1..3k]| \geq 2k$, the ω-LCP values in the range $[b+1, e]$ are at least $2k$. Therefore, x_b and x_e are the only two possible positions for which y occurs as $x_b^\infty[1-k'..3k-k']$ or $x_e^\infty[1-k'..3k-k']$, and $y[k']$ is charged for an ω-LCP value $k' \in [k, 2k)$ at position b or $e+1$ in the ω-LCP array.

From the above arguments, the total cost $\sum_{i \in \mathcal{I}_k} \mathsf{cost}_k(i)$ cannot be more than twice the number of positions of strings in S_{3k} that can be charged, and is upper-bounded

by $|S_{3k}| \cdot 2 \log n = O(kz \log n)$, and thus the number of irreducible ω-LCP positions is upper-bounded by $O(z \log n)$. □

Despite sharing common traits, r and r_B can be asymptotically different:

Theorem 2 *There exists a family of strings with $r_B = \Omega(\log n)$ and $r = 2$.*

Proof Define the Fibonacci words as follows: $F_0 = \mathtt{b}$, $F_1 = \mathtt{a}$, $F_i = F_{i-1}F_{i-2}$. The infinite Fibonacci word is $\lim_{k\to\infty} F_k$. Melançon [25] showed that the k-th factor (the first factor being the 0-th) of the Lyndon factorization of the infinite Fibonacci word, has length f_{2k+2}, where $f_i = |F_i|$. Now, $\sum_{k=0}^{i} f_{2k+2} = -f_1 + (\cdots(((f_1 + f_2) + f_4) + f_6) + \cdots) + f_{2i+2} = f_{2i+3} - 1$. Therefore, the word obtained by deleting the last symbol of F_{2i+3} has $i + 1$ distinct Lyndon factors. Noticing that the last symbol of F_{2i+3} must be 'a' and will form a distinct Lyndon factor, we have that the size of the Lyndon factorization of F_{2i+3} is $i + 2$. Since $\ell(w) \leq r_B(w)$ by Lemma 2, the BBWT of the k-th Fibonacci word F_k for odd k has $\Omega(k)$ runs, while the BWT of any Fibonacci word has $r(F_k) = 2$ runs [24]. □

Very recently, the opposite case, i.e., a family of strings where $r = \Omega(\log n)$ and $r_B = O(1)$ has been reported [6] as well.

4 RLBBWT and Rotation

While r_B can be larger or smaller compared to r, if we are to incorporate a rotation operation, which can be encoded as a single $\log n$-bit integer, we can always obtain a representation that is at least the same and possibly smaller than r, using BBWT. This is because we have $r(x) = r_B(\mathsf{rot}^*(x))$ due to Observation 1. The lexicographically smallest rotation of a word can be computed in linear time [29].

For any string w, let $\hat{w} = \arg\min_{uv=w}\{r_B(vu)\}$ be the *optimal* rotation with respect to r_B. We observe that $\hat{w}$ is not always the Lyndon rotation of w. For example, for the Lyndon word $w = \mathtt{aaabaabaaabaabb}$, we have that $\mathsf{BWT}(w) = \mathsf{BBWT}(w) = \mathtt{bbbaabaaaabaaaa}$, thus $r_B(w) = 6$. However, we have that $\hat{w} = \mathsf{rot}(w) = \mathtt{baaabaabaaabaab}$ and $\mathsf{BBWT}(\hat{w}) = \mathtt{bbbbaaaaaaaaaab}$, thus $r_B(\hat{w}) = 3$.

Since $\mathsf{BBWT}(w)$ (and hence $r_B(w)$) can be computed in $O(n)$ time, we can potentially get a rotation giving smaller r_B still in linear time by trying $O(1)$ more rotations. For computing the optimal rotation giving the smallest r_B, it is straightforward to compute $\hat{w}$ (and hence $r_B(\hat{w})$) in $O(n^2)$ time. A subquadratic time algorithm for this problem is unknown to us. For LZ77, it was recently shown that indeed subquadratic time computation for the optimal rotation is possible [4]. While we have not yet been able to achieve this, we give partial results: We first show a linear time algorithm for computing the Lyndon factorizations, a precursor to computing BBWT, of all cyclic rotations, using Lyndon trees. We then show that the optimal rotation can be computed in $O(nh \log n)$ time using $O(n)$ space, or $O(nh \log\log n)$ using $O(nh)$ space, where h is the height of the Lyndon trees.

The *right (resp. left) Lyndon tree* [3, 5] of a Lyndon word w is a binary tree defined recursively on the *right (resp. left) standard factorization*. The *right standard factorization* of a Lyndon word w with at least two symbols is given by two strings u and v

such that $w = uv$ and v is the longest proper Lyndon suffix of w. The *left standard factorization* of a Lyndon word w is defined similarly with the difference that u is the longest proper Lyndon prefix of w. It is known that u and v are both Lyndon words in either of the two standard factorizations.

Now, if w is a single symbol, then the right (resp. left) Lyndon tree of w is just a leaf representing w. Otherwise, w is represented by an internal node of the right (resp. left) Lyndon tree whose left and right child are respectively the right (resp. left) Lyndon trees of u and v, where $w = uv$ is the right (resp. left) standard factorization of w.

Theorem 3 *We can compute the sizes of the Lyndon factorizations of all cyclic rotations of w in time linear in the length of w.*

Proof Assume that w is Lyndon, consider the string $W = ww$, and view the cyclic rotations of w as substrings of length $|w|$ of W. Any Lyndon factorization of such substrings consists of the Lyndon factorization of a suffix of w and a prefix of w, since any $x = uv$ such that u is a suffix of w and v is a prefix of w cannot be Lyndon: it would imply $uv \prec v \prec w \prec u$, a contradiction. Therefore, we can consider the Lyndon factorizations of the suffixes of w and the prefixes of w independently.

We observe that the factors of the Lyndon factorization for any suffix of w, are the sequence of maximal right sub-trees of the standard (right) Lyndon tree of w that are contained in the suffix. Similarly, for any prefix of w, they are the maximal left sub-trees of the left Lyndon tree of w that are contained in the prefix.

By definition of the Lyndon trees, and the property of the Lyndon factorization which states that the first (resp. last) factor is the longest prefix (resp. suffix) that is a Lyndon word, it is a simple observation that the Lyndon factorization of a suffix of w is the sequence of maximal right nodes of the right Lyndon tree that are contained in the suffix, and the Lyndon factorization of a prefix of w is the sequence of maximal left nodes of the left Lyndon tree that are contained in the prefix. In other words, the Lyndon factorization of a proper suffix $w[i..|w|]$ of w is the sequence of right children on the path from the leaf at position $i-1$ to the root in the right Lyndon tree, and the Lyndon factorization of a proper prefix $w[1..i]$ of w is the sequence of left children on the path from the leaf at position $i+1$ to the root in the left Lyndon tree. See Fig. 2 for an example.

Both trees can be computed in linear time [3, 5]. It is not difficult to see that the changes in the factorization and thus the sizes of the Lyndon factorizations can be computed in total linear time for each of the suffixes and prefixes, by a left-to-right traversal on the trees. More specifically, we maintain the Lyndon factors on a stack; The top element on the stack will be the left-most Lyndon factor when computing the factorization for suffixes of w, and the right-most when computing the factorization for prefixes of w. When removing the left-most symbol from a suffix of w, we first pop the Lyndon factor on top, and push, if any, each right child on the (possibly empty) left-path from the removed factor leading to the removed symbol. When adding a symbol to a prefix of w, we pop, if any, the Lyndon factors on the stack that are a left child of nodes that have a (possibly empty) right-path leading to the added symbol, and add the highest node with such right-path.

It is clear that the total time is linear, since each node is pushed and popped once. □

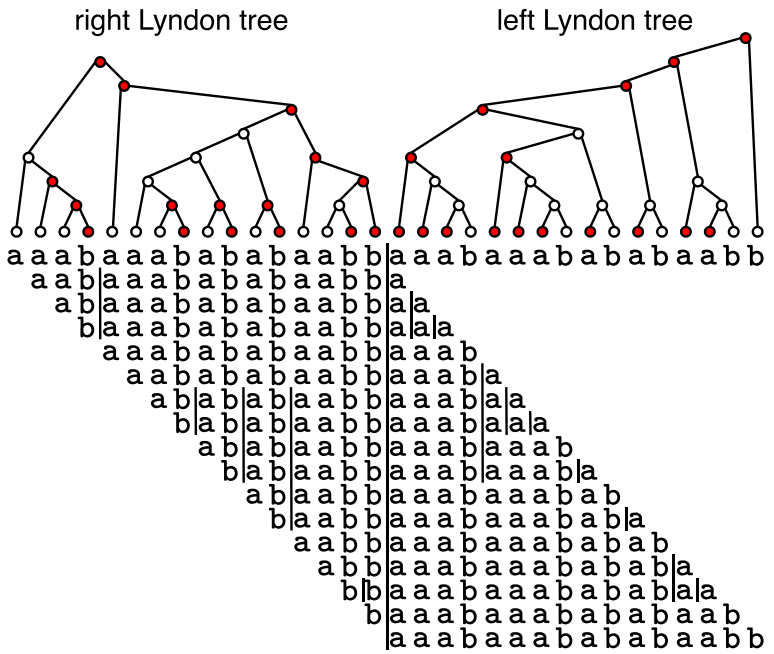

Fig. 2 The left and right Lyndon trees of the string $w = \texttt{aaabaaababababb}$. The Lyndon factorization of cyclic rotations of w are shown below, where factors are delimited by vertical bars. Right-nodes of the right Lyndon tree, and left-nodes of the left Lyndon tree are marked in red. We observe that the Lyndon factorization of suffixes of w is the sequence of maximal right nodes of the right Lyndon tree that are substrings of the suffix, and the Lyndon factorization of prefixes of w is the sequence of maximal left nodes of the left Lyndon tree that are substrings of the prefix

Theorem 4 *We can compute in $O(nh \log n)$ time using $O(n)$ space, or in $O(nh \log\log n)$ time using $O(nh)$ space, the optimal rotation of w, where h is the maximum height of the right and left Lyndon trees of w.*

Proof Following the proof of Theorem 3, we assume $w = \mathsf{rot}^*(w)$, and compute the right and left Lyndon trees of w, $\mathsf{BBWT}(w)$ in linear time, and $r_B(w)$.

To achieve $O(nh \log n)$ time using $O(n)$ space, we preprocess w in linear time for constant time Longest Common Extension (LCE) queries [9]. An LCE query (i, j) on string w returns $\max\{l \geq 0 \mid w[i..i+l-1] = w[j..j+l-1]\}$. Then, comparisons for the lexicographic order and ω-order between any two substrings of w or between any two rotations of a substring of w, can be computed in a constant time, i.e., a constant number of LCE queries.

We start from w, and incrementally compute $\mathsf{BBWT}(\mathsf{rot}^{-k}(w))$ and $r_B(\mathsf{rot}^{-k}(w))$ for $k = 1, \ldots, n-1$. We maintain the BBWT as a dynamic string with insert/delete operations and predecessor/successor queries in $O(\log n)$ time per operation, which can be implemented by any standard balanced binary search tree. When computing BBWT and r_B for the next rotation, we delete all entries corresponding to the Lyndon

factor that is removed and insert all entries corresponding to the new Lyndon factors. Updates on r_B after an insert/delete operation can be done by simply comparing the inserted/deleted symbol and the adjacent symbols which can be obtained by predecessor/successor operations. Since the ω-order of an element can be determined in constant time, all the above operations and queries can be done in $O(\log n)$ time per operation/query.

It remains to bound the total number of insertion/deletions, which is the total length of all the Lyndon factors that are updated during all the rotations. This can be bounded by $O(nh)$ since each position is involved in at most h different Lyndon factors during the updates. Therefore, the total time is $O(nh \log n)$.

To achieve $O(nh \log \log n)$ time using $O(nh)$ space, we first construct the eBWT of all the factors that occur in the Lyndon factorization of all rotations. These correspond to the right nodes of the right Lyndon tree, and the left nodes of the left Lyndon tree, and their total length sums up to $O(nh)$ as mentioned in the previous paragraph, so can be constructed in $O(nh)$ time. Then, the BBWT of each rotation of w is a subsequence of the eBWT. During the construction, we can associate for each factor, at which rotation the symbols in the factor were inserted and at which rotation they were subsequently deleted. Thus, for each rotation, we can later identify positions in the eBWT of the factors that will be inserted or deleted in constant time. The problem now lies in maintaining the size of the run-length encoding of the subsequence. This can be solved using van Emde Boas trees [32] to implement dynamic predecessor queries using $O(nh)$ space, in $O(\log \log n)$ time per update and query. □

Unfortunately, the heights of both trees can be $\Theta(n)$ in the worst case, for example, `abcdef`.... We note that the expected height of the right Lyndon tree for a random Lyndon word is known to be $\Theta(\log n)$ [26]. We are not aware of any similar analysis of the height of the left Lyndon tree.

5 BBWT Reachability

Further developing the idea of combining rotation and BBWT in order to obtain a smaller representation of a string, we give the following conjecture.

Conjecture 1 *Given two words with the same Parikh vector, we can transform one into the other by using only rotation and* BBWT *operations.*

Here, a *Parikh vector* of a string is an array of $\sigma = |\Sigma|$ non-negative integers that represent the number of occurrences of each alphabet symbol in the string. If Conjecture 1 is true, then we can represent a string w based on w's Parikh vector, and a sequence of integers where each integer alternately represents the offset of the rotation or the number of times BBWT is applied, to reach w from the lexicographically smallest string with the same Parikh vector.

We have computationally confirmed the conjecture for ternary strings of up to length 17, with code available at https://github.com/koeppl/bbwtreachability, and

prove below for the specific cases where the alphabet is binary, or, when all symbols are distinct. We first give the following lemma which shows that, under the condition of the lemma, we can obtain a lexicographically smaller string using rotation operations and an inverse BBWT operation.

Lemma 4 *Let x be a word of length n that satisfies $x = \mathsf{rot}^*(x)$ over the alphabet $\{c_1, \ldots, c_\sigma\}$ where $c_1 \prec \cdots \prec c_\sigma$, and let $(e_1, \ldots, e_\sigma)$ be the Parikh vector of x. Let $y = c_1^{e_1} \cdots c_\sigma^{e_\sigma}$, i.e., the lexicographically smallest string with the same Parikh vector as x, and $i = lcp(x, y)$. If $x[i] \neq x[n]$, then, $\mathsf{rot}^*(\mathsf{BBWT}^{-1}(\mathsf{rot}(x))) \prec x$.*

Proof Note that $x[i] \neq x[n]$ implies $y \prec x$ since $y = x$ implies $i = n$. Let $x[1..i] = c_1^{e_1} \cdots c_k^{e'}$, where $e' \leq e_k$. This implies that symbols smaller than c_k are all used up in $x[1..i] = y[1..i]$, and cannot occur in $x[i+1..n]$ nor $y[i+1..n]$. Thus, for all $j \in [i+1, n]$, it holds that $x[j] \succeq x[i] = c_k$, in particular, for all $j \in [1, i]$, it holds that $x[n] \succ x[i] \succeq x[j]$ since $x[i] \neq x[n]$ is assumed.

Next, consider traversing the symbols of $\hat{x} = \mathsf{rot}(x) = x[n]x[1..n-1]$ using the LF mapping $\psi_{\hat{x}}$ starting from position i' such that $\psi_{\hat{x}}(i') = i+1$ to recover a cyclic substring of $\mathsf{BBWT}^{-1}(\hat{x})$, whose smallest rotation (Lyndon rotation) will be a substring of $\mathsf{BBWT}^{-1}(\hat{x})$. We start from the symbol $\hat{x}[i'] = y[i+1]$. Since $\hat{x}[1] = x[n] \neq x[j] = y[j]$ for any $j \in [1, i]$ and $\hat{x}[2..i+1] = x[1..i] = y[1..i]$, it follows that $\psi_{\hat{x}}(j) = j-1$ for any $j \in [2, i+1]$. Therefore, we have that $y[i+1]$ is preceded by $x[1..i]$, i.e., $x[1..i]y[i+1] \prec x[1..i+1]$ is a cyclic substring of $\mathsf{BBWT}^{-1}(\hat{x})$, where the inequality follows from the definition of y and i. It follows that $\mathsf{BBWT}^{-1}(\hat{x})$ will contain a length $i+1$ substring that is smaller than $x[1..i+1]$, and therefore a cyclic rotation smaller than $x[1..i+1] \prec x[1..n]$, and the lemma holds. □

Example 2 For $x = \mathtt{aacb}$, $\mathsf{BBWT}^{-1}(\mathsf{rot}(x)) = \mathsf{BBWT}^{-1}(\mathtt{baac}) = \mathtt{caab}$. Then, $\mathsf{rot}^*(\mathtt{caab}) = \mathtt{aabc}$, which is lexicographically smaller than x.

For $x = \mathtt{abbcb}$, the conditions of Lemma 4 do *not* hold. Indeed, $\mathsf{BBWT}^{-1}(\mathsf{rot}(x)) = \mathsf{BBWT}^{-1}(\mathtt{babbc}) = \mathtt{cbbab}$, and its smallest rotation $\mathsf{rot}^*(\mathtt{cbbab}) = \mathtt{abcbb}$ is lexicographically larger than x.

In Example 2, caab coincides with the string y in the proof of Lemma 4. A schematic sketch of this proof is given in Fig. 3.

The following theorem is a consequence of Lemma 4.

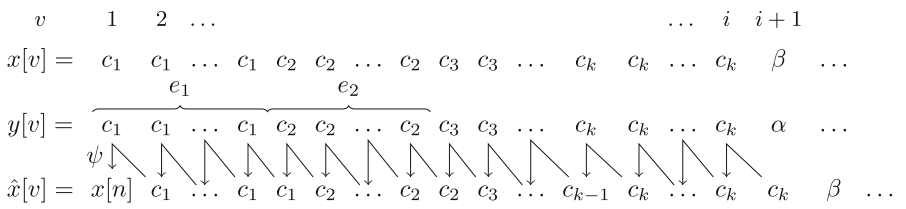

Fig. 3 Schematic sketch of the proof of Lemma 4. Here, $\alpha = y[i+1]$ and $\beta = x[i+1]$ with $c_k \leq y[i+1] = \alpha < x[i+1] = \beta$ produces a string that contains $x[1..i]y[i+1]$. $x[n]$ does not "interfere" with this mapping when $x[n] > c_k$

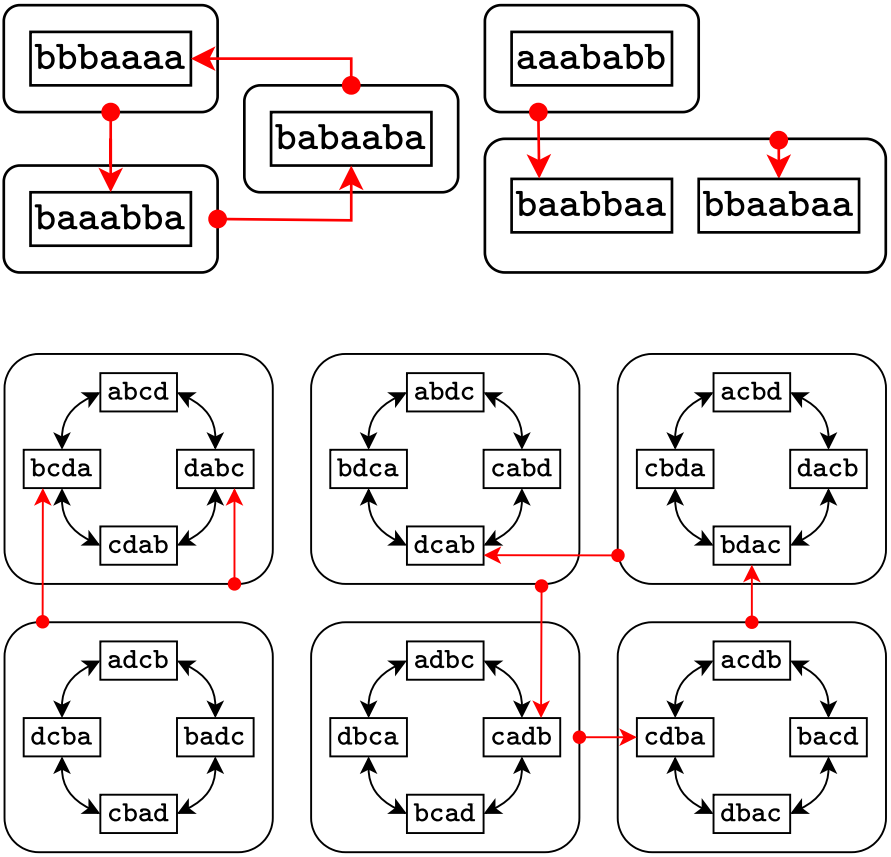

Fig. 4 Example of non-reachability for standard BWT and BWT^{-1}, for binary strings of length 7 with 4 a's and 3 b's (above), and permutations of length 4 (below). Strings are enclosed in rectangles, and rounded edge rectangles group conjugacy classes. For binary strings, not all strings are shown for each conjugacy class. For permutations, all strings are shown (since there exists a conjugacy class with no BWT images (bottom left)) and the black arrows represent rotation operation rot or rot^{-1}. The red arrows represent BWT from a conjugacy class to a string. BWT^{-1} can be considered as a mapping back to one of the strings in the conjugacy class. We see that for binary strings, the left three conjugacy classes cannot reach or be reached from the right two conjugacy classes. For permutations, the left two conjugacy classes cannot reach or be reached from the right four conjugacy classes

Theorem 5 *Given two words of the same length with the same Parikh vector, it is possible to transform one into the other by using only rotations and* BBWT *transformations if all symbols are distinct, or if the alphabet is binary.*

Proof Given any word x, let y be the smallest word with the same Parikh vector. Since BBWT and rotations are bijections, it is easy to see that $\mathsf{BBWT}^{-1}(x)$ (resp. $\mathsf{rot}^{-1}(x)$) can be represented by a sequence of $\mathsf{BBWT}(x)$ (resp. $\mathsf{rot}(x)$) operations. Therefore, it suffices to show that we can reach y from x using any of these operations. If $y \prec x$, using Lemma 4, we can always obtain a strictly lexicographically smaller string using rotations and BBWT^{-1} and thus eventually reach y: when all symbols are distinct, it is

easy to see that the condition of Lemma 4 holds. If the alphabet is binary, i.e., {a, b}, we have that $x[n] = \mathtt{b}$ since x is a smallest rotation. Furthermore, if $i = lcp(x, y)$, then, since $x[i] = \mathtt{b}$ would imply $x = y$, we have $x[i] = \mathtt{a} \neq \mathtt{b} = x[n]$. □

6 Discussion

We analyzed r_B as a repetitiveness measure and showed similar results that were known for r. We further showed that by combining rotation with BBWT, we are always able to achieve compression at least as small as BWT, and possibly better. We showed how to compute the optimal rotation in $O(nh \log n)$ time using $O(n)$ space, or $O(nh \log \log n)$ using $O(nh)$ space, where h is the maximum height of the right and left Lyndon trees. Since h can be $\Theta(n)$, this is still $\Omega(n^2)$ in the worst case. Further developing the idea of combining rotation and BBWT, we conjecture that any string can be transformed from another with the same Parikh vector, by a combination of rotation and BBWT operations, and prove the conjecture holds for binary strings and strings where all symbols are distinct.

We note that the analog of this conjecture does not hold for the standard BWT and BWT^{-1}, even for the same cases of Theorem 5. For binary strings, for example, from `aaaabbb` we cannot reach `aaababb` and vice versa using BWT, BWT^{-1}, and rotations. Similarly, for permutations, for example, from `abcd` we cannot reach `acbd` and vice versa. See Fig. 4.

Acknowledgements This work was supported by JSPS KAKENHI Grant Numbers JP24K02899 (HB), JP22K11907 (TI) and JP23H04378, JP25K21150 (DK).

Author Contributions All authors contributed significantly to the contents. H.B. and D.K. wrote the main manuscript. All authors reviewed the manuscript.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing Interests The authors declare no competing interests.

References

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

2. Badkobeh, G., Bannai, H., Köppl, D.: Bijective BWT based compression schemes. In: Lipták, Zs., de Moura, E.S., Figueroa, K., Baeza-Yates, R. (eds.) String Processing and Information Retrieval - 31st International Symposium, SPIRE 2024, Puerto Vallarta, Mexico, September 23-25, 2024, Proceedings, vol. 14899 of Lecture Notes in Computer Science, pp. 16–25. Springer (2024)
3. Badkobeh, G., Crochemore, M.: Linear construction of a left Lyndon tree. Inf. Comput. **285**(Part), 104884 (2022)
4. Bannai, H., Charalampopoulos, P., Radoszewski, J.: Maintaining the size of LZ77 on semi-dynamic strings. In: Inenaga, S., Puglisi, S.J. (eds.) 35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024, June 25-27, 2024, Fukuoka, Japan, vol. 296 of LIPIcs, pp. 3:1–3:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024)
5. Bannai, H., I, T., Inenaga, S., Nakashima, Y., Takeda, M., Tsuruta, K.: The "runs" theorem. SIAM J. Comput. **46**(5), 1501–1514 (2017)
6. Bannai, H., I, T., Nakashima, Y.: On the compressiveness of the Burrows-Wheeler transform. In 36th Annual Symposium on Combinatorial Pattern Matching (CPM 2025). Leibniz International Proceedings in Informatics (LIPIcs), vol. 331, pp. 17:1-17:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2025). https://doi.org/10.4230/LIPIcs.CPM.2025.17
7. Bannai, H., Kärkkäinen, J., Köppl, D., Piątkowski, M.: Indexing the bijective BWT. In Pisanti, N., Pissis, S.P. (eds.) 30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy, vol. 128 of LIPIcs, pp. 17:1–17:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
8. Bannai, H., Kärkkäinen, J., Köppl, D., Piątkowski, M.: Constructing and indexing the bijective and extended Burrows-Wheeler transform. Inf. Comput. **297**, 105153 (2024)
9. Bender, M.A., Farach-Colton, M., Pemmasani, G., Skiena, S., Sumazin, P.: Lowest common ancestors in trees and directed acyclic graphs. J. Algorithms **57**(2), 75–94 (2005)
10. Biagi, E., Cenzato, D., Lipták, Zs., Romana, G.: On the number of equal-letter runs of the bijective Burrows-Wheeler transform. In: Castiglione, G., Sciortino, M. (eds.) Proceedings of the 24th Italian Conference on Theoretical Computer Science, Palermo, Italy, September 13-15, 2023, vol. 3587 of CEUR Workshop Proceedings, pp. 129–142. CEUR-WS.org (2023)
11. Biagi, E., Cenzato, D., Lipták, Zs., Romana, G.: On the number of equal-letter runs of the bijective Burrows-Wheeler transform. Theo. Comput. Sci. **1027**, 115004 (2025)
12. Boucher, C., Cenzato, D., Lipták, Zs., Rossi, M., Sciortino, M.: Computing the original eBWT faster, simpler, and with less memory. In Lecroq, T., Touzet, H. (eds.) String Processing and Information Retrieval - 28th International Symposium, SPIRE 2021, Lille, France, October 4-6, 2021, Proceedings, vol. 12944 of Lecture Notes in Computer Science, pp. 129–142. Springer (2021)
13. Boucher, C., Cenzato, D., Lipták, Zs., Rossi, M., Sciortino, M.: r-indexing the eBWT. Inf. Comput. **298**, 105155 (2024)
14. Burrows, M., Wheeler, D.J.: A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California (1994)
15. Chen, K.T., Fox, R.H., Lyndon, R.C.: Free differential calculus, IV. The quotient groups of the lower central series. Annals Math. **68**(1), 81–95 (1958)
16. Gessel, I.M., Reutenauer, C.: Counting permutations with given cycle structure and descent set. J. Comb. Theory A **64**(2), 189–215 (1993)
17. Gil, J.Y., Scott, D.A.: A bijective string sorting transform (2012). CoRR arXiv:1201.3077
18. Kärkkäinen, J., Kempa, D., Piątkowski, M.: Tighter bounds for the sum of irreducible LCP values. Theor. Comput. Sci. **656**, 265–278 (2016)
19. Kempa, D., Kociumaka, T.: Resolution of the Burrows-Wheeler transform conjecture. Commun. ACM **65**(6), 91–98 (2022)
20. Kufleitner, M.: On bijective variants of the Burrows–Wheeler transform. In: Proc. PSC pp. 65–79 (2009)
21. Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE Trans. Inf. Theory **22**(1), 75–81 (1976)
22. Lyndon, R.C.: On Burnside's problem. Trans. Am. Math. Soc. **77**(2), 202–215 (1954)
23. Mantaci, S., Restivo, A., Rosone, G., Sciortino, M.: An extension of the Burrows–Wheeler transform. Theor. Comput. Sci. **387**(3), 298–312 (2007)
24. Mantaci, S., Restivo, A., Sciortino, M.: Burrows–Wheeler transform and Sturmian words. Inf. Process. Lett. **86**(5), 241–246 (2003)

25. Melançon, G.: Lyndon words and singular factors of Sturmian words. Theor. Comput. Sci. **218**(1), 41–59 (1999)
26. Mercier, L., Chassaing, P.: The height of the Lyndon tree. In: Goupil, A., Schaeffer, G. (eds.) 25th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2013), DMTCS Proceedings, pp. 957–968, Paris, France (2013). Discrete Mathematics and Theoretical Computer Science
27. Navarro, G.: Indexing highly repetitive string collections, part I: repetitiveness measures. ACM Comput. Surv. **54**(2), 29:1–29:31 (2021)
28. Navarro, G., Ochoa, C., Prezza, N.: On the approximation ratio of ordered parsings. IEEE Trans. Inf. Theory **67**(2), 1008–1026 (2021)
29. Shiloach, Y.: Fast canonization of circular strings. J. Algorithms **2**(2), 107–121 (1981)
30. Storer, J.A., Szymanski, T.G.: Data compression via textual substitution. J. ACM **29**(4), 928–951 (1982)
31. Urabe, Y., Nakashima, Y., Inenaga, S., Bannai, H., Takeda, M.: On the size of overlapping Lempel-Ziv and Lyndon factorizations. In Pisanti, N., Pissis, S.P. (eds.) 30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy, vol. 128 of LIPIcs, pp. 29:1–29:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
32. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time and linear space. Inf. Process. Lett. **6**(3), 80–82 (1977)
33. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theory **23**(3), 337–343 (1977)