View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

Indexing the Bijective BWT

Hideo Bannai
Department of Informatics, Kyushu University, Fukuoka, Japan
bannai@inf.kyushu-u.ac.jp

Juha Kärkkäinen
Helsinki Institute of Information Technology (HIIT), Finland
juha.karkkainen@cs.helsinki.fi

Dominik Köppl
Department of Informatics, Kyushu University, Japan Society for Promotion of Science (JSPS)
dominik.koeppl@inf.kyushu-u.ac.jp

Marcin Piątkowski
Nicolaus Copernicus University, Toruń, Poland
marcin.piatkowski@mat.umk.pl

Abstract

The Burrows-Wheeler transform (BWT) is a permutation whose applications are prevalent in data compression and text indexing. The *bijective BWT* is a bijective variant of it that has not yet been studied for text indexing applications. We fill this gap by proposing a self-index built on the bijective BWT. The self-index applies the backward search technique of the FM-index to find a pattern P with $\mathcal{O}(|P| \lg |P|)$ backward search steps.

2012 ACM Subject Classification Theory of computation; Mathematics of computing → Combinatorics on words

Keywords and phrases Burrows-Wheeler Transform, Lyndon words, Text Indexing

Digital Object Identifier 10.4230/LIPIcs.CPM.2019.17

Funding This research received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 690941.
Dominik Köppl: JSPS KAKENHI Grant Number JP18F18120.

1 Introduction

The Burrows-Wheeler transform (BWT) [6] is a transformation permuting all symbols of a given string T. It is obtained by sorting all cyclic rotations (conjugates) of T with respect to the lexicographical order and writing the last character of the i-th sorted cyclic rotation in a linear manner from $i = 1$ to $i = |T|$. The BWT tends to group identical characters together. All cyclic rotations of a given string share the same BWT. However, there are strings that are not the BWT of any string (e.g., `bccaab` cannot be reversed). A variant, called the bijective BWT [15], is a *bijective* transformation. It is based on the Lyndon factorization [7] of the input string. In this variant, the output consists of the last symbols of the lexicographically sorted cyclic rotations of all Lyndon factors of the input. Since the Lyndon factorization of a string is uniquely defined, the bijective BWT induces a bijection between strings of a given length n and multisets of Lyndon words of total length n.

In the following, we call the BWT *traditional* to ease the distinguishability of both transformations. It is well known that the traditional BWT has many applications in data compression [1] and text indexing [9, 10, 11]. For the latter, the algorithms for pattern searching are based on the backward search [20]: given a pattern P and the traditional BWT


© Hideo Bannai, Juha Kärkkäinen, Dominik Köppl, and Marcin Piątkowski;
licensed under Creative Commons License CC-BY

of T, the occurrences of P in a text T can be computed with $\mathcal{O}(|P|)$ backward search steps. In this light, one may ask whether it is possible to build similar index data structures by exchanging the traditional BWT with the bijective BWT.

In this article, we answer affirmatively the above question: We show that searching a pattern P on the bijective BWT can be conducted with $\mathcal{O}(|P|\hat{p})$ backward search steps, where $\hat{p}$ is the number of distinct factors in the Lyndon factorization of the longest pre-Lyndon suffix of P, where $\hat{p}$ is known to be in $\mathcal{O}(\lg|P|)$ [13]. Thus, we can reduce the number of backward search steps to $\mathcal{O}(|P|\lg|P|)$.

Our results are based on combinatoric properties of Lyndon words and the bijective BWT. They may have applications in distributed implementations of the BWT index [14] or in practical database systems storing dynamic yet compressed data [4, 5].

2 Preliminaries

Our computational model is the word RAM model with word size $\Omega(\lg n)$. Accessing a word costs $\mathcal{O}(1)$ time. We write $[\mathsf{b}(I)..\mathsf{e}(I)] = I$ for an interval I of natural numbers.

2.1 Strings

Let Σ denote a finite alphabet. We call an element $T \in \Sigma^*$ a *string*. Its length is denoted by $|T|$. Given an integer j with $1 \leq j \leq |T|$, we access the j-th character of T with $T[j]$. Concatenating a string $T \in \Sigma^*$ k times is abbreviated by T^k. A *bit vector* is a string on the binary alphabet $\{0, 1\}$.

When T is represented by the concatenation of $X, Y, Z \in \Sigma^*$, i.e., $T = XYZ$, then X, Y and Z are called a *prefix*, *substring* and *suffix* of T, respectively; A prefix X, substring Y, or suffix Z is called *proper* if $X \neq T$, $Y \neq T$, or $Z \neq T$, respectively. For two integers i, j with $1 \leq i \leq j \leq |T|$, let $T[i..j]$ denote the substring of T that begins at position i and ends at position j in T. If $i > j$, then $T[i..j]$ is the empty string. In particular, the suffix starting at position j of T is called the *j-th suffix* of T, and denoted with $T[j..]$. An occurrence of a substring S in T is treated as a sub-interval of $[1..|T|]$ such that $S = T[\mathsf{b}(S)..\mathsf{e}(S)]$.

The *longest common prefix (LCP)* of two strings S and T is the longest string that is a prefix of both S and T. The length of the LCP of two strings S and T is given by the function $\mathsf{lcp}(S, T)$ returning an integer ℓ such that $T[1..\ell] = S[1..\ell]$ and either (a) $T[\ell+1] \neq S[\ell+1]$ or (b) $\ell = \min(|T|, |S|)$ holds.

Lexicographic Order. We denote the *lexicographic order* with $\prec$. Given two string S and T, then $S \prec T$ if S is a prefix of T or there exists an integer ℓ with $1 \leq \ell \leq \min(|S|, |T|)$ such that $S[1..\ell-1] = T[1..\ell-1]$ and $S[\ell] < T[\ell]$. We write $S \prec_\omega T$ if the infinite concatenation $S^\omega := SSS\cdots$ is lexicographically smaller than $T^\omega := TTT\cdots$. For instance, $\mathtt{ab} \prec \mathtt{aba}$ but $\mathtt{aba} \prec_\omega \mathtt{ab}$.

Support Data Structures. Given a string $T \in \Sigma^*$, a character $\mathsf{c} \in \Sigma$, and an integer j, the *rank* query $T.\mathsf{rank}_\mathsf{c}(j)$ counts the occurrences of c in $T[1..j]$, and the *select* query $T.\mathsf{select}_\mathsf{c}(j)$ gives the position of the j-th c in T. We stipulate that $\mathsf{rank}_\mathsf{c}(0) = \mathsf{select}_\mathsf{c}(0) = 0$.

2.2 Lyndon Words

Given a string $T = T[1..n]$, its i-th *conjugate* $\mathsf{conj}_i(T)$ is defined as $T[i+1..n]T[1..i]$ for an integer i with $0 \leq i \leq n-1$. We say that T and every of its conjugates belongs to the *conjugate class* $\mathsf{conj}(T) := \{\mathsf{conj}_0(T), \ldots, \mathsf{conj}_{n-1}(T)\}$. If a conjugate class contains *exactly*

one conjugate that is lexicographically smaller than all other conjugates, then this conjugate is called a *Lyndon word* [16]. Equivalently, a string T is said to be a Lyndon word if and only if $T \prec S$ for every proper suffix S of T. A consequence is that a Lyndon word is border-free, i.e., there is no Lyndon word $T = SUS$ with $S \in \Sigma^+$ and $U \in \Sigma^*$. A *pre-Lyndon* word is a string that is a prefix of a Lyndon word.

The *Lyndon factorization* [7] of $T \in \Sigma^+$ is the factorization of T into a sequence of lexicographically non-increasing Lyndon words $T_1 \cdots T_t$, where (a) each $T_x \in \Sigma^+$ is a Lyndon word, and (b) $T_x \geq T_{x+1}$ for each $1 \leq x < t$.

▶ **Lemma 1** ([8, Algo. 2.1]). *The Lyndon-factorization of a string can be computed in linear time.*

Each Lyndon word T_x is called a *Lyndon factor*. We denote the multiset of T's Lyndon factors by $\mathsf{LynF}(T) := \{T_1, \ldots, T_t\}$. There is a bijection between $\mathsf{LynF}(T)$ and T in such a sense that $\mathsf{LynF}(T)$ uniquely defines T. That is because we can restore T by

1. sorting the Lyndon factors of $\mathsf{LynF}(T)$ in lexicographically descending order, and
2. subsequently concatenating them.

The last factor T_t is special, as it has the following property:

▶ **Lemma 2** ([8, Prop. 1.9]). *The last Lyndon factor of a string T is the smallest suffix of T.*

We borrow from [13, Sect. 2.2] the notation $\mathrm{lfs}_T(j) := T_j \cdots T_t$ for the suffix of T starting with the j-th Lyndon factor. For what follows, we fix a string $T[1..n]$ over an alphabet Σ with size σ. We use the string $T := \mathtt{acababdababcababbab}$ as our running example. Its Lyndon factorization is $\mathsf{LynF}(T) = \{\mathtt{ac}, \mathtt{ababd}, \mathtt{ababc}, \mathtt{ababb}, \mathtt{ab}\}$. The suffix $\mathrm{lfs}_T(4)$ is $T_4T_5 = \mathtt{ababbab}$.

2.3 Bijective Burrows-Wheeler transform

We denote the bijective BWT of T by BBWT, where $\mathsf{BBWT}[i]$ is the last character of the i-th string in the list storing the conjugates of all Lyndon factors $T_1, \ldots, T_t$ of T sorted with respect to $\prec_\omega$. Figure 1 shows BBWT for our running example.

Gill and Scott [12] and Mantaci et al. [19] postulated that BBWT can be built in linear time. However, we could only verify an algorithm running in $\mathcal{O}(n \lg n / \lg\lg n)$ time in the word RAM model with $\sigma = n^{\mathcal{O}(1)}$, provided by Bonomo et al. [3]. This algorithm processes all Lyndon factors in their lexicographical order, starting with the largest one. Since the algorithm processes all Lyndon factors in text order, we can use it to build BBWT *online*, i.e., we can build BBWT up to the Lyndon factor of a string T that starts with the longest pre-Lyndon suffix of the currently read T. That is because all previous Lyndon factors cannot change when appending characters to T.

▶ **Theorem 3.** *The* BBWT *can be constructed* online *in $\mathcal{O}(n \lg n / \lg\lg n)$ time for a string of length n.*

This is an interesting property, since the only known online technique [23, 22] for computing the traditional BWT needs the text to be given in reversed order (starting with the last character).

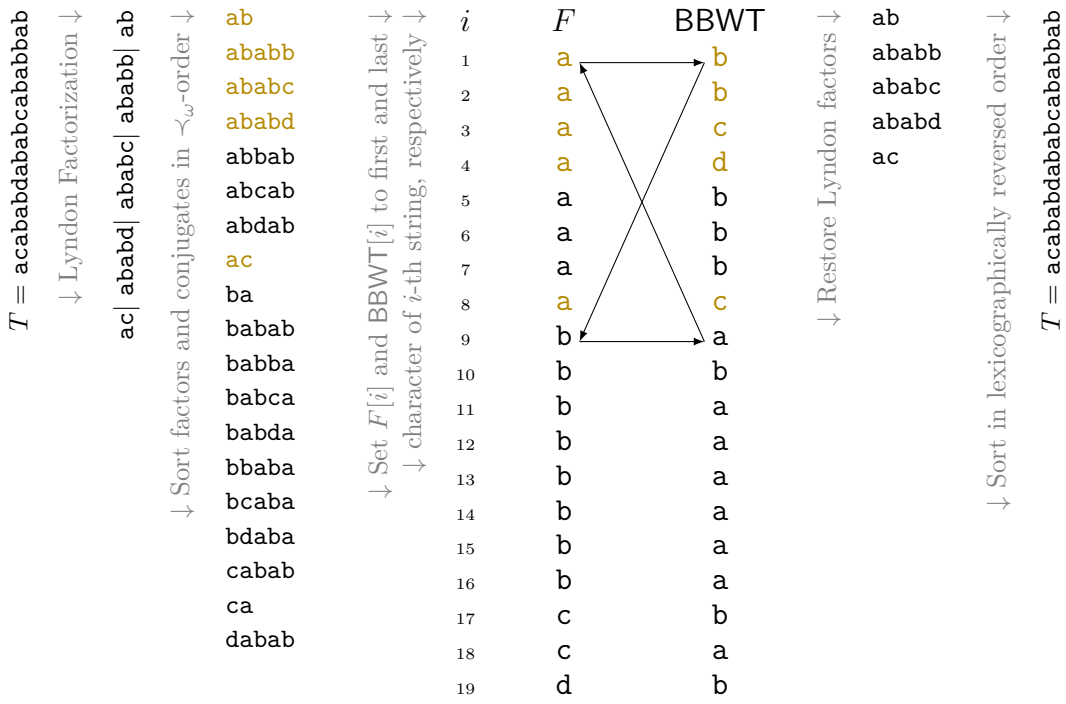

Figure 1 Constructing BBWT and restoring the original input. The Lyndon factors are colored in dark yellow. Middle: Restoring the Lyndon factor `ab` with the backward search, where the array F is defined by $F[i] := \mathtt{c}$ if $C[c-1]+1 \leq i \leq C[c]$. Right: Lyndon factors of T restored by visiting all cycles of BBWT.

3 Backward Search Algorithm

For finding patterns, our index applies the same backward search as the FM-index [9], which we briefly review. Prior to that, we define some necessary data structures:

Text Data Structures. Let SA and ISA denote the *suffix array* [18] and the *inverse suffix array* of T, respectively. The entry $\mathsf{SA}[i]$ is the starting position of the i-th lexicographically smallest suffix such that $T[\mathsf{SA}[i]..] \prec T[\mathsf{SA}[i+1]..]$ for all integers i with $1 \leq i \leq n-1$. The *Burrows-Wheeler transform (BWT)* [6] of T is the string BWT with $\mathsf{BWT}[i] = T[n]$ if $\mathsf{SA}[i] = 1$ and $\mathsf{BWT}[i] = T[\mathsf{SA}[i]-1]$ otherwise, for every i with $1 \leq i \leq n$.

FM-Index. The FM-index uses BWT with the following auxiliary data structures:

- an array C with $\sigma \lg n$ bits, where $C[c]$ is the number of occurrences of those characters in T that are smaller than c (for each character c with $1 \leq c \leq \sigma$), and
- a data structure that supports rank queries on BWT.

Given a pattern P whose characters are drawn from Σ, the occurrences of P in T are represented by $\mathsf{range}(P)$ storing an interval of SA such that $\mathsf{SA}[i]$ is a starting position of an occurrence of P for each $i \in \mathsf{range}(P)$. More formally, $\mathsf{range}(P)$ denotes the range in BWT such that

$$T[\mathsf{SA}[j]..\mathsf{SA}[j]+|P|-1] = P \text{ if and only if } j \in \mathsf{range}(P). \tag{1}$$

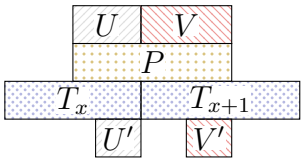

Figure 2 Setting of the proof of Lemma 6.

We obtain $R_i = \mathsf{range}(P[i..])$ from $R_{i+1} = \mathsf{range}(P[i+1..])$ with a *backward search* step

$$\mathsf{b}(R_i) = C[P[i]] + \mathsf{BWT.rank}_{P[i]}(\mathsf{b}(R_{i+1}) + 1) \text{ and } \mathsf{e}(R_i) = C[P[i]] + \mathsf{BWT.rank}_{P[i]}(\mathsf{e}(R_{i+1})). \tag{2}$$

We stipulate that the range of the empty string is $[1..n]$. Starting with the range of the empty string $\mathsf{range}(P[|P|+1..])$ and applying Equation (2) iteratively, we can find all occurrences of the pattern P in T with $|P|$ rank operations.

If we exchange BWT with BBWT, we need to take special care of a so-called *rewinding*. Suppose that we matched an occurrence of $P[i+1..]$ starting at position $j+1$ in T.

- If both text positions j and $j+1$ are contained in a Lyndon factor T_x for an integer x with $1 \le x \le t$, the backward search step

$$C[P[i]] + \mathsf{BBWT.rank}_{P[i]}(\mathsf{ISA}[i+1]) \tag{3}$$

  yields the occurrence of $P[i..]$ starting at position j in T.
- Otherwise, j and $j+1$ are contained in two different Lyndon factors. Let T_x be the Lyndon factor with $\mathsf{b}(T_x) = j+1$ (and hence the text position j is contained in T_{x-1}). Then the backward search gives $\mathsf{ISA}[\mathsf{e}(T_x)]$, i.e., the starting position of the last conjugate of T_x (in SA-order, cf. the cycle representing the Lyndon factor `ab` in Figure 1).

We call the second case *rewinding*, as the backward search counts down from the i-th conjugate to the $(i-1)$-th conjugate, but *rewinds* from the zeroth-conjugate (i.e., T_x itself) to the last conjugate. Whenever we expect that no rewinding will happen, we can find a pattern with the backward search of the FM-index:

▶ **Lemma 4.** *Given a text T and a pattern P such that each occurrence of P in T is contained in a Lyndon factor of T, we can compute these occurrences with the backward search of the FM-index on the* BBWT *with $|P|$ rank operations.*

Proof. Since all occurrences of P are contained in Lyndon factors of T, the backward search finds no occurrence of $P[i..]$ starting at the beginning $\mathsf{b}(T_x)$ of a Lyndon factor T_x in T, for $2 \le i \le |P|$ and $1 \le x \le t$. ◀

3.1 Lyndon Patterns

We first focus on the special case that the pattern itself is a Lyndon word. Subsequently, we show the general case (Section 3.2) by applying the Lyndon factorization to the pattern P and introduce an enhancement to the backward search for obtaining $\mathsf{range}(P[i..])$ from $\mathsf{range}(P[i+1..])$ in the case that the suffix $P[i+1..]$ starts with a Lyndon factor of T. For all this we need a little helper lemma:

▶ **Lemma 5** ([8, Prop. 1.10])**.** *The longest prefix of T that is a Lyndon word is the first Lyndon factor T_1 of T. Given* $\mathrm{LynF}(T) = \{T_1, \ldots, T_t\}$, $\mathrm{LynF}(T) = \{T_1\} \cup \mathrm{LynF}(T_2 \cdots T_t)$.

Figure 3 Suffix $P[i..]$ of pattern P matches the beginnings of some Lyndon factors of T. These Lyndon factors $T_x, \ldots, T_z$ are all consecutive.

▶ **Lemma 6.** *Let T be a string with* $\mathrm{LynF}(T) = \{T_1, \ldots, T_t\}$*, and let P be a pattern. If P is a Lyndon word, then there is no occurrence of P in T that crosses the border of two Lyndon factors, i.e., each occurrence of P in T is contained in a Lyndon factor T_x $(1 \leq x \leq t)$.*

Proof. Assume to the contrary that $P = UV$, where $U \in \Sigma^+$ is a suffix of T_x and $V \in \Sigma^+$ is a prefix of $T_{x+1} \cdots T_t$ for an integer x with $1 \leq x < t$. This setting is illustrated in Figure 2.

Since T_x is the longest Lyndon prefix of $T_x \cdots T_t$ (see Lemma 5), it is not possible that $U = T_x$ (otherwise we could extend T_x to UV to form a longer Lyndon word). We conclude that U is a proper suffix of T_x. Since T_x is a Lyndon word, we have $T_x \prec U'$ for every proper suffix U' of T_x (including U). This implies that $T_xV \prec U'V$, and in particular $T_xV \prec UV$. Since the pattern P is a Lyndon word, we have $V' \succ P = UV \succ T_xV$ for every suffix V' of V (including V itself).

Putting everything together, we have that T_xV is lexicographically smaller than its proper suffixes, and T_xV thus is a Lyndon word. However, this again contradicts the setting that T_x is the longest Lyndon prefix of $T_x \cdots T_t$. ◀

Combining this result with Lemma 4 yields:

▶ **Corollary 7.** *Given a pattern P that is a Lyndon word, we can find all its occurrences with $|P|$ rank operations of the FM-index built on* BBWT*.*

3.2 General Case

To find arbitrary patterns, we need to understand what happens during the rewinding. Suppose that we matched $P[i..]$ in T with the backward search. Further suppose that an occurrence of $P[i..]$ starts at position $\mathsf{b}(T_y)$ in T. Then we claim that T_y belongs to a consecutive set of Lyndon factors $T_x, \ldots, T_z$ with $x \leq y \leq z$ such that there is an occurrence of $P[i..]$ starting at position $\mathsf{b}(T_{y'})$ in T for each Lyndon factor $T_{y'}$ with $x \leq y' \leq z$. Figure 3 visualizes this setting. Assume that our claim is not true. Then there is an index y' with $x \leq y' \leq z$ and there is no occurrence of $P[i..]$ starting at position $\mathsf{b}(T_{y'})$ in T. This contradicts $T_x \succeq T_{y'} \succeq T_z$. We conclude our observation with the following lemma.

▶ **Lemma 8.** *If a string P is a prefix of T_x and T_z, then P is a prefix of T_y for each integer y with $x \leq y \leq z$.*

Suppose that we matched $P[i..]$ and that there are occurrences of $P[i..]$ starting with Lyndon factors of T. These Lyndon factors are consecutive according to Lemma 8. Let these Lyndon factors be $T_x, \ldots, T_z$. Moreover, $P[i..]$ starts with a Lyndon factor of P according to Lemma 6, i.e., $P[i..] = \mathrm{lfs}_P(w)$ for an integer w with $1 \leq w \leq p$. A further backward search step causes a rewinding for all occurrences of $P[i..]$ starting at $\mathsf{b}(T_x), \ldots, \mathsf{b}(T_z)$, where the following cases can occur:

- If $T[\mathsf{e}(T_z)] = P[i-1]$, but $T[\mathsf{b}(T_{z+1})..]$ does not have $\mathrm{lfs}_P(w)$ as a prefix, then the backward search carries on a *false occurrence*.
- If $T[\mathsf{b}(T_x) - 1] = P[i-1]$, we would expect that the backward search reports that an occurrence of $P[i-1..]$ starts at $T[\mathsf{b}(T_x) - 1]$ (we assume that $T[\mathsf{e}(T_x)] = P[i-1]$).

However, this is not the case because of the rewinding, either reporting the text position $\mathsf{e}(T_x)$ or dismissing this occurrence if $\mathsf{e}(T_x) \neq P[i-1]$. In either case, we say that there is a *missed occurrence* of $P[i-1..]$ starting at $\mathsf{b}(T_x) - 1$.

- If $T[\mathsf{e}(T_y)] = P[i-1]$ but $T[\mathsf{e}(T_{y+1})] \neq P[i-1]$ for an integer y with $x \leq y \leq z-1$, then the rewinding discards the occurrence of $P[i..]$ starting at $T[\mathsf{b}(T_{y+1})]$ although $T[\mathsf{b}(T_{y+1})-1] = T[\mathsf{e}(T_y)] = P[i-1]$. This looks like that the occurrence of $P[i..]$ starting at $T[\mathsf{b}(T_{y+1})]$ becomes a missed occurrence. However, since $T[\mathsf{b}(T_y)]$ and $T[\mathsf{b}(T_{y+1})]$ are the starting positions of occurrences of $P[i..]$, the occurrence of $P[i..]$ starting at $T[\mathsf{b}(T_y)]$ takes over the job from the occurrence starting at $T[\mathsf{b}(T_{y+1})]$ after the rewinding, i.e., we obtain the starting position $T[\mathsf{e}(T_y)] = T[\mathsf{b}(T_{y+1}) - 1]$ of the occurrence of $P[i-1..]$ after rewinding it.
- Similarly, the setting $T[\mathsf{e}(T_y)] \neq P[i-1]$ but $T[\mathsf{e}(T_{y+1})] = P[i-1]$ for an integer y with $x \leq y \leq z-1$ seems to cause a false occurrence after rewinding the occurrence of $P[i..]$ starting at $\mathsf{b}(T_y)$, but actually this occurrence takes over the job from the occurrence starting at $T[\mathsf{b}(T_{y+1})]$.
- In all other cases, for $x+1 \leq y \leq z$, $T[\mathsf{e}(T_{y-1})]\mathrm{lfs}_P(w) = T[\mathsf{e}(T_y)]\mathrm{lfs}_P(w)$, i.e., the rewound positions are beginning positions of occurrences of $P[i-1..]$, where the occurrence of $P[i..]$ starting at $\mathsf{b}(T_{y-1})$ takes the job from the occurrence starting at $\mathsf{b}(T_y)$ after the rewinding.

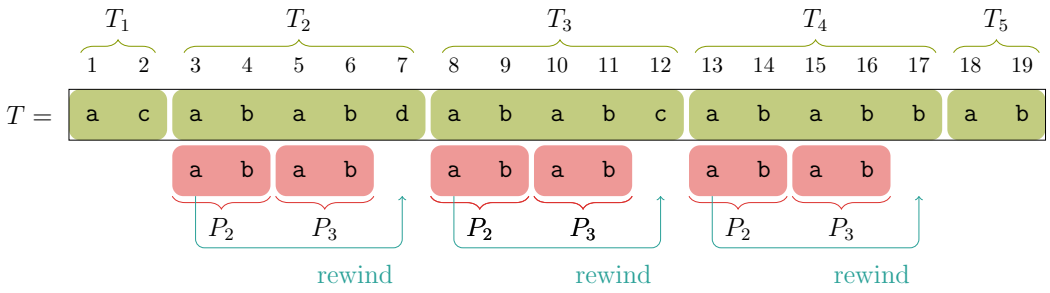

Figure 4 Backward search of a pattern P with $\mathrm{LynF}(P) = \{P_1, P_2 = \mathtt{ab}, P_3 = \mathtt{ab}\}$ in our running example $T = \mathtt{acababdababcababbab}$ after $|P_2P_3|$ steps. The sub-pattern P_2P_3 has occurrences starting at the starting positions of the Lyndon factors T_2, T_3, and T_4 of the text. The effects of the rewinding depend on P_1. If P_1 ends with $\mathtt{c}$, then we derive a *missed* occurrence from $\mathrm{lfs}_P(2)$ and T_2. If P_1 ends with $\mathtt{b}$, then we derive a *false* occurrence from $\mathrm{lfs}_P(2)$ and T_4.

In what follows, we study ways to limit the number of false and missed occurrences. We say that a false (resp. missed) occurrence of P is *derived from* P_w and T_z (resp. T_x) if it emerges on the rewinding at $T[\mathsf{b}(T_z)]$ (resp. $T[\mathsf{b}(T_x)]$). See Figure 4 for an example. According to Lemma 6 there are at most p rewindings, and hence at most p false and missed occurrences. (We lower this upper bound in the subsequent section). The false occurrences can be easily maintained in a separate list, in which each element corresponds to a false occurrence (more precisely, applying SA to such an element yields its corresponding starting position in the text). Each element of the list is subject to the backward search (Equation (3)) like the range itself (Equation (2)). Whenever a backward search step of an element of the list yields not an occurrence (e.g., we obtain the element $\mathsf{ISA}[j]$ by a backward search step from $P[i+1..]$ to $P[i..]$, but find out that $T[j] \neq P[i]$), then the false occurrence will also vanish from the range such that we no longer need to manage that element. Similarly, we keep track of the missed occurrences. For that, we take advantage of the fact that the entries of BBWT corresponding to Lyndon factors are lexicographically sorted (see the dark

yellow marked entries in Figure 1). To move from the beginning of a Lyndon factor to the end of its preceding Lyndon factor, it suffices to locate the previously larger Lyndon factor and apply a backward search step on it (to intentionally cause a rewinding). For that, we add a bit vector B_{L} marking the entries in BBWT corresponding to a Lyndon factor (and not to one of its conjugates) with '1'. Then $B_{\mathrm{L}}.\mathsf{select}_1(t-x+1)$ corresponds to T_x and the position $\mathsf{ISA}[\mathsf{b}(T_x)-1] = \mathsf{ISA}[\mathsf{e}(T_{x-1})]$ is found by applying a backward search step to $B_{\mathrm{L}}.\mathsf{select}_1(t-x)$. Again, we keep the missed occurrences in a list whose elements are (each individually) subject to the backward search. Finally, when we want to report all occurrences of the complete pattern, we take the computed range $\mathsf{range}(P)$, add all elements of the list of missed occurrences, and remove all elements of the list of the false occurrences. By doing so, we can restore the property of Equation (1). With the lists for the missed and false occurrences and the bit vector B_{L}, we can state the following theorem generalizing the backward search for arbitrary patterns.

▶ **Theorem 9.** *Given a text T and a pattern P, we can compute all occurrences of P in T with the FM-index built on* BBWT *with $\mathcal{O}(|P|p)$ rank operations, where p is the number of Lyndon factors of P.*

In the following, we improve the $\mathcal{O}(|P|p)$ bound on the number of rank operations. There is a problem with matching a pattern whose Lyndon factorization consists of the same Lyndon factor that is equal to some Lyndon factors of the text. An example for such a case is given by $T = P = \mathtt{a}^n$. Here, P has n Lyndon factors, and therefore our current upper bound on the number of rank operations stated in Theorem 9 is only $\mathcal{O}(n^2)$. Multiple occurrences of the same Lyndon factors (a) in the text as well as (b) in the pattern make the matching difficult. However, as we will see, we can cope with both individually.

First, we start with (a) the text; (b) is treated in Section 3.3. Our solution is to build the bijective BWT on all *distinct* Lyndon factors of T (along with their conjugates), remembering the number of occurrences of a Lyndon factor, such that the Lyndon factorization $T = T_1 \cdots T_t$ becomes $T = \tilde{T}_1^{\tau_1} \cdots \tilde{T}_{t'}^{\tau_{t'}}$, where $\tilde{T}_1, \cdots \tilde{T}_{t'}$ are distinct Lyndon words with $\tilde{T}_x \prec \tilde{T}_{x+1}$ for $1 \le x \le t'-1 \le t-1$, and for every $1 \le x \le t'$ it holds that (a) $\tau_j \ge 1$ and (b) there is an integer y with $y \ge x$ such that $\tilde{T}_x = T_y$. The set $\{\tilde{T}_1^{\tau_1}, \ldots, \tilde{T}_{t'}^{\tau_{t'}}\}$ is called the *composed* Lyndon factorization of T. Given $\tilde{T}_x = T_y$, we stipulate that $\mathsf{b}(\tilde{T}_x)$ is the starting position of the leftmost Lyndon factor $T_{y-\tau_x+1}$ with $T_{y-\tau_x+1} = \tilde{T}_x$. For instance, the composed Lyndon factorization of $T = \mathtt{bbabababa}$ is $T = \tilde{T}_1^2\, \tilde{T}_2^3\, \tilde{T}_3$ with $\tilde{T}_1 = \mathtt{b}$, $\tilde{T}_2 = \mathtt{ab}$, and $\tilde{T}_3 = \mathtt{a}$. The starting position $\mathsf{b}(\tilde{T}_2)$ is 3.

Now suppose that the Lyndon factor P_w occurs k_w times in $\mathsf{LynF}(P)$, and suppose that P_w is the rightmost occurrence of them, i.e., $P_{w-k_w} \neq P_{w-k_w+1} = \ldots = P_{w-1} = P_w \neq P_{w+1}$. Whenever we match $\mathrm{lfs}_P(w)$ with the beginning of the rightmost Lyndon factor T_y equal to $\tilde{T}_x$ with $|\tilde{T}_x| \le |\mathrm{lfs}_P(w)|$ occurring τ_x times in T, we can directly match $P_w^{k_w-1}\mathrm{lfs}_P(w)$ if $\tau_x \ge k_w$, skipping the backward search for $P[\mathsf{b}(P_{w-k_w+1})..\mathsf{b}(P_w)]$ such that we directly match $P[\mathsf{b}(P_{w-k_w+1})..]$ for one occurrence O starting at $T[\mathsf{b}(\tilde{T}_x)]$. For that, we assumed that $\tilde{T}_x = P_{w-j}$ for every integer j with $0 \le j \le k_w - 1$. This is true due to the following lemma:

▶ **Lemma 10.** *Given an occurrence of P_w that starts at position $\mathsf{b}(T_x)$ in T, $\mathrm{lfs}_P(w)$ is not a proper prefix of T_x if and only if $P_w = T_x$.*

Proof. Assume that $\mathrm{lfs}_P(w)$ is not a proper prefix of T_x. Switching the roles of P and T in Lemma 6, we yield that T_x cannot cross the border between P_w and P_{w+1}. Since $|\mathrm{lfs}_P(w)| \ge |T_x|$, it holds that $P_w = T_x$ (otherwise we could extend T_x to a longer Lyndon factor). ◀

The further matching of the occurrence O is conducted separately to the backward search with the range of occurrences $\mathsf{range}(P[\mathsf{b}(P_w)..])$. If $\tau_x < k_w$, then we cannot extend the currently matched occurrence, and thus can ignore to follow this occurrence. We call this technique of skipping consecutive Lyndon factors a *composed jump*.

The composed jump allows us to proceed as follows: We only count missed occurrences O that were not derived from a missed occurrence (i.e., an occurrence belonging to the range and not part of the list of missed occurrences), which we call in the following *freshly* missed occurrences. We do not count a missed occurrence O' that is derived from a missed occurrence O. Instead, we only update the position of O to O' in the list of missed occurrences. This is justified as we cannot create a freshly missed occurrence during a later backward search step:

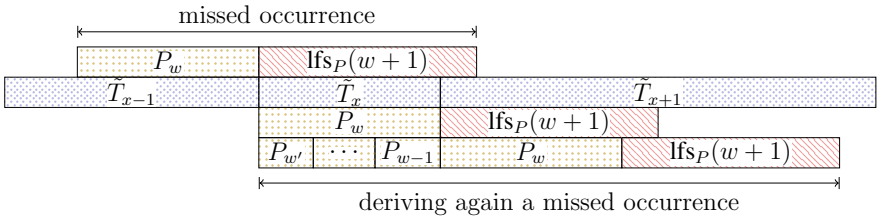

Figure 5 Setting of the proof of Lemma 11 where a false occurrence from $\mathrm{lfs}_P(w')$ and $\tilde{T}_x$ is derived after a false occurrence was derived from $\mathrm{lfs}_P(w)$ and $\tilde{T}_x$ with $w' < w$. However, this is not possible since then $\tilde{T}_{x-1} = \tilde{T}_x$.

▶ **Lemma 11.** *Let an occurrence of* $\mathrm{lfs}_P(w)$ *start at position* $\mathsf{b}(\tilde{T}_x)$ *in* T *and let* $|\tilde{T}_x| < |\mathrm{lfs}_P(w)|$. *If there is a missed occurrence derived from* $\mathrm{lfs}_P(w)$ *and* $\tilde{T}_x$, *there is no* $w' < w$ *such that* $\mathrm{lfs}_P(w')$ *and* $\tilde{T}_x$ *derive a* freshly *missed occurrence.*

Proof. By Lemma 10, $P_w = \tilde{T}_x$. Assume that there is a freshly missed occurrence derived from $\mathrm{lfs}_P(w')$ and $\tilde{T}_x$ for the largest such w' with $w' < w$. Then $P_{w'} \cdots P_{w-1} = P_w$ and $\mathrm{lfs}_P(w') = P_w\mathrm{lfs}_P(w) = P_wP_w\mathrm{lfs}_P(w+1)$. Hence, $P_w = \tilde{T}_x$ is a suffix of $\tilde{T}_{x-1}$ (cf. Figure 5). Since $\tilde{T}_{x-1}$ is a Lyndon word with $\tilde{T}_x \succeq \tilde{T}_{x-1}$, $\tilde{T}_{x-1} = \tilde{T}_x = P_w = P_{w-1}$ must hold. However, this contradicts the distinctness of $\tilde{T}_x$ in the composed Lyndon factorization. ◀

3.3 Improving the Number of Ranks

In this section, we study the case of multiple occurrences of the same Lyndon factor in the pattern to improve the bound to $\mathcal{O}(|P|p')$ rank operations, where p' is the number of *different* Lyndon factors of P. For that we show two lemmas:

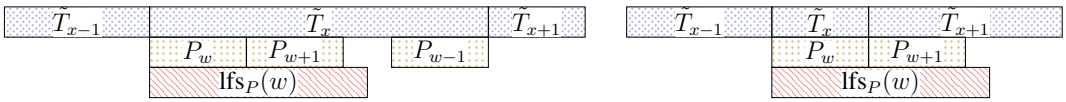

Figure 6 Setting of the proof of Lemma 12 that seems to derive a false occurrence. A necessary condition to derive a false occurrence from $\mathrm{lfs}_P(w)$ and $\tilde{T}_x$ is that $\tilde{T}_x$ is the last Lyndon factor having $\mathrm{lfs}_P(w)$ as a prefix (left). Since $\tilde{T}_x$ must be border-free, $\tilde{T}_x = P_w$ holds (right).

▶ **Lemma 12.** *If* $P_{w-1} = P_w = P_{w+1}$, *then a* false *occurrence derived from* $\mathrm{lfs}_P(w)$ *disappears after matching* $|P_w|$ *characters.*

Proof. Assume that there is a false occurrence derived from $\mathrm{lfs}_P(w)$ and $\tilde{T}_x$. Then (a) an occurrence of $\mathrm{lfs}_P(w)$ starts at position $\mathsf{b}(\tilde{T}_x)$ in T and (b) P_{w-1} is a suffix of $\tilde{T}_x$. See also Figure 6. Since a Lyndon word is border-free, $\tilde{T}_x = P_w$. However, we derived a false occurrence from $\mathrm{lfs}_P(w)$ and $\tilde{T}_x$ such that $\tilde{T}_{x+1}$ cannot start with P_{w+1}. This is a contradiction, since we found an occurrence of $\mathrm{lfs}_P(w)$ starting at position $\mathsf{b}(\tilde{T}_x)$ in T, $\tilde{T}_x = P_w$, and therefore $\tilde{T}_{x+1}$ must start with P_{w+1}. ◀

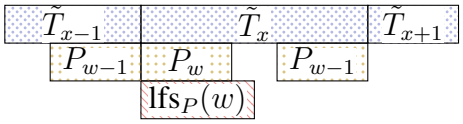

Figure 7 Setting of the proof of Lemma 13 that seems to derive a missed occurrence.

▶ **Lemma 13.** *Given an occurrence of P_w starts at position $\mathsf{b}(\tilde{T}_x)$ in T, a* missed *occurrence derived from $\mathrm{lfs}_P(w)$ and $\tilde{T}_x$ disappears after matching $|P_w|$ characters if $|\mathrm{lfs}_P(w)| \leq |\tilde{T}_x|$ and $P_{w-1} = P_w$.*

Proof. Suppose that there is a missed occurrence derived from $\mathrm{lfs}_P(w)$ and $\tilde{T}_x$. We have the following setting, which is sketched in Figure 7:

- $\tilde{T}_x$ is the leftmost Lyndon factor of the composed Lyndon factorization of T that starts with $\mathrm{lfs}_P(w)$, and
- P_{w-1} is a suffix of $\tilde{T}_{x-1}$.

$$\begin{aligned} \text{Then } P_w \preceq \mathrm{lfs}_P(w) &\preceq \tilde{T}_x && (P_w \text{ is a prefix of } \mathrm{lfs}_P(w) \text{ and } \mathrm{lfs}_P(w) \text{ is a prefix of } \tilde{T}_x) \\ &\prec \tilde{T}_{x-1} && (\text{Definition of the composed Lyndon factorization}) \\ &\preceq P_{w-1}, && (\tilde{T}_{x-1} \text{ is a Lyndon word and } P_{w-1} \text{ one of its suffixes}) \end{aligned}$$

contradicting the assumption $P_{w-1} = P_w$. ◀

A conclusion is that all longest consecutive appearances of the same Lyndon factors $P_w = \ldots = P_{w+j}$ for integers $1 \leq w \leq p$ and $j \geq 0$ can cause at most one newly missed and one false occurrence in total (which we need to keep track of). In other words, we know that we only have to care about p' freshly missed and false occurrences. Thus, we can improve the number of rank operations to $\mathcal{O}(|P|p')$.

3.4 Longest Pre-Lyndon Word

To obtain $\mathcal{O}(|P| \lg |P|)$ rank operations, we need the notion of the *longest pre-Lyndon suffix* λ_P, which is the smallest integer such that $\mathrm{lfs}_P(w+1)$ is a prefix of P_w for every $\lambda_P \leq w \leq p$. In our running example (cf. Figure 1), $\lambda_T = 4$ since T_5 is a prefix of T_4, but T_4 is not a prefix of T_3.

$$\mathrm{lfs}_P(w)c^{|X|+1} =$$

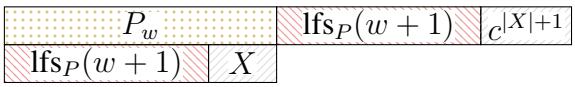

Figure 8 Setting of the proofs of Lemmas 14 and 18, where $\mathrm{lfs}_P(w+1)c^{|X|+1}$ is a Lyndon word because $\mathrm{lfs}_P(w+1)$ is a prefix of P_w and $c^{|X|+1} \succ X$.

▶ **Lemma 14.** *The string* $\mathrm{lfs}_P(w)$ *is a pre-Lyndon word for every* $\lambda_P \le w \le p$.

Proof. Since $\mathrm{lfs}_P(w+1)$ is a prefix of P_w, there is a suffix X of P_w with $P_w = \mathrm{lfs}_P(w+1)X$ (cf. Figure 8). Given a $c \in \Sigma$ with $c^{|X|+1} \succ X$, $\mathrm{lfs}_P(w)c^{|X|+1} = \mathrm{lfs}_P(w+1)X\mathrm{lfs}_P(w+1)c^{|X|+1}$ is a Lyndon word. ◀

We borrow the following facts from literature:

▶ **Lemma 15** ([13, Lemma 11])**.** P_w *is* not *a proper prefix of* $\mathrm{lfs}_P(w+1)$ *for every integer* w *with* $1 \le w \le \lambda_P - 1$.

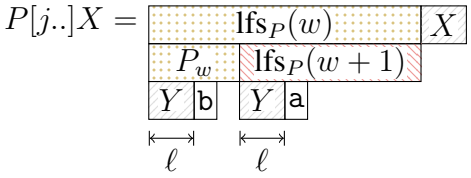

Figure 9 Setting of the proofs of Corollary 16 and Lemma 19, where a and b are characters with $\mathtt{a} \prec \mathtt{b}$, and $Y = P_w[1..\ell] = \mathrm{lfs}_P(w+1)[1..\ell]$. There is no such string X that $P[j..]X$ is a Lyndon word.

▶ **Corollary 16.** $\mathrm{lfs}_P(\lambda_P)$ *is the longest pre-Lyndon suffix of* P.

Proof. Assume that there is a longer pre-Lyndon suffix $P[j..]$. This suffix has to start with a Lyndon factor P_w for an integer w with $1 \le w \le p$, otherwise $\mathrm{lfs}_P(w) \prec P[j..]$ with w such that $\mathsf{b}(P_w) < j \le \mathsf{e}(P_w)$ (since every proper suffix of P_w is lexicographically larger than P_w), and therefore $\mathrm{lfs}_P(w)$ would be a longer pre-Lyndon suffix.

According to Lemma 15, there is an $\ell := \mathrm{lcp}(P_w, \mathrm{lfs}_P(w+1))$ with $\ell < \min(|P_w|, |\mathrm{lfs}_P(w+1)|)$. Then $P_w[\ell+1] > \mathrm{lfs}_P(w+1)[\ell+1]$ and therefore $\mathsf{conj}_{|P_w|}(\mathrm{lfs}_P(w)X) = \mathrm{lfs}_P(w+1)XP_w \prec P_w\mathrm{lfs}_P(w+1)X = \mathrm{lfs}_P(w)X$, regardless of the choice of the string X (cf. Figure 9). ◀

▶ **Lemma 17** ([13, Lemma 12])**.** $p' - \lambda_P = \mathcal{O}(\lg |P|)$ *where* p' *is the number of* distinct *Lyndon factors of* P.

The next lemmas show the usefulness of λ_P:

▶ **Lemma 18.** *Given an integer* w *with* $1 \le w \le \lambda_P - 1$, $\mathrm{lfs}_P(w+1)$ *is not a prefix of* P_w.

Proof. Assume to the contrary that $\mathrm{lfs}_P(w+1)$ is a prefix of P_w, and $P_w = \mathrm{lfs}_P(w+1)X$ for a string $X \in \Sigma^+$. Given a character $c \in \Sigma$ with $c^{|X|+1} \succ X$, $\mathrm{lfs}_P(w)c^{|X|+1} = \mathrm{lfs}_P(w+1)X\mathrm{lfs}_P(w+1)c^{|X|+1}$ is a Lyndon word, contradicting the fact that $\mathrm{lfs}_P(\lambda_P)$ is the longest pre-Lyndon word of P (cf. Corollary 16 and Figure 8). ◀

▶ **Lemma 19.** *Given an occurrence of* $\mathrm{lfs}_P(w)$ *with* $w < \lambda_P$ *that starts at position* $\mathsf{b}(\tilde{T}_x)$ *in* T *for an integer* x *with* $1 \le x \le t'$, *we have* $\tilde{T}_x = P_w$.

Proof. Since $\mathrm{lfs}_P(\lambda_P)$ is the longest pre-Lyndon suffix of P (see Corollary 16), $\mathrm{lfs}_P(w)$ is not a pre-Lyndon suffix of P. According to Lemma 18, $\mathrm{lfs}_P(w+1)$ is not a prefix of P_w. Let $\ell := \mathrm{lcp}(P_w, \mathrm{lfs}_P(w+1)) < \min(|P_w|, |\mathrm{lfs}_P(w+1)|)$. Then $P_w[\ell+1] > \mathrm{lfs}_P(w+1)[\ell+1]$ according to the Lyndon factorization of P, and therefore P_w is the longest Lyndon word of T having an occurrence that starts at position $\mathsf{b}(\tilde{T}_x)$ in T, i.e., $\tilde{T}_x = P_w$. That is because a longer Lyndon factor $\tilde{T}_x$ would contain $P_w\mathrm{lfs}_P(w+1)[1..\ell+1]$, which is lexicographically larger than $\mathsf{conj}_{|P_w|}(P_w\mathrm{lfs}_P(w+1)[1..\ell+1]) = \mathrm{lfs}_P(w+1)[1..\ell+1]P_w$ (cf. Figure 9). ◀

The above lemmas allow us to derive the following consequence:

▶ **Corollary 20.** *There is no freshly missed occurrence derived from* $\mathrm{lfs}_P(w)$ *for every integer* w *with* $w < \lambda_P$*. If one of those* $\mathrm{lfs}_P(w)$ *derives a false occurrence, then this false occurrence disappears until the range* $\mathsf{range}(P)$ *is matched.*

Proof. Suppose there is an occurrence of $\mathrm{lfs}_P(w)$ starting at position $\mathsf{b}(\tilde{T}_x)$ in T, for a composed Lyndon factor $\tilde{T}_x$ with $1 \leq x \leq t'$. According to Lemma 19, $P_w = \tilde{T}_x$. Since $w < \lambda_P \leq p$, we have that $|\tilde{T}_x| = |P_w| < |\mathrm{lfs}_P(w)|$. Then with Lemma 11 we obtain that $\mathrm{lfs}_P(w)$ and $\tilde{T}_x$ cannot derive a freshly missed occurrence. By Lemma 15, P_{w-1} is not a proper prefix of $\mathrm{lfs}_w(P)$, such that after matching $\mathsf{lcp}(P_{w-1}, \mathrm{lfs}_w(P))$ characters, a possibly derived false occurrence will disappear, and thus does not need to be tracked. ◀

With Lemma 17 we obtain:

▶ **Theorem 21.** *Given a text* T *and a pattern* P*, we can compute all occurrences of* P *in* T *with the FM-index built on the bijective BWT of the composed Lyndon factors of* T *with* $\mathcal{O}(|P| \lg |P|)$ *rank operations.*

Finally, we explain how to detect whether an occurrence of a suffix of the pattern starts at the beginning of a Lyndon factor of the text. For that, after each backward search step, we use the bit vector B_{L} introduced in Section 3.2 marking now the entries corresponding to the *composed* Lyndon factors in BBWT. If $\mathsf{range}(P[i..]) = [b..e]$ and $B_{\mathrm{L}}.\mathsf{rank}_1(e) - B_{\mathrm{L}}.\mathsf{rank}_1(b-1)$ is positive, then there is an occurrence of $P[i..]$ starting at position $\mathsf{b}(\tilde{T}_x)$ in T (after applying a composed jump) for every x with $t' - B_{\mathrm{L}}.\mathsf{rank}_1(e) + 1 \leq x \leq t' - B_{\mathrm{L}}.\mathsf{rank}_1(b-1)$. In this case, $P[i..] = \mathrm{lfs}_P(w)$ for an integer w with $1 \leq w \leq p$ due to Lemma 6.

4 Construction and Outlook

Having the backward search technique of Theorem 21, we can construct and use an online BBWT index data structure when combining the BBWT construction algorithm of Theorem 3 with a dynamic data structure for rank and select queries.

▶ **Theorem 22.** *Given a text* T *of length* n *whose characters are drawn from an alphabet of size* σ*, we can build a text index on the bijective BWT of* T *online in* $\mathcal{O}(n \lg n / \lg\lg n)$ *time. The text index supports searching for all occurrences of a pattern* P *in* $\mathcal{O}(|P| \lg |P| \lg n / \lg\lg n)$ *time. The index uses* $|\widetilde{\mathsf{BBWT}}|(H_0(\widetilde{\mathsf{BBWT}}) + H_0(B_{\mathrm{L}})) + o(n \lg \sigma) + \mathcal{O}(\sigma \lg n)$ *bits, where* $\widetilde{\mathsf{BBWT}}$ *is the bijective BWT of the composed Lyndon factorization. It returns a range and a list of positions in* SA *corresponding to starting positions of suffixes having* P *as a prefix.*

Proof. We use the dynamic representation of Navarro and Nekrich [21] for the wavelet tree of the FM-index, as well as for the bit vector B_{L}. This data structure can be constructed in $\mathcal{O}(n \lg n / \lg\lg n)$ time and can answer each type of query in optimal $\mathcal{O}(\lg n / \lg\lg n)$ time. Our space bound is due to this data structure. The array C is stored in a plain form using $\sigma \lg n$ bits. ◀

To actually report the matched positions in the text, we can use the approach of Mäkinen and Navarro [17] who apply run-length compression on the traditional BWT and store a suffix array entry for each run in the BWT, thus achieving $r \lg n$ bits additional cost for this suffix array sampling, where r is the number of runs in the BWT. It is straight-forward to adapt this technique for the bijective BWT: For that, we keep B_{L}, but run-length compress BBWT. The time bounds remain the same if $\sigma = \mathcal{O}(\lg^{\mathcal{O}(1)} n)$ [17].

Open Problems

We are unaware of an algorithm for which it is proven that it can build BBWT in linear time. It seems hard to find a relation to suffix array construction algorithms, since the context of the suffixes with respect to the lexicographic order and the context of the Lyndon words with respect to $\prec_\omega$ is different.

We are aware of a recently found redundancy [2] in the traditional BWT, and wonder whether this result translates to the bijective variant.

References

1 Donald Adjeroh, Timothy Bell, and Amar Mukherjee. *The Burrows-Wheeler Transform:: Data Compression, Suffix Arrays, and Pattern Matching.* Springer, 2008.

2 Uwe Baier. On Undetected Redundancy in the Burrows-Wheeler Transform. In *Proc. CPM*, volume 105 of *LIPIcs*, pages 3:1–3:15, 2018.

3 Silvia Bonomo, Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Sorting conjugates and Suffixes of Words in a Multiset. *Int. J. Found. Comput. Sci.*, 25(8):1161, 2014.

4 Stefan Böttcher, Alexander Bültmann, Rita Hartel, and Jonathan Schlüßler. Fast Insertion and Deletion in Compressed Texts. In *Proc. DCC*, page 393, 2012.

5 Stefan Böttcher, Alexander Bültmann, Rita Hartel, and Jonathan Schlüßler. Implementing Efficient Updates in Compressed Big Text Databases. In *Proc. DEXA*, volume 8056 of *LNCS*, pages 189–202, 2013.

6 M. Burrows and D. J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

7 Kuo Tsai Chen, Ralph H. Fox, and Roger C. Lyndon. Free differential calculus, IV. The quotient groups of the lower central series. *Annals of Mathematics*, pages 81–95, 1958.

8 Jean-Pierre Duval. Factorizing Words over an Ordered Alphabet. *J. Algorithms*, 4(4):363–381, 1983.

9 Paolo Ferragina and Giovanni Manzini. Opportunistic Data Structures with Applications. In *Proc. FOCS*, pages 390–398, 2000.

10 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005.

11 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-Time Text Indexing in BWT-runs Bounded Space. In *Proc. SODA*, pages 1459–1477, 2018.

12 Joseph Yossi Gil and David Allen Scott. A Bijective String Sorting Transform. *ArXiv 1201.3077*, 2012. `arXiv:1201.3077`.

13 Tomohiro I, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster Lyndon factorization algorithms for SLP and LZ78 compressed text. *Theor. Comput. Sci.*, 656:215–224, 2016.

14 Masaru Ito, Hiroshi Inoue, and Kenjiro Taura. Fragmented BWT: An Extended BWT for Full-Text Indexing. In *Proc. SPIRE*, volume 9954 of *LNCS*, pages 97–109, 2016.

15 Manfred Kufleitner. On Bijective Variants of the Burrows-Wheeler Transform. In *Proc. PSC*, pages 65–79, 2009.

16 R. C. Lyndon. On Burnside's Problem. *Transactions of the American Mathematical Society*, 77(2):202–215, 1954.

17 Veli Mäkinen and Gonzalo Navarro. Succinct Suffix Arrays based on Run-Length Encoding. *Nord. J. Comput.*, 12(1):40–66, 2005.

18 Udi Manber and Eugene W. Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM J. Comput.*, 22(5):935–948, 1993.

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

20 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007.

21 Gonzalo Navarro and Yakov Nekrich. Optimal Dynamic Sequence Representations. *SIAM J. Comput.*, 43(5):1781–1806, 2014.

22 Tatsuya Ohno, Yoshimasa Takabatake, Tomohiro I, and Hiroshi Sakamoto. A Faster Implementation of Online Run-Length Burrows-Wheeler Transform. In *Proc. IWOCA*, volume 10765 of *LNCS*, pages 409–419, 2017.

23 Alberto Policriti and Nicola Prezza. Computing LZ77 in Run-Compressed Space. In *Proc. DCC*, pages 23–32, 2016.