Sorting Conjugates and Suffixes of Words in a Multiset*

Silvia Bonomo, Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, Marinella Sciortino
University of Palermo, Dipartimento di Matematica e Informatica,
Via Archirafi 34, 90123 Palermo, ITALY.
{sabrina,restivo,giovanna,mari}@math.unipa.it

In this paper we are interested in the study of the combinatorial aspects related to the extension of the Burrows-Wheeler transform to a multiset of words. Such study involves the notion of suffixes and conjugates of words and is based on two different order relations, denoted by $<_{lex}$ and $\prec_\omega$, that, even if strictly connected, are quite different from the computational point of view. In particular, we introduce a method that only uses the $<_{lex}$ sorting among suffixes of a multiset of words in order to sort their conjugates according to $\prec_\omega$-order. In this study an important role is played by Lyndon words. This strategy could be used in applications specially in the field of Bioinformatics, where for instance the advent of "next-generation" DNA sequencing technologies has meant that huge collections of DNA sequences are now commonplace.

Keywords: Lyndon words; Burrows-Wheeler Transform; Extended Burrows-Wheeler Transform; Circular words; Conjugates; Suffixes; Sorting.

1. Introduction

The Burrows-Wheeler Transform (`bwt`) [5, 1] is a classical tool introduced in Data Compression, which finds application in many fields of research. The `bwt` of a word w is a word $\texttt{bwt}(w)$ obtained by a letters permutation of w induced by the list of the conjugates of w sorted in lexicographical order (denoted by $<_{lex}$). Actually, in several implementations of the `bwt` in order to improve the efficiency of the algorithm, an end-marker symbol is appended to the end of the word and $\texttt{\$-bwt}(w)$ is derived by the sorting the suffixes of $w\$$. In this way, the computation of $\texttt{\$-bwt}(w)$ can be reduced to the suffix array (SA) construction. The SA of a word w is defined as the permutation of integers giving the starting positions of the suffixes of $w\$$ in lexicographical order.

In the last years, a strong interest has been devoted to find strategies that can handle collections of words. For instance, the advent of "next-generation" DNA sequencing technologies has meant that collections of DNA sequences are now commonplace in Bioinformatics. Since, many algorithms and data structures for com-

*Electronic version of an article published as International Journal of Foundations of Computer Science, 25:08, 2014, 1161-1175 https://doi.org/10.1142/S0129054114400309 ©2014 World Scientific Publishing Company https://www.worldscientific.com/doi/10.1142/S0129054114400309. Partially supported by Italian MIUR Project PRIN 2010LYA9RH, "Automi e Linguaggi Formali: Aspetti Matematici e Applicativi".

pression and analysis of a text have the Burrows-Wheeler transform at their heart, it would be of great interest to explore their applications to string collections such as these.

Recently, an extension of the `bwt` to a multiset of words has been introduced. In [16], the authors give an extension of the `bwt` (called `ebwt`) to a multiset $\mathcal{S}$ of words on the alphabet Σ, suggested by the remark given in [6], that the `bwt` coincides with a particular case of a bijection defined in [10] by Gessel and Reutenauer. The `ebwt` of a multiset $\mathcal{S}$ is a word obtained by a letters permutation of the words in $\mathcal{S}$ induced by the sorting of the conjugates of words in $\mathcal{S}$ according with an order relation (denoted by $\prec_\omega$) defined by using lexicographic order among infinite words. The `ebwt` is a reversible transformation, that has been used for circular words comparison [18, 20] and for circular pattern matching [13]. It is also used as preprocessing of compressors on a single text, where the `ebwt` is applied to the words obtained by a factorization of the text. For instance, in [16], the input is factorized into blocks of equal length, whereas in [12, 14] the Lyndon factorization of the text is used.

Unfortunately, the efficiency of `ebwt` computation is spoiled by the sorting step of the conjugates (i.e. $\prec_\omega$-order). For this reason other variants of the `ebwt` have been considered. In such approaches, the input is modified in order to use the $<_{lex}$-order among suffixes instead of $\prec_\omega$-order among conjugates. For instance, a straightforward strategy consists in concatenating all words of $\mathcal{S}$ separated by ordered and distinct end-markers $\$_i \notin \Sigma$ (for $i = 1, \ldots, m$). The `$-bwt` is then applied to the obtained word. Another strategy (called `bcr`) has been given in [3], where ordered and distinct end-markers are appended to the end of each word and the output `bcr`($\mathcal{S}$) is obtained by lexicographically sorting the suffixes of such words. However, in both cases, some negative aspects can be remarked. Firstly, as consequence of the sorting of suffixes of $\mathcal{S}$, the circular nature of the `ebwt` is destroyed. Moreover, the introduction of the end-markers increases both the size of the input and the cardinality of the alphabet. Finally, the output is strongly affected by the order of the end-markers.

The goal of the paper is to introduce a method that only uses the lexicographic sorting among suffixes of a multiset of words in order to sort their conjugates according to $\prec_\omega$-order without appending any end-marker.

In particular, Section 2 is devoted to introduce some preliminary notions. Section 3 contains the definition of the Burrows-Wheeler Transform, whereas in Section 4 we discuss the extension of Burrows-Wheeler Transform to a multiset of words. In Section 5 we show how the $\prec_\omega$-sorting of the conjugates of two Lyndon words and the lexicographic sorting among their suffixes are related. In this study an important role is played by the notion of Lyndon word. In Section 6 we describe an example of a possible strategy for the construction of an algorithm that, by using the results of Section 5, produces the `ebwt` by implicitly finding the $\prec_\omega$ order among all the conjugates of a multiset of lexicographically sorted Lyndon words by a suitable comparison among the suffixes of such words.

A preliminary version of some results of the present paper are contained in [4].

2. Preliminaries

Let $\Sigma = \{c_1, c_2, \ldots, c_\sigma\}$ be a finite alphabet with $c_1 < c_2 < \ldots < c_\sigma$. Given a finite word $w = a_1 a_2 \cdots a_n$, $a_i \in \Sigma$ for $i = 1, \ldots, n$, a *factor* of w is written as $w[i,j] = a_i \cdots a_j$. A factor $w[1,j]$ is called a *prefix*, while a factor $w[i,n]$ is called a *suffix*. In this paper, we also denote by $suf_k(w)$ (resp. $pref_k(w)$) the suffix (resp. prefix) of w that has length k.

We say that $x, y \in \Sigma^*$ are *conjugate* or *y is a conjugate of x* if $x = uv$ and $y = vu$ for some $u, v \in \Sigma^*$. Recall that conjugacy is an equivalent relation. We denote by $conj_k(w)$ the conjugate of w starting from the position $|w| - k + 1$, i.e. if $w = xy$ and $|y| = k$ then $yx = conj_k(w)$. A word $v \in \Sigma^*$ is *primitive* if $v = u^n$ implies $v = u$ and $n = 1$. In what follows we only deal with primitive words.

If u is a word in Σ^* we denote by u^ω the infinite word obtained by infinitely iterating u, i.e. $u^\omega = uuuuu\cdots$. The usual lexicographic order $<_{lex}$ between finite words can be naturally extended to infinite words. If u and v are primitive words, we say that $u \preceq_\omega v \Leftrightarrow u^\omega <_{lex} v^\omega$. Note that this order can also be defined for non-primitive words, but such a case is not considered in this paper.

Although the $\prec_\omega$ order of u and v is defined by using infinite words, the following theorem (cf. [17]), that is a consequence of Fine and Wilf theorem in [9], shows that this order can be established with a bounded number of symbol comparison.

Theorem 1. *Let u, v be two primitive words over a finite alphabet Σ. If $k = |u| + |v| - \gcd(|u|, |v|)$ we have that:*

$$u \prec_\omega v \Leftrightarrow pref_k(u^\omega) <_{lex} pref_k(v^\omega).$$

The following example shows that the $<_{lex}$ order can be different from the $\prec_\omega$ order when one word is a prefix of the other one and also that the bound is tight.

Example 2. *Consider $u = abaab$ and $v = abaababa$. Although $u <_{lex} v$, we have $v \prec_\omega u$. Moreover u^ω and v^ω differ for the character in position $k = 12 = 5 + 8 - 1$. Remark that, for any $h < k$, $pref_h(u^\omega) = pref_k(v^\omega)$, i.e. k is tight.*

$$\overbrace{abaab}^{u}\,\overbrace{abaab}^{u}\,\overbrace{a\boldsymbol{b}\cdots}^{u}$$
$$\underbrace{abaababa}_{v}\,\underbrace{aba\boldsymbol{a}\cdots}_{v}$$

A *Lyndon* word is a primitive word which is also the minimum in its conjugacy class, with respect to the $<_{lex}$ order. In [15, 7], one can find a linear algorithm that for any word $w \in \Sigma^*$ computes the Lyndon word of its conjugacy class.

3. Burrows-Wheeler transform

The Burrows-Wheeler Transform (`bwt`) [5] is intuitively described as follows: given a word $w \in \Sigma^*$, $\texttt{bwt}(w)$ is a word obtained by sorting (with respect to the $<_{lex}$

order) the list of the conjugates of w and giving as output the concatenation of the last symbol of each conjugate in the sorted list. With the additional information of the position of w in the sorted list (i.e. the index I), the `bwt` becomes an invertible transformation, i.e., we can recover w from `bwt`(w). For instance, one can verify that the output of `bwt` of $w = abraca$ is the word $bwt(w) = caraab$ and $I = 2$.

The heaviest step in terms of time complexity for the computation of `bwt` of a word w is the sorting of the conjugates of w. In order to speed up the computation, an approximation of the `bwt` of a word w is given (cf. [5]), where an end-marker symbol \$ is added to the end of w; it is also assumed that \$ does not occur in the input alphabet and it is alphabetically smaller than any symbol of the alphabet. In order to avoid confusion, in what follows we refer to this transform as `$-bwt`. Because the assumption on end-marker symbol reduces the sorting of the conjugates of $w\$$ to the sorting of its suffixes, `$-bwt`(w) is computed by sorting only the suffixes of $w\$$. For instance, `$-bwt`$(abraca) = ac\$raab$. Note that the addition of the end-marker is essential, otherwise the transformation is not reversible. Moreover in order to recover the word w from `$-bwt`(w), we do not need the additional information of the position of w in the sorted list of the suffixes.

Remark that, although `$-bwt` is still a reversible transformation, it gives in general a different output than `bwt` (for instance `bwt`$(abraca) = caraab$). In [11], the authors show that also the computation of `bwt`(w) can be reduced to the sorting of *only* suffixes of w when w is a Lyndon word. The presence of the end marker in fact forces the word $\$w$ to be the Lyndon word in its conjugacy class.

Notice that, for each pair of mutually conjugate words u and v, `bwt`$(u) =$ `bwt`(v) We remark that this property does not hold if the sorting of the suffixes is used. As a consequence `$-bwt`(u) and `$-bwt`(v) are in general different. This highlights the fact that `bwt` is a transformation on circular words, whereas `$-bwt` is defined on linear words.

In the next section, we analyze the extension of the Burrows-Wheeler Transform to a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of m words. Actually, nowadays, the `$-bwt` is also used to manage collections. In order to do this, one needs to concatenate all the words separated by distinct end-markers $\$_i \notin \Sigma$ (for $i = 1, \ldots, m$) and then compute the `$-bwt` of the word $\bar{s}$ obtained in this way. By the definition of `$-bwt`, one also needs to append the end-marker $\$ \notin \Sigma$ at the end of the concatenated word. It is also assumed that $\$ < \$_i < \$_j < a$ for any $a \in \Sigma$ and $i, j = 1, \ldots, m$ with $i < j$.

As a consequence of the use the end-markers, such strategy could have the drawback that if the words in the collection are concatenated in two different orders, the output of the `$-bwt` could be different.

4. Generalizations of `bwt` to a multiset of words

In this section we recall the definition of an extension of the Burrows-Wheeler transform (called `ebwt`) to a multiset $\mathcal{S}$ of primitive words that, as the original `bwt`,

Sorted infinite powers $<_{lex}$-order			Sorted conjugates $\prec_\omega$-order
$a\ b\ a\ c\ a\ b \cdots$		$\rightarrow$	$a\ b\ a\ \mathbf{c}$
$a\ b\ c\ a\ b\ c \cdots$			$a\ b\ \mathbf{c}$
$a\ b\ c\ b\ a\ b \cdots$			$a\ b\ c\ \mathbf{b}$
$a\ c\ a\ b\ a\ c \cdots$			$a\ c\ a\ \mathbf{b}$
$a\ c\ b\ a\ c\ b \cdots$			$a\ c\ \mathbf{b}$
$b\ a\ b\ c\ b\ a \cdots$			$b\ a\ b\ \mathbf{c}$
$b\ a\ c\ a\ b\ a \cdots$	$\Longrightarrow$		$b\ a\ c\ \mathbf{a}$
$b\ a\ c\ b\ a\ c \cdots$			$b\ a\ \mathbf{c}$
$b\ c\ a\ b\ c\ a \cdots$		$\rightarrow$	$b\ c\ \mathbf{a}$
$b\ c\ b\ a\ b\ c \cdots$			$b\ c\ b\ \mathbf{a}$
$c\ a\ b\ a\ c\ a \cdots$			$c\ a\ b\ \mathbf{a}$
$c\ a\ b\ c\ a\ b \cdots$			$c\ a\ \mathbf{b}$
$c\ b\ a\ b\ c\ b \cdots$		$\rightarrow$	$c\ b\ a\ \mathbf{b}$
$c\ b\ a\ c\ b\ a \cdots$		$\rightarrow$	$c\ b\ \mathbf{a}$

Fig. 1. The output is the pair $(\texttt{ebwt}(\mathcal{S}), \mathcal{I})$ where $\texttt{ebwt}(\mathcal{S}) = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

$\texttt{bcr}(\mathcal{S})$	Sorted suffixes
c	$\$_1$
b	$\$_2$
a	$\$_3$
a	$\$_4$
c	$a\ \$_3$
b	$a\ \$_4$
b	$a\ b\ \$_2$
$\$_1$	$a\ b\ a\ c\ \$_1$
b	$a\ c\ \$_1$
a	$b\ \$_2$
c	$b\ a\ \$_4$
c	$b\ a\ b\ \$_2$
a	$b\ a\ c\ \$_1$
$\$_3$	$b\ c\ a\ \$_3$
a	$c\ \$_1$
b	$c\ a\ \$_3$
$\$_4$	$c\ b\ a\ \$_4$
$\$_2$	$c\ b\ a\ b\ \$_2$

Fig. 2. The output is $\texttt{bcr}(\mathcal{S}) = cbaacbb\$_1bacca\$_3ab\$_4\$_2$.

is defined on conjugates (see [16, 17]). The `ebwt` has been inspired by a remark in [6], where the authors pointed out that the `bwt` coincides with a particular case of a bijection defined in [10]. Actually `ebwt` is somehow the algorithmic version of the general transformation given by Gessel and Reutenauer in [10].

We also describe other transformations that generalize the Burrows-Wheeler transform to a multiset of words, that make use of the $<_{lex}$ sorting of the suffixes instead than the $\prec_\omega$ order among conjugates used for the construction `ebwt`. We highlight advantages and weakness of each of these transformations.

4.1. *The Extended Burrows-Wheeler Transform*

Here, we describe a transformation that generalizes the Burrows-Wheeler Transform to a multiset of circular words. Given a multiset $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of m words, the output of the Extended Burrows-Wheeler transform (`ebwt`) is obtained by sorting the list of conjugates of $\mathcal{S}$ according to the $\prec_\omega$ order relation, and it is defined as the pair $(\texttt{ebwt}(\mathcal{S}), \mathcal{I})$, where $\texttt{ebwt}(\mathcal{S})$ is the concatenation of the last symbol of each conjugate in the sorted list and $\mathcal{I}$ consists of the set of indices representing the position in the sorted list of the original words in $\mathcal{S}$.

Example 3. *Let $\mathcal{S} = \{abac, cbab, bca, cba\}$. As shown in Figure 1, the output of the Extended Burrows-Wheeler transform is the pair consisting of* $\texttt{ebwt}(\mathcal{S}) = ccbbbcacaaabba$ *and $\mathcal{I} = \{1, 9, 13, 14\}$.*

If we do not care about the indices, we can associate to each word $\texttt{ebwt}(\mathcal{S}) \in \Sigma^*$ a unique multiset of conjugacy classes of primitive words in Σ^*, i.e. a unique multiset of circular words. For this reason, the set $\mathcal{I}$ can be neglected when we are interested to the circular words.

4.2. *A variant of the ebwt with end-markers*

The hardest computational step of ebwt consists in the $\prec_\omega$ sorting the conjugates of words in $\mathcal{S}$. In fact, the use of the $\prec_\omega$ order overloads the already heavy process of the lexicographic sorting of the conjugates. A naive implementation of ebwt can be found in the original paper [17], and a more precise implementation is given in [13]. By a similar approach used to speed-up the bwt computation, in [2, 3] the authors defined a variant of ebwt (denoted by bcr), where ordered and distinct end-markers are appended to the end of each word. In particular, if $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, $\texttt{bcr}(\mathcal{S}) = \texttt{ebwt}(\{S_1\$_1, S_2\$_2, \ldots, S_m\$_m\})$. This trick allows to reduce the $\prec_\omega$ sorting of the *conjugates* to the $<_{lex}$ order of the *suffixes* of such words. Then the use of distinct end-markers guarantees the reversibility of the transformation. We represent in Figure 2 the sorted list of the suffixes of the multiset $\mathcal{S}$ used in Example 3 and $\texttt{bcr}(\mathcal{S}) = cbaacbb\$_1bacca\$_3ab\$_4\$_2$.

Notice that this approach would require m additional distinct end-markers and this could be not supportable in many applications where many millions of words are processed. For this reason, the authors use the same end-marker \$ for all words and they establish an implicit order among them.

As in the case of a single word, the use of end-markers destroys the circular nature of the ebwt as shown in the following example.

Example 4. *Consider the set* $\mathcal{S} = \{acbcc, aaacab\}$*, and the other set* $\mathcal{S}' = \{bccac, abaaac\}$ *obtained by replacing each word in* $\mathcal{S}$ *with one of its conjugates, we have* $\texttt{ebwt}(\mathcal{S}) = \texttt{ebwt}(\mathcal{S}') = bacacacacab$*, whereas* $\texttt{bcr}(\mathcal{S}) = cb\$_2aca\$_1accaab \neq \texttt{bcr}(\mathcal{S}') = ccba\$_2caa\$_1aacb$.

The use of *distinct* end-markers implies that the multiset of words becomes an ordered collection, and then the output of bcr depends on the position of the words in the collection (or, in other words, by the alphabetic order of the end-markers) as shown in the following examples.

Example 5. *The output* $\texttt{bcr}(\mathcal{S})$ *differs from the output* $\texttt{ebwt}(\mathcal{S})$ *for a number of symbols that does not only depend on the number of words in the collection. For instance, let* $\mathcal{S} = \{a, aaaab, aaabb, aabab, aabbb, ababb, abbbb, b\}$*. We consider the following permutation of the sequences in* $\mathcal{S}$*:* $\mathcal{S}' = \{b, ababb, abbbb, a, aaaab, aabbb, aabab, aaabb\}$*. We obtain* $\texttt{ebwt}(\mathcal{S}) = \texttt{ebwt}(\mathcal{S}') = (ab)^{16}$*, whereas*

$$\texttt{bcr}(\mathcal{S}) = abbbbbbb\$_1\$_2a\$_3a\$_4a\$_5aba\$_6aba\$_7ababbb\$_8aaabababa$$

and

$$\texttt{bcr}(\mathcal{S}') = bbbabbbb\$_4\$_5a\$_8a\$_7a\$_6aba\$_2baa\$_3\$_1bbababaaabbabaa.$$

In conclusion, we have shown that $\texttt{bcr}(\mathcal{S})$ is affected by the order in which the words appear in the collection $\mathcal{S}$, whereas the $\texttt{ebwt}(\mathcal{S})$ is invariant with respect to it. Moreover ebwt is a transformation on conjugacy classes, i.e. on circular words, whereas `bcr`$(\mathcal{S})$ is a transformation on linear words. For these reasons `bcr` could not be a good alternative to `ebwt`.

Unfortunately, as remarked before, the bottleneck of `ebwt` is the sorting of conjugates, that is a very time-expensive step. Since the sorting of suffixes of words in $\mathcal{S}$ can be performed more efficiently than the sorting of their conjugates, one might ask whether, as in the case of a single word, the use of the Lyndon words allows to reduce the sorting of the conjugates of more than two words to the sorting of the corresponding suffixes. The following example shows that the two sorting are different.

Example 6. *Let $\mathcal{S} = \{aa\underline{dabc}, abbabc\underline{d}, abdacb\underline{d}\}$. The underlined suffix $\underline{dabc}$ of S_1 is lexicographically greater than both the underlined suffix $\underline{d}$ of S_2 and the underlined suffix $\underline{d}$ of S_3, but if we consider the conjugates starting with these underlined suffixes, the conjugate $\underline{dabc}aa$ of S_1 is greater than the conjugate $\underline{d}abbabc$ of S_2, but it is smaller than the conjugate $\underline{d}abdacb$ of S_3.*

Nevertheless, in the next sections we show a strategy to obtain the $\prec_\omega$ sorting of the conjugates from the $<_{lex}$ sorting of the suffixes.

5. Comparing conjugates and suffixes of Lyndon words

In this paper we are interested in relating the $<_{lex}$ order of the suffixes of a multiset of words and the $\prec_\omega$ order of their conjugates. In particular, in this section we consider the special case in which the multiset consists of two Lyndon words T_1 and T_2. The results stated in this section will be used to give a more general solution in the next section.

The case of a single word is synthesized in the following lemma.

Lemma 7. *Let T be a Lyndon word. For any integers h, k with $1 \leq h, k \leq |T|$, the following statements are equivalent:*

i) $conj_h(T) <_{lex} conj_k(T)$;
ii) $conj_h(T) \prec_\omega conj_k(T)$;
iii) $suf_h(T) <_{lex} suf_k(T)$.

Proof. The equivalence between *i)* and *ii)* is a trivial consequence of the fact that, for words of the same length, the $<_{lex}$ order coincides with the $\prec_\omega$ order. The equivalence between *i)* and *iii)* is an easy consequence of the properties of Lyndon words (cf. [11]). □

Consider now two distinct Lyndon words T_1 and T_2. We first show that the $<_{lex}$ order and the $\prec_\omega$ order coincide for Lyndon words.

Theorem 8. *Let T_1 and T_2 be two Lyndon words, then $T_1 \leq_{lex} T_2$ if and only if $T_1 \prec_\omega T_2$.*

Proof. ($\Rightarrow$) Let $T_1 \leq_{lex} T_2$. If $T_1 = T_2$ then $T_1 \prec_\omega T_2$ by definition. Consider $T_1 \neq T_2$. If T_1 is a prefix of T_2, then $T_2 = T_1 y$, where y is a non-empty word. From $T_1 <_{lex} T_2$ and $T_2 <_{lex} y$ (since T_2 is a Lyndon word) we get $T_1^2 <_{lex} T_1 y$, i.e. $T_1 \prec_\omega T_2$. If T_1 is not a prefix of T_2, then obviously $T_1 \prec_\omega T_2$.
($\Leftarrow$) Let $T_1 \prec_\omega T_2$. By contradiction, we suppose $T_2 <_{lex} T_1$. By the first part of the proof we would have $T_2 \prec_\omega T_1$, a contradiction. □

The following lemmas take into consideration the generic conjugates $conj_h(T_1)$ and $conj_k(T_2)$ of two distinct Lyndon words.

Lemma 9. *Let T_1 and T_2 be two distinct Lyndon words. If $T_1 <_{lex} T_2$ and $h \leq k$ then*

$$conj_h(T_1) \prec_\omega conj_k(T_2) \Leftrightarrow suf_h(T_1) \leq_{lex} suf_k(T_2).$$

Proof. Let $u = suf_h(T_1)$ and $v = suf_k(T_2)$. If u is not prefix of v, one symbol distinguishes u from v and there is nothing to prove. If u is a prefix of v we have $v = uv'$, $v' \in \Sigma^*$. If $v' = \varepsilon$ (i.e., $h = k$), then one has $conj_k(T_1) \prec_\omega conj_k(T_2) \Leftrightarrow uT_1^\omega <_{lex} uT_2^\omega \Leftrightarrow T_1 \prec_\omega T_2 \Leftrightarrow T_1 <_{lex} T_2$. Obviously $suf_h(T_1) \leq_{lex} suf_h(T_2)$.

If $v' \neq \varepsilon$ (i.e., $h < k$), u is a proper prefix of v, then we can write $T_1 = su$ and $T_2 = tuv'$. One can see that, since $T_1 <_{lex} T_2$ and by Theorem 8, $T_1^\omega <_{lex} T_2^\omega$. Moreover, as T_2 is a Lyndon word, it follows that $T_2 <_{lex} v'T_2$. Then $T_1 <_{lex} v'T_2$. So $T_1^\omega <_{lex} v'T_2^\omega$. As u is a prefix of v, then $suf_h(T_1) \leq_{lex} suf_k(T_2)$. Moreover, $conj_h(T_1) \prec_\omega conj_k(T_2) \Leftrightarrow uT_1^\omega <_{lex} uv'T_2^\omega \Leftrightarrow T_1^\omega <_{lex} v'T_2^\omega$, then the thesis follows. That is $u \leq_{lex} v \Leftrightarrow suf_h(T_1) \leq_{lex} suf_k(T_2) \Leftrightarrow conj_h(T_1) \prec_\omega conj_k(T_2)$. □

The following example shows that if $k < h$ and $suf_k(T_2)$ is a prefix of $suf_h(T_1)$, the relative order of the suffixes is not sufficient to establish the $\prec_\omega$ order between the conjugates (cf. also Example 6).

Example 10. *Let $\Sigma = \{a, b, c\}$, $T_1 = aacab$ and $T_2 = acbcc$, where $T_1 \prec_\omega T_2$. We consider the following conjugates: $conj_3(T_1) = cabaa$, $conj_1(T_2) = cacbc$. In this case we have $suf_1(T_2) <_{lex} suf_3(T_1)$ and $conj_3(T_1) \prec_\omega conj_1(T_2)$. Consider now the Lyndon words $T_1 = aacab$ and $T_2 = aacbc$, where $T_1 \prec_\omega T_2$. We consider the following conjugates: $conj_3(T_1) = cabaa$, $conj_1(T_2) = caacb$. In this case we have $suf_1(T_2) <_{lex} suf_3(T_1)$ and $conj_1(T_2) \prec_\omega conj_3(T_1)$.*

Lemma 11. *Let T_1 and T_2 be two distinct Lyndon words. If $T_1 <_{lex} T_2$ and $h > k$, then*

$$conj_h(T_1) \prec_\omega conj_k(T_2) \Leftrightarrow suf_h(T_1) \leq_{lex} suf_k(T_2) pref_{h-k}(T_2^\omega).$$

Proof. If $suf_k(T_2)$ is not a prefix of $suf_h(T_1)$ then there exists a symbol at position $t \leq k$ that distinguishes $suf_h(T_1)$ from $suf_k(T_2)$ hence $conj_h(T_1) \prec_\omega conj_k(T_2) \Leftrightarrow suf_h(T_1) <_{lex} suf_k(T_2)$. Since $t \leq k$, $suf_h(T_1) \leq_{lex} suf_k(T_2)$ is equivalent to $suf_h(T_1) \leq_{lex} suf_k(T_2) pref_{h-k}(T_2^\omega)$. If $suf_k(T_2)$ is a proper prefix of $suf_h(T_1)$,

then we pose $v = suf_k(T_2)$ and $vu = suf_h(T_1)$, with $u \neq \varepsilon$. In order to compare the corresponding conjugates according with $\prec_\omega$ relation, we need to compare vuT_1^ω with vT_2^ω, that is uT_1^ω with T_2^ω. Two cases can occur:
Case 1. There exists a character in u that allows to distinguish u from T_2^ω. So the problem of conjugates comparison is solved in at most $|u|$ symbol comparisons, i.e. $conj_h(T_1) \prec_\omega conj_k(T_2) \Leftrightarrow suf_{h-k}(T_1) \leq_{lex} pref_{h-k}(T_2^\omega)$. Since v is a common prefix of length k, this relation is equivalent to $suf_h(T_1) \leq_{lex} suf_k(T_2)pref_{h-k}(T_2^\omega)$.
Case 2. The word u is a prefix of T_2^ω. Therefore, let l be the smallest integer such that $T_2^l = uz$. If $z \neq \varepsilon$, we have that $T_2 <_{lex} z$ (since T_2 is a Lyndon word) and $T_1 <_{lex} T_2$ by hypothesis. Then we can state, without any additional comparison that $conj_h(T_1) \prec_\omega conj_k(T_2)$ and $suf_{h-k}(T_1) \leq_{lex} pref_{h-k}(T_2^\omega)$. If z is the empty word, then $T_2^l = u$ means that we have to compare $T_2^l T_1^\omega$ with $T_2^l T_2^\omega$, that is T_1^ω with T_2^ω. So $conj_h(T_1) \prec_\omega conj_k(T_2)$ and $suf_{h-k}(T_1) <_{lex} pref_{h-k}(T_2^\omega)$.
In both cases, since v is a common prefix, the relation $suf_h(T_1) \leq_{lex} suf_k(T_2)pref_{h-k}(T_2^\omega)$ holds. □

Remark that, both in Lemma 9 and in Lemma 11 we need at most h symbol comparisons in order to establish the $\prec_\omega$ order between $conj_h(T_1)$ and $conj_k(T_2)$. Hence, from previous lemmas one can derive the following theorem that considers two arbitrary primitive words.

Theorem 12. *Let u and v be primitive words, let T_u and T_v be their corresponding Lyndon words. Let suppose $T_u <_{lex} T_v$ and let r be the integer such that $u = conj_r(T_u)$. Then*

$$u \prec_\omega v \Leftrightarrow pref_r(u^\omega) \leq_{lex} pref_r(v^\omega).$$

Proof. We let s denote the integer such that $v = conj_s(T_v)$. We can distinguish the cases $r \leq s$ and $r > s$. If $r \leq s$, by Lemma 9 it follows that $conj_r(T_u) \prec_\omega conj_s(T_v) \Leftrightarrow suf_r(T_u) \leq_{lex} pref_r(suf_s(T_v))$. The thesis follows because $suf_r(T_u) = pref_r(u^\omega)$ and $pref_r(suf_s(T_v)) = pref_r(v^\omega)$. If $r > s$, by Lemma 11 we have that $conj_r(T_u) \prec_\omega conj_s(T_v) \Leftrightarrow suf_r(T_u) \leq_{lex} suf_s(T_v)pref_{r-s}(T_v^\omega)$. Since $suf_r(T_u) = pref_r(u^\omega)$ and $suf_s(T_v)pref_{r-s}(T_v^\omega) = pref_r(v^\omega)$, the thesis follows. □

The theorem states that we can determine the $\prec_\omega$ order of two primitive words u and v only by looking at the first r characters of the infinite words u^ω and v^ω: if there is a mismatch within the first r characters, then the $\prec_\omega$ order is determined by the order of the letters corresponding to such a mismatch; otherwise, the $\prec_\omega$ order is decided by the order of the corresponding Lyndon words T_u and T_v.

Remark that even if r can be much smaller than $|u|+|v|-gcd(|u|,|v|)$, Theorem 12 does not contradict the tightness of the bound given in Theorem 1, since here we have the supplementary information on the order of the corresponding Lyndon words. Consider, for instance, the words in Example 2, $u = abaab$ and $v = abaababa$. The corresponding Lyndon words are $T_u = aabab$ and $T_v = aabaabab$. We have that

$T_v <_{lex} T_u$, $u = conj_2(T_u)$ and $v = conj_7(T_v)$. Consider the infinite words

$$u^\omega = abaababaabab\cdots$$
$$v^\omega = abaababaabaa\cdots$$

The first mismatch between u^ω and v^ω is in the position 12. Nevertheless since $pref_7(v^\omega) = pref_7(u^\omega)$, by Theorem 12 we can conclude that $v \prec_\omega u$.

Theorem 12 suggests that, in order to establish the $\prec_\omega$ order of two primitive words u and v, one could use the following procedure:first determine the $<_{lex}$ order of T_u and T_v, and then decide the $\prec_\omega$ order by looking only the first r characters of u^ω and v^ω. Anyway the above example shows that the total number of comparisons with this procedure is the same as the one given by the bound in Theorem 1. In the above example we need 5 comparisons in order to state that $T_v <_{lex} T_u$ and 7 comparisons to state that $v \prec_\omega u$, i.e. 12 comparisons, which is the same number of comparisons that we need in order to find a mismatch between u^ω and v^ω.

However, such a strategy takes advantage with respect the usual comparison between conjugates when more than two pairs of conjugates have to be compared. In fact in this case the cost of comparing the Lyndon words is amortized on the total number of comparisons among conjugates. Such a consideration is enforced when multisets of words are considered. This is a starting point for the algorithm proposed in the next section.

6. Sorting the conjugates of a multiset of Lyndon words

In this section we assume that $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$ is a lexicographically sorted multiset of Lyndon words. Such an hypothesis, although strong, is not restrictive because if $\mathcal{S}$ is a generic multiset of primitive words we can obtain $\mathcal{T}$ in linear time by computing for each word the corresponding Lyndon conjugate (cf. [15]), and then by sorting this multiset. Moreover $\texttt{ebwt}(\mathcal{T}) = \texttt{ebwt}(\mathcal{S})$.

As shown in previous section, the hypothesis that the elements of $\mathcal{T}$ are Lyndon words suggests to connect the problem of the $\prec_\omega$ sorting of the conjugates to the lexicographic sorting of the suffixes of the elements of $\mathcal{T}$. In this connection, the results of the previous section also show an asymmetry in the roles played by the Lyndon words involved into the sorting.

Our strategy analyzes the conjugates of the words in $\mathcal{T}$ starting from the greatest Lyndon word to the smallest one. Due the circular nature of the conjugacy relation, we assume that both $conj_0(T_i)$ and $conj_{|T_i|}(T_i)$ denote the word T_i.

For $i = m, \ldots, 1$, we denote by $B_i(\mathcal{T})$ the array of characters such that, for each position γ, $B_i(\mathcal{T})[\gamma]$ contains the last character of the γ-th smallest conjugate in the $\prec_\omega$ sorted list of conjugates after that all the conjugates of $T_i, T_{i+1}, \ldots, T_m$ have been processed. Moreover, for each h ranging from 0 to $|T_i| - 1$, we let $B_i^h(\mathcal{T})$ denote the partial array of $B_i(\mathcal{T})$ obtained when the position of $conj_h(T_i)$ in the $\prec_\omega$ sorted list has been determined. Remark that the concatenation of the symbols in $B_1(\mathcal{T})$ coincides with the $\texttt{ebwt}(\mathcal{S}) = \texttt{ebwt}(\mathcal{T})$. So, we can consider the arrays $B_i(\mathcal{T})$ as partial $\texttt{ebwt}$'s.

Our algorithm consists of m steps and at each step i (from m down to 1) the partial `ebwt` $B_i(\mathcal{T})$ is computed. During each step i the partial `ebwt` is incrementally built by considering all the conjugates of the word T_i from the rightmost to the leftmost one, and by adding to the partial `ebwt` computed in the previous steps the symbol associated with the current conjugate according with the $\prec_\omega$-order. Such insertion does not affect the relative order of the symbols already inserted in the partial `ebwt`. In this way the arrays $B_i^h(\mathcal{T})$ are built.

More formally, at each step i from m down to 1 we consider the word T_i and we distinguish two different phases: 1. determining the position of the word $T_i = conj_0(T_i) = conj_{|T_i|}(T_i)$ into the sorted partial list; 2. determining the position of the conjugate $conj_r(T_i)$, with $r = 1, \ldots, |T_i| - 1$ into the sorted partial list.

In the algorithm we also use the following notations. Let w be a word. For any character $x \in \Sigma$, let $C(w, x)$ denote the number of characters in w that are smaller than x, and let $rank(w, x, t)$ denote the number of occurrences of x in $pref_t(w)$. Such functions have been introduced in [8] for the FM-index. The function NEW takes as input an array T and a position r and allocates a new cell in T at the position r. Figure 3 gives a sketch of the algorithm BUILD_EBWT($\mathcal{T}$).

```
Algorithm 13.   BUILD_EBWT($\mathcal{T}$);
$B_{m+1} = NULL$;
$i = m$;
for each Lyndon word $T_i \in \mathcal{T}$, for $i = m, \ldots, 1$ {
    $B_i = B_{i+1}$;
    NEW($B, 1$); $B_i[1] = T_i[|T_i|]$;
    $prevPos = 1$; $prevSymb = B_i[1]$;
    for each conjugate $conj_h(T_i)$ for $h = 1, \ldots, |T_i| - 1$ {
        $\gamma = C[prevSymb] + rank(B_i, prevPos, prevSymb) + 1$;
        NEW($B, \gamma$); $B_i[\gamma] = conj_h[|T_i|]$;
        $prevPos = \gamma$; $prevSymb = B_i[\gamma]$;
    }
}
```

Fig. 3. Construction of the `ebwt` of a multiset of lexicographically sorted Lyndon words.

Theorem 14. *Given the multiset $\mathcal{T}$ of lexicographically sorted Lyndon words, the algorithm* BUILD_EBWT($\mathcal{T}$) *correctly constructs* $ebwt(\mathcal{T})$.

The proof of the theorem can be deduced by the following lemmas. Such lemmas show, for each step, the correctness of the two phases above described.

Lemma 15. *For each step i from m to 1, the position of symbol $T_i[|T_i|]$ in the partial* `ebwt` *$B_{i+1}(\mathcal{T})$ is* 1.

In the phase 2 of each step i we have to establish the position in the list where we have to insert the last symbol of the conjugate $conj_h(T_i)$, with $h \geq 1$. Clearly, the conjugate $conj_h(T_i)$ is greater than all conjugates that starting with a symbol y smaller than the initial of $conj_h(T_i)$. The order between two conjugates $conj_h(T_i)$ and $conj_k(T_j)$ starting with the same symbol $a \in \Sigma$ is established by using the (already known) order between the conjugates $conj_{h-1}(T_i)$ and $conj_{k-1}(T_j)$.

Lemma 16. *For each step i from m to 1 and for $h = 1, \dots, |T_i| - 1$ the position of the last symbol of the conjugate $conj_h(T_i)$ in the array $B_i^h(\mathcal{T})$ is*

$$\gamma = C(B_i^{h-1}(\mathcal{T}), x) + rank(B_i^{h-1}(\mathcal{T}), x, t) + 1. \qquad (1)$$

where x is the first character of $conj_h(T_i)$ and t is the position in $B_i^{h-1}(\mathcal{T})$ of the last symbol of the conjugate $conj_{h-1}(T_i)$.

Note that the algorithm BUILD_EBWT($\mathcal{T}$) could be adapted in order to compute the so called *conjugate array* of $\mathcal{T}$, i.e. the list of the positions of all the conjugates of the words of $\mathcal{T}$ sorted according to the $\prec_\omega$ relation (cf. [4]).

In the following, we describe an instance of running of the algorithm.

Let $\mathcal{T} = \{aaacab, acbcc\}$ an ordered collection of Lyndon words. It can be represented by a right justified table in which the rows are the Lyndon words and the columns represent the starting points of the conjugates of the words. Such a table is depicted in Figure 4.

We describe the last step of the algorithm BUILD_EBWT($\mathcal{T}$), i.e. we have already computed $B_2(\mathcal{T})$ in which the symbols associated with the conjugates of T_2 have been inserted with respect to the $\prec_\omega$ relation. We have to compute $B_1(\mathcal{T}) = B(\mathcal{T})$. Firstly we pose $B_1(\mathcal{T}) = B_2(\mathcal{T})$. So the first phase of the step requires to insert the symbol associated with the Lyndon word T_1 into $B_1(\mathcal{T})$. We know that T_1 is smaller than all conjugates of T_2, so we have to insert b at the first position of $B_1(\mathcal{T})$ (see Lemma 15). We describe below the sorted list of conjugates for sake of clarity in the description of the algorithm. In order to make visible the index h of the conjugate $conj_h(T) = yx$ of $T_i = xy$, we write in italic the characters of y and in boldface the characters of x.

B_2	Sorted conjugates $\prec_\omega$-order
c	$a\ c\ b\ c\ c$
c	$b\ c\ c\ \mathbf{a}\ \mathbf{c}$
c	$c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}\ \mathbf{c}$
a	$c\ b\ c\ c\ \mathbf{a}$
b	$c\ c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}$

B_1	Sorted conjugates $\prec_\omega$-order
b	$a\ a\ a\ c\ a\ b$
c	$a\ c\ b\ c\ c$
c	$b\ c\ c\ \mathbf{a}\ \mathbf{c}$
c	$c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}\ \mathbf{c}$
a	$c\ b\ c\ c\ \mathbf{a}$
b	$c\ c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}$

The second phase requires to determine the correct position where the symbols associated with the other conjugates $conj_h(T_1)$ should be inserted with respect to the $\prec_\omega$ sorting. For instance, when $h = 1$, $conj_1(T_1) = aacaba$, from Lemma 16 we have that the previous symbol (stored in *prevSymb*) is b, the previous position (stored in *prevPos*) is 1 and $\gamma = C(B_1(\mathcal{T}), b) + rank(B_1(\mathcal{T}), b, 1) + 1 = 1 + 1 + 1 = 3$, so we have to insert a at the third position, as depicted below.

	6	5	4	3	2	1
1	a	a	a	c	a	b
2		a	c	b	c	c

Fig. 4. Table representing $\mathcal{T} = \{aaacab, acbcc\}$.

B_1	Sorted conjugates $\prec_\omega$-order
b	$a\ a\ a\ c\ a\ b$
c	$a\ c\ b\ c\ c$
c	$b\ c\ c\ \mathbf{a}\ \mathbf{c}$
c	$c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}\ \mathbf{c}$
a	$c\ b\ c\ c\ \mathbf{a}$
b	$c\ c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}$

B_1	Sorted conjugates $\prec_\omega$-order
b	$a\ a\ a\ c\ a\ b$
c	$a\ c\ b\ c\ c$
a	$b\ \mathbf{a}\ \mathbf{a}\ \mathbf{a}\ \mathbf{c}\ \mathbf{a}$
c	$b\ c\ c\ \mathbf{a}\ \mathbf{c}$
c	$c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}\ \mathbf{c}$
a	$c\ b\ c\ c\ \mathbf{a}$
b	$c\ c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}$

The final result is $B_1(\mathcal{T}) = \texttt{ebwt}(\mathcal{S}) = bacacacacab$, as depicted below.

B_1	Sorted conjugates $\prec_\omega$-order
b	$a\ a\ a\ c\ a\ b$
a	$a\ a\ c\ a\ b\ \mathbf{a}$
c	$a\ b\ \mathbf{a}\ \mathbf{a}\ \mathbf{a}\ \mathbf{c}$
a	$a\ c\ a\ b\ \mathbf{a}\ \mathbf{a}$
c	$a\ c\ b\ c\ c$
a	$b\ \mathbf{a}\ \mathbf{a}\ \mathbf{a}\ \mathbf{c}\ \mathbf{a}$
c	$b\ c\ c\ \mathbf{a}\ \mathbf{c}$
a	$c\ a\ b\ \mathbf{a}\ \mathbf{a}\ \mathbf{a}$
c	$c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}\ \mathbf{c}$
a	$c\ b\ c\ c\ \mathbf{a}$
b	$c\ c\ \mathbf{a}\ \mathbf{c}\ \mathbf{b}$

With reference to the computational complexity of the algorithm we can formulate the following theorem.

Theorem 17. *Let $\mathcal{T} = \{T_1, \ldots, T_m\}$ a lexicographically sorted multiset of Lyndon words and let n be the size of $\mathcal{T}$ (i.e. $n = \sum_{i=1}^{m} |T_i|$)). The algorithm* Build_EBWT($\mathcal{T}$) *runs in $O(n(t_1 + t_2))$ where t_1, t_2 are the cost of the operations insertion and rank in the array $B_i(\mathcal{T})$, respectively.*

One can deduce from the previous theorem that the complexity of the algorithm depends on the suitable data structures used for the rank operations and for the insertion operations. Navarro and Nekrich's recent result [19] on optimal representations of dynamic sequences shows that one can insert symbols at arbitrary positions and compute the rank function in the optimal time $O(\frac{\log n}{\log \log n})$ within essentially $nH_0(s) + O(n)$ bits of space for a sequence s of length n, where $H_0(s)$ denotes the 0-th order empirical entropy of s. Moreover, it is possible to give also an external memory implementation of the algorithm Build_EBWT($\mathcal{T}$) by adapting the strategy used in [3]. In this case the used memory is negligible and the time complexity depends on the time of I/O operations.

References

[1] D. Adjeroh, T. Bell, and A. Mukherjee. *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching.* Springer, 2008.

[2] M. J. Bauer, A. J. Cox, and G. Rosone. Lightweight BWT construction for very large string collections. In *CPM*, volume 6661 of *LNCS*, pages 219–231. Springer, 2011.

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

[4] S. Bonomo, S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. Suffixes, Conjugates and Lyndon words. In *DLT*, volume 7907 of *LNCS*, pages 131–142. Springer, 2013.

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

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

[7] J.-P. Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363 – 381, 1983.

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

[9] N. J. Fine and H. S. Wilf. Uniqueness theorem for periodic functions. *Proc. Amer. Math. Soc.*, (16):109–114, 1965.

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

[11] R. Giancarlo, A. Restivo, and M. Sciortino. From first principles to the Burrows and Wheeler transform and beyond, via combinatorial optimization. *Theoret. Comput. Sci.*, 387(3):236 – 248, 2007.

[12] J. Y. Gil and D. A. Scott. A bijective string sorting transform. *CoRR*, abs/1201.3077, 2012.

[13] W.-K. Hon, T.-H. Ku, C.-H. Lu, R. Shah, and S. V. Thankachan. Efficient Algorithm for Circular Burrows-Wheeler Transform. In *CPM*, volume 7354 of *LNCS*, pages 257–268. Springer, 2012.

[14] M. Kufleitner. On bijective variants of the Burrows-Wheeler transform. pages 65–79, 2009.

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

[16] S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. An Extension of the Burrows Wheeler Transform and Applications to Sequence Comparison and Data Compression. In *CPM*, volume 3537 of *LNCS*, pages 178–189. Springer, 2005.

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

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

[19] G. Navarro and Y. Nekrich. Optimal dynamic sequence representations. In *SODA 2013*, ACM-SIAM Symposium on Discrete Algorithms, pages 865–876. SIAM, 2013.

[20] L. Yang, G. Chang, X. Zhang, and T. Wang. Use of the Burrows-Wheeler similarity distribution to the comparison of the proteins. *Amino Acids*, 39(3):887–898, 2010.