

When a Dollar in a Fully Clustered Word Makes a BWT

Sara Giuliani^{1,*}, Zsuzsanna Lipták¹ and Francesco Masillo¹

¹University of Verona, Italy

Abstract

The Burrows-Wheeler Transform (BWT) is a powerful transform widely used in string compression and string processing. It produces a reversible permutation of the characters of the input string, often allowing for easier compression of the string, while also enabling fast pattern matching. While the BWT is defined for every word, not every word is the BWT of some word. A characterization of BWT images was given in [Likhomanov and Shur, CSR, 2011], based on the standard permutation of the string.

Often an end-of-file character \$ is added to mark the end of a string. Given a string w , it is an interesting combinatorial question where a \$ can be inserted to make w the BWT of some string $v\$$. This question was answered in [Giuliani et al. Theor. Comput. Sci. 2021], where an efficient algorithm was presented for computing all such positions (called *nice positions*), and a characterization of nice positions was given, based on *pseudo-cycles* in the standard permutation of w .

In this paper, we give a stronger characterization of nice positions: We show that these can be characterized using only *essential* pseudo-cycles, which constitute a small subset of all possible pseudo-cycles. We present an algorithm to compute all essential pseudo-cycles of a word w .

In the second part of the paper, we study nice positions of *fully clustered words*: these are words w whose number of runs equals the number of distinct characters occurring in w . Fully clustered words are of particular interest due to their extreme compressibility, and words whose BWT is fully clustered have been studied extensively. We are interested in the number of nice positions of fully clustered words, as well as in the number of fully clustered words with k nice positions, for fixed k and word length.

Keywords

combinatorics on words, Burrows-Wheeler-Transform, permutations, string algorithms, fully clustered words


Proceedings of the 23rd Italian Conference on Theoretical Computer Science, Rome, Italy, September 7-9, 2022

*Corresponding author.

✉ sara.giuliani_01@univr.it (S. Giuliani); zsuzsanna.liptak@univr.it (Zs. Lipták); francesco.masillo@univr.it (F. Masillo)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

1. Introduction

The Burrows-Wheeler Transform (BWT) [1] is a widely used reversible transformation on strings, which is central in the context of lossless compression and string processing. It is a permutation of the input word that tends to group together equal characters. Usually, the more repetitive the string, the longer the groups of consecutive equal characters, called *runs*. This is known as *clustering effect* [2], and allows for easier compression of the input [3, 4]. Due to its power with very repetitive strings and to its reversibility, the BWT is fundamental in fields where large amounts of textual data need to be processed, such as bioinformatics, where BWT-based tools are among those most frequently used [5, 6, 7, 8]. Understanding the underlying combinatorics is crucial in applications, since they are necessary for advancements on data structures based on the BWT, such as the FM-index [9] or the more recent *r*-index [10].

The BWT has also been studied intensively from a combinatorics on words point of view, with [11] showing that it was a special case of the Gessel-Reutenauer bijection [12]. Many papers examined the clustering effect of the BWT, most recently [13, 14, 15, 16], while other transforms with similar properties were investigated in [17, 18, 19]. While the BWT is defined for every word, not every word is the BWT of some word; characterizations of BWT images are known [20, 21].

In this paper, we continue the study of the following combinatorial problem, introduced in [22]: Given a word w , where, if at all, can a sentinel character $\$$ be inserted into w to make it the BWT of some word ending with $\$$? We call such positions *nice positions*. The question is motivated by the fact that in the application context, often such a sentinel character $\$$ is appended to mark the end of the input string.

In [22], an $\mathcal{O}(n \log n)$ -time algorithm for computing all nice positions was given, as well as a characterization using so-called *pseudo-cycles*. In this paper, we give a stronger characterization of nice positions, restricting the set of pseudo-cycles to a subset which can be exponentially smaller than the complete set. We also present an $\mathcal{O}(n^2)$ -time algorithm that computes nice positions via this restricted set of pseudo-cycles. Although the new algorithm is slower than the previous one in the worst case, it gives more information in output, thus opening up new possibilities of insights into the problem.

We then apply this algorithm to the study of nice positions in *fully clustered words*. These are words whose number of equal-letter-runs equals the number of distinct characters, such as $w = \text{bbbbcaa}$. Words whose BWT is fully clustered are of special interest because of their high compressibility via the BWT; e.g. such a word is abbacbb , with BWT w . Over a binary alphabet, it is known that a word has fully clustered BWT if and only if it is conjugate of a standard word or of a power of a standard word [20]. Over a ternary alphabet, there exists a characterization via morphisms, of words whose BWT has the form $c^i b^j a^k$ (there called *Type I*) [23], while a combinatorial property (circular palindromic richness) was shown to be a necessary but not sufficient condition for such words [24, 25]. This result extends to larger alphabets, i.e. words whose BWT has the form $a_\sigma^{n_\sigma} a_{\sigma-1}^{n_{\sigma-1}} \dots a_1^{n_1}$, with alphabet $\Sigma = \{a_1, a_2, \dots, a_\sigma\}$ and $a_1 < a_2 < \dots < a_\sigma$. Finally, a characterization of words with fully clustered BWT, over arbitrary alphabets, was given in terms of discrete interval exchanges in [26].

We present both theoretical and experimental results on nice positions in fully clustered words, as well as on the number of words with k nice positions, for fixed k . We studied binary

and ternary alphabets. Due to space restrictions, we present only the results on the binary alphabet, as well as having to omit some proofs. Both will be included in the full version.

Overview of paper: In Section 2, we introduce the necessary notation and basic facts. In Section 3, we give the new stronger characterization of nice positions, and in Section 3.1, we describe the algorithm exploiting the new characterization. In Section 4, we present some properties and conjectures related to nice positions for fully clustered words for the unary and binary alphabets. We close with a brief outline of future work in Section 5.

2. Preliminaries

Strings and permutations. Given a finite ordered alphabet Σ , a *string* (or *word*) over Σ is a finite sequence $w = w_0 \cdots w_{n-1}$ of characters from Σ . We denote by σ the size of Σ , by $|w| = n$ the length of a word $w_0 \cdots w_{n-1}$, and by ε the empty word, the only string of length 0. Note that we index strings from 0. We denote by $\text{alph}(w) = \{a \mid \text{there exists } 1 \leq i \leq |w| : w_i = a\}$ the set of characters occurring in w . For two words u, v , uv denotes their concatenation, given by $uv = u_0 \cdots u_{|u|-1} v_0 \cdots v_{|v|-1}$. If $w = xvy$, then x is called a *prefix*, v a *substring*, and y a *suffix* of w . Given a word u and a positive integer c , $u^c = u \cdots u$ denotes the c -fold concatenation of u . A word v is called a *power* if it can be written as $v = u^c$ for some $c > 1$, otherwise it is called *primitive*. Often a *sentinel character*, denoted $\$$, is appended to mark the end of a string, where $\$$ does not occur elsewhere in the string, and is assumed to be smaller than all $a \in \Sigma$. Clearly, every word of the form $v\$$ is primitive.

Two words w, w' are called *conjugates* (or *rotations*) if there exist u, v , possibly empty, such that $w = uv$ and $w' = vu$. Given a word $w = w_0 \cdots w_{n-1}$, the i 'th rotation of w , for $0 \leq i \leq n-1$, is $w_i \cdots w_{n-1} w_0 \cdots w_{i-1}$. A word of length n is primitive if and only if it has n distinct conjugates. The set of all words over Σ is totally ordered by the *lexicographic order*: Let $v, w \in \Sigma^*$, then $v \leq w$ if v is a prefix of w , or there exists an index j s.t. for all $i < j$, $v_i = w_i$, and $v_j < w_j$ according to the order on Σ .

A *run* in a word w is a maximal substring consisting of the same character. A *fully clustered word* is a word w with $|\text{alph}(w)|$ runs. Given a word $w = b_0^{i_0} b_1^{i_1} \cdots b_{r-1}^{i_{r-1}}$ with r runs, we define its *pattern* as $\text{pat}(w) = b_0 b_1 \cdots b_{r-1}$. For example, the fully clustered word $w = \text{bbccccca}$ has pattern $\text{pat}(w) = \text{bca}$.

Let n be a positive integer. A *permutation* π of n is a bijection from the set $\{0, \dots, n-1\}$ to itself. A common way to represent permutations is the two-line notation: $\pi = \begin{pmatrix} 0 & 1 & \dots & n-1 \\ \pi(0) & \pi(1) & \dots & \pi(n-1) \end{pmatrix}$. (Note that we index permutations from 0.) A *cycle* of π is a minimal subset C of the elements of the permutation such that $\pi(C) = C$. A cycle of length 1 is called a *fixpoint*, one of length 2 is called a *transposition*. It is a basic result on permutations that every permutation can be uniquely decomposed into disjoint cycles, giving rise to another common representation of permutations, the *cycle representation*. For example, $\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 0 & 5 & 4 & 1 \end{pmatrix}$ has cycle representation $(0\ 2)(1\ 3\ 5)(4)$, where $(0\ 2)$ is a transposition, $(1\ 3\ 5)$ is a cycle of length 3, and (4) is a fixpoint. A permutation that consists of one cycle only is called *cyclic*. For more details on permutations, see [27]. Given a word w , the *standard permutation* of w , denoted π_w , is the permutation defined by: $\pi_w(i) < \pi_w(j)$ if and only if either $w_i < w_j$, or $w_i = w_j$ and $i < j$. For example, the standard permutation of banana is $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 3 & 0 & 4 & 1 & 5 & 2 \end{pmatrix} = (0\ 3\ 1)(2\ 4\ 5)$.

Burrows-Wheeler-Transform. Given a word v of length n , the Burrows-Wheeler-Transform (BWT) [1] of v is the concatenation of the final characters of its rotations in lexicographic order. This is often visualized using the so-called BW-matrix, consisting of the lexicographically sorted rotations of v : $\text{bwt}(v)$ is the last column of this matrix, read from top to bottom. For example, the BWT of banana is nbbaaa . It follows from the definition that two words u, v have the same BWT if and only if they are conjugates.

While the BWT is defined for every word, not every word w is the BWT of some word. Likhomanov and Shur have shown that a word is the BWT of some word v if and only if the number of cycles of its standard permutation equals the greatest common divisor of the runlengths of w [21] (see also [20, 23] for earlier, more restricted versions of this statement). For example, the standard permutation of nbbaaa is $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 3 & 0 & 1 & 2 \end{pmatrix} = (0\ 4\ 1\ 5\ 2\ 3)$. It has one cycle and this equals $\gcd(2, 1, 3)$; indeed, nbbaaa is the BWT of the word banana. Conversely, banana's standard permutation has two cycles, but the \gcd of its runlengths is 1, and therefore, it is not the BWT of any word.

Nice positions and pseudo-cycles. When is a word the BWT of some word of the form $v\$,$ where $\$$ is the sentinel character? It is clear that such a word must have exactly one occurrence of $\$$. Thus, it follows from the result of Likhomanov and Shur that its standard permutation must be cyclic.

The following question was studied in [22]: Given a word w , where can $\$$ be inserted into w such that the resulting word is the BWT of some $v\$$? Such positions are called *nice*, formally: Given $w = w_0 \cdots w_{n-1}$, a position i is *nice*, $0 \leq i \leq n$, if the $(n+1)$ -length word w' is the BWT of some word of the form $v\$,$ where $w'_j = w_j$ for $j < i$, $w'_i = \$$, and $w'_j = w_{j-1}$ for $j \geq i$. For example, the word nbbaaa has two nice positions, 1 and 3: $\text{n\$nbbaaa} = \text{bwt}(\text{abanan\$})$ and $\text{nnb\$aaaa} = \text{bwt}(\text{anaban\$})$, while banana has none: nowhere can a $\$$ be inserted to make it the BWT of some word.

Let π be a permutation of n . A non-empty subset $S \subseteq \{0, 1, \dots, n-1\}$ is called a *pseudo-cycle* if it can be partitioned into two sets S_{left} and S_{right} , where $S_{\text{left}} < S_{\text{right}}$ (elementwise), such that $\pi(S) = \{x-1 \mid x \in S_{\text{left}}\} \cup S_{\text{right}}$. The *critical interval* of pseudo-cycle S is $R_S = [a+1, b]$, where $a = \max S_{\text{left}}$ and $b = \min S_{\text{right}}$. If S_{left} is empty, we set $a = -1$, and if S_{right} is empty, we set $b = n$. An example of a pseudo-cycle is given in Fig. 1. When $\$$ is inserted in the critical interval, S becomes a cycle, hence the resulting standard permutation is not cyclic. The following characterization of nice positions was given in [22]:

Theorem 1 (Thm. 6 in [22]). *Let w be a word of length n over Σ , and $0 \leq i \leq n$. Then i is nice if and only if there is no pseudo-cycle S w.r.t. the standard permutation π_w whose critical interval contains i .*

We will denote by R_w the *critical set* of w , the union of the critical intervals of all pseudo-cycles. By Thm. 1 then i is nice if and only if $i \notin R_w$. We will further say that pseudo-cycle S *blocks* position i if $i \in R_S$. We will need one more result from [22]. Recall that here we index permutations from 0, while in [22], they are indexed from 1.

Lemma 1 (Thm. 5 in [22]). *Let w be the BWT of some word, and c the number of cycles of π_w . Then c is a nice position.*

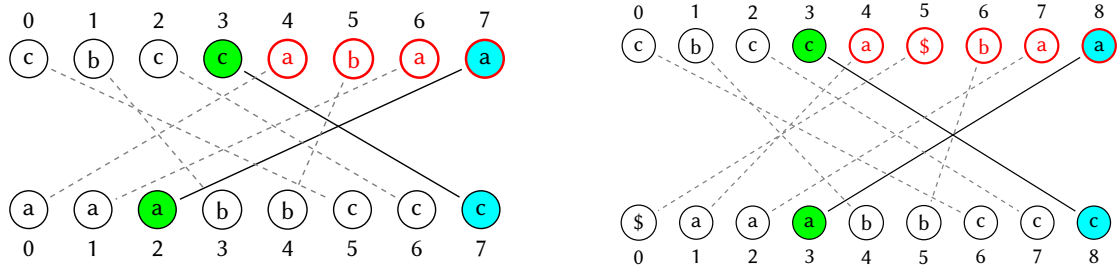


Figure 1: Standard permutation of $w = cbccabaa$ with $\pi(w) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 3 & 6 & 7 & 0 & 4 & 1 & 2 \end{pmatrix}$ (on the left) and of $w' = cbcca\$baa$ with $\pi(w') = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 3 & 6 & 7 & 0 & 4 & 1 & 2 & 3 \end{pmatrix}$ (on the right). In particular, the pseudo-cycle $S = \{3, 7\}$ of π (with S_{left} in green, S_{right} in blue) and its critical interval $R = [4, 7]$ (in red) are shown. Inserting $\$$ in position 5 in the word transforms the pseudo-cycle of $\pi(w)$ in a cycle in $\pi(w')$.

3. New Results on Pseudo-cycles

In this section we will give a stricter characterization of nice positions via a subset of all possible pseudo-cycles. We need some new definitions.

Definition 1. Let S be a pseudo-cycle, $S = S_{left} \cup S_{right}$. If $S_{left} = \emptyset$ then S is called right-only, if $S_{right} = \emptyset$ then S is called left-only. If $S_{left} \neq \emptyset$, then we refer to $a = \max S_{left}$ as the boundary of S . A right-only pseudo-cycle is called minimal if no proper subset is a cycle.

Example 1. Consider the word $cbccabaa$. The pseudo-cycle $S = \{1, 2, 3, 6, 7\}$, with $\pi(S) = \{1, 2, 3, 6, 7\}$ is a right-only pseudo-cycle with critical interval $R_S = [0, 1]$, while $S' = \{5\}$, with $\pi(S') = \{4\}$ is a left-only pseudo-cycle with boundary 4 and critical interval $R_{S'} = [6, 8]$. Finally, $S'' = \{2, 3, 6, 7\}$, with $\pi(S'') = \{1, 2, 6, 7\}$ has boundary 3 and critical interval $R_{S''} = [4, 6]$. This last pseudo-cycle can be divided in $S''_{left} = \{2, 3\}$ and $S''_{right} = \{6, 7\}$.

Lemma 2. Let S and S' be two pseudo-cycles with boundary i . Then $S \cap S'$ is also a pseudo-cycle with boundary i .

Proof. If $S \subseteq S'$ or $S' \subseteq S$, then the statement is trivial. Else, we have to show that (1) $i \in S \cap S'$, and (2) if $x \in S \cap S'$ then $x - 1 \in \pi(S \cap S')$ if $x \leq i$, and $x \in \pi(S \cap S')$ if $x > i$. (1) follows from the fact that both S and S' have boundary i . Ad (2): Let $x \in S \cap S'$, $x \leq i$. Since S is a pseudo-cycle with boundary i , it follows that $\pi^{-1}(x - 1) \in S$. Similarly, since S' is a pseudo-cycle with boundary i , $\pi^{-1}(x - 1) \in S'$, thus $\pi^{-1}(x - 1) \in S \cap S'$. Now let $x \in S \cap S'$, $x > i$. Then it follows that $\pi^{-1}(x) \in S$ and $\pi^{-1}(x) \in S'$, thus $\pi^{-1}(x) \in S \cap S'$. \square

Note that not every i is the boundary of some pseudo-cycle. But with Lemma 2, we can now define a unique pseudo-cycle for each i which is.

Definition 2. Let $0 \leq i \leq n - 1$. A pseudo-cycle $S = S_{left} \cup S_{right}$, with $S_{left} \neq \emptyset$ is called i -essential if i is the boundary of S and every pseudo-cycle S' with boundary i is a superset of S . A pseudo-cycle is called essential if it is i -essential for some i .

Clearly, for every i , there exists at most one i -essential pseudo-cycle. This implies that altogether there are at most $n - 2$ essential pseudo-cycles.

Example 2. Consider again the word $cbccabaa$. $S = \{3, 7\}$, with $\pi(S) = \{2, 7\}$ is the 3-essential pseudo-cycle with critical interval $R = [4, 7]$. In this pseudo-cycle we have $S_{\text{left}} = \{3\}$ and $S_{\text{right}} = \{7\}$.

Proposition 1. Given an i -essential pseudo-cycle S , its critical interval R_S is maximal w.r.t. the boundary i . In other words, if S' is a pseudo-cycle with the same boundary i , then $R_{S'} \subseteq R_S$.

Proof. Follows immediately from Lemma 2. \square

Corollary 1. Let w be a word. Then R_w equals the union of critical intervals of those pseudo-cycles which are either minimal right-only or essential.

3.1. Algorithm

In this section we will describe an alternative algorithm for finding all nice positions of a given word w . Even though this algorithm is slower than the one in [22], it is of interest due to its greater informativeness w.r.t. pseudo-cycles. The algorithm takes advantage of Corollary 1. This is useful from a computational point of view, as can be deduced from the next lemma.

Lemma 3. The set of essential pseudo-cycles can be exponentially smaller than the complete set of pseudo-cycles.

Proof. Let $w = \mathbf{b}^{\lceil n/2 \rceil} \mathbf{a}^{\lfloor n/2 \rfloor}$, thus $|w| = n$. The total number of pseudo-cycles is $2^{|w|_a} = 2^{\lfloor n/2 \rfloor}$, since every subset of the index set of w containing at least one \mathbf{a} and excluding the first \mathbf{b} is a pseudo-cycle with S_{left} and S_{right} not empty. On the other hand, there are only $\lfloor n/2 \rfloor$ essential pseudo-cycles, namely sets of the form $S = \{i, \lfloor n/2 \rfloor + i\}$ for $i \in \{1, 2, 3, \dots, \lfloor n/2 \rfloor\}$. \square

From Cor. 1 we know that it suffices to compute right-only pseudo-cycles and essential pseudo-cycles to identify nice positions of a word. However, our experiments show that on 98% of words with 0 nice positions, all positions are blocked by left-only and right-only pseudo-cycles (data not shown). Based on experimental evidence from [22] we know that a large fraction of words have 0 nice positions: e.g. for binary words of length $n = 20$, around 65% of words have 0 nice positions, while over a ternary alphabet, 61% of words of length 20 have 0 nice positions. Since right-only and left-only pseudo-cycles can be computed in linear time, our algorithm does this as a first step (Phase 1 and Phase 2). Only if at this point there are still positions which have not been blocked does it proceed to computing essential pseudo-cycles (Phase 3).

We next give a description of the three phases of the algorithm.

Phase 1 consists in computing the cycle decomposition of π_w , since the cycles of the permutation are exactly the minimal right-only pseudo-cycles.

Phase 2 starts with a filtering step: It removes from the index set all indices which cannot be part of any left-only pseudo-cycle. The idea is that 0 cannot be part of a left-only pseudo-cycle because, since then -1 would be in the image of the pseudo-cycle. This cannot happen because both the pseudo-cycle and its image must be a subset of the set of indices $\{0, 1, \dots, n-1\}$. By using π on 0 we have the information that $\pi(0) + 1$ must not be in the pseudo-cycle itself, because otherwise it would imply also the previous statement. If we continue to apply π iteratively we can block every index connected to 0, avoiding them in the following operations.

Algorithm 1: Pseudo-cycles finder

Input: Word w
Output: Set of right-only, left-only, and essential pseudo-cycles

```
1  $n \leftarrow |w|$ 
2  $\pi \leftarrow stdPerm(w)$ 
3  $\pi^{-1} \leftarrow invPerm(\pi)$ 
4  $setRight \leftarrow rightPCycles(\pi)$  // cycle decomposition
5  $setLeft \leftarrow leftPCycles(\pi, \pi^{-1})$  // compute left-only pseudo-cycles
6 if  $|R_w| = n + 1$  then // if  $R_w$  contains all indices, exit
7 | return  $setRight \cup setLeft$ 
8  $setEss \leftarrow essPCycles(\pi, \pi^{-1})$  // compute  $i$ -essential pseudo-cycles
9 return  $setRight \cup setLeft \cup setEss$ 
```

With the remaining set of indices the left-only pseudo-cycles are built. We use a procedure resembling the cycle decomposition of the standard permutation. Starting from position i , we traverse the permutation and we insert $\pi(i) + 1$ in S . From this index we continue until we loop back to the starting index i , namely we build a “shifted” cycle. This procedure is repeated until no indices are left to create disjoint left-only pseudo-cycles.

Phase 3 computes i -essential pseudo-cycles which have a non-empty right part, since essential pseudo-cycles which are left-only have already been computed in Phase 2. Let B be the set of indices for which such a pseudo-cycle has already been computed. For each $i > 0, i \notin B$, the algorithm computes the unique i -essential pseudo-cycle, if it exists. During this step, the algorithm traverses the permutation iteratively starting from $j = \pi^{-1}(i - 1)$ and using this trajectory:

$$j \leftarrow \begin{cases} \text{abort} & \text{if } j = 0 \\ \pi^{-1}(j - 1) & \text{if } j < i \\ \pi^{-1}(j) & \text{if } j > i \\ \text{stop - ps.-c. closed} & \text{if } j = i \end{cases} \quad (1)$$

The algorithm inserts the indices that it touches either in S_{left} (if the index is on the left of the boundary) or in S_{right} (if the index is on the right of the boundary). It stops either because it returns to the starting point (boundary) during the traversal of the permutation or because it fails the construction of such pseudo-cycle, ending up in position 0 (which cannot be part of S_{left} , otherwise we would have -1 in $\pi(S)$). An example of the construction of i -essential pseudo-cycles can be seen in Figure 2.

Proposition 2. *Algorithm 1 outputs exactly the essential pseudo-cycles and the minimal right-only pseudo-cycles of the input word. Therefore, the algorithm computes all pseudo-cycles which are necessary to identify all nice positions.*

Proposition 3. *Algorithm 1 runs in $\mathcal{O}(n^2)$ time.*

The following example shows that this bound is tight: In fact, there is an infinite number of words on which the algorithm takes $\Theta(n^2)$ time. The reason is that the running time of Phase

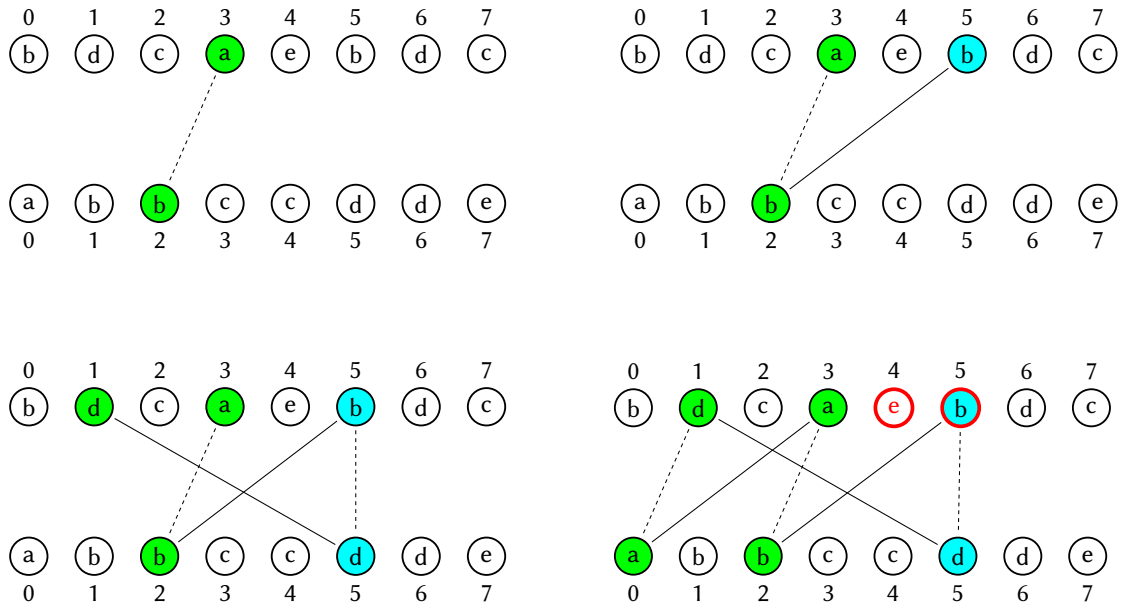


Figure 2: An example for the construction of the i -essential pseudo-cycle of the word $bdcaebdc$ with boundary $i = 3$ (Section 3.1). S_{left} in green, S_{right} in blue, critical interval in red.

3 is proportional to the sum of the lengths of the essential pseudo-cycles of the input word. E.g. for $\sigma = 2$ and $n = 20$, consider $w = \text{bbaaaaaaaabbbbbbbba}$: the sum of the sizes of the essential pseudo-cycles of w is 302. We can produce such example words for every even length by adding an a and a b, respectively, to the first run of a's and to the second run of b's.

4. On Nice Positions in Fully Clustered Words

In this section, we present properties of nice positions in fully clustered words. Recall that a fully clustered word is one which has one run per character, such as cccabb . We will study these questions for different alphabets, where we restrict our attention to words w s.t. $\text{alph}(w) = \Sigma$, i.e. words in which every character appears at least once. We are interested in the number $h(w)$ of nice positions of a word w , and in the number of words $H_n(k)$ which have k nice positions, for given length n .

As an example, for $\sigma = 2$, there are 10 fully clustered words of length 6, 6 of which have 1 nice position, 2 have 2 nice positions, and 2 have 3 (see Table 1). Of the 60 fully clustered words for $\sigma = 3$, 5 have 0 nice positions, 26 have 1, 16 have 2, and 13 have 3 (data not shown).

First we gather some useful properties using pseudo-cycles:

Lemma 4. *Let w be any word and π_w its standard permutation. The following hold:*

1. *If $n - 1$ is a fixpoint, then no $i < n$ is nice.*

word	$h(w)$	nice positions
abbbbb	1	6
aabbbb	1	6
aaabbb	1	6
aaaabb	1	6
aaaaab	1	6

word	$h(w)$	nice positions
baaaaa	1	1
bbaaaa	3	2, 4, 6
bbbaaa	2	3, 5
bbbbaa	2	2, 4
bbbbaa	3	1, 3, 5

Table 1

Nice positions of fully clustered binary words of length 6. We report the number of nice positions and the nice positions for each word.

2. If $\pi(1) = 0$, then no $i > 1$ is nice.
3. n is nice if and only if π_w has no left-only pseudo-cycle.
4. 0 is not nice.

Proof. 1. If $(n - 1)$ is a fixpoint, then $\{n - 1\}$ is a right-only pseudo-cycle, so it blocks all elements from 0 to $n - 1$. 2. If $\pi(1) = 0$, then $\{1\}$ is a left-only pseudo-cycle, blocking any $i > 1$. 3. If n is nice, then it cannot be in the critical interval of any pseudo-cycle. But the pseudo-cycles whose critical interval contains n are exactly the left-only pseudo-cycles, since if there is a non-empty right part, then its minimum must be smaller than n . 4. The entire set $\{0, \dots, n - 1\}$ is a right-only pseudo-cycle, so its minimum 0 blocks all $i \leq 0$. \square

Lemma 5. *Let w be any word, $|w| = n$. If π_w is the identity permutation, then $h(w) = 1$. In particular, n is the only nice position.*

Proof. $\pi_w = (0)(1) \cdots (n - 1)$ (in cycle representation), consisting of n fixpoints. Since $n - 1$ is a fixpoint, by Lemma 4, no $i < n$ can be nice. Clearly, π_w has no left-only pseudo-cycles, so again by Lemma 4, n is nice. \square

Since the standard permutation of any word over a unary alphabet is the identity permutation, these words have exactly one nice position:

Corollary 2. *If w is a word over an alphabet of size $\sigma = 1$, then $h(w) = 1$.*

4.1. Number of Nice Positions over Alphabet Size 2

Our first result says that every fully clustered word over a binary alphabet has at least one nice position.

Proposition 4. *If w begins with an a , then $h(w) = 1$. If w begins with a b then $h(w) \geq 1$.*

Proof. First, let $w = a^i b^{n-i}$, for some $1 \leq i \leq n - 1$. Then $\pi_w = id$, thus, by Lemma 5, there is exactly one nice position, namely n .

Now let $w = b^i a^{n-i}$, for some $1 \leq i \leq n - 1$. It is known that a word of this form is the BWT of a standard word, or of a power of a standard word [20], with the first case if $\gcd(i, n - i) = 1$. By the result of Likhomanov and Shur [21], the number of cycles c of π_w equals the greatest common divisor of the runlengths, namely $c = \gcd(i, n - i)$. By Lemma 1 then, c is nice. \square

Proposition 5. Let $w = \text{ba}^{n-1}$. Then $h(w) = 1$.

Proof. The standard permutation of w is $\pi_w = (0, n-1, n-2, \dots, 2, 1)$, in cycle notation. By Lemma 1, 1 is nice, since π_w has one cycle. 0 is not nice by Lemma 4. Every $i > 0$ is a singleton left-only pseudo-cycle, since $\pi(i) = i-1$, while all other pseudo-cycles are subsets of $\{1, \dots, n-1\}$, and thus left-only. Every left-only pseudo-cycle $\{i\}$ is i -essential, blocking $[i+1, n]$, i.e. the only position which is nice is 1 (Fig. 3). \square

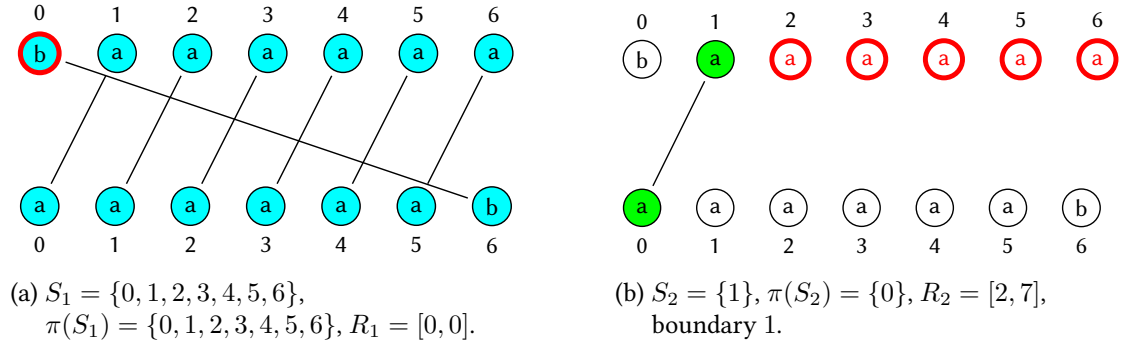


Figure 3: Two pseudo-cycles S_1, S_2 in the word baaaaaa and their critical intervals R_1, R_2 , which block all positions of the word except for 1. (S_{left} in green, S_{right} in blue, critical intervals in red.)

Proposition 6. Let $w = \text{b}^i \text{a}^{n-i}$ where $c = \gcd(i, n-i) > 1$. Then w has at least two nice positions, namely c and $c+2$.

Proposition 7. Let $w = \text{b}^{j+1} \text{a}^j$. Then $h(w) = 2$, in particular, 1 and n are nice.

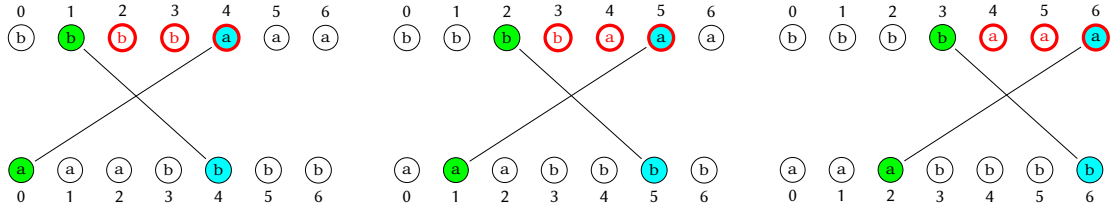
Proof. First, since w is a BWT, the number of cycles c of π_w is nice, by Lemma 1. Since the runlengths j and $j+1$ are relatively prime, $c = 1$, so 1 is nice. Let $0 < i \leq j$. The i -essential pseudo-cycle is $S_i = \{i, i+j\}$, since $\pi(i) = i+j$ and $\pi(j) = i+j - (j+1) = i-1$. The critical interval of S_i is $R_i = [i+1, i+j]$. The union of these critical intervals is $[2, n-1] \subseteq R_w$, since $n = |w| = 2j+1$. Finally, no i -essential pseudo-cycles exist for $i > j$. Therefore, there are no left-only pseudo-cycles, and thus, by Lemma 4, n is nice (see Figure 4). \square

4.2. Number of Words with k Nice Positions over Alphabet Size 2

We now turn to the number of words with k nice positions.

Definition 3. For $n > 0, k \geq 0$, let $H_n^\sigma(k)$ denote the number of fully clustered words with exactly k nice positions over an alphabet of size σ .

From Prop. 4 it follows that $H_n^2(0) = 0$, and from Prop. 4 and 5 it follows that $H_n^2(1) \geq n$, since the $n-1$ words beginning with a and the word ba^{n-1} have 1 nice position. For a better understanding of the words with k nice positions, we ran experiments on fully clustered words up to length 100, and studied their pseudo-cycles. These led to the following conjectures:



- (a) $S_1 = \{1, 4\}$,
 $\pi(S_1) = \{0, 4\}$, $R_1 = [2, 4]$,
boundary 1.
- (b) $S_2 = \{2, 5\}$,
 $\pi(S_2) = \{1, 5\}$, $R_2 = [3, 5]$,
boundary 2.
- (c) $S_3 = \{3, 6\}$,
 $\pi(S_3) = \{2, 6\}$, $R_3 = [4, 6]$,
boundary 3.

Figure 4: Three pseudo-cycles S_1, S_2, S_3 of the same form in the word bbbbaaa, and their critical intervals R_1, R_2, R_3 . (S_{left} in green, S_{right} in blue, critical intervals in red.)

Conjecture 1. For all n , $H_n^2(1) = n$.

Conjecture 2. For n even, $n \geq 8$, $H_n^2(2) = 0$. For n odd, $H_n^2(2) = 1$.

Conjecture 3. For every n , $H_n^2(\lceil \frac{n}{2} \rceil) = 2$.

The two words with $\lceil n/2 \rceil$ nice positions are bba^{n-2} and $b^{n-1}a$. This is because both have $\lceil n/2 \rceil$ pseudo-cycles whose critical intervals contain exactly one position each. From [22], we know that for words that are BWT images, the parity of nice positions is the same as the parity of the number of cycles. In these two cases, the pseudo-cycles block every position which have the opposite parity, leaving all the remaining positions available.

All our conjectures are confirmed by the histograms we produced for words up to length 100 and k nice positions, k up to 19 (data not shown). Furthermore, the words show a regular behaviour with distinct k 's. In particular, we can see that the 8 longest words (4 of even length, 4 of odd length) are disconnected from all the others. Moreover, the greater k , the farther this group of 8 words is from the others. The first k where this group is visible is $k = 6$, and we identify the 4 words of even length as follows: two primitive words ($w_1 = b^7a^{13}$, $w_2 = b^{15}a^7$), two power words ($w_3 = b^{14}a^6$, $w_4 = b^8a^{14}$); and the 4 primitive words of odd length ($w_5 = b^8a^{15}$, $w_6 = b^{16}a^7$, $w_7 = b^9a^{16}$, $w_8 = b^{17}a^8$).

These words give an idea of the regularity of $H_n^2(k)$ across different k . Consider $w_1 = b^7a^{13}$, and the same word with 4 more a's and 2 more b's, namely $w'_1 = b^9a^{17}$. In both cases there are $|w_1|_b - 1$ number of pseudo-cycles which have the same form, and starting from the one with the smallest boundary, they are shifted by one position to the right. Moreover, the critical interval of each of these pseudo-cycles has length $|w_1|_b - 1$ (in total they cover $2 \cdot |w_1|_b - 1$ positions). On the other hand, there are $\frac{|w_1| - 2 \cdot |w_1|_b - 1}{2}$ additional pseudo-cycles blocking just one position each, e.g. 5 and 7 in w resp. w' . For this reason, $h(w_1) = |w_1| - \frac{|w_1| - 2 \cdot |w_1|_b - 1}{2} - 2 \cdot |w_1|_b - 1$ and $h(w'_1) = |w'_1| - \frac{|w'_1| - 2 \cdot |w'_1|_b - 1}{2} - 2 \cdot |w'_1|_b - 1$, and therefore $h(w'_1) = h(w_1) + 1$. We observed this phenomenon holds adding iteratively the same number of a's and b's (up to length 100).

k	3	4	5	6	7	8	9	10	11	12	13	14	15	16
n_k	10	16	22	28	34	40	46	52	58	64	70	76	82	88

Table 2

Conj. 4: For $\sigma = 2$, there are no binary words of length greater than n_k with k nice positions.

Further, the same happens adding 2 b's and 4 a's to w_4, w_5, w_7 , and adding 4 b's and 2 a's to w_2, w_3, w_6, w_8 . Our experiments suggest that for fixed k , there are no words with greater length than those above, see Table 2. On the basis of these observations, we conjecture that the set $\mathcal{F}^2(k)$ of binary words with k nice positions is finite. Formally:

Conjecture 4. *For every $k \geq 3$, there exists a length n_k such that no word of length greater than n_k has exactly k nice positions.*

Finally, we noticed from our experiments that the smallest nice position is always a divisor of n . Let d divide n . As we have seen before, $w = b^i a^j$ is the BWT of a word u^d with u a standard word and $d = \gcd(j, i)$ [20]. Thus d is nice by Lemma 1. The standard permutation of w has d cycles, and the largest minimum of a cycle is $d - 1$, blocking all positions $i \leq d - 1$. Thus, given n, d where d is a divisor of n , e.g. the word $b^d a^{(\frac{n}{d}-1)d}$ has smallest nice position d . We conjecture that the converse is true also:

Conjecture 5. *An integer d can occur as a smallest nice position for some fully clustered word of length n if and only if d divides n .*

5. Conclusion and Ongoing Work

In this paper we continued the study of a combinatorial problem introduced in [22]: which are the positions where a dollar can be inserted into a word to make it a BWT (so-called nice positions). We strengthened the characterization via pseudo-cycles of nice positions given there. Pseudo-cycles are particular subsets of the indices; here we showed that a much smaller subset of pseudo-cycles suffices to characterize the set of nice positions of a word.

We further presented an algorithm that returns the smallest set of pseudo-cycles needed to identify all nice positions of the input word, and using this algorithm, we studied nice positions of fully clustered words over a binary alphabet. We presented properties on the number of nice positions for distinct fully clustered words, and we gave some conjectures on the number of words with fixed number of nice positions, for which we have experimental evidence.

As future work we are planning to extend the study to larger alphabets. We already have extensive experimental data for $\sigma = 3$, and plan to prove related properties as for the binary alphabet. For example, w.r.t. Conj. 4 we could characterize a consistent phenomenon we observe in the histograms for the ternary alphabets. For $k \geq 3$ we reach a plateau after a certain length, which we believe to be strongly related to the conjectured n_k . Moreover the words appearing in this plateau seem to be consisting of only two types, namely $a^i c^j b^\ell$ and $c^i b^j a^\ell$.

References

- [1] M. Burrows, D. J. Wheeler, A block-sorting lossless data compression algorithm, Technical Report, DIGITAL System Research Center, 1994.
- [2] G. Rosone, M. Sciortino, The Burrows-Wheeler transform between data compression and combinatorics on words, in: 9th Conference on Computability in Europe (CiE 2013), Springer, 2013, pp. 353–364.
- [3] G. Manzini, An analysis of the Burrows-Wheeler transform, *J. ACM* 48 (2001) 407–430.
- [4] G. Navarro, Indexing highly repetitive string collections, part I: repetitiveness measures, *ACM Computing Surveys* 54 (2021) 29:1–29:31.
- [5] H. Li, R. Durbin, Fast and accurate long-read alignment with Burrows-Wheeler transform, *Bioinformatics* 26 (2010) 589–595.
- [6] B. Langmead, C. Trapnell, M. Pop, S. L. Salzberg, Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, *Genome Biology* 10 (2009) R25.
- [7] T. W. Lam, R. Li, A. Tam, S. Wong, E. Wu, S. M. Yiu, High throughput short read alignment via bi-directional BWT, in: 2009 IEEE International Conference on Bioinformatics and Biomedicine, 2009, pp. 31–36.
- [8] C. Boucher, T. Gagie, A. Kuhnle, B. Langmead, G. Manzini, T. Mun, Prefix-free parsing for building big BWTs, *Algorithms Mol. Biol.* 14 (2019) 13:1–13:15.
- [9] P. Ferragina, G. Manzini, Indexing compressed text, *J. ACM* 52 (2005) 552–581.
- [10] T. Gagie, G. Navarro, N. Prezza, Optimal-time text indexing in BWT-runs bounded space, in: Proc. of 39th ACM-SIAM Symposium on Discrete Algorithms (SODA 2018), 2018, pp. 1459–1477.
- [11] M. Crochemore, J. Désarménien, D. Perrin, A note on the Burrows-Wheeler transformation, *Theor. Comput. Sci.* 332 (2005) 567–572.
- [12] I. M. Gessel, C. Reutenauer, Counting permutations with given cycle structure and descent set, *Journal of Combinatorial Theory* 64 (1993) 189–215.
- [13] S. Mantaci, A. Restivo, G. Rosone, M. Sciortino, L. Versari, Measuring the clustering effect of BWT via RLE, *Theor. Comput. Sci.* 698 (2017) 79–87.
- [14] S. Brlek, A. Frosini, I. Mancini, E. Pergola, S. Rinaldi, Burrows-Wheeler transform of words defined by morphisms, in: 30th International Workshop on Combinatorial Algorithms (IWOCOA 2019), volume 11638 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 393–404.
- [15] S. Giuliani, S. Inenaga, Zs. Lipták, N. Prezza, M. Sciortino, A. Toffanello, Novel results on the number of runs of the Burrows-Wheeler-Transform, in: Proc. of 47th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2021), volume 12607 of *LNCS*, 2021, pp. 249–262.
- [16] A. Frosini, I. Mancini, S. Rinaldi, G. Romana, M. Sciortino, Logarithmic equal-letter runs for BWT of purely morphic words, in: 26th International Conference on Developments in Language Theory (DLT 2022), volume 13257 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 139–151.
- [17] R. Giancarlo, A. Restivo, M. Sciortino, From first principles to the Burrows and Wheeler transform and beyond, via combinatorial optimization, *Theoretical Computer Science* 387 (2007) 236–248.

- [18] I. M. Gessel, A. Restivo, C. Reutenauer, A bijection between words and multisets of necklaces, *European Journal of Combinatorics* 33 (2012) 1537–1546.
- [19] J. W. Daykin, R. Groult, Y. Guesnet, T. Lecroq, A. Lefebvre, M. Léonard, É. Prieur-Gaston, A survey of string orderings and their application to the Burrows–Wheeler transform, *Theoretical Computer Science* 710 (2018) 52–65.
- [20] S. Mantaci, A. Restivo, M. Sciortino, Burrows–Wheeler transform and Sturmian words, *Information Processing Letters* 86 (2003) 241–246.
- [21] K. M. Likhomanov, A. M. Shur, Two Combinatorial Criteria for BWT Images, in: *Proceeding of the 6th International Computer Science Symposium in Russia (CSR 2011)*, 2011, pp. 385–396.
- [22] S. Giuliani, Zs. Lipták, F. Masillo, R. Rizzi, When a dollar makes a BWT, *Theoretical Computer Science* 857 (2021) 123–146.
- [23] J. Simpson, S. J. Puglisi, Words with simple Burrows-Wheeler Transforms, *Electronic Journal of Combinatorics* 15 (2008).
- [24] A. Restivo, G. Rosone, Burrows-Wheeler transform and palindromic richness, *Theor. Comput. Sci.* 410 (2009) 3018–3026.
- [25] A. Restivo, G. Rosone, Balancing and clustering of words in the Burrows–Wheeler transform, *Theoretical Computer Science* 412 (2011) 3019–3032.
- [26] S. Ferenczi, L. Q. Zamboni, Clustering words and interval exchanges, *Journal of Integer Sequences* 16 (2013) 3.
- [27] M. Bóna, *Combinatorics of Permutations, Second Edition*, Discrete mathematics and its applications, CRC Press, 2012.