Linking BWT and XBW via Aho-Corasick Automaton: Applications to Run-Length Encoding
Bastien Cazaux, Eric Rivals

To cite this version:

HAL Id: lirmm-02382066
https://hal-lirmm.ccsd.cnrs.fr/lirmm-02382066
Submitted on 27 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Linking BWT and XBW via Aho-Corasick Automaton: Applications to Run-Length Encoding

Bastien Cazaux
Department of Computer Science, University of Helsinki, Finland
L.I.R.M.M., CNRS, Université Montpellier, France
https://www.mv.helsinki.fi/home/cazaux/
bastien.cazaux@helsinki.fi

Eric Rivals
L.I.R.M.M., CNRS, Université Montpellier, France
https://www.lirmm.fr/~rivals/
rivals@lirmm.fr

Abstract
The boom of genomic sequencing makes compression of sets of sequences inescapable. This underlies the need for multi-string indexing data structures that helps compressing the data. The most prominent example of such data structures is the Burrows-Wheeler Transform (BWT), a reversible permutation of a text that improves its compressibility. A similar data structure, the eXtended Burrows-Wheeler Transform (XBW), is able to index a tree labelled with alphabet symbols. A link between a multi-string BWT and the Aho-Corasick automaton has already been found and led to a way to build a XBW from a multi-string BWT. We exhibit a stronger link between a multi-string BWT and a XBW by using the order of the concatenation in the multi-string. This bijective link has several applications: first, it allows one to build one data structure from the other; second, it enables one to compute an ordering of the input strings that optimises a Run-Length measure (i.e., the compressibility) of the BWT or of the XBW.

2012 ACM Subject Classification
Mathematics of computing → Discrete mathematics; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Data structures and algorithms for data management

Keywords and phrases
Data Structure, Algorithm, Aho-Corasick Tree, compression, RLE

Digital Object Identifier 10.4230/LIPIcs.CPM.2019.24

Supplement Material Source code at: https://framagit.org/bcazaux/compressbwt

Funding Support from the Institut de Biologie Computationnelle (ANR-11-BINF-0002). Eric Rivals: thanks the GEM Flagship project funded from Labex NUMEV (ANR-10-LABX-0020).

1 Introduction

A seminal, key data structure, which was used for searching a set of words in a text, is the Aho-Corasick (AC) automaton [1]. Its states form a tree that indexes all the prefixes of the words, and each node in the tree is equipped with another kind of arc, called a Failure Link. A failure link of a node/prefix \( v \) points to the node representing the largest proper suffix of \( v \) in the tree. In a way, the Aho-Corasick automaton can be viewed as a multi-string indexing data structure.

In the early 90’s, the Burrows-Wheeler Transform (BWT) of a text \( T \), which is a reversible permutation of \( T \), was introduced for the sake of compressing a text. Indeed, the BWT permutation tends to group identical symbols in runs, which favours compression [5]. However, the BWT can also be used as an index for searching in \( T \), using the Backward Search
procedure [10]. In fact, the BWT of T is the last column of a matrix containing all cyclic-shifts of T sorted in lexicographical order. As sorting the cyclic shifts of T is equivalent to sorting its suffixes, there exists a natural link between the suffix array of T and the BWT of T. Starting in 2005, the radical increase in textual data and in biological sequences raised the need for multi-string indexes. In the multi-string case, a simple approach is to concatenate all input strings and separate them using the same termination symbol (which does not belong to the alphabet), and then to index the result with a traditional indexing data structure (e.g., a suffix array or a FM-index). Another approach is to add a unique termination symbol for each string without concatenation [3, 21]. In both approaches, an initial order on the strings is given by the permutation of the strings in the first approach, or by the order between the termination symbols in the second. All our results are made with the first approach (which minimises the alphabet size) in mind, and can be adapted to the second. Such multi-string indexes are heavily exploited in bioinformatics: first, to index all chromosomes of a genome [17], or a large collection of similar genomes, which allows aligning sequence reads simultaneously to several reference genomes [19], or second, to store and mine whole sets of raw DNA/RNA sequence reads for the purpose of comparing biological conditions or of identifying splice junctions in RNA [7, 14]. In fact, managing compressed and searchable read data sets is now crucial for bioinformatic analyses.

Initially viewed as a simple extension of single-string BWT construction, the efficient construction of multi-string BWT is not trivial and has been investigated per se. Bauer et al. proposed, among others, a lightweight incremental algorithm for their construction [3]. Then, Holt et al. devised algorithms for directly merging several, already built multi-string BWTs efficiently [14, 15], which has recently been improved to build simultaneously the companion Longest Common Prefix (LCP) table [8], or to scale up to terabyte datasets [23].

The notion of BWT has been extended into the XBW to index trees whose arcs are labelled by alphabet symbols [9]. The XBW comprises two arrays, which compactly represent the tree and offer navigational operations.

Recently, Gagie et al. propose the notion of Wheeler graphs to subsume several variants of the BWT, including the XBW of a trie for a set of strings [12]. The relation between a multi-string BWT and the XBW representation of the Aho-Corasick automaton has already been studied and exploited. Hon et al. first use the XBW representation of the Aho-Corasick trie to speed-up dictionary matching [16] building up on [4]. Manzini gave an algorithm that computes the failure links for the trie using the Suffix Array and LCP tables, and an algorithm to build the XBW of the trie with failure links from the BWT of a multi-string [22]. However, none of these established a bijective link between a multi-string BWT and the XBW of the Aho-Corasick automaton. To generalise these results, one needs to consider the order in which the strings are concatenated to form the multi-string. This idea enables us to exhibit a bijection between a multi-string BWT and the XBW of the Aho-Corasick automaton, which allows building one structure from the other in either direction (from BWT to XBW or from XBW to BWT). Finally, we exploit this bijection to find an optimal string order that minimises a Run-Length Encoding (i.e., maximises the compressibility) of these two data structures. Numerous proofs are in appendix (A preprint version of this work is available as [6]).

2 Preliminaries

In this section, we introduce basic concepts and notation, present the Burrows-Wheeler Transform and an extension of it for a set of strings. Finally, we provide a notation for the Aho-Corasick automaton and the eXtended Burrows-Wheeler Transform.
2.1 General notation

Set and Permutation. Let $i$ and $j$ be two integers such that $i \leq j$. The interval $[i,j]$ is the set of all integers between $i$ and $j$. An integer interval partition of $[i,j]$ is a set of intervals $\{[i_1,j_1], \ldots, [i_n,j_n]\}$ such that $i_1 = i$, $j_n = j$, and for all $k \in [1,n - 1]$, $j_k + 1 = i_{k+1}$. We also define the order $<$ on intervals such that for two intervals $u := [i,j]$ and $v := [i',j']$, $u < v$ iff $j < j'$. Let $E$ be a finite set and let $\#E$ (or $\#(E)$) denote its cardinality. A permutation of $E$ is an automorphism of $E$. A permutation $\sigma$ of $E$ is said to be circular iff for all $i$ and $j \in E$, there exists a positive integer $k$ such that $\sigma^k(i) = j$ (where $\sigma^1(i) = \sigma(i)$ and $\sigma^k(i) = \sigma(\sigma^{k-1}(i))$). For a circular permutation $\sigma$ of $E$ and an element $e \in E$, we denote by $\sigma_\tau$ the function from $[1,\#E]$ to $E$ such that for all $i \in [1,\#E]$, $\sigma_\tau(i) := \sigma(i+\#E-1)(e)$. Given a total order $<$ for $E$, we define $<_\sigma$ as the order on $E$ such that $e <_\sigma f$ iff $\sigma(e) < \sigma(f)$ for any $e, f \in E$.

String. Let $\Sigma$ be a finite alphabet. A string $w$ of length $n$ over $\Sigma$ is a sequence of symbols $w[1] \ldots w[n]$ where $w[i] \in \Sigma$ for all $i \in [1,n]$. $\Sigma^*$ is the set of all finite strings over $\Sigma$. The length of a string $w$ is denoted by $|w|$. A substring of $w$ is written as $w[i,j] := w[i] \ldots w[j]$ for some $i \leq j$. A prefix of $w$ is a substring of $w$ starting at position 1, and a suffix of $w$ is a substring of $w$ ending at position $|w|$. A prefix $x$ (or a suffix) of a string $y$ is said proper is $x$ is different from $y$. The reverse of a string $w$, denoted by $\overline{w}$, is the string $w[n] \ldots w[2]w[1]$. We define the lexicographic order $<$ on strings as usual.

Ordered Set of Strings. Let $S = \{s_1, \ldots, s_n\}$ be a set of strings. The norm of $S$, denoted $\|S\|$, is the sum of the length of strings of $S$, i.e. $\|S\| := \sum_{i \in S} |s_i|$. Let $\text{Prefix}(S)$, (respectively $\text{Suffix}(S)$) denote the set of all prefixes (resp. all suffixes) of strings of $S$. We denote by $\overline{S}$ the set of all reverse strings of strings of $S$, i.e. $\overline{S} := \{\overline{s_1}, \ldots, \overline{s_n}\}$. An ordered set of strings $P$ is a pair $(S, \sigma)$ where $S$ is a set of strings in lexicographic order, and $\sigma$ a circular permutation of $S$. We denote by $P, S$ the set of strings $S$ and by $P, \sigma$ the circular permutation $\sigma$. We denote by $\overline{P}$ the pair $(\overline{P}, S, \sigma)$.

2.2 Burrows-Wheeler Transform

BWT of a string. Let $w$ be a string and $i$ be an integer satisfying $1 \leq i \leq |w|$. The Suffix Array (SA) of $w$ [20], denoted $SA(w)$, is the array of integers that stores the starting positions of the $|w|$ suffixes of $w$ sorted in lexicographic order. The Burrows-Wheeler Transform (BWT) [5] of $w$, denoted $BWT(w)$, is the array containing a permutation of the symbols of $w$ which satisfies $BWT(w)[i] := w[SA(w)[i] - 1]$ if $SA(w)[i] > 1$, and $BWT(w)[i] := w[i]$ otherwise. The Longest Common Prefix table (LCP) [5] of $w$, denoted $LCP(w)$, is the array of integers such that $LCP(w)[i]$ equals the length of the longest common prefix between the suffixes of $w$ starting at positions $SA(w)[i]$ and $SA(w)[i-1]$ if $i > 0$, and 0 otherwise.

For any string $s \in \Sigma^*$ and any symbol $c \in \Sigma$, one defines the functions denoted $\text{rank}$ and $\text{select}$ as follows: $\text{rank}_c(s,i)$ is the number of occurrences of $c$ in $s[1,i]$, and $\text{select}_c(s,j)$ is the position of the $j$th occurrence of $c$ in $s$. The arrays $BWT(w)$ and $LCP(w)$ can be computed in $O(|w|)$ time [5, 10]. Simultaneously, one can compute $\text{rank}$ and $\text{select}$ for $BWT(w)$ at no additional cost and implement them such that any $\text{rank}$ or $\text{select}$ query takes constant time [13, 11, 18]. We use such state-of-the-art data structure to store a BWT.

Let $C$ denote the array of length $\#\Sigma$ such that $C[c]$ equals the number of symbols of $w$ that are alphabetically strictly smaller than $c$. The Last-To-First column mapping (LF) [10] of $w$ is the function such that for any $1 \leq i \leq |w|$ one has $LF(w)[i] := C[BWT(w)[i]] + \text{rank}_{BWT(w)[i]}(BWT(w), i)$. It is proved that $SA(w)[LF(w)[i]] = SA(w)[i] - 1$ if $SA(w)[i] \geq 2$ and $SA(w)[LF(w)[i]] = |w|$ otherwise [5, 10].
BWT of a set of strings. From now on, let \( P := (S, \sigma) \) be an ordered set of strings. We assume the symbol \( \$ \) is not in \( \Sigma \) and is alphabetically smaller than all other symbols. We denote by \( m_P \) the string obtained by concatenating the strings of \( P : S \) separated by a \( \$ \) and following the order \( \overline{P}.\sigma \). i.e., \( m_P := s_{\overline{P}(1)}\$s_{\overline{P}(2)}\$...s_{\overline{P}(n)}\$ \) (See Figure 1). We extend the notion of BWT of a string to an ordered set of strings \( P \); the BWT of \( P \) is the BWT of the string \( m_P \), i.e., \( \text{BWT}(P) := \text{BWT}(m_P) \). We extend similarly the LF function by setting that \( \text{LF}(P) := \text{LF}(m_P) \) (by using \( \text{rank} \) on \( \text{BWT}(P) \)).

![Figure 1 Running example with \( P.S := \{abaa, abba, baba, bbaa\} \) and the corresponding \( m_P \).]

We define the Longest Representative Suffix table (LRS) of \( P \) as the array of \(|m_P|\) integers satisfying: for any \( i \in [1, |m_P|] \) one has \( \text{LRS}(P)[i] := \text{select}_{\$}([|m_P|/\lceil|m_P|\rceil] - 1) \). The entry \( \text{LRS}(P)[i] \) gives the length of the substring of \( m_P \) starting at position \( \text{SA}(m_P)[i] \) up to, but not including the next \( \$ \). Using the LRS table, we extend the notion of LCP table to an ordered set of strings. For \( P \), we set \( \text{LCP}(P)[i] := \text{min}(\text{LCP}(m_P)[i], \text{LRS}(P)[i]) \) for any \( i \in [1, |m_P|] \). Using tables \( \text{BWT}(P) \) and \( \text{LCP}(m_P) \), we can compute the tables \( \text{LRS}(P) \) and \( \text{LCP}(P) \) in linear time in \(|m_P|\) (see Appendix). Figure 2 illustrates the LCP and LRS tables.

### 2.3 Aho-Corasick automaton and eXtended Burrows-Wheeler Transform

**Tree.** Let \( \mathcal{T} \) be a tree and \( u \) be a node of \( \mathcal{T} \). Let \( \perp \) denote the root of \( \mathcal{T} \). We denote by \( \text{Parent}_{\mathcal{T}}(u) \) the parent of \( u \) in \( \mathcal{T} \), by \( \text{Children}_{\mathcal{T}}(u) \) the set of children of \( u \) in \( \mathcal{T} \), and by \( \text{Leaves}_{\mathcal{T}}(u) \) the set of leaves in the subtree of \( u \) in \( \mathcal{T} \). Let \( v \) be a leaf of \( \mathcal{T} \); we denote by \( \mathcal{B}_{\mathcal{T}}(v) \) the subtree of \( \mathcal{T} \) containing all nodes comprised between \( \perp \) and \( v \) included. As for a leaf \( v \) in the subtree of \( u \) in \( \mathcal{T} \), \( \#\text{Children}_{\mathcal{B}_{\mathcal{T}}(v)}(u) = 1 \), we denote by \( \text{Child}_{\mathcal{T}}(v) \) the unique element of \( \text{Children}_{\mathcal{B}_{\mathcal{T}}(v)}(u) \). Given a total order \( \prec \) on \( \text{Leaves}_{\mathcal{T}}(\perp) \), we extend this order on the set \( \text{Children}_{\mathcal{T}}(v) \) for any node \( v \) of \( \mathcal{T} \) by: for any \( x, y \) in \( \text{Children}_{\mathcal{T}}(v) \), \( x \prec y \iff \min_{x' \in \text{Leaves}_{\mathcal{T}}(x)} x' \prec \min_{y' \in \text{Leaves}_{\mathcal{T}}(y)} y' \).

**Aho-Corasick tree.** The Aho-Corasick automaton (AC) [1] of a set of strings \( S \) is a digraph whose set of nodes is the set of all prefixes of the strings of \( S \). This graph is composed of two trees on the same node set. The first tree, which we called the Aho-Corasick Tree (ACT), has an arc from a prefix \( u \) to a different prefix \( v \) iff \( u \) is the longest proper prefix of \( v \) in \( \text{Prefix}(S) \) (see Figure 4). The second tree, termed Aho-Corasick Failure link (ACFL), has an arc from a prefix \( u \) to a different prefix \( v \) iff \( v \) is the longest proper suffix of \( u \) in \( \text{Prefix}(S) \).

**eXtended Burrows-Wheeler Transform.** Let \( \mathcal{T} \) be an ordered tree (i.e., with a total order on the set of children of each node) such that every node of \( \mathcal{T} \) is labelled with a symbol from an alphabet \( \Sigma \). We define the functions \( \delta \) and \( \pi \) on the set of nodes of \( \mathcal{T} \) except for the root such that for a node \( v \) of \( \mathcal{T} \), \( \delta(v) \) is the label of \( v \), and \( \pi(v) \) is the string obtained by concatenating the labels from \( v \)'s parent up to the root of \( \mathcal{T} \). Let \( \prec \) be the total order between the nodes of \( \mathcal{T} \) such that for \( u \) and \( v \) two nodes of \( \mathcal{T} \), \( u \prec v \) iff \( \pi(u) \) is strictly lexicographically smaller than \( \pi(v) \) or \( u \) is before \( v \) in the order of \( \mathcal{T} \).

The node $i$ of $T$ (except the root of $T$) gives the label of the node $i$ of length $LCP(P)$, where $LCP(P)$ is an array of symbols of $T$ except the root of $T$. The eXtended Burrows-Wheeler Transform (XBWT) [9] of a tree $T$ is an array of symbols of $\Sigma$ of length $#PA(T)$, such that the entry at position $i$ gives the label of the node $PA(T)[i]$. The eXtended Burrows-Wheeler Last (XBWL) [9] of a tree $T$ is the bit array of length $#PA(T)$ such that $XBWL(T)[i]$ equals 1 if the node $PA(T)[i]$ is the last child of its parent, and 0 otherwise. Figure 5a illustrates XBWT and XBWL.

eXtended Burrows-Wheeler Transform of Aho-Corisick automaton. For a set of strings $S := \{s_1, \ldots, s_n\}$, we denote by $S^8$ the set $\{s_1$, $\ldots$, $s_n, \}$.

Figure 2: Tables $LCP(P)$, $LRS(P)$, $SA(m_P)$ and $BWT(P)$ for the running example. One has $LCP(P)[10] = LRS(P)[10] = 2$ although the longest common prefix between suffixes of rank 9 and 10 is $aa$ of length 5 (i.e., although $LCP(m_P)[10] = 5$).

Example 1. With the tree of Figure 3, we have $\delta(\varnothing) = b$ and $\pi(\varnothing) = aba = \delta(\varnothing)\delta(\varnothing)\delta(\varnothing)$, and also $\delta(\varnothing) = b$ and $\pi(\varnothing) = baa = \delta(\varnothing)\delta(\varnothing)\delta(\varnothing)$. Thus, $\varnothing \prec \varnothing$ since $\pi(\varnothing) < \pi(\varnothing)$.

The Prefix Array (PA) of an ordered tree $T$ is the array of pointers to the nodes of $T$ (except the root of $T$) sorted in $\prec$ order. The eXtended Burrows-Wheeler Transform (XBWT) [9] of a tree $T$ is an array of symbols of $\Sigma$ of length $#PA(T)$, such that the entry at position $i$ gives the label of the node $PA(T)[i]$. The eXtended Burrows-Wheeler Last (XBWL) [9] of a tree $T$ is the bit array of length $#PA(T)$ such that $XBWL(T)[i]$ equals 1 if the node $PA(T)[i]$ is the last child of its parent, and 0 otherwise. Figure 5a illustrates XBWT and XBWL.

eXtended Burrows-Wheeler Transform of Aho-Corisick automaton. For a set of strings $S := \{s_1, \ldots, s_n\}$, we denote by $S^8$ the set $\{s_1$, $\ldots$, $s_n, \}$. Let $P$ be an ordered set of strings. We define $ACT(P)$ as the variant of the Aho-Corisick tree of $P.S^8$ where the label of an arc is shifted on the deepest node of this arc (i.e., the label of a node $v$ of $ACT(P)$ is the label of $(u,v)$ in $ACT(P.S)$ where $u$ is the parent of $v$ in $ACT(P)$) and equipped with the order $<_{P.S}$. Indeed, $<_{P.S}$ is the order on the leaves satisfying: for $u$ and $v$ two leaves of $ACT(P)$, $u <_{P.S} v$

1 In [9], the XBW-transform is defined as $XBW(T)[i] := (XBWT[i], XBWL[i])$, for any position $i$.  

CPM 2019
We define \( m \) which is impossible. This concludes the proof.

Let us start the proof of Proposition 2 with the following Lemma.

**Lemma 3.** Let \( u = [i, j] \in \text{Decomp}_\text{BWT}(P) \). For all \( k \in [i, j] \):

\[
\text{Dec}_{\text{Pre}}[u] = m_P[S_A(m_P)[k] : S_A(m_P)[k] + LRS(P)[k] - 1].
\]

**Proof.** Let us show by contraposition that \( \text{LRS}(P)[k - 1] = \text{LCP}(P)[k] \) for all \( k \in [i + 1, j] \). Assume that there exists \( k \in [i + 1, j] \) such that \( \text{LRS}(P)[k - 1] \neq \text{LCP}(P)[k] \). Whenever \( \text{LRS}(P)[k - 1] < \text{LCP}(P)[k] \), we get by definition that \( \text{LRS}(P)[k] \geq \text{LCP}(P)[k] \), and thus \( \text{LRS}(P)[k - 1] < \text{LRS}(P)[k] \). By the definition of \( \text{LRS}(P) \), \( m_P[S_A(m_P)[k - 1] + \text{LRS}(P)[k - 1]] = \$. By the definition of \( \text{LCP}(P) \), for all \( j \in [1, \text{LCP}(P)[k - 1] - 1] \), \( m_P[S_A(m_P)[k] + j] = m_P[S_A(m_P)[k - 1] + j] \). As \( \text{LRS}(P)[k - 1] < \text{LCP}(P)[k] \), we have \( m_P[S_A(m_P)[k] + \text{LRS}(P)[k - 1]] = m_P[S_A(m_P)[k - 1] + \text{LRS}(P)[k - 1]] = \$, which is impossible since \( \text{LRS}(P)[k - 1] < \text{LRS}(P)[k] \). Whenever \( \text{LRS}(P)[k - 1] > \text{LCP}(P)[k] \), as \( \text{LCP}(P)[k] = \text{LRS}(P)[k] \), the string \( m_P[S_A(m_P)[k] : m_P] \) is lexicographically strictly smaller than \( m_P[S_A(m_P)[k - 1] : m_P] \), which is impossible. This concludes the proof.

**Figure 3** Tree \( \text{ACT}(P) \) for the running example, and links between the orders \( <_{\text{P,σ}} \) and \( \prec_{\text{P,σ}} \).
Proof. Let \( u = [i, j] \) be an interval of \( \text{Decomp}_\text{BWT}(P) \). First, we prove that \( \text{Dec}_\text{Pre}[u] \in \text{Prefix}(\overrightarrow{P.S}) \), and then to prove the bijection, we show \( \text{Dec}_\text{Pre} \) is injective and surjective.

By definition, \( \text{LRS}(P)[i] = \text{select}_4(m_P[\text{SA}(m_P)[i] : |m_P|, 1] - 1, m_P[\text{SA}(m_P)[i] + \text{LRS}(P)[i]] = \$ \) and for any \( j \in \{ \text{SA}(m_P)[i], \text{SA}(m_P)[i] + \text{LRS}(P)[i] - 1 \} \), \( m_P[j] \neq \$. Hence, \( m_P[\text{SA}(m_P)[i] : \text{SA}(m_P)[i] + \text{LRS}(P)[i] - 1] \) is a suffix of a string \( w \) of \( P.S \), and thus \( \text{Dec}_\text{Pre}[u] \) is a prefix of \( \overrightarrow{w} \) in \( \overrightarrow{P.S} \).

Injectiveness. Let \( u_1 = [i_1, j_1] \) and \( u_2 = [i_2, j_2] \) be two elements of \( \text{Decomp}_\text{BWT}(P) \).

Without loss of generality, we take \( i_1 \leq i_2 \). Assume that \( \text{Dec}_\text{Pre}[u_1] = \text{Dec}_\text{Pre}[u_2] \), we have that \( m_P[\text{SA}(m_P)[i_1] : \text{SA}(m_P)[i_1] + \text{LRS}(P)[i_1] - 1] = m_P[\text{SA}(m_P)[i_2] : \text{SA}(m_P)[i_2] + \text{LRS}(P)[i_2] - 1] \) and thus for all \( k \in [i_1, i_2] \), \( m_P[\text{SA}(m_P)[i_1] : \text{SA}(m_P)[i_1] + \text{LRS}(P)[i_1] - 1] = m_P[\text{SA}(m_P)[k] : \text{SA}(m_P)[k] + \text{LRS}(P)[k] - 1] \). Hence, we have \( \text{LCP}(P)[k] = \text{LRS}(P)[i_1] \) and \( \text{LRS}(P)[k] = \text{LRS}(P)[i_1] \). Therefore by the definition of \( \text{Decomp}_\text{BWT}(P) \), we get \( u_1 = u_2 \).

Surjectiveness. Let \( v \) be a prefix of a string of \( \overrightarrow{P.S} \). The string \( \overrightarrow{v} \) is a suffix of a string of \( P.S \).

By the definition of \( \text{BWT}(P) \), the table \( \text{SA}(m_P) \) gives, for a position \( i \), the starting position of the \( i \)th suffix of \( m_P \) in lexicographic order. Hence, there is a bijection between \( \text{Suffix}(m_P) \) and the set of positions in \( \text{SA}(m_P) \). Let \( k \in [1, |m_P|] \) such that \( s = m_P[\text{SA}(m_P)[k] : |m_P|] \). As \( \overrightarrow{v} \) is a suffix of a string of \( P.S \) and a prefix of \( s \), we have \( \overrightarrow{v} = m_P[\text{SA}(m_P)[k] : \text{SA}(m_P)[k] + \text{LRS}(P)[k] - 1] \). We take \( u = [i, j] \in \text{Decomp}_\text{BWT}(P) \) such that \( k \in [i, j] \). By Lemma 3, \( \overrightarrow{w} = \text{Dec}_\text{Pre}[u] \).

By Proposition 2, there exists an integer interval partition of \( [1, |m_P|] \) (i.e., \( \text{Decomp}_\text{BWT}(P) \)) that is in bijection with the set of nodes of \( \text{AC}(\overrightarrow{P.S}) \). The following theorem extends this bijection by building an isomorphic graph of \( \text{AC}(P.S) \) whose set of nodes is \( \text{Decomp}_\text{BWT}(\overrightarrow{P}) \). This states how to simulate an Aho-Corasick automaton using the BWT (similarly to [22]).

**Theorem 4.** Using tables \( \text{BWT}(\overrightarrow{P}), \text{LCP}(\overrightarrow{P}), \text{LRS}(\overrightarrow{P}) \) and the function \( \text{LF}(\overrightarrow{P}) \), we can build the graph \( (\text{Decomp}_\text{BWT}(\overrightarrow{P}), A_T(\overrightarrow{P}) \cup A_F(\overrightarrow{P})) \) that is isomorphic to \( \text{AC}(P.S) \).

\[ A_T(P) := \{ (u, v) \in \text{Decomp}_\text{BWT}(P)^2 \mid \exists x \in u \text{ such that } \text{LF}(P)[x] \neq 0 \text{ and } \text{LF}(P)[x] < v \}, \]

\[ A_F(P) := \{ (u, v) \in \text{Decomp}_\text{BWT}(P)^2 \mid \max_{k < i} \text{LRS}(P)[k] = \min_{k < i} \text{LCP}(P)[i] \} \]

More precisely, the graph \( (\text{Decomp}_\text{BWT}(\overrightarrow{P}), A_T(\overrightarrow{P})) \) is isomorphic to the tree \( \text{ACT}(P.S) \) and the graph \( (\text{Decomp}_\text{BWT}(\overrightarrow{P}), A_F(\overrightarrow{P})) \) is isomorphic to the tree \( \text{ACFL}(P.S) \) (see Appendix).

**Link between BWT and XBW.** After giving a new proof of the relation between Aho-Corasick automaton and BWT, we exhibit a new (bijective) link between the BWT and the XBW, which takes into account the order in the multi-string (Theorem 5). This leads to both, another construction algorithm of the XBW from the BWT, and to a construction of the BWT from the XBW, and thereby extends Manzini’s results (Corollary 6).

Alike the definition of \( \text{Decomp}_\text{BWT}(P) \) for an ordered set of strings \( P \), we define \( \text{Decomp}_\text{XBW}(T) \) of a tree \( T \) of \( t \) nodes as the integer interval partition of \( [1, t] \) such that

\[ [i, j] \in \text{Decomp}_\text{XBW}(T) \text{ iff } \begin{cases} \text{XBWL}(T)[k] = 1, & \text{for } k \in \{i - 1, j\} \\ \text{XBWL}(T)[k] = 0, & \text{for } k \in [i, j - 1] \end{cases} \]
In this section, we show how to use Theorem 5 to compute the permutation that leads to a BWT and a XBWT having the minimum Run Length Encoding.
4.1 Minimum permutation problem for BWT and XBWT

Run-Length Encoding [24] is a widely used method to compress strings. For a string \( w \), the Run-Length Encoding splits \( w \) into the minimum number of substrings containing a single symbol. The size of the Run-Length Encoding of \( w \) is the cardinality of the minimum decomposition. For example for \( abbaaccabbb = a^1b^2a^3c^2a^1b^3 \) (where the power notation \( \alpha^n \) means \( n \) copies of symbol \( \alpha \)), the size of the Run-Length Encoding is 6 (for the decomposition has 6 blocks).

We define Run-Length measures: \( d_B \) for a BWT, and \( d_X \) for a XBW (similar to those of [15]). For \( P \) an ordered set of strings, let \( d_B(P) \) be the cardinality of the set \( \{ i \in [1, \#BWT(P) - 1] \mid BWT(P)[i] \neq BWT(P)[i + 1] \} \). Similarly, let \( d_X(P) \) be the cardinality of the set \( \{ i \in [1, \#XBWT(ACT(P)) - 1] \mid XBWT(ACT(P))[i] \neq XBWT(ACT(P))[i + 1] \} \).

Given two ordered sets of strings, \( P_1 \) and \( P_2 \) such that \( P_1.S = P_2.S \) (i.e., they contain the same set of strings), Theorem 5 implies that their BWT may differ, and thus \( d_B(P_1) \) and \( d_B(P_2) \) may also differ. We define the following minimisation problems. As the Run-Length Encoding of BWT(P) has size \( d_B(P) + 1 \), finding an optimal solution of Min-Permutation-BWT can help compressing BWT(P).

**Definition 7** (Min-Permutation-BWT and Min-Permutation-XBWT). Let \( S \) be a set of strings. The problem **Min-Permutation-BWT** asks for an ordered set of strings \( P \) that minimises \( d_B(P) \) and such that \( P.S = S \). The problem **Min-Permutation-XBWT** asks for an ordered set of strings \( P \) that minimises \( d_X(P) \) and such that \( P.S = S \).

To simplify Min-Permutation-BWT, we consider specific ordered sets of strings. Let \( P \) be an ordered set of strings and let \( \perp \) denote the root of \( ACT(\overrightarrow{P}) \). We say that \( P \) is **topologically planar** if for each node \( u \) in \( ACT(\overrightarrow{P}) \) and \( v \in Leaves_{ACT(\overrightarrow{P})}(\perp) \setminus Leaves_{ACT(\overrightarrow{P})}(u) \), there does not exist \( u_1 \) and \( u_2 \) in \( Leaves_{ACT(\overrightarrow{P})}(u) \) such that \( u_1 <_{P.\sigma} v <_{P.\sigma} u_2 \). In other words, \( P \) is **topologically planar** if we can draw the tree \( ACT(\overrightarrow{P}) \) by ordering the leaves with \( <_{P.\sigma} \) without arcs crossing each other.
Let $P$ be an ordered set of strings, which is not necessarily topologically planar. We denote by $P_{tp}$ the ordered set of strings such that $P_{tp}=P.S$, and such that $P_{tp}.\sigma$ satisfies for any $u$ in $\text{ACT}(\overrightarrow{P})$, any $v \in \text{Leaves}_{\text{ACT}(\overrightarrow{P})}(u)$, and any $u_1,u_2$ in $\text{Leaves}_{\text{ACT}(\overrightarrow{P})}(u)$ such that $u_1 <_{P,\sigma} v <_{P,\sigma} u_2$, we have $u_1 <_{P,\sigma} v <_{P,\sigma} u_2$. As we have a bijection between the set of circular permutations of $P.S$ and the set of leaves of $\text{ACT}(\overrightarrow{P})$, we can unambiguously define the ordered set of strings $P_{tp}$ that is topologically planar.

**Proposition 8.** Let $P$ be an ordered set of strings. We have $d_B(P_{tp}) \leq d_B(P)$ and $d_B(P_{tp}) = d_X(\overrightarrow{P_{tp}})$.

**Proof.** For the inequality, let us prove that any modification of the order used to create $P_{tp}$ decreases the value of $d_B$. Let $P$ be an ordered set of strings that is not topologically planar. Let $u$ in $\text{ACT}(\overrightarrow{P})$, $v \in \text{Leaves}_{\text{ACT}(\overrightarrow{P})}(u)$, and let $u_1$ and $u_2$ in $\text{Leaves}_{\text{ACT}(\overrightarrow{P})}(u)$ such that $u_1 <_{P,\sigma} v <_{P,\sigma} u_2$. Let $P'$ be the copy of $P$ where the only difference is $u_1 <_{P_{tp},\sigma} u_2 <_{P_{tp},\sigma} v$. Let $x \in \text{Decomp}_{\text{BWT}}(P)$ such that $\text{Dec}_{\text{Pre}}[x] = u$. By Theorem 5, for all $y \in \text{Decomp}_{\text{BWT}}(P) \setminus \{x\}$, we have $\text{BWT}(P)[y] = \text{BWT}(P')[y]$. $\text{BWT}(P)[x] = \ldots \delta[u_1]\delta[v_1]\delta[u_2]\ldots$ and $\text{BWT}(P')[x] = \ldots \delta[u_1]\delta[v_1]\delta[u_2]\ldots$ with $v_1 = \text{Child}_{\text{ACT}(\overrightarrow{P})}(v)$.

As $\delta[u_1] = \delta[u_2]$ and $\delta[v_1] \neq \delta[v_1]$, we have $d_B(P') \leq d_B(P)$.

For the equality, it is enough to see that for any element $u$ in $\text{Decomp}_{\text{BWT}}(P_{tp})$, the numbers of distinct successive symbols are identical in $\text{BWT}(P_{tp})[u]$ and in $\text{XBWT}(\text{ACT}(\overrightarrow{P_{tp}}))$ $[\text{BWT}_{\text{XBW}}[u]]$. Thus, for two successive elements $u = [i_1,j_1]$ and $v = [i_2,j_2]$ of $\text{Decomp}_{\text{BWT}}(P_{tp})$, we obtain an equivalence between $\text{XBWT}[u] = [i'_1,j'_1]$ and $\text{XBWT}[v] = [i'_2,j'_2]$:

$$\text{BWT}(P_{tp})[j_1] = \text{BWT}(P_{tp})[i_2] \quad \text{iff} \quad \text{BWT}(\text{ACT}(\overrightarrow{P_{tp}}))[j'_1] = \text{BWT}(\text{ACT}(\overrightarrow{P_{tp}}))[i'_2]$$

Thanks to Proposition 8, we can restrict the search to ordered sets of strings that are topologically planar when solving $\text{Min-Permutation-BWT}$ or $\text{Min-Permutation-XBWT}$. Furthermore, an optimal solution of $\text{Min-Permutation-BWT}$ for $S$ is also an optimal solution of $\text{Min-Permutation-XBWT}$ for $\overrightarrow{S}$, and vice versa. This yields the following theorem.

**Theorem 9.** Let $S$ be a set of strings. We can find an optimal solution for $\text{Min-Permutation-BWT}$ and for $\text{Min-Permutation-XBWT}$ in $O(|S|\times\#\Sigma)$ time.

### 4.2 Proof of Theorem 9

As a reminder, Proposition 8 states that an optimal solution of $\text{Min-Permutation-BWT}$ is also an optimal solution of $\text{Min-Permutation-XBWT}$, and vice versa. In the following of this proof, we only prove the result regarding $\text{Min-Permutation-XBWT}$.

To start, let us give an overview of the algorithm:

1. we take a random permutation $\sigma$ of $S$ and define $P$ such that $P.S = S$ and $P.\sigma = \sigma$,
2. we build $\text{ACT}(P),\text{XBWT}(\text{ACT}(P))$ and $\text{Decomp}_{\text{XBW}}(\text{ACT}(P))$,
3. we find $P'$ which is an optimal solution of $\text{Min-Permutation-XBWT}$.

In the following, we define the problem $\text{Min-Permutation-Table}$ and explicit its link to $\text{Min-Permutation-XBWT}$ (Lemma 10). Lemma 10 gives us a linear time algorithm for finding an optimal solution of $\text{Min-Permutation-Table}$, and thus we can apply this algorithm to obtain an optimal solution for $\text{Min-Permutation-XBWT}$. Given $A$ an array of symbols of $\Sigma$, we define $\text{Char}(A)$ as the set of (different) symbols in $A$. Given $T$ an array of $n$ symbols of $\Sigma$ and $D$ an integer interval partition of $[1,n]$ such for each interval $[i,j]$ of $D$, $\#(\text{Char}(T[i,j])) = j - i + 1$ (i.e., all the symbols of
Proof. The problem Min-Permutation-Table is to find a \(T'\) such that for all \([i, j] \in D\), \(\#(\text{Char}(T'[i, j])) = \#(\text{Char}(T'[i, j]))\) and which minimises \(d_A(T', D) := \#(i \in [1, n - 1] \mid T'[i] \neq T'[i + 1])\). A proof of the following lemma is given in Appendix.

**Lemma 10.** Let \(S\) be a set of strings and let \(P\) be an ordered set of strings such that \(P.S = S\). For an optimal solution \(T'\) of Min-Permutation-Table for XBWT(ACT(P)) and for Decomp_XBWT(ACT(P)), there exists an optimal solution \(P'\) of Min-Permutation-XBWT for \(S\) such that XBWT(ACT(P')) = \(T'\).

Let \(T\) be an array of \(n\) symbols of \(\Sigma\) and let \(D\) be an integer interval partition of \([1, n]\) such for each interval \([i, j]\) of \(D\), \(\#(\text{Char}(T[i, j])) = j - i + 1\). Let \(A(D)\) be the array of all intervals in \(D\) in the order \(<\) and \(B(T, D)\) the array of size \(#(D) - 1\) such that the position \(i\) of \(B(T, D)\) is \(B(T, D)[i] = \text{Char}(A(D))[i] \cap \text{Char}(A(D))[i + 1])\). For any set of symbols \(C := \{c_1, \ldots, c_m\}\) where \(c_1 < \ldots < c_m\), we define \(\text{word}(C)\) as the string \(c_1 \ldots c_m\).

**Lemma 11.** Let \(T\) be an array of \(n\) symbols of \(\Sigma\) and let \(D\) be an integer interval partition of \([1, n]\) such for any interval \([i, j]\) of \(D\), \(\#(\text{Char}(T[i, j])) = j - i + 1\).

If there exists \(i\) in \([1, n]\) such that \([i, i]\) \(\in D\), then \(T'_1[1, i - 1]T'_2\) is an optimal solution of Min-Permutation-Table for \(T\) and for \(D\), where \(T'_1\) is an optimal solution of Min-Permutation-Table for \(T[1, i]\) and for \([i', j']\) \(\in D\ | j' \leq i\), and \(T'_2\) is an optimal solution of Min-Permutation-Table for \(T[i, n]\) and for \([i', j']\) \(\in D\ | j' \geq i\).

If there exists \(i\) in \([1, \#B(T, D)]\) such that \(\#(\text{Char}(T[D[i])]) = 0\), then \(T'_1T'_2\) is an optimal solution of Min-Permutation-Table for \(T\) and for \(D\), where \(T'_1\) is an optimal solution of Min-Permutation-Table for \(T[1, A(D)[i][1])\) and for \([i', j']\) \(\in D\ | j' \leq A(D)[i][1])\), and \(T'_2\) is an optimal solution of Min-Permutation-Table for \(T[A(D)[i][1])\) and for \([i', j']\) \(\in D\ | j' \geq A(D)[i][1])\).

If there exists \(i\) in \([1, \#B(T, D)]\) such that \(\#(\text{Char}(T[D[i])]) = 1\), then \(T'_1\text{char}T'_2\) is an optimal solution of Min-Permutation-Table for \(T\) and for \(D\), where \(B(T, D)[i] = \{a\}\), \(T'_1\) is an optimal solution of Min-Permutation-Table for \(T[1, A(D)[i][1])\) \(\cup \{\text{char}(A(D)[i])\}\) and for \([i', j']\) \(\in D\ | j' \leq A(D)[i][1])\) \(\cup \{A(D)[i][1])\} A(D)[i][1])\) \(\cup \{\text{char}(A(D)[i])\}\) and \(T'_2\) is an optimal solution of Min-Permutation-Table for \(T[1, A(D)[i][1])\) \(\cup \{\text{char}(A(D)[i])\}\) and for \([i', j']\) \(\in D\ | j' \leq A(D)[i][1])\) \(\cup \{A(D)[i][1])\} A(D)[i][1])\) \(\cup \{\text{char}(A(D)[i])\}\).

Proof. All the proofs are derived from the equality \(d_A(T_1[1, n], D) = d_A(T_1[1, i], \{(i', j') \in D \mid j' \leq i\}) + d_A(T_1[i, n], \{(i', j') \in D \mid j' \geq i\})\) for all \(i \in [1, n]\).

**Lemma 12.** Let \(T\) be an array of \(n\) symbols of \(\Sigma\) and let \(D\) be an integer interval partition of \([1, n]\) such that for any interval \([i, j]\) of \(D\), \(\#(\text{Char}(T[i, j])) = j - i + 1\). Whenever for any \(i\) in \([1, \#B(T, D)]\) one has \(\#(B(T, D)[i]) \geq 2\), Algorithm 1 gives an optimal solution for Min-Permutation-Table in \(O(\#T \times \#\Sigma)\) time.

Proof.

**Complexity.** To build the tables \((A(D)\) and \(B(T, D)\), we need \(O(\#T \times \#\Sigma)\) time. As both tables are smaller than \(T\), the for loop of Algorithm 1 also takes \(O(\#T \times \#\Sigma)\) time.

**Optimality.** As for any \(i\) in \([1, \#B(T, D)]\), one has \(\#(B(T, D)[i]) \geq 2\), we get \(B(T, D)[i] \{\text{last}\} \neq \emptyset\) (within the for loop of Algorithm 1). Let \(T^*\) be a string such that for all \([i, j] \in D\), \(\text{Char}(T'[i, j]) = \text{Char}(T'[i, j])\). The size of \(T^*\) is \(\#T\) and the number of intervals of \(D\) is \(\#D\), i.e., the maximum number of positions where two consecutive letters can be identical. Hence, we have \(d_A(T^*, D) \leq \#T - \#D + 1\). Let \(T'\) be the string given by Algorithm 1. We have \(d_A(T', D) = \sum_{u \in D}(\#u - 1) + 1 = \#T - \#D + 1\). This concludes the proof.
Algorithm 1: Computation of an array $T'$ of symbols satisfying for any $[i,j] \in D$, $\text{Char}(T[i,j]) = \text{Char}(T'[i,j])$.

**Input**: An instance of Min-Permutation-Table $T$ and $D$  
**Output**: A string $T'$

1. $\text{last} \leftarrow \$ such that $\$ \notin \Sigma$;
2. $T' \leftarrow$ empty string;
3. for $i \in \llbracket 1, \#B(T, D) \rrbracket$ do
   1. let $\text{lettre} \leftarrow$ random element of $B(T, D)[i] \setminus \{\text{last}\}$;
   2. $T' \leftarrow T' \text{word}(\text{Char}(A(D)[i]) \setminus \{\text{lettre}, \text{last}\})$;
   3. $T' \leftarrow T' \text{lettre lettre}$;
   4. $\text{last} \leftarrow \text{lettre}$;
   5. $T' \leftarrow T' \text{word}(\text{Char}(A(D)[\#A(D)]) \setminus \{\text{last}\})$;

4. return $T'$;

Lemma 13. Let $T$ be an array of $n$ symbols of $\Sigma$ and let $D$ be an integer interval partition of $[1, n]$ such that for each interval $[i, j]$ of $D$, $\#(\text{Char}(T[i, j])) = j - i + 1$. The problem Min-Permutation-Table can be solved in $O(\#T \times \#\Sigma)$ time.

**Proof.** By Lemma 12 and Lemma 11, we can compute an optimal solution of Min-Permutation-Table by splitting the interval, applying Algorithm 1 on each part, and then merging the strings output by Algorithm 1. ▶

5 Conclusion and Perspectives

Here, we present a new view of the Burrows-Wheeler Transform: as the text representation of an Aho-Corasick automaton that depends on the concatenation order. This induces a link between the Burrows-Wheeler Transform and the eXtended Burrows-Wheeler Transform, via the Aho-Corasick automaton. This link allows one to transform one structure into the other (for which we provide algorithms). We also exploit this link to find in linear time an ordering of input strings that optimises the compression of the concatenated strings.

Of course, it would be interesting to evaluate even empirically the gain of this compression on real life data. In bioinformatics, one wishes to index a collection of genomes from individual of the same species. For instance, it can be all variants of a virus genome within a host (e.g., HIV or Ebola virus in human); their number and relative frequency may change along time [2]. An application is then to compare the sequencing reads obtained from any new infected individual with this index to determine whether some variants are becoming more frequent or resistant to some treatment. Some viruses evolve rapidly to circumvent immune response, the number of potential variants can be large. Hence, it is practically relevant to reduce the index space by finding an optimal permutation of the genomes that would maximised the Run-Length Encoding of the index.

References


A Appendix

A.1 Linear time construction of tables \( \text{LRS}(P) \) and \( \text{LCP}(P) \)

To build in linear time the tables \( \text{LRS}(P) \) and \( \text{LCP}(P) \), we use Algorithm 2, which is proved in Lemma 14. We summarise this property in Proposition 15. The tables for the running example are illustrated in Figure 2.

\[ \text{Lemma 14. Let } i \in [1, |mP|]. \]

\[
\text{LRS}(P)[i] = \begin{cases} 0 & \text{if } i \in [1, \#(P.S)], \\ \text{LRS}(P)[\text{LF}(P)[i]] - 1 & \text{otherwise.} \end{cases}
\]

\[ \text{Proof. For any } i \in [1, |w|], \text{we know that } \text{SA}(w)[\text{LF}(w)[i]] = \text{SA}(w)[i] - 1 \text{ if } \text{SA}(w)[i] \geq 2 \text{ and } \text{SA}(w)[\text{LF}(w)[i]] = |w| \text{ otherwise; hence, we get} \]

\[ = \text{if } \text{SA}(w)[i] \geq 2, \]

\[ \text{LRS}(P)[\text{LF}(P)[i]] = \text{select}_b(mP|\text{SA}(mP)|\text{LF}(P)[i] : |mP|), 1) - 1 \\
= \text{select}_b(mP|\text{SA}(w)[i] - 1 : |mP|), 1) - 1 \\
= \begin{cases} 0 & \text{if } mP|\text{SA}(w)[i] - 1] = $, \\
\text{LRS}(P)[i] + 1 & \text{otherwise.} \end{cases} \]

\[ = \text{if } \text{SA}(w)[i] = 1, \]

\[ \text{LRS}(P)[\text{LF}(P)[i]] = \text{select}_b(mP|\text{SA}(mP)|\text{LF}(P)[i] : |mP|), 1) - 1 \\
= \text{select}_b(mP|\text{SA}(w)[i] : |mP|), 1) - 1 \\
= 0. \]

We define the function Letter from \([1, |mP|]\) to \( \Sigma \) such that \( C[\text{Letter}[i]] < i \leq C[\text{Letter}[i + 1]] \) (see the definition of LF on page 3) where \( \Sigma = \{c_1, \ldots, c_{\#\Sigma}\} \) with \( c_1 < \ldots < c_{\#\Sigma} \). Thus, we define \( \text{RLF}(P) \) from \([1, |mP|]\) to \([1, |mP|]\) such that

\[ \text{RLF}(P)[i] = \text{select}_{\text{Letter}[i]}(\text{BWT}(P), i - C[\text{Letter}[i]]). \]

Let \( i \) be an integer between 1 and \( |mP| \). We have

\[ \text{RLF}(P)[\text{LF}(P)[i]] = \text{select}_{\text{Letter}[\text{LF}(P)[i]]}(\text{BWT}(P), \text{LF}(P)[i] - C[\text{Letter}[\text{LF}(P)[i]]]) \\
= \text{select}_{\text{BWT}(P)[i]}(\text{BWT}(P), \text{LF}(P)[i] - C[\text{BWT}(P)[i]]) \\
= \text{select}_{\text{BWT}(P)[i]}(\text{BWT}(P), \text{rank}_{\text{BWT}(w)[i]}(\text{BWT}(w), i)) \\
= i. \]

Hence, the function \( \text{RLF}(P) \) is the reverse bijection of \( \text{LF}(P) \), and as \( mP|\text{SA}(mP)[i] - 1] = \text{BWT}(P)[i] \), one gets

\[ \text{LRS}(P)[i] = 0 \iff \text{BWT}(P)[\text{RLF}(P)[i]] = $ \\
\iff \text{BWT}(P)[\text{select}_{\text{Letter}[i]}(\text{BWT}(P), i - C[\text{Letter}[i]])] = $ \\
\iff \text{Letter}[i] = $ \\
\iff i \in [1, \#(P.S)]. \]

Therefore, we derive the desired equality, which concludes the proof. \( \blacksquare \)
Proposition 15. Let \( P \) be an ordered set of strings. Using tables \( \text{BWT}(P) \) and \( \text{LCP}(m_P) \), we can compute the tables \( \text{LRS}(P) \) and \( \text{LCP}(P) \) in a time that is linear in \( |m_P| \).

Proof. As the value of \( \text{LCP}(P) \) at each position corresponds to the minimum between the values at same positions in \( \text{LCP}(m_P) \) and in \( \text{LRS}(P) \), we only need to prove that the table \( \text{LRS}(P) \) can be computed from \( \text{BWT}(P) \) in linear time.

Using Algorithm 2 and Lemma 14, we can compute table \( \text{LRS}(P) \) in \( O(|m_P|) \) time.

\[ \text{Algorithm 2: Computation of table \( \text{LRS}(P) \). } \]

\begin{itemize}
  \item \textbf{Input}: The ordered set of strings \( P \)
  \item \textbf{Output}: The table \( \text{LRS}(P) \)
  \item Compute the table \( \text{BWT}(P) \);
  \item Initialise \( \text{LRS} \) as an empty table of length \( |m_P| \);
  \item for \( i \in [1, \#(P.S)] \) do
    \begin{itemize}
      \item \( \text{position} \leftarrow i; \)
      \item \( \text{nb} \leftarrow 0; \)
      \item \( \text{LRS}[\text{position}] \leftarrow \text{nb}; \)
      \item while \( \text{BWT}(P)[\text{position}] \neq \$ \) do
        \begin{itemize}
          \item \( \text{LRS}[\text{position}] \leftarrow \text{nb}; \)
          \item \( \text{nb} \leftarrow \text{nb} + 1; \)
          \item \( \text{position} \leftarrow \text{LF}[\text{position}]; \)
        \end{itemize}
    \end{itemize}
  \item return \( \text{LRS} \);
\end{itemize}

A.2 Proof of Theorem 4

The next propositions exhibit the links between \( \text{Decomp}_\text{BWT}(P) \) and first, the arcs of \( \text{ACT}(P.S) \) (Proposition 16), and second, the arcs of \( \text{ACFL}(P.S) \) (Proposition 17).

Proposition 16 (See Figure 4). The graph \( G_T(P) = (\text{Decomp}_\text{BWT}(P), A_T(P)) \) is isomorphic to the tree \( \text{ACT}(P.S) \), where

\[ A_T(P) := \{(u, v) \in (\text{Decomp}_\text{BWT}(P))^2 | \exists x \in u \text{ such that } \text{LF}(P)[x] \neq 0 \text{ and } \text{LF}(P)[x] \in v \}. \]

Proof. First, we show that there exists a bijection between the node set of \( \text{ACT}(P.S) \) and that of \( G_T(P) \). We reuse the bijection \( \text{Dec}_\text{Pre} \), which served in Proposition 2. Let us show that for each arc \((u, v)\) of \( A_T(P) \), \((\text{Dec}_\text{Pre}[u], \text{Dec}_\text{Pre}[v])\) is an arc of \( \text{ACT}(P.S) \), and vice versa. Let \((u, v) \in A_T(P), \ i.e., \ (u, v) \in (\text{Decomp}_\text{BWT}(P))^2 \) such that there exists \( x \in u \) satisfying \( \text{LF}(P)[x] \in v \).

According to [5], we know that \( \text{SA}(m_P)[\text{LF}(m_P)[i]] = \text{SA}(m_P)[i] - 1 \). By Lemma 14, we have \( \text{LRS}(P)[\text{LF}(P)[x]] = \text{LRS}(P)[x] + 1 \) for all \( x \) such that \( \text{LF}(P)[x] \neq 0 \). With both equalities and Lemma 3, we obtain

\[
\text{Dec}_\text{Pre}[v] = \begin{cases}
  m_P[\text{SA}(m_p)[\text{LF}(P)[x]] : \text{SA}(m_P)[\text{LF}(P)[x]] + \text{LRS}(P)[\text{LF}(P)[x]] - 1] \\
  m_P[\text{SA}(m_P)[x] - 1 : \text{SA}(m_P)[x] - 1 + \text{LRS}(P)[\text{LF}(P)[x]] - 1] \\
  m_P[\text{SA}(m_P)[x] - 1 : \text{SA}(m_P)[x] - 1 + \text{LRS}(P)[x]] \\
  m_P[\text{SA}(m_P)[x] : \text{SA}(m_P)[x] + \text{LRS}(P)[x] - 1] m_P[\text{SA}(m_P)[x] - 1] \\
  \text{Dec}_\text{Pre}[u] m_P[\text{SA}(m_P)[x] - 1].
\end{cases}
\]
The string Dec\_Pre[u] is thus the longest prefix of Dec\_Pre[v].

Let \((x, y)\) be an arc of \(\text{ACT}(\overrightarrow{P}, S)\). We take \(z\) a leaf in the subtree of \(\text{ACT}(\overrightarrow{P}, S)\) in \(y\). As \(x\) is the parent of \(y\) in \(\text{ACT}(\overrightarrow{P}, S)\), \(z\) is also a leaf in the subtree of \(\text{ACT}(\overrightarrow{P}, S)\) in \(x\). We take \(i \in [1, |m_P|]\) such that \(z\) is a prefix of \(m_P[i : |m_P|]\). Hence \(z\) is a prefix of \(m_P[i + |z| - |y| : |m_P|]\) and \(\overrightarrow{y}\) is a prefix of \(m_P[i + |z| - |x| : |m_P|]\). As \((x, y)\) is an arc of \(\text{ACT}(\overrightarrow{P}, S)\), \(|y| - |z| = 1\). Thus, choosing \(k\) such that \(\text{SA}(m_P)[k] = i + |z| - |y| + 1\), and \(u, v\) in Decom\_BWT(P)^2 such that \(k \in u\) and \(\text{LF}(P)[k] \in v\), we get \(\overrightarrow{y} = \text{Dec\_Pre}[u]\) and \(\overrightarrow{x} = \text{Dec\_Pre}[v]\). This concludes the proof. □

**Proposition 17** (See Figure 4). The graph \(G_F(P) = (\text{Decomp\_BWT}(P), A_F(P))\) is isomorphic to the tree \(\text{ACFL}(\overrightarrow{P}, S)\), where

\[
A_F(P) := \{ (u, v) \in \text{Decomp\_BWT}(P)^2 \mid \left( \max_{k < i} (k) \right) \in v \text{ with } u = [i, j] \}.
\]

**Proof.** First, let us show the following equivalence. Let \(u = [i, j] \in \text{Decomp\_BWT}(P)\).

\[\begin{align*}
w & \in \text{Decomp\_BWT}(P) \text{ such that } \exists k \in w \text{ with } k < i \text{ and } \text{LRS}(P)[k] = \min_{k \leq i \leq l} (\text{LCP}(P)[l]) \\
\iff \text{Dec\_Pre}[u]\text{ is a suffix of Dec\_Pre}[v].
\end{align*}\]

Let \(w \in \text{Decomp\_BWT}(P)\) such that \(\exists k \in w \text{ with } k < i \text{ and } \text{LRS}(P)[k] = \min_{k \leq l \leq i} (\text{LCP}(P)[l])\). Hence, we have for all \(l \in [k - 1, i]\), \(\text{LRS}(P)[k] \leq \text{LCP}(P)[l]\), and thus

\[
\begin{align*}
\text{Dec\_Pre}[u] &= m_P[\text{SA}(m_P)[i] : \text{SA}(m_P)[i] + \text{LRS}(P)[i] - 1] \\
&= m_P[\text{SA}(m_P)[i] + \text{LRS}(P)[k] : \text{SA}(m_P)[i] + \text{LRS}(P)[i] - 1] \\
&= m_P[\text{SA}(m_P)[i] + \text{LRS}(P)[k] : \text{SA}(m_P)[i] + \text{LRS}(P)[i] - 1] \\
&= m_P[\text{SA}(m_P)[i] + \text{LRS}(P)[k] : \text{SA}(m_P)[i] + \text{LRS}(P)[i] - 1] \text{Dec\_Pre}[w].
\end{align*}
\]

Let \(w = [i_1, j_1]\) and \(u = [i_2, j_2]\) be two elements of \(\text{Decomp\_BWT}(P)\) such that \(\text{Dec\_Pre}[w]\) is a suffix of \(\text{Dec\_Pre}[u]\). Hence, we have that \(m_P[\text{SA}(m_P)[i_1] : \text{SA}(m_P)[i_1] + \text{LRS}(P)[i_1] - 1]\) is a prefix of \(m_P[\text{SA}(m_P)[i_2] : \text{SA}(m_P)[i_2] + \text{LRS}(P)[i_2] - 1]\). By the definition of \(\text{BWT}(P)\), for all \(l \in [i_1, i_2]\), \(\text{LRS}(P)[i_1] \leq \text{LCP}(P)[l]\). This concludes the proof of the equivalence. By the equivalence, given \(u\) and \(v\) in \(\text{Decomp\_BWT}(P)\) such that \(\text{Dec\_Pre}[u]\) is a suffix of \(\text{Dec\_Pre}[v]\), for all \(k_1 \in u\) and \(k_2 \in v\), we have \(k_1 \leq k_2\). Hence, by taking the largest \(w\) satisfying the first step of the inequality, we obtain the longest suffix and vice versa. □

### A.3 Proof of Theorem 5

**Proof.** We define \(T_B\) (resp. \(T_X\)) as the array of intervals of \(\text{Decomp\_BWT}(P)\) (resp. \(\text{Decomp\_XBW}(\text{ACT}(\overrightarrow{P}))\)) sorted in the interval order. Let us prove that \(T_B\) and \(T_X\) have the same length, and that at the same position \(i\), \(T_B[i]\) and \(T_X[i]\) represent the same prefix of \(\overrightarrow{P}\). By Proposition 2, the length of \(T_B\) is \#\text{Prefix}(\overrightarrow{P}, S).\) By the definition of \(\text{Decomp\_XBW}\), the length of \(T_X\) is the number of \(1\) in \(\text{XBWL}(\text{ACT}(\overrightarrow{P}))\), i.e., the number of internal nodes of \(\text{ACT}(\overrightarrow{P})\), and thus is equal to \#\text{Prefix}(\overrightarrow{P}, S).\) Hence, \(T_B\) and \(T_X\) have the same cardinalities.

Let \(i \in [1, \#\text{Prefix}(\overrightarrow{P}, S)]\). By Proposition 2, \(T_B[i]\) represents the \(i^{\text{th}}\) suffix of strings of \(P.S\) in lexicographic order. By the definition of \(\text{XBWL}(\text{ACT}(\overrightarrow{P}))\), all the nodes \(\text{PA}(\text{ACT}(\overrightarrow{P}))[k]\) for \(k \in T_X[i]\) have the same parent \(z\) in \(\text{ACT}(\overrightarrow{P})\), and \(z\) represents the \(i^{\text{th}}\) node in \(\prec\) order, i.e., the \(i^{\text{th}}\) suffix of strings of \(P.S\) in lexicographic order.
As for a given position \(i\), \(T_B[i]\) and \(T_X[i]\) represent the same prefix of \(\hat{P}\), we define the bijection \(\text{BWT}_X \hat{P}\) such that for any \(i\) in \([1, \#\text{Prefix} (\hat{P}, S)]\), one has \(\text{BWT}_X \hat{P}[T_B[i]] = T_X[i]\).

As the tree \(\text{ACT}(\hat{P})\) represents the Aho-Corasick tree of \(\hat{P} \preceq S\), we have a bijection \(b_1\) from the node set of \(\text{ACT}(\hat{P})\) onto the set of prefixes of \(\hat{P} \preceq S\). By the definition of functions \(\pi\) and \(\delta\) (see page 4), for any node \(v\) of \(\text{ACT}(\hat{P})\), we have \(b_1(v) = \pi(v) \delta(v)\).

By the definition of \(\text{XBWT}(\text{ACT}(\hat{P}))\), we have that for \(T_X[i] = [i', j']\)
\[
\#T_X[i] = \#\{v \text{ node of } \text{ACT}(\hat{P}) | \pi(v) = \delta(\text{Dec}_{\text{Pre}[\text{BWT}_X \hat{P}]^{-1}[T_X[i]]})
\]
\[
= \#\{v \text{ node of } \text{ACT}(\hat{P}) | v \text{ is a child of } b_1^{-1}(\text{Dec}_{\text{Pre}[\text{BWT}_X \hat{P}]^{-1}[T_X[i]])}\}.
\]

As \(i' \in T_X[i]\) and \(b_1^{-1}(\text{Dec}_{\text{Pre}[\text{BWT}_X \hat{P}]^{-1}[T_X[i]]}) = \text{Parent}_{\text{ACT}(\hat{P})}(\text{PA}(\text{ACT}(\hat{P}))[i'])), \)
\[
\#T_X[i] = \#\{v \text{ node of } \text{ACT}(\hat{P}) | v \text{ is a child of } \text{Parent}_{\text{ACT}(\hat{P})}(\text{PA}(\text{ACT}(\hat{P}))[i'])))
\]
\[
= \#\text{Children}_{\text{ACT}(\hat{P})}(\text{PA}(\text{ACT}(\hat{P}))(i'))\).
\]

Let \(\{x_1, x_2, \ldots, x_{\#T_X[i]}\}\) be the set of children of \(\text{Parent}_{\text{ACT}(\hat{P})}(\text{PA}(\text{ACT}(\hat{P}))[i']))\) sorted such that \(x_1 <_{P, \sigma} \ldots <_{P, \sigma} x_{\#T_X[i]}\). By the definition of \(\text{XBWT}(\text{ACT}(\hat{P}))\), for any \(k \in [1, \#T_X[i]]\), we have \(\text{XBWT}(\text{ACT}(\hat{P}))[i' + k - 1] = \pi(x_k)\). By the definition of \(\text{BWT}(P)\), for \(T_B[i] = [i'', j'']\), we get
\[
\#T_B[i] = \#\{w \in \hat{P} \preceq S | \text{Dec}_{\text{Pre}[T_B[i]]} \text{ is a prefix of } w\}
\]
\[
= \#\{v \text{ node of } \text{ACT}(\hat{P}) | v \text{ is an ancestor of } \text{Dec}_{\text{Pre}[T_B[i]]})\}.
\]

As \(b_1^{-1}(\text{Dec}_{\text{Pre}[T_B[i]]}) = b_1^{-1}(\text{Dec}_{\text{Pre}[\text{BWT}_X \hat{P}]^{-1}[T_X[i]]}) = \text{Parent}_{\text{ACT}(\hat{P})}(\text{PA}(\text{ACT}(\hat{P}))[i']), \)
\[
\#T_B[i] = \#\{v \text{ node of } \text{ACT}(\hat{P}) | v \text{ is an ancestor of } \text{Dec}_{\text{Pre}[T_B[i]]})\}
\]
\[
= \#\text{Leaves}_{\text{ACT}(\hat{P})}(\text{PA}(\text{ACT}(\hat{P}))[i']))\).
\]

Let \(\{y_1, y_2, \ldots, y_{\#T_B[i]}\}\) be the set of leaves of the subtree of \(\text{Parent}_{\text{ACT}(\hat{P})}(\text{PA}(\text{ACT}(\hat{P}))[i'))\) in \(\text{ACT}(\hat{P})\) sorted such that \(y_1 <_{P, \sigma} \ldots <_{P, \sigma} y_{\#T_B[i]}\). Given \(k \in [i'', j'']\), we define \(w_P[k]\) as the string \(w_P[\text{select}_S(m_P, \text{rank}_S(m_P, \text{SA}(P)[k] - 1) + 1 : \text{SA}(P)[k] + \text{LRS}(P)[k] - 1]\). For all \(k \in [i'', j'']\), the string \(w_P[k]\) is a string of \(P, S\). Moreover, the definition of \(<_{P, \sigma}\) implies \(w_P[i''] <_{P, \sigma} \ldots <_{P, \sigma} w_P[j'']\). As for \(l \in [1, \#T_B[i]]\), the string \(\pi(y_l)\) is also a string of \(P, S\), we get \(\pi(y_l) = w_P[l'' + l - 1]\). Given \(x\) and \(y\) in \([i'', j''] \in \text{Decomp}_B(P, W)\), we obtain the following equivalences between orders:
\[
x < y \Leftrightarrow w_P[x] <_{P, \sigma} w_P[y] \Leftrightarrow \pi(x) <_{P, \sigma} \pi(y) \Leftrightarrow y <_{P, \sigma} y + i'' + 1.
\]
By the definition of \(\text{BWT}(P)\), for all \(k \in [1, \#T_B[i]]\), we get that
\[
\text{BWT}(P)[i'' + k - 1] = m_P[\text{SA}(m_P)[i'' + k - 1] - 1]
\]
\[
= w_P[i'' + k - 1][w_P[i'' + k - 1] - \text{LRS}[i'' + k - 1] + 1]
\]
\[
= \pi(y_l)[w_P[i'' + k - 1] - \text{LRS}[i'' + k - 1] + 1]
\]
\[
= \delta(\text{Child}_{\text{ACT}(\hat{P})}(y_l) \text{Parent}_{\text{ACT}(\hat{P})}(\text{PA}(\text{ACT}(\hat{P}))[i']))\).
\]
A.4 Proof of Corollary 6

Let $P$ be an ordered set of strings. To compute tables $XBWT(\overrightarrow{ACT(P)})$ and $XBWL(\overrightarrow{ACT(P)})$ using only $BWD(P)$, $LCP(P)$ and $LRS(P)$, we first define a new table $BWD(P)$.

The Burrows-Wheeler Decomposition of $P$, denoted by $BWD(P)$, is the array of length $\#\text{Decomp}_BWT(P)$ such that for each position $i$, $BWD(P)[i]$ is the cardinality of the $i$th element of $\text{Decomp}_BWT(P)$ in interval order.

\bf Lemma 18. Using tables $LCP(P)$ and $LRS(P)$, Algorithm 3 computes $BWD(P)$ in linear time in $\|P.S\|$ and the table $BWD(P)$ can be stored with $\|P.S\| \times \log(\#(P.S))$ bits.

\textbf{Proof of Lemma 18.} For each $i$ in $[1,|m_P|]$, at the beginning of the loop for, we have that $LCP(P)[i-j] \neq LRS(P)[i-j]$ and for all $k \in [i-j+1,i-1]$, $LCP(P)[k] \neq LRS(P)[k]$. Hence, if $LCP(P)[i] \neq LRS(P)[i]$, the interval $[i-j,i-1]$ is an element of $\text{Decomp}_BWT(P)$ and the cardinality of $[i-j,i-1]$ is $j$. Otherwise, we increase $j$ by 1 because the position $i$ does not correspond to a new interval of $\text{Decomp}_BWT(P)$. For the complexity, as each step of the loop can be computed in constant time, Algorithm 3 computes $BWD(P)$ in linear time in $\|P.S\|$. As for each position $i$ of $BWD(P)$, $BWD(P)[i]$ represents the number of strings of $P.S$ having as suffix $\text{Dec}_Pre[i]$, where $u$ is the $i$th element of $\text{Decomp}_BWT(P)$ sorted in interval order, it follows that $BWD(P)[i] \leq \#(P.S)$. This concludes the proof. \hfill \blacksquare

\begin{algorithm}
\DontPrintSemicolon
\KwIn{The tables $LCP(P)$ and $LRS(P)$}
\KwOut{The table $BWD(P)$}
\text{BWD} \leftarrow \text{empty list;}
\text{j} \leftarrow 0;
\text{for } i \in [1,|m_P|] \text{ do}\]
\text{if } LCP(P)[i] \neq LRS(P)[i] \text{ then}
\text{insert } j \text{ at the end of } \text{BWD;}
\text{j} \leftarrow 1;
\text{else } j \leftarrow j + 1 ;
\text{return } \text{BWD;}
\end{algorithm}

\bf Lemma 19. Using tables $BWT(P)$ and $BWD(P)$, Algorithm 4 computes the tables $XBWT(\overrightarrow{ACT(P)})$ and $XBWL(\overrightarrow{ACT(P)})$ in linear time of $\|P.S\| \times \#\Sigma$.

\textbf{Proof of Lemma 19.} Algorithm 4 is an application of Theorem 5. \hfill \blacksquare

Using Algorithm 3 to build $BWD(P)$ and Algorithm 4, we can compute tables $XBWT(\overrightarrow{ACT(P)})$ and $XBWL(\overrightarrow{ACT(P)})$ in linear time of $\|P.S\| \times \#\Sigma$.

We define the equivalent of $BWD$ for $\text{Decomp}_XBW(P)$. The eXtended Burrows Wheeler Decomposition ($XBWD$) of a tree $T$ is the array of length $\#XBWL(T)$ such that for any position $i$, $XBWD(P)[i]$ equals the cardinality of the $i$th element of $\text{Decomp}_XBW(T)$ sorted in interval order.

\bf Lemma 20. From the table $XBWL(\overrightarrow{ACT(S)})$, Algorithm 5 computes $XBWD(\overrightarrow{ACT(S)})$ in linear time in $\|P.S\|$ and the table $XBWD(\overrightarrow{ACT(S)})$ can be stored in $\|P.S\| \times \log(\#\Sigma)$ bits.
Algorithm 4: Computation of tables $\text{XBWT}(\text{ACT}(\overrightarrow{P}))$ and $\text{XBWL}(\text{ACT}(\overrightarrow{P}))$.

**Input**: The tables $\text{BWT}(P)$ and $\text{BWD}(P)$
**Output**: The tables $\text{XBWT}(\text{ACT}(\overrightarrow{P}))$ and $\text{XBWL}(\text{ACT}(\overrightarrow{P}))$

Initialise both $\text{XBWT}$ and $\text{XBWL}$ as empty lists;

\[ nb \leftarrow 1; \]

for \( i \in [1, \#\text{BWD}(P)] \) do

\[
D \leftarrow \text{Dictionary such that for all } l \in \Sigma, D[l] \leftarrow \text{true}; \\
\text{begin} \leftarrow nb; \\
\text{last} \leftarrow \text{begin} + \text{BWD}(P)[i] - 1; \\
nb \leftarrow \text{last} + 1; \\
\text{for } j \in [\text{begin}, \text{last}] \text{ do} \\
\quad \text{if } D[\text{BWT}(P)[j]] \text{ then} \\
\quad \quad \text{insert } \text{BWT}(P)[j] \text{ at the end of } \text{XBWT}; \\
\quad \quad \text{insert } 0 \text{ at the end of } \text{XBWL}; \\
\quad \quad D[\text{BWT}(P)[j]] \leftarrow \text{false}; \\
\text{XBWL}[\#\text{XBWL}] \leftarrow 1; \\
\text{return } \text{XBWT} \text{ and } \text{XBWL};
\]

Proof. The proof of Lemma 20 is similar to that of Lemma 18. As for each position \( i \) of $\text{XBWD}(\text{ACT}(S))$, $\text{XBWD}(\text{ACT}(S))[i]$ represents the number of the right extensions of the strings $\text{PA}(\text{ACT}(S))[i]$ in \( S \) (i.e., the number of different strings $\text{PA}(\text{ACT}(S))[i]$ a which are substrings of a string of \( S \) for some $a \in \Sigma$), we have $\text{XBWD}(\text{ACT}(S))[i] \leq \#\Sigma$. \( \blacklozenge \)

Algorithm 5: Computation of table $\text{XBWD}(\text{ACT}(S))$.

**Input**: The table $\text{XBWL}(\text{ACT}(S))$; **Output**: The table $\text{XBWD}(\text{ACT}(S))$

\[
\text{XBWD} \leftarrow \text{empty list}; \\
j \leftarrow 1; \\
\text{for } i \in [1, \#(\text{XBWL}(\text{ACT}(S)))] \text{ do} \\
\quad \text{if } \text{XBWL}(\text{ACT}(S)) = 1 \text{ then} \\
\quad \quad \text{insert } j \text{ at the end of } \text{XBWD}; \\
\quad \quad j \leftarrow 1; \\
\quad \text{else } j \leftarrow j + 1; \\
\text{return } \text{XBWD};
\]

Lemma 21. Using tables $\text{XBWT}(\text{ACT}(S))$ and $\text{XBWD}(\text{ACT}(S))$, we can build the tables $\text{BWT}(\overrightarrow{P})$, $\text{LCP}(\overrightarrow{P})$ and $\text{LRS}(\overrightarrow{P})$ in linear time of $\|P.S\| \times \#\Sigma$, where $P$ is a topologically planar, ordered set of strings such that $P.S = S$.

Proof. In [9], Ferragina et al. prove that with both tables $\text{XBWT}(\text{ACT}(S))$ and $\text{XBWD}(\text{ACT}(S))$ one can access in constant time the children and the parents in $\text{ACT}(S)$. Hence, we can compute in linear time in $\|P.S\|$, the table $\text{TL}(\text{ACT}(S))$, where in each position of $i$ we store the number of leaves in the subtree of the node $\text{PA}(\text{ACT}(S))[i]$. We finish the proof using the results of Theorem 5 and an algorithm similar to Algorithm 4. \( \blacklozenge \)
A.5 Proof of Lemma 10

Proof. Let $T'$ be an optimal solution of Min-Permutation-Table for $XBWT(ACT(P))$ and for $Decomp_{XBW}(ACT(P))$. By Theorem 5, for each $[[i,j]] \in Decomp_{XBW}(ACT(P))$, the order of the symbols in $XBWT(ACT(P))[i,j]$ depends on the order on the children of the parent of $PA(ACT(P))[i]$. Hence, the choice of $T'$ corresponds to the choice of an order for each internal node of $ACT(P)$ over all its children. As we can extend this order to a total order on the leaves of $ACT(P)$, we can build $P'$ the ordered set of strings satisfying $P'.S = S^\#$ and $P'.\sigma(\pi(f_i)) = s_i$, where the order on the leaves of $ACT(P)$ is $f_1 < \ldots < f_{\#S}$ and $s_1 < \ldots < s_{\#S}$ are the strings of $S$ in lexicographic order. ◀