Linking BWT and XBW via Aho-Corasick automaton : Applications to run-length encoding

Cazaux, B.

Schloss Dagstuhl - Leibniz-Zentrum für Informatik
2019

Cazaux , B & Rivals , E 2019 , Linking BWT and XBW via Aho-Corasick automaton : Applications to run-length encoding . in N Pisanti & S P Pissis (eds) , 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019) . , 24 , Leibniz International Proceedings in Informatics (LIPIcs) , vol. 128 , Schloss Dagstuhl - Leibniz-Zentrum für Informatik , Dagstuhl , Annual Symposium on Combinatorial Pattern Matching , Pisa , Italy , 18/06/2019 . https://doi.org/10.4230/LIPIcs.CPM.2019.24

http://hdl.handle.net/10138/313200
https://doi.org/10.4230/LIPIcs.CPM.2019.24

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

Related Version A full version of the paper is available at https://arxiv.org/abs/1805.10070.

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 $\{[\![i_1, j_1]\!], \dots [\![i_n, j_n]\!]\}$ 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 $j < i'$. 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 σ 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 σ of E and an element $e \in E$, we denote by $\overline{\sigma_e}$ the function from $[\![1, \#E]\!]$ to E such that for all $i \in [\![1, \#E]\!]$, $\overline{\sigma_e}(i) := \sigma^{(i+\#E-1)}(e)$. 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 Σ be a finite alphabet. A *string* w of length n over Σ is a sequence of symbols $w[1] \dots w[n]$ where $w[i] \in \Sigma$ for all $i \in [\![1, n]\!]$. Σ^* is the set of all finite strings over Σ. The *length* of a string w is denoted by $|w|$. A *substring* of w is written as $w[i, j] := w[i] \dots w[j]$ 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 $\overleftarrow{w}$, is the string $w[n] \dots w[2]w[1]$. We define the *lexicographic order* $<$ on strings as usual.

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

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 $\texttt{SA}(w)$, 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 $\texttt{BWT}(w)$, is the array containing a permutation of the symbols of w which satisfies $\texttt{BWT}(w)[i] := w[\texttt{SA}(w)[i] - 1]$ if $\texttt{SA}(w)[i] > 1$, and $\texttt{BWT}(w)[i] := w[|w|]$ otherwise. The *Longest Common Prefix* table (LCP) [5] of w, denoted $\texttt{LCP}(w)$, is the array of integers such that $\texttt{LCP}(w)[i]$ equals the length of the longest common prefix between the suffixes of w starting at positions $\texttt{SA}(w)[i]$ and $\texttt{SA}(w)[i-1]$ if $i > 1$, and 0 otherwise.

For any string $s \in \Sigma^*$ and any symbol $c \in \Sigma$, one defines the functions denoted `rank` and `select` as follows: $\texttt{rank}_c(s, i)$ is the number of occurences of c in $s[1, i]$, and $\texttt{select}_c(s, j)$ is the position of the j^{th} occurence of c in s. The arrays $\texttt{BWT}(w)$ and $\texttt{LCP}(w)$ can be computed in $O(|w|)$ time [5, 10]. Simultaneously, one can compute `rank` and `select` for $\texttt{BWT}(w)$ at no additional cost and implement them such that any `rank` or `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 $\texttt{LF}(w)[i] := C[\texttt{BWT}(w)[i]] + \texttt{rank}_{\texttt{BWT}(w)[i]}(\texttt{BWT}(w), i)$. It is proved that $\texttt{SA}(w)[\texttt{LF}(w)[i]] = \texttt{SA}(w)[i] - 1$ if $\texttt{SA}(w)[i] \geq 2$ and $\texttt{SA}(w)[\texttt{LF}(w)[i]] = |w|$ 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 Σ and is alphabetically smaller than all other symbols. We denote by m_P the string obtained by concatenating the strings of $P.S := \{s_1, \ldots, s_{\#(P.S)}\}$ separated by a \$ and following the order $\overline{P.\sigma}$. i.e., $m_P := s_{\overline{P.\sigma}(1)}\$s_{\overline{P.\sigma}(2)}\$\ldots\$s_{\overline{P.\sigma}(n)}\$$ (See Figure 1). We extend the notion of $\texttt{BWT}$ of a string to an ordered set of strings P: the $\texttt{BWT}$ of P is the $\texttt{BWT}$ of the string m_P, i.e., $\texttt{BWT}(P) := \texttt{BWT}(m_P)$. We extend similarly the $\texttt{LF}$ function by setting that $\texttt{LF}(P) := \texttt{LF}(m_P)$ (by using $\texttt{rank}$ on $\texttt{BWT}(P)$).

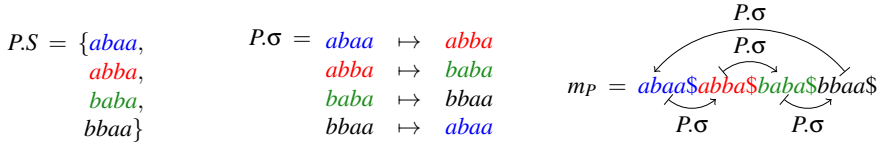

Figure 1 Running example with $P.S := \{abaa, abba, baba, bbaa\}$ and the corresponding m_P.

We define the *Longest Representative Suffix* table ($\texttt{LRS}$) of P as the array of $|m_P|$ integers satisfying: for any $i \in [\![1, |m_P|]\!]$ one has $\texttt{LRS}(P)[i] := \texttt{select}_\$(m_P[\texttt{SA}(m_P)[i] : |m_P|], 1) - 1$. The entry $\texttt{LRS}(P)[i]$ gives the length of the substring of m_P starting at position $\texttt{SA}(m_P)[i]$ up to, but not including the next \$. Using the $\texttt{LRS}$ table, we extend the notion of $\texttt{LCP}$ table to an ordered set of strings. For P, we set $\texttt{LCP}(P)[i] := \min(\texttt{LCP}(m_P)[i], \texttt{LRS}(P)[i])$ for any $i \in [\![1, |m_P|]\!]$. Using tables $\texttt{BWT}(P)$ and $\texttt{LCP}(m_P)$, we can compute the tables $\texttt{LRS}(P)$ and $\texttt{LCP}(P)$ in linear time in $|m_P|$ (see Appendix). Figure 2 illustrates the $\texttt{LCP}$ and $\texttt{LRS}$ 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 $\texttt{Parent}_\mathcal{T}(u)$ the parent of u in $\mathcal{T}$, by $\texttt{Children}_\mathcal{T}(u)$ the set of children of u in $\mathcal{T}$, and by $\texttt{Leaves}_\mathcal{T}(u)$ 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 $\mathcal{T}$, $\#\texttt{Children}_{\mathcal{B}_\mathcal{T}(v)}(u) = 1$, we denote by $\texttt{Child}_{\mathcal{T}(v)}(u)$ the unique element of $\texttt{Children}_{\mathcal{B}_\mathcal{T}(v)}(u)$. Given a total order $\prec$ on $\texttt{Leaves}_\mathcal{T}(\perp)$, we extend this order on the set $\texttt{Children}_\mathcal{T}(v)$ for any node v of $\mathcal{T}$ by: for any x, y in $\texttt{Children}_\mathcal{T}(v)$, $x \prec y$ **iff** $\min_{x' \in \texttt{Leaves}_\mathcal{T}(x)} x' \prec \min_{y' \in \texttt{Leaves}_\mathcal{T}(y)} y'$.

Aho-Corasick tree. The *Aho-Corasick automaton* ($\texttt{AC}$) [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* ($\texttt{ACT}$), has an arc from a prefix u to a different prefix v iff u is the longest proper prefix of v in $\texttt{Prefix}(S)$ (see Figure 4). The second tree, termed *Aho-Corasick Failure link* ($\texttt{ACFL}$), has an arc from a prefix u to a different prefix v iff v is the longest proper **suffix** of u in $\texttt{Prefix}(S)$.

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 Σ. We define the functions δ and π 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 $\mathcal{T}$.

LCP(P)	LRS(P)	SA(m_P)																			BWT(P)
0	0	20	$	a	b	a	a	$	a	b	b	a	$	b	a	b	a	$	b	b	a	a
0	0	5	$	a	b	b	a	$	b	a	b	a	$	b	b	a	a	$	a	b	a	a
0	0	10	$	b	a	b	a	$	b	b	a	a	$	a	b	a	a	$	a	b	b	a
0	0	15	$	b	b	a	a	$	a	b	a	a	$	a	b	b	a	$	b	a	b	a
0	1	19	a	$	a	b	a	a	$	a	b	b	a	$	b	a	b	a	$	b	b	a
1	1	4	a	$	a	b	b	a	$	b	a	b	a	$	b	b	a	a	$	a	b	a
1	1	9	a	$	b	a	b	a	$	b	b	a	a	$	a	b	a	a	$	a	b	b
1	1	14	a	$	b	b	a	a	$	a	b	a	a	$	a	b	b	a	$	b	a	b
1	2	18	a	a	$	a	b	a	a	$	a	b	b	a	$	b	a	b	a	$	b	b
2	2	3	a	a	$	a	b	b	a	$	b	a	b	a	$	b	b	a	a	$	a	b
1	3	12	a	b	a	$	b	b	a	a	$	a	b	a	a	$	a	b	b	a	$	b
3	4	1	a	b	a	a	$	a	b	b	a	$	b	a	b	a	$	b	b	a	a	$
2	4	6	a	b	b	a	$	b	a	b	a	$	b	b	a	a	$	a	b	a	a	$
0	2	8	b	a	$	b	a	b	a	$	b	b	a	a	$	a	b	a	a	$	a	b
2	2	13	b	a	$	b	b	a	a	$	a	b	a	a	$	a	b	b	a	$	b	a
2	3	17	b	a	a	$	a	b	a	a	$	a	b	b	a	$	b	a	b	a	$	b
3	3	2	b	a	a	$	a	b	b	a	$	b	a	b	a	$	b	b	a	a	$	a
2	4	11	b	a	b	a	$	b	b	a	a	$	a	b	a	a	$	a	b	b	a	$
1	3	7	b	b	a	$	b	a	b	a	$	b	b	a	a	$	a	b	a	a	$	a
3	4	16	b	b	a	a	$	a	b	a	a	$	a	b	b	a	$	b	a	b	a	$

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

▶ **Example 1.** With the tree of Figure 3, we have $\delta(\textcircled{13}) = b$ and $\pi(\textcircled{13}) = aba = \delta(\textcircled{12})\delta(\textcircled{8})\delta(\textcircled{1})$, and also $\delta(\textcircled{4}) = b$ and $\pi(\textcircled{4}) = baa = \delta(\textcircled{3})\delta(\textcircled{2})\delta(\textcircled{1})$. Thus, $\textcircled{13} \prec \textcircled{4}$ since $\pi(\textcircled{13}) < \pi(\textcircled{4})$.

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

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

[1] In [9], the XBW-*transform* is defined as $\texttt{XBW}(T)[i] := \langle \texttt{XBWT}[i], \texttt{XBWL}[i] \rangle$, for any position i.

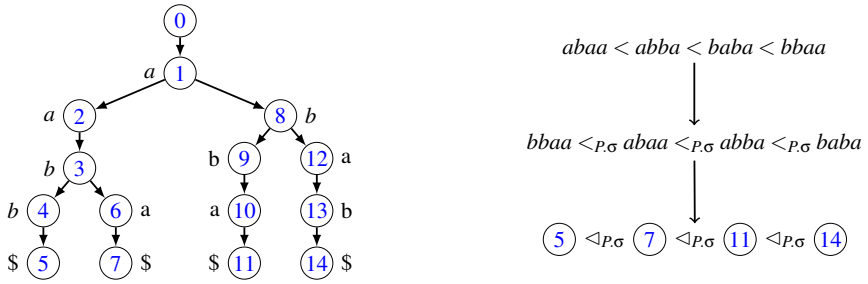

Figure 3 Tree $\texttt{ACT}(\overleftarrow{P})$ for the running example, and links between the orders $<$, $<_{P.\sigma}$, and $\lhd_{P.\sigma}$.

iff $\pi(u) <_{P.\sigma} \pi(v)$. We extend this order to the set of children of all nodes (see Figure 3). Note that $\texttt{ACT}(P)$ differs from $\texttt{ACT}(P.S)$ (defined above) to correspond to an input of the eXtended Burrows-Wheeler Transform.

3 Link between BWT and XBW of Aho-Corasick automaton

Here, we introduce a decomposition of a multi-string BWT that leads us to exhibit a bijection with the Aho-Corasick automaton (see Proposition 2). This builds on and extends Manzini's work [22]. We extend this bijection to the XBWT of the Aho-Corasick automaton (see Figure 4). For space reasons, many proofs are given in the Appendix (or the preprint [6]).

Link between BWT and AC. Given an ordered set of strings P, $\texttt{Decomp\_BWT}(P)$ is the integer interval partition of $[\![1, |m_P|]\!]$ such that

$$[\![i,j]\!] \in \texttt{Decomp\_BWT}(P) \quad \textbf{iff} \quad \begin{cases} \texttt{LCP}(P)[k] \neq \texttt{LRS}(P)[k], & \text{for } k \in \{i, j+1\} \\ \texttt{LCP}(P)[k] = \texttt{LRS}(P)[k], & \text{for } k \in [\![i+1, j]\!]. \end{cases}$$

We define Dec_Pre as the function from $\texttt{Decomp\_BWT}(P)$ to Σ^* such that

$$\texttt{Dec\_Pre}[u] := \overleftarrow{m_P[\texttt{SA}(m_P)[i] : \texttt{SA}(m_P)[i] + \texttt{LRS}(P)[i] - 1]},$$
$$\text{for all } u := [\![i,j]\!] \in \texttt{Decomp\_BWT}(P).$$

▶ **Proposition 2.** Dec_Pre *is a bijection between* $\texttt{Decomp\_BWT}(P)$ *and* $\texttt{Prefix}(\overleftarrow{P.S})$.

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

▶ **Lemma 3.** *Let* $u = [\![i,j]\!] \in \texttt{Decomp\_BWT}(P)$. *For all* $k \in [\![i,j]\!]$,

$$\texttt{Dec\_Pre}[u] \;=\; \overleftarrow{m_P[\texttt{SA}(m_P)[k] : \texttt{SA}(m_P)[k] + \texttt{LRS}(P)[k] - 1]}.$$

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

Proof. Let $u = [\![i, j]\!]$ be an interval of $\texttt{Decomp\_BWT}(P)$. First, we prove that $\texttt{Dec\_Pre}[u] \in \texttt{Prefix}(\overleftarrow{P.S})$, and then to prove the bijection, we show $\texttt{Dec\_Pre}$ is injective and surjective. By definition, $\texttt{LRS}(P)[i] = \texttt{select}_{\$}(m_P[\texttt{SA}(m_P)[i] : |m_P|], 1) - 1$, we get that $m_P[\texttt{SA}(m_P)[i] + \texttt{LRS}(P)[i]] = \$$ and for any $j \in [\![\texttt{SA}(m_P)[i], \texttt{SA}(m_P)[i] + \texttt{LRS}(P)[i] - 1]\!]$, $m_P[j] \neq \$$. Hence, $m_P[\texttt{SA}(m_P)[i] : \texttt{SA}(m_P)[i] + \texttt{LRS}(P)[i] - 1]$ is a suffix of a string w of $P.S$, and thus $\texttt{Dec\_Pre}[u]$ is a prefix of $\overleftarrow{w}$ in $\overleftarrow{P.S}$.

Injectiveness. Let $u_1 = [\![i_1, j_1]\!]$ and $u_2 = [\![i_2, j_2]\!]$ be two elements of $\texttt{Decomp\_BWT}(P)$. Without loss of generality, we take $i_1 \leq i_2$. Assume that $\texttt{Dec\_Pre}[u_1] = \texttt{Dec\_Pre}[u_2]$, we have that $m_P[\texttt{SA}(m_P)[i_1] : \texttt{SA}(m_P)[i_1] + \texttt{LRS}(P)[i_1] - 1] = m_P[\texttt{SA}(m_P)[i_2] : \texttt{SA}(m_P)[i_2] + \texttt{LRS}(P)[i_2] - 1]$ and thus for all $k \in [\![i_1, i_2]\!]$, $m_P[\texttt{SA}(m_P)[i_1] : \texttt{SA}(m_P)[i_1] + \texttt{LRS}(P)[i_1] - 1] = m_P[\texttt{SA}(m_P)[k] : \texttt{SA}(m_P)[k] + \texttt{LRS}(P)[k] - 1]$. Hence, we have $\texttt{LCP}(P)[k] = \texttt{LRS}(P)[i_1]$ and $\texttt{LRS}(P)[k] = \texttt{LRS}(P)[i_1]$. Therefore by the definition of $\texttt{Decomp\_BWT}(P)$, we get $u_1 = u_2$.

Surjectiveness. Let v be a prefix of a string of $\overleftarrow{P.S}$. The string $\overleftarrow{v}$ is a suffix of a string of $P.S$. By the definition of m_P, $\overleftarrow{v}$ is a prefix of a suffix s of m_P such that $s[|\overleftarrow{v}| + 1] = \$$. By the definition of $\texttt{BWT}(P)$, the table $\texttt{SA}(m_P)$ 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 $\texttt{Suffix}(m_P)$ and the set of positions in $\texttt{SA}(m_P)$. Let $k \in [\![1, |m_P|]\!]$ such that $s = m_P[\texttt{SA}(m_P)[k] : |m_P|]$. As $\overleftarrow{v}$ is a suffix of a string of $P.S$ and a prefix of s, we have $\overleftarrow{v} = m_P[\texttt{SA}(m_P)[k] : \texttt{SA}(m_P)[k] + \texttt{LRS}(P)[k] - 1]$. We take $u = [\![i, j]\!] \in \texttt{Decomp\_BWT}(P)$ such that $k \in [\![i, j]\!]$. By Lemma 3, $\overleftarrow{v} = \texttt{Dec\_Pre}[u]$. ◀

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

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

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

$$A_F(P) := \{(u, v) \in \texttt{Decomp\_BWT}(P)^2 |\ \Big(\max_{\substack{k<i \\ \texttt{LRS}(P)[k] = \min_{k<l\leq i}(\texttt{LCP}(P)[l])}} (k) \Big) \in v \text{ with } u := [\![i, j]\!]\}.$$

More precisely, the graph $(\texttt{Decomp\_BWT}(\overleftarrow{P}), A_T(\overleftarrow{P}))$ is isomorphic to the tree $\texttt{ACT}(P.S)$ and the graph $(\texttt{Decomp\_BWT}(\overleftarrow{P}), A_F(\overleftarrow{P}))$ is isomorphic to the tree $\texttt{ACFL}(P.S)$ (see Appendix).

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

Alike the definition of $\texttt{Decomp\_BWT}(P)$ for an ordered set of strings P, we define $\texttt{Decomp\_XBW}(\mathcal{T})$ of a tree $\mathcal{T}$ of t nodes as the integer interval partition of $[\![1, t]\!]$ such that

$$[\![i, j]\!] \in \texttt{Decomp\_XBW}(\mathcal{T}) \quad \textbf{iff} \quad \begin{cases} \texttt{XBWL}(\mathcal{T})[k] = 1, & \text{for } k \in \{i - 1, j\} \\ \texttt{XBWL}(\mathcal{T})[k] = 0, & \text{for } k \in [\![i, j - 1]\!]. \end{cases}$$

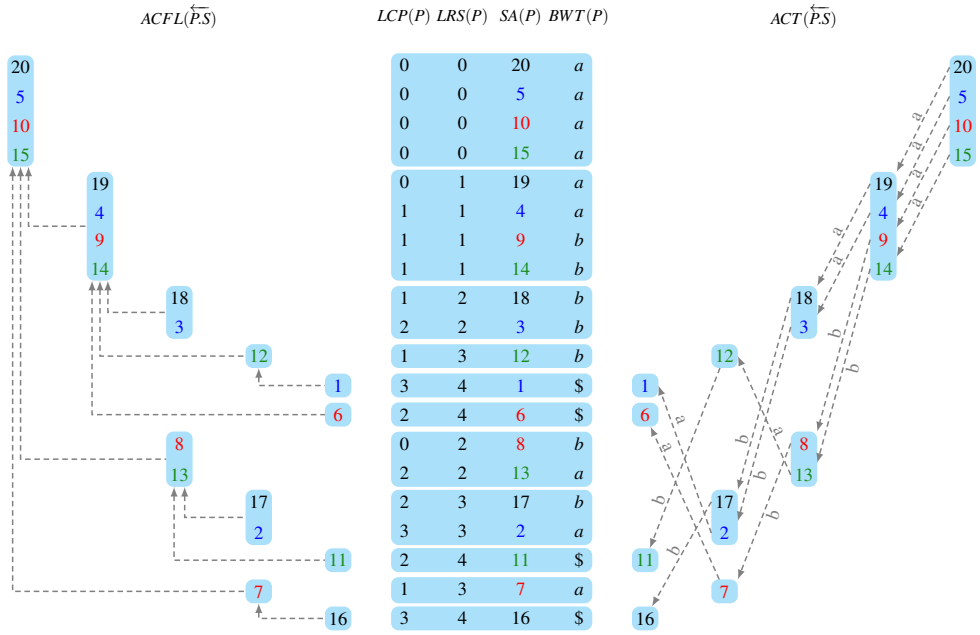

Figure 4 Link between $\texttt{BWT}(P)$, $\texttt{ACT}(\overleftarrow{P.S})$ and $\texttt{ACFL}(\overleftarrow{P.S})$ for the running example. The blocks in the tables SA, LCP, LRS and BWT show the decomposition $\texttt{Decomp\_BWT}(P)$.

▶ **Theorem 5** (See Figure 5b). *There exists a bijection, denoted* $\texttt{BWT\_XBW}$*, between* $\texttt{Decomp\_BWT}(P)$ *and* $\texttt{Decomp\_XBW}(\texttt{ACT}(\overleftarrow{P}))$ *such that for all* $u \in \texttt{Decomp\_BWT}(P)$ *with* $u := [\![i, j]\!]$, $\texttt{BWT\_XBW}(u) := [\![i', j']\!]$ *and* $z := \texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P})[i']))$*:*

(1) *Let* $\{y_1, y_2, \dots y_{\#u}\} := \texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(z)$ *such that* $y_k \lhd_{P.\sigma} y_l \Rightarrow k < l$*; then*

$$\texttt{BWT}(P)[i, j] = \delta[\texttt{Child}_{\texttt{ACT}(\overleftarrow{P})(y_1)}(z)]\delta[\texttt{Child}_{\texttt{ACT}(\overleftarrow{P})(y_2)}(z)] \dots \delta[\texttt{Child}_{\texttt{ACT}(\overleftarrow{P})(y_{\#u})}(z)].$$

(2) *Let* $\{x_1, x_2, \dots, x_{\#(\texttt{BWT\_XBW}(u))}\} := \texttt{Children}_{\texttt{ACT}(\overleftarrow{P})}(z)$ *such that* $x_k \lhd_{P.\sigma} x_l \Rightarrow k < l$*; then*

$$\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))[i', j'] = \delta[x_1]\delta[x_2] \dots \delta[x_{\#(\texttt{BWT\_XBW}(u))}].$$

Theorem 5 provides us with a strong link between $\texttt{BWT}(P)$ and $\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))$, which allows transforming one structure into the other. This leads to the following corollary.

▶ **Corollary 6.** *Let* P *be an ordered set of strings.*

(1) *Using tables* $\texttt{BWT}(P)$, $\texttt{LCP}(P)$ *and* $\texttt{LRS}(P)$ *of an ordered set of strings* P*, we can build the tables* $\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))$ *and* $\texttt{XBWL}(\texttt{ACT}(\overleftarrow{P}))$ *in linear time of* $\|P.S\| \times \#\Sigma$*.*

(2) *Using tables* $\texttt{XBWT}(\texttt{ACT}(S))$ *and* $\texttt{XBWL}(\texttt{ACT}(S))$ *of a set of strings* S*, we can build the tables* $\texttt{BWT}(\overleftarrow{P})$, $\texttt{LCP}(\overleftarrow{P})$ *and* $\texttt{LRS}(\overleftarrow{P})$ *in linear time of* $\|S\| \times \#\Sigma$ *where* P *is an ordered set of strings such that* $P.S = S$*.*

The idea behind the algorithms is to exploit the link of Theorem 5 to compute each substring of the BWT or of the XBWT associated to each element of $\texttt{Decomp\_BWT}$ or of $\texttt{Decomp\_XBW}$.

4 Optimal ordering of strings for maximising compression

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.

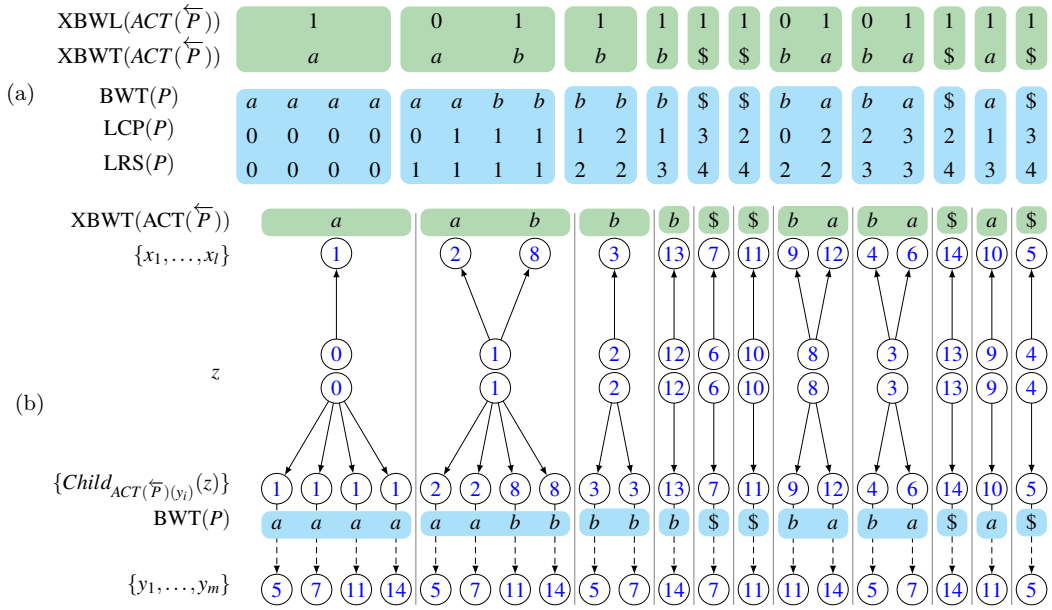

Figure 5 (a) Link between `Decomp_BWT`(P) (in blue) and `Decomp_XBW`$(\texttt{ACT}(\overleftarrow{P}))$ (in green). (b) Illustration of Theorem 5 for the running example.

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 $abbaaaccabbb = a^1b^2a^3c^2a^1b^3$ (where the power notation α^n means n copies of symbol α), 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 $\{i \in [\![1, \#\texttt{BWT}(P) - 1]\!] \mid \texttt{BWT}(P)[i] \neq \texttt{BWT}(P)[i+1]\}$. Similarly, let $d_X(P)$ be the cardinality of the set $\{i \in [\![1, \#\texttt{XBWT}(\texttt{ACT}(P)) - 1]\!] \mid \texttt{XBWT}(\texttt{ACT}(P))[i] \neq \texttt{XBWT}(\texttt{ACT}(P))[i+1]\}$.

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 $\texttt{BWT}(P)$ has size $d_B(P)+1$, finding an optimal solution of `Min-Permutation-BWT` can help compressing $\texttt{BWT}(P)$.

▶ **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 $\texttt{ACT}(\overleftarrow{P})$. We say that P is *topologically planar* if for each node u in $\texttt{ACT}(\overleftarrow{P})$ and $v \in \texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(\perp) \setminus \texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(u)$, there does not exists u_1 and u_2 in $\texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(u)$ such that $u_1 \lhd_{P.\sigma} v \lhd_{P.\sigma} u_2$. In other words, P is *topologically planar* if we can draw the tree $\texttt{ACT}(\overleftarrow{P})$ by ordering the leaves with $\lhd_{P.\sigma}$ 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 $P_{tp}.S = P.S$, and such that $P_{tp}.\sigma$ satisfies for any u in $\texttt{ACT}(\overleftarrow{P})$, any $v \in \texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(\bot) \setminus \texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(u)$, and any u_1, u_2 in $\texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(u)$ such that $u_1 \lhd_{P.\sigma} v \lhd_{P.\sigma} u_2$, we have $u_1 \lhd_{P_{tp}.\sigma} u_2 \lhd_{P_{tp}.\sigma} v$. As we have a bijection between the set of circular permutations of $P.S$ and the set of leaves of $\texttt{ACT}(\overleftarrow{P})$, 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 $d_B(P_{tp}) = d_X(\overleftarrow{P_{tp}})$.*

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 $\texttt{ACT}(\overleftarrow{P})$, $v \in \texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(\bot) \setminus \texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(u)$, and let u_1 and u_2 in $\texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(u)$ such that $u_1 \lhd_{P.\sigma} v \lhd_{P.\sigma} u_2$. Let P' be the copy of P where the only difference is $u_1 \lhd_{P_{tp}.\sigma} u_2 \lhd_{P_{tp}.\sigma} v$. Let $x \in \texttt{Decomp\_BWT}(P)$ such that $\texttt{Dec\_Pre}[x] = u$. By Theorem 5, for all $y \in \texttt{Decomp\_BWT}(P) \setminus \{x\}$, we have $\texttt{BWT}(P)[y] = \texttt{BWT}(P')[y]$, $\texttt{BWT}(P)[x] = \ldots\delta[u_1]\delta[v_1]\delta[u_2]\ldots$ and $\texttt{BWT}(P')[x] = \ldots\delta[u_1]\delta[u_2]\ldots\delta[v_1]\ldots$ with $v_1 = \texttt{Child}_{\texttt{ACT}(\overleftarrow{P})(v)}(u)$. As $\delta[u_1] = \delta[u_2]$ and $\delta[u_1] \neq \delta[v_1]$, we have $d_B(P') \leq d_B(P)$.

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

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

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

▶ **Theorem 9.** *Let S be a set of strings. We can find an optimal solution for* Min-Permutation-BWT *and for* 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 Min-Permutation-BWT is also an optimal solution of Min-Permutation-XBWT, and vice versa. In the following of this proof, we only prove the result regarding Min-Permutation-XBWT.

To start, let us give an overview of algorithm:

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

In the following, we define the problem Min-Permutation-Table and explicit its link to Min-Permutation-XBWT (Lemma 10). Lemma 10 gives us a linear time algorithm for finding an optimal solution of Min-Permutation-Table, and thus we can apply this algorithm to obtain an optimal solution for Min-Permutation-XBWT.

Given A an array of symbols of Σ, we define $\texttt{Char}(A)$ as the set of (different) symbols in A. Given T an array of n symbols of Σ and D an integer interval partition of $[\![1, n]\!]$ such for each interval $[\![i, j]\!]$ of D, $\#(\texttt{Char}(T[i, j])) = j - i + 1$ (i.e., all the symbols of

$T[i,j]$ are different), the problem `Min-Permutation-Table` is to find a T' such that for all $[\![i,j]\!] \in D$, $\#(\texttt{Char}(T[i,j])) = \#(\texttt{Char}(T'[i,j]))$ 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* $\texttt{XBWT}(\texttt{ACT}(P))$ *and for* $\texttt{Decomp\_XBW}(\texttt{ACT}(P))$*, there exists an optimal solution P' of* `Min-Permutation-XBWT` *for S such that* $\texttt{XBWT}(\texttt{ACT}(P')) = T'$.

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

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

- *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 $\{[\![i',j']\!] \in D \mid j' \leq i\}$, and T'_2 is an optimal solution of* `Min-Permutation-Table` *for $T[i,n]$ and for $\{[\![i',j']\!] \in D \mid i' \geq i\}$.*
- *If there exists i in $[\![1, \#B(T,D)]\!]$ such that $\#(B(T,D)[i]) = 0$, 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 $\{[\![i',j']\!] \in D \mid j' \leq A(D)[i][1]\}$, and T'_2 is an optimal solution of* `Min-Permutation-Table` *for $T[A(D)[i+1][0], n]$ and for $\{[\![i',j']\!] \in D \mid i' \geq A(D)[i+1][0]\}$.*
- *If there exists i in $[\![1, \#B(T,D)]\!]$ such that $\#(B(T,D)[i]) = 1$, then $T'_1aaT'_2$ 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 $T[1, A(D)[i-1][1]]\texttt{word}(\texttt{Char}(A(D)[i] \setminus \{a\})$ and for $\{[\![i',j']\!] \in D \mid j' < A(D)[i][1]\} \cup \{[\![A(D)[i][0], A(D)[i][1]-1]\!]\}$, and T'_2 is an optimal solution of* `Min-Permutation-Table` *for $\texttt{word}(\texttt{Char}(A(D)[i+1] \setminus \{a\})T[A(D)[i+2][0], n]$ and for $\{[\![i',j']\!] \in D \mid i' > A(D)[i+1][0]\} \cup \{[\![A(D)[i+1][0]+1, A(D)[i][1]]\!]\}$.*

Proof. All the proofs are derived from the equality $d_A(T[1,n], D) = d_A(T[1,i], \{[\![i',j']\!] \in D \mid j' \leq i\}) + d_A(T[i,n], \{[\![i',j']\!] \in D \mid i' \geq i\})$ for all $i \in [\![1,n]\!]$. ◀

▶ **Lemma 12.** *Let T be an array of n symbols of Σ and let D be an integer interval partition of $[\![1,n]\!]$ such that for any interval $[\![i,j]\!]$ of D, $\#(\texttt{Char}(T[i,j])) = j - i + 1$. 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 $B(T,D)[i] \setminus \{last\} \neq \emptyset$ (within the **for** loop of Algorithm 1). Let T^* be a string such that for all $[\![i,j]\!] \in D$, $\texttt{Char}(T[i,j]) = \texttt{Char}(T^*[i,j])$. 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 $[\![i,j]\!] \in D$, $\texttt{Char}(T[i,j]) = \texttt{Char}(T'[i,j])$.

Input : An instance of `Min-Permutation-Table` T and D
Output : A string T'
$last \leftarrow \$$ such that $\$ \notin \Sigma$;
$T' \leftarrow$ empty string;
for $i \in [\![1, \#B(T,D)]\!]$ **do**
  $lettre \leftarrow$ random element of $B(T,D)[i] \setminus \{last\}$;
  $T' \leftarrow T'\ \texttt{word}(\texttt{Char}(A(D)[i]) \setminus \{lettre, last\})$;
  $T' \leftarrow T'\ lettre\ lettre$;
  $last \leftarrow lettre$;
$T' \leftarrow T'\ \texttt{word}(\texttt{Char}(A(D)[\#A(D)]) \setminus \{last\})$;
return T';

▶ **Lemma 13.** *Let T be an array of n symbols of Σ and let D be an integer interval partition of $[\![1,n]\!]$ such that for each interval $[\![i,j]\!]$ of D, $\#(\texttt{Char}(T[i,j])) = j - i + 1$. 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 indidual 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

1 Alfred V. Aho and Margaret J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. *Communications of the ACM*, 18(6):333–340, 1975.

2 Jasmijn A. Baaijens, Amal Zine El Aabidine, Eric Rivals, and Alexander Schönhuth. De novo assembly of viral quasispecies using overlap graphs. *Genome Research*, 27(5):835–848, 2017.

3 Markus J. Bauer, Anthony J. Cox, and Giovanna Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theoretical Computer Science*, 483:134–148, 2013.

4 Djamal Belazzougui. Succinct Dictionary Matching with No Slowdown. In *Combinatorial Pattern Matching*, pages 88–100, 2010.

5 Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation, 1994.

6 Bastien Cazaux and Eric Rivals. Strong link between BWT and XBW via Aho-Corasick automaton and applications to Run-Length Encoding. *CoRR*, abs/1805.10070, 2018. `arXiv:1805.10070`.

7 Anthony J. Cox, Markus J. Bauer, Tobias Jakobi, and Giovanna Rosone. Large-scale compression of genomic sequence databases with the Burrows–Wheeler Transform. *Bioinformatics*, 28(11):1415–1419, 2012.

8 Lavinia Egidi and Giovanni Manzini. Lightweight BWT and LCP Merging via the Gap Algorithm. In *String Processing and Information Retrieval*, pages 176–190, 2017.

9 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *Journal of the ACM*, 57(1):1–33, 2009.

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

11 Luca Foschini, Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. When indexing equals compression: Experiments with compressing suffix arrays and applications. *ACM Transactions on Algorithms*, 2(4):611–639, 2006.

12 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theoretical Computer Science*, 698:67–78, 2017.

13 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *ACM-SIAM Symp. on Discrete Algorithms*, pages 841–850, 2003.

14 James Holt and Leonard McMillan. Constructing Burrows-Wheeler Transforms of large string collections via merging. In *ACM Conf. on Bioinformatics, Computational Biology, and Health Informatics*, pages 464–471, 2014.

15 James Holt and Leonard McMillan. Merging of multi-string BWTs with applications. *Bioinformatics*, 30(24):3524–3531, 2014.

16 Wing-Kai Hon, Tsung-Han Ku, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Faster compressed dictionary matching. *Theoretical Computer Science*, 475:113–119, 2013.

17 Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

18 Veli Mäkinen and Gonzalo Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387(3):332–347, 2007.

19 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and Retrieval of Individual Genomes. In *RECOMB, Tucson, AZ, USA*, pages 121–137, 2009.

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

21 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. An Extension of the Burrows Wheeler Transform and Applications to Sequence Comparison and Data Compression. In *Combinatorial Pattern Matching*, pages 178–189, 2005.

22 Giovanni Manzini. XBWT tricks. In *String Processing and Information Retrieval*, pages 80–92, 2016.

23 Jouni Sirén. Burrows-Wheeler Transform for Terabases. In *Data Compression Conf.*, pages 211–220, 2016.

24 Jouni Sirén, Niko Välimäki, Veli Mäkinen, and Gonzalo Navarro. Run-Length Compressed Indexes Are Superior for Highly Repetitive Sequence Collections. In *String Processing and Information Retrieval*, pages 164–175, 2009.

A Appendix

A.1 Linear time construction of tables LRS(P) and LCP(P)

To build in linear time the tables $\texttt{LRS}(P)$ and $\texttt{LCP}(P)$, 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.

▶ **Lemma 14.** *Let $i \in [\![1, |m_P|]\!]$.*

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

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

- if $\texttt{SA}(w)[i] \geq 2$,

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

- If $\texttt{SA}(w)[i] = 1$,

$$\begin{aligned} \texttt{LRS}(P)[\texttt{LF}(P)[i]] &= \texttt{select}_{\$}(m_P[\texttt{SA}(m_P)[\texttt{LF}(P)[i]] : |m_P|], 1) - 1 \\ &= \texttt{select}_{\$}(m_P[|m_P| : |m_P|], 1) - 1 \\ &= 0. \end{aligned}$$

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

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

Let i be an integer between 1 and $|m_P|$. We have

$$\begin{aligned} \texttt{RLF}(P)[\texttt{LF}(P)[i]] &= \texttt{select}_{\texttt{Letter}[\texttt{LF}(P)[i]]}(\texttt{BWT}(P), \texttt{LF}(P)[i] - C[\texttt{Letter}[\texttt{LF}(P)[i]]]) \\ &= \texttt{select}_{\texttt{BWT}(P)[i]}(\texttt{BWT}(P), \texttt{LF}(P)[i] - C[\texttt{BWT}(P)[i]]) \\ &= \texttt{select}_{\texttt{BWT}(P)[i]}(\texttt{BWT}(P), \texttt{rank}_{\texttt{BWT}(w)[i]}(\texttt{BWT}(w), i)) \\ &= i. \end{aligned}$$

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

$$\begin{aligned} \texttt{LRS}(P)[i] = 0 &\Leftrightarrow \texttt{BWT}(P)[\texttt{RLF}(P)[i]] = \$ \\ &\Leftrightarrow \texttt{BWT}(P)[\texttt{select}_{\texttt{Letter}[i]}(\texttt{BWT}(P), i - C[\texttt{Letter}[i]])] = \$ \\ &\Leftrightarrow \texttt{Letter}[i] = \$ \\ &\Leftrightarrow i \in [\![1, \#(P.S)]\!]. \end{aligned}$$

Therefore, we derive the desired equality, which concludes the proof. ◀

▶ **Proposition 15.** *Let P be an ordered set of strings. Using tables* $\texttt{BWT}(P)$ *and* $\texttt{LCP}(m_P)$*, we can compute the tables* $\texttt{LRS}(P)$ *and* $\texttt{LCP}(P)$ *in a time that is linear in* $|m_P|$*.*

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

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

Algorithm 2: Computation of table $\texttt{LRS}(P)$.

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

◀

A.2 Proof of Theorem 4

The next propositions exhibit the links between $\texttt{Decomp\_BWT}(P)$ and first, the arcs of $\texttt{ACT}(\overleftarrow{P.S})$ (Proposition 16), and second, the arcs of $\texttt{ACFL}(\overleftarrow{P.S})$ (Proposition 17).

▶ **Proposition 16** (See Figure 4)**.** *The graph* $G_T(P) = (\texttt{Decomp\_BWT}(P), A_T(P))$ *is isomorphic to the tree* $\texttt{ACT}(\overleftarrow{P.S})$*, where*

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

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

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

$$\begin{aligned}
\texttt{Dec_Pre}[v] &= \overleftarrow{m_P[\texttt{SA}(m_P)[\texttt{LF}(P)[x]] : \texttt{SA}(m_P)[\texttt{LF}(P)[x]] + \texttt{LRS}(P)[\texttt{LF}(P)[x]] - 1]} \\
&= \overleftarrow{m_P[\texttt{SA}(m_P)[x] - 1 : \texttt{SA}(m_P)[x] - 1 + \texttt{LRS}(P)[\texttt{LF}(P)[x]] - 1]} \\
&= \overleftarrow{m_P[\texttt{SA}(m_P)[x] - 1 : \texttt{SA}(m_P)[x] - 1 + \texttt{LRS}(P)[x]]} \\
&= \overleftarrow{m_P[\texttt{SA}(m_P)[x] : \texttt{SA}(m_P)[x] + \texttt{LRS}(P)[x] - 1]}\ m_P[\texttt{SA}(m_P)[x] - 1] \\
&= \texttt{Dec_Pre}[u]\ m_P[\texttt{SA}(m_P)[x] - 1].
\end{aligned}$$

The string $\texttt{Dec\_Pre}[u]$ is thus the longest prefix of $\texttt{Dec\_Pre}[v]$.

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

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

$$A_F(P) := \{(u, v) \in \texttt{Decomp\_BWT}(P)^2 \mid \big(\max_{\substack{k<i \\ \texttt{LRS}(P)[k]=\min_{k-1\le l\le i}(\texttt{LCP}(P)[l])}} (k)\big) \in v \text{ with } u = [\![i, j]\!]\}.$$

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

$$w \in \texttt{Decomp\_BWT}(P) \text{ such that } \exists k \in w \text{ with } k < i \text{ and } \texttt{LRS}(P)[k] = \min_{k-1\le l\le i}(\texttt{LCP}(P)[l])$$
$$\Leftrightarrow$$
$$\texttt{Dec\_Pre}[w] \text{ is a suffix of } \texttt{Dec\_Pre}[u].$$

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

$$\begin{aligned}
\texttt{Dec_Pre}[u] &= \overleftarrow{m_P[\texttt{SA}(m_P)[i] : \texttt{SA}(m_P)[i] + \texttt{LRS}(P)[i] - 1]} \\
&= \overleftarrow{m_P[\texttt{SA}(m_P)[i] + \texttt{LRS}(P)[k] : \texttt{SA}(m_P)[i] + \texttt{LRS}(P)[i] - 1]} \\
&\quad \overleftarrow{m_P[\texttt{SA}(m_P)[i] : \texttt{SA}(m_P)[i] + \texttt{LRS}(P)[k] - 1]} \\
&= \overleftarrow{m_P[\texttt{SA}(m_P)[i] + \texttt{LRS}(P)[k] : \texttt{SA}(m_P)[i] + \texttt{LRS}(P)[i] - 1]} \\
&\quad \overleftarrow{m_P[\texttt{SA}(m_P)[k] : \texttt{SA}(m_P)[k] + \texttt{LRS}(P)[k] - 1]} \\
&= \overleftarrow{m_P[\texttt{SA}(m_P)[i] + \texttt{LRS}(P)[k] : \texttt{SA}(m_P)[i] + \texttt{LRS}(P)[i] - 1]}\ \texttt{Dec_Pre}[w].
\end{aligned}$$

Let $w = [\![i_1, j_1]\!]$ and $u = [\![i_2, j_2]\!]$ be two elements of $\texttt{Decomp\_BWT}(P)$ such that $\texttt{Dec\_Pre}[w]$ is a suffix of $\texttt{Dec\_Pre}[u]$. Hence, we have that $m_P[\texttt{SA}(m_P)[i_1] : \texttt{SA}(m_P)[i_1] + \texttt{LRS}(P)[i_1] - 1]$ is a prefix of $m_P[\texttt{SA}(m_P)[i_2] : \texttt{SA}(m_P)[i_2] + \texttt{LRS}(P)[i_2] - 1]$. By the definition of $\texttt{BWT}(P)$, for all $l \in [\![i_1, i_2]\!]$, $\texttt{LRS}(P)[i_1] \le \texttt{LCP}(P)[l]$. This concludes the proof of the equivalence. By the equivalence, given u and v in $\texttt{Decomp\_BWT}(P)$ such that $\texttt{Dec\_Pre}[u]$ is a suffix of $\texttt{Dec\_Pre}[v]$, for all $k_1 \in u$ and $k_2 \in v$, we have $k_1 \le k_2$. 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 $\texttt{Decomp\_BWT}(P)$ (resp. $\texttt{Decomp\_XBW}(\texttt{ACT}(\overleftarrow{P}))$) 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 $\overleftarrow{P}$. By Proposition 2, the length of T_B is $\#\texttt{Prefix}(\overleftarrow{P.S})$. By the definition of $\texttt{Decomp\_XBW}$, the length of T_X is the number of 1 in $\texttt{XBWL}(\texttt{ACT}(\overleftarrow{P}))$, i.e., the number of internal nodes of $\texttt{ACT}(\overleftarrow{P})$, and thus is equal to $\#\texttt{Prefix}(\overleftarrow{P.S})$. Hence, T_B and T_X have the same cardinalities.

Let $i \in [\![1, \#\texttt{Prefix}(\overleftarrow{P.S})]\!]$. By Proposition 2, $T_B[i]$ represents the i^{th} suffix of strings of $P.S$ in lexicographic order. By the definition of $\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))$, all the nodes $\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[k]$ for $k \in T_X[i]$ have the same parent z in $\texttt{ACT}(\overleftarrow{P})$, and z represents the i^{th} node in $\prec$ order, i.e., the i^{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 $\overleftarrow{P}$, we define the bijection $\texttt{BWT\_XBW}$ such that for any i in $[\![1, \#\texttt{Prefix}(\overleftarrow{P.S})]\!]$, one has $\texttt{BWT\_XBW}[T_B[i]] = T_X[i]$. As the tree $\texttt{ACT}(\overleftarrow{P})$ represents the Aho-Corasick tree of $\overleftarrow{P.S}^{\$}$, we have a bijection b_1 from the node set of $\texttt{ACT}(\overleftarrow{P})$ onto the set of prefixes of $\overleftarrow{P.S}^{\$}$. By the definition of functions π and δ (see page 4), for any node v of $\texttt{ACT}(\overleftarrow{P})$, we have $b_1(v) = \overleftarrow{\pi(v)}\ \delta(v)$.

By the definition of $\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))$, we have that for $T_X[i] = [\![i', j']\!]$

$$\begin{aligned} \#T_X[i] &= \#\{v \text{ node of } \texttt{ACT}(\overleftarrow{P}) \mid \pi(v) = \overleftarrow{\texttt{Dec\_Pre}[\texttt{BWT\_XBW}^{-1}[T_X[i]]]}\} \\ &= \#\{v \text{ node of } \texttt{ACT}(\overleftarrow{P}) \mid v \text{ is a child of } b_1^{-1}(\texttt{Dec\_Pre}[\texttt{BWT\_XBW}^{-1}[T_X[i]]])\}. \end{aligned}$$

As $i' \in T_X[i]$ and $b_1^{-1}(\texttt{Dec\_Pre}[\texttt{BWT\_XBW}^{-1}[T_X[i]]]) = \texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[i'])$,

$$\begin{aligned} \#T_X[i] &= \#\{v \text{ node of } \texttt{ACT}(\overleftarrow{P}) \mid v \text{ is a child of } \texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[i'])\} \\ &= \#\texttt{Children}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[i'])). \end{aligned}$$

Let $\{x_1, x_2, \ldots, x_{\#T_X[i]}\}$ be the set of children of $\texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[i'])$ sorted such that $x_1 \lhd_{P.\sigma} \ldots \lhd_{P.\sigma} x_{\#T_X[i]}$. By the definition of $\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))$, for any $k \in [\![1, \#T_X[i]]\!]$, we have $\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))[i' + k - 1] = \delta[x_k]$. By the definition of $\texttt{BWT}(P)$, for $T_B[i] = [\![i'', j'']\!]$, we get

$$\begin{aligned} \#T_B[i] &= \#\{w \in \overleftarrow{P.S}^{\$} \mid \texttt{Dec\_Pre}[T_B[i]] \text{ is a prefix of } w\} \\ &= \#\{v \text{ leaf of } \texttt{ACT}(\overleftarrow{P}) \mid b_1^{-1}(\texttt{Dec\_Pre}[T_B[i]]) \text{ is an ancestor of } v \text{ in } \texttt{ACT}(\overleftarrow{P})\}. \end{aligned}$$

As $b_1^{-1}(\texttt{Dec\_Pre}[T_B[i]]) = b_1^{-1}(\texttt{Dec\_Pre}[\texttt{BWT\_XBW}^{-1}[T_X[i]]]) = \texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[i'])$,

$$\begin{aligned} \#T_B[i] &= \#\{v \text{ a leaf of } \texttt{ACT}(\overleftarrow{P}) \mid \texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[i']) \\ &\qquad \text{is an ancestor of } v \text{ in } \texttt{ACT}(\overleftarrow{P})\} \\ &= \#\texttt{Leaves}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[i'])). \end{aligned}$$

Let $\{y_1, y_2, \ldots, y_{\#T_B[i]}\}$ be the set of leaves of the subtree of $\texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[i'])$ in $\texttt{ACT}(\overleftarrow{P})$ sorted such that $y_1 \lhd_{P.\sigma} \ldots \lhd_{P.\sigma} y_{\#T_B[i]}$. Given $k \in [\![i'', j'']\!]$, we define $w_P[k]$ as the string $m_P[\texttt{select}_{\$}(m_P, \texttt{rank}_{\$}(m_P, \texttt{SA}(P)[k] - 1)) + 1 : \texttt{SA}(P)[k] + \texttt{LRS}(P)[k] - 1]$. 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 $\pi(y_l) = w_P[i'' + l - 1]$. Given x and y in $[\![i'', j'']\!] \in \texttt{Decomp\_BWT}(P)$, we obtain the following equivalences between orders:

$$x < y \Leftrightarrow w_P[x] <_{P.\sigma} w_P[y] \Leftrightarrow \pi(y_{x-i''+1}) <_{P.\sigma} \pi(y_{y-i''+1}) \Leftrightarrow y_{x-i''+1} \lhd_{P.\sigma} y_{y-i''+1}.$$

By the definition of $\texttt{BWT}(P)$, for all $k \in [\![1, \#T_B[i]]\!]$, we get that

$$\begin{aligned} \texttt{BWT}(P)[i'' + k - 1] &= m_P[\texttt{SA}(m_P)[i'' + k - 1] - 1] \\ &= w_P[i'' + k - 1][|w_P[i'' + k - 1]| - \texttt{LRS}[i'' + k - 1] + 1] \\ &= \pi(y_k)[|w_P[i'' + k - 1]| - \texttt{LRS}[i'' + k - 1] + 1] \\ &= \delta[\texttt{Child}_{\texttt{ACT}(\overleftarrow{P})(y_k)}(\texttt{Parent}_{\texttt{ACT}(\overleftarrow{P})}(\texttt{PA}(\texttt{ACT}(\overleftarrow{P}))[i']))]. \end{aligned}$$

◀

A.4 Proof of Corollary 6

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

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

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

Proof of Lemma 18. For each i in $[\![1, |m_P|]\!]$, at the begining of the loop **for**, we have that $\texttt{LCP}(P)[i-j] \neq \texttt{LRS}(P)[i-j]$ and for all $k \in [\![i-j+1, i-1]\!]$, $\texttt{LCP}(P)[k] \neq \texttt{LRS}(P)[k]$. Hence, if $\texttt{LCP}(P)[i] \neq \texttt{LRS}(P)[i]$, the interval $[\![i-j, i-1]\!]$ is an element of $\texttt{Decomp\_BWT}(P)$ 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 $\texttt{Decomp\_BWT}(P)$. For the complexity, as each step of the loop can be computed in constant time, Algorithm 3 computes $\texttt{BWD}(P)$ in linear time in $\|P.S\|$. As for each position i of $\texttt{BWD}(P)$, $\texttt{BWD}(P)[i]$ represents the number of strings of $P.S$ having as suffix $\texttt{Dec\_Pre}[u]$, where u is the i^{th} element of $\texttt{Decomp\_BWT}(P)$ sorted in interval order, it follows that $\texttt{BWD}(P)[i] \leq \#(P.S)$. This concludes the proof. ◀

Algorithm 3: Computation of table $\texttt{BWD}(P)$.

Input : The tables $\texttt{LCP}(P)$ and $\texttt{LRS}(P)$
Output : The table $\texttt{BWD}(P)$
$\texttt{BWD} \leftarrow$ empty list;
$j \leftarrow 0$;
for $i \in [\![1, |m_P|]\!]$ **do**
  if $\texttt{LCP}(P)[i] \neq \texttt{LRS}(P)[i]$ **then**
    insert j at the end of BWD;
    $j \leftarrow 1$;
  else $j \leftarrow j + 1$;
return BWD;

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

Proof of Lemma 19. Algorithm 4 is an application of Theorem 5. ◀

Using Algorithm 3 to build $\texttt{BWD}(P)$ and Algorithm 4, we can compute tables $\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))$ and $\texttt{XBWL}(\texttt{ACT}(\overleftarrow{P}))$ in linear time of $\|P.S\| \times \#\Sigma$.
We define the equivalent of BWD for $\texttt{Decomp\_XBW}(P)$. The *eXtended Burrows Wheeler Decomposition* (XBWD) of a tree $\mathcal{T}$ is the array of length $\#\texttt{XBWL}(\mathcal{T})$ such that for any position i, $\texttt{XBWD}(P)[i]$ equals the cardinality of the i^{th} element of $\texttt{Decomp\_XBW}(\mathcal{T})$ sorted in interval order.

▶ **Lemma 20.** *From the table* $\texttt{XBWL}(\texttt{ACT}(S))$*, Algorithm 5 computes* $\texttt{XBWD}(\texttt{ACT}(S))$ *in linear time in* $\|P.S\|$ *and the table* $\texttt{XBWD}(\texttt{ACT}(S))$ *can be stored in* $\|P.S\| \times log(\#\Sigma)$ *bits.*

Algorithm 4: Computation of tables $\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))$ and $\texttt{XBWL}(\texttt{ACT}(\overleftarrow{P}))$.

Input : The tables $\texttt{BWT}(P)$ and $\texttt{BWD}(P)$
Output : The tables $\texttt{XBWT}(\texttt{ACT}(\overleftarrow{P}))$ and $\texttt{XBWL}(\texttt{ACT}(\overleftarrow{P}))$
Initialise both XBWT and XBWL as empty lists;
$nb \leftarrow 1$;
for $i \in [\![1, \#\texttt{BWD}(P)]\!]$ **do**
  $D \leftarrow$ Dictionnary such that for all $l \in \Sigma$, $D[l] \leftarrow true$;
  $begin \leftarrow nb$;
  $last \leftarrow begin + \texttt{BWD}(P)[i] - 1$;
  $nb \leftarrow last + 1$;
  for $j \in [\![begin, last]\!]$ **do**
    if $D[\texttt{BWT}(P)[j]]$ **then**
      insert $\texttt{BWT}(P)[j]$ at the end of XBWT ;
      insert 0 at the end of XBWL ;
      $D[\texttt{BWT}(P)[j]] \leftarrow false$;
  $\texttt{XBWL}[\#\texttt{XBWL}] \leftarrow 1$;
return XBWT and XBWL;

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

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

Input : The table $\texttt{XBWL}(\texttt{ACT}(S))$; **Output**: The table $\texttt{XBWD}(\texttt{ACT}(S))$
$\texttt{XBWD} \leftarrow$ empty list;
$j \leftarrow 1$;
for $i \in [\![1, \#(\texttt{XBWL}(\texttt{ACT}(S)))]\!]$ **do**
  if $\texttt{XBWL}(\texttt{ACT}(S)) = 1$ **then**
    insert j at the end of XBWD ;
    $j \leftarrow 1$;
  else $j \leftarrow j + 1$;
return XBWD;

▶ **Lemma 21.** *Using tables* $\texttt{XBWT}(\texttt{ACT}(S))$ *and* $\texttt{XBWD}(\texttt{ACT}(S))$*, we can build the tables* $\texttt{BWT}(\overleftarrow{P})$*,* $\texttt{LCP}(\overleftarrow{P})$ *and* $\texttt{LRS}(\overleftarrow{P})$ *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 $\texttt{XBWT}(\texttt{ACT}(S))$ and $\texttt{XBWD}(\texttt{ACT}(S))$ one can access in constant time the children and the parents in $\texttt{ACT}(S)$. Hence, we can compute in linear time in $\|P.S\|$, the table $\texttt{TL}(\texttt{ACT}(S))$, where in each position of i we store the number of leaves in the subtree of the node $\texttt{PA}(\texttt{ACT}(S))[i]$. We finish the proof using the results of Theorem 5 and an algoritm similar to Algorithm 4. ◀

A.5 Proof of Lemma 10

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