Extending the Parameterized Burrows–Wheeler Transform

Eric M. Osterkamp[†] and Dominik Köppl[*]
[†]University of Münster, Münster, Germany, osterkamp@uni-muenster.de
[*]University of Yamanashi, Kōfu, Japan, dkppl@yamanashi.ac.jp

Abstract

The Burrows–Wheeler transform (BWT) provides a succinct way to index text for pattern matching queries. Notable variants are (a) the extended BWT (eBWT) capable to index multiple input texts for circular pattern matching, or (b) the parameterized BWT (pBWT) for parameterized pattern matching. A natural extension is the combination of the virtues of both variants into a new data structure, whose name we coin with *extended parameterized BWT* (epBWT). We show that the epBWT supports circular pattern matching in context of parameterized pattern matching on multiple texts, within the same complexities as known solutions presented for the pBWT [Kim and Cho, IPL'21] for patterns shorter than the shortest indexed text.

Keywords: extended BWT, parameterized pattern matching, circular pattern matching

1 Introduction

The seek for compact indexing data structures stems from the need to manage and analyze the ever-increasing amount of available data. A mainstream line of research is devoted to fast yet space-economical variations of the FM-index, a full-text index built upon the Burrows–Wheeler transform (BWT). For a given pattern, the FM-index allows us to query for the number of its occurrences in the indexed text, called *count* query, or the starting positions of these occurrences, called *locate* query. For the former, it suffices to encode the BWT with a rank/select data structure. For the latter, we add suffix array samples to map found positions in the BWT into text-range. The few number of tools make the FM-index a predestined data structure for compact text indexing. However, the standard variant only supports exact pattern matching in the classic sense. A common variation in bioinformatics and computational geometry is to search for circular patterns. For instance, the genetic data of some viruses can be modelled as circular strings. With the aim for an index for circular pattern matching, Mantaci et al. [7] proposed the extended BWT (eBWT), which can additionally index multiple circular strings. On the other hand, structural analysis of RNA data involves a different kind of pattern matching, coined as *parameterized pattern*

matching by Baker [1], and introduced to the context of RNA data by Shibuya [9]. Again, BWT-based indexes such as the *parameterized BWT* (pBWT) by Ganguly et al. [2] emerge as compact indexes for parameterized pattern matching (see Mendivelso [8] for an overview of other parameterized pattern matching indexes). Recently, Iseri et al. [5] showed how to build the pBWT efficiently.

In the light that bioinformatic problems ask for RNA indexes that can cope with circular as well as parameterized pattern matching, a natural question to ask is whether an extension of the FM-index exists that allows a combination of both types of queries. We here answer this question affirmative by presenting the *extended parameterized BWT* (epBWT), which is based on the ideas of Mantaci et al. [7] for the eBWT and the tricks used by Iseri et al. [5] for the pBWT.

2 Preliminaries

Let $\mathbb{N}$ denote the natural numbers starting with 1, and let Σ denote an *alphabet*. An element of Σ is called a *symbol*, a sequence of symbols from Σ a *string over* Σ and a subsequence U of a consecutive run of elements from a string V over Σ a *substring* of V. A substring U of V is *proper* if $U \neq V$. Let Σ^* denote the set of all finite sequences of symbols from Σ, Σ^ω the set of all countably infinite sequences of symbols from Σ and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Then an element of Σ^* is called a *finite* string over Σ and an element of Σ^ω an *infinite* string over Σ.

For $U \in \Sigma^*$, we denote by U^ω the infinite string obtained by infinitely iterating U, i.e. $U^\omega = UUU\cdots \in \Sigma^\omega$. We call a string $V \in \Sigma^*$ *primitive* if $V = U^k$ for $U \in \Sigma^*$ and $k \in \mathbb{N}$ already implies $V = U$ and $k = 1$. For every $V \in \Sigma^*$ there exists a unique primitive $U \in \Sigma^*$ and a unique $k \in \mathbb{N}$ such that $V = U^k$, denoted by $\mathrm{root}(V)$ and $\exp(V)$, respectively. For $U \in \Sigma^\infty$, let $|U|$ denote the *length* of U. Let ε denote the unique string of length 0 and denote the length of the longest common prefix of $U, V \in \Sigma^\infty$ by $\mathrm{lcp}(U, V)$.

Fix some $V \in \Sigma^\infty$ and $1 \leq i, j \leq |V|$ with $i, j \in \mathbb{N}$. A substring $V[i]\cdots V[j]$ starting at position i and ending at position j of V is denoted by $V[i..j]$, where we set $V[i..j] = \varepsilon$ if $j < i$. Let $V[..i]$ denote the prefix $V[1..i]$ of V and $V[i..]$ the suffix W of V such that $V = V[..i-1]W$. Assume $V \in \Sigma^*$. Then $\mathrm{Rot}(V, 0) = V$ and $\mathrm{Rot}(V, k+1) = \mathrm{Rot}(V, k)[2..]\mathrm{Rot}(V, k)[1]$ denote the kth *left rotation* of V for $k \in \mathbb{N}$.

Let $<$ denote the alphabetical order on Σ. For $U, W \in \Sigma^*$, we write $U < W$ if and only if U is a proper prefix of W or $U[\mathrm{lcp}(U, W) + 1] < W[\mathrm{lcp}(U, W) + 1]$. For $U, W \in \Sigma^\omega$, we write $U < W$ if and only if there exists $q \in \mathbb{N}$ such that $U[..q-1] = W[..q-1]$ and $U[q] < W[q]$.

Let $c \in \Sigma$ and $1 \leq i \leq j \leq |V|$. The query $\mathrm{rank}_c(V, i)$ returns the number of occurrences of c in $V[..i]$; the query $\mathrm{select}_c(V, i)$ returns the position of the ith occurrence of c in V, for $1 \leq i \leq \mathrm{rank}_c(V, |V|)$; the query $\mathrm{rangecount}_V(a, b, x, y)$ returns the number of entries in $V[a..b]$ having a value in $[x..y]$; the *range maximum query* $\mathrm{RMQ}_V(i, j)$ returns the position of a maximal element in $V[i..j]$; the *find previous query* FPQ_c returns the largest position $k \leq i$ at which $V[k] = c$; the *find next query* FNQ_c returns the smallest position $\ell \geq i$ at which $V[\ell] = c$ — such positions might not exist, and must be treated as border cases

$i =$	1	2	3	4	5	6	7	8
$V[i] =$	7	14	5	1	11	27	11	7

$\mathrm{rank}_{11}(V, 5) = 1$, $\mathrm{select}_{11}(V, 2) = 7$,
$\mathrm{rangecount}_V(2, 7, 7, 14) = 3$, $\mathrm{RMQ}_V(2, 5) = 2$,
$\mathrm{FNQ}_{11}(V, 5) = 5$, $\mathrm{FNQ}_7(V, 2) = 8$, $\mathrm{FPQ}_7(V, 2) = 1$.

Figure 1: Example integer string V to illustrate the queries defined in Sect. 2.

(which we here omit for the sake of space and simplicity). See Fig. 1 for some examples.

2.1 Parameterized Burrows–Wheeler Transform

Let ∞ represent a symbol larger than any integer and let Σ_s and Σ_p denote two disjoint alphabets such that $(\{\infty\} \cup \mathbb{N}) \cap \Sigma_\mathrm{s} = \emptyset$ and $c < i$ for any $c \in \Sigma_\mathrm{s}$ and $i \in \mathbb{N}$. A symbol from Σ_s is called a *static symbol (s-symbol)* and a symbol from Σ_p a *parameterized symbol (p-symbol)*. Let $\sigma = |\Sigma_\mathrm{s} \cup \Sigma_\mathrm{p}|$ and $\sigma_\mathrm{p} = |\Sigma_\mathrm{p}|$. We call a string over $\Sigma_\mathrm{s} \cup \Sigma_\mathrm{p}$ a *parameterized string (p-string)*. Any examples of p-strings use $\Sigma_\mathrm{s} = \{\mathtt{a}, \mathtt{b}\}$, $\Sigma_\mathrm{p} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$ and a selection of additional s-symbols, which will be introduced in due course. Two p-strings U and V of equal length are said to *parameterized match (p-match)* if there exists a bijection $\psi : \Sigma_\mathrm{p} \to \Sigma_\mathrm{p}$ such that $U[i] = V[i]$ if $V[i] \in \Sigma_\mathrm{s}$ and $U[i] = \psi(V[i])$ otherwise, for each $1 \le i \le |U|$. We write $U \approx V$ if and only if we have a p-match of U and V. The relation $\approx$ is an equivalence relation on $(\Sigma_\mathrm{s} \cup \Sigma_\mathrm{p})^*$. For example, $\mathtt{CABaAC} \approx \mathtt{BCAaCB}$, where $\psi(\mathtt{A}) = \mathtt{B}$, $\psi(\mathtt{B}) = \mathtt{C}$ and $\psi(\mathtt{C}) = \mathtt{A}$.

p-matching. The *indexing problem for p-matching* is to reversibly preprocess a text p-string T so that we can efficiently count and locate the occurrences of substrings of T that p-match with a given pattern p-string. In what follows, we review the pBWT as an index for p-matching, and subsequently show a straightforward adaptation to circular matching. To this end, we introduce two convenient p-string encoding techniques.

First, Kim and Cho's [6] *prev-encoding* $\langle V \rangle$ of a p-string V is a string of length $|V|$ over $\Sigma_\mathrm{s} \cup [1..|V|-1] \cup \{\infty\}$ such that

$$\langle V \rangle[i] = \begin{cases} V[i] & \text{if } V[i] \in \Sigma_\mathrm{s}, \\ \infty & \text{if } V[i] \in \Sigma_\mathrm{p} \wedge V[i] \neq V[j] \text{ for all } 1 \le j < i, \text{ and} \\ i - \mathrm{FPQ}_{V[i]}(V, i-1) & \text{otherwise,} \end{cases}$$

for every $1 \le i \le |V|$, i.e. the leftmost occurrence of a p-symbol in V is replaced by ∞ and each successive occurrence with the distance to its previous occurrence.

Second, for any p-string W, let $|W|_\mathrm{p} = \mathrm{rank}_\infty(\langle W \rangle, |W|)$ denote the number of distinct p-symbols in W. The encoding $[\![V]\!]$ of a p-string V proposed by Hashimoto et al. [3] is given by a string of length $|V|$ over $\Sigma_\mathrm{s} \cup [1..\sigma_\mathrm{p}]$ such that $[\![V]\!][i] = V[i]$ if $V[i] \in \Sigma_\mathrm{s}$, or $\big|\mathrm{Rot}(V, i)\,[..\mathrm{select}_{V[i]}(\mathrm{Rot}(V, i), 1)]\big|_\mathrm{p}$ otherwise, for each $1 \le i \le |V|$. In other words, we view V circularly and replace each occurrence of a p-symbol in V by the number of distinct p-symbols until its next occurrence. The convenience of this encoding technique stems from the fact that it, unlike the prev-encoding, commutes with the rotation operator, i.e., $[\![\mathrm{Rot}(S, i)]\!] = \mathrm{Rot}([\![S]\!], i)$ for all p-strings S and indices i, which eases an adaptation of Burrows–Wheeler techniques requiring the sorting of rotations. We also leverage the following lemma.

Lemma 2.1. *Let U, W denote p-strings. Then $\langle U\rangle = \langle W\rangle \Leftrightarrow U \approx W \Leftrightarrow [\![U]\!] = [\![W]\!]$.*

Proof. The first equivalence is a corollary to Proposition 1 by Baker [1] and the second equivalence is Proposition 1 by Hashimoto et al. [3]. □

Parameterized Rotation Array. Let \$ denote a static symbol smaller than any other static symbol and let S denote a p-string of length n that has exactly one occurrence of \$ and satisfies $S[n] = \$$. The *parameterized rotation array* RA_S of S is an array of size n such that $\mathrm{RA}_S[i] = j$ with $1 \leq j \leq n$ if and only if $\langle \mathrm{Rot}(S, j-1)\rangle$ is the ith lexicographically smallest string in $\{\langle \mathrm{Rot}(S, k-1)\rangle \mid 1 \leq k \leq n\}$. We say that RA_S maps from *lex-range* to text-range, meaning that it assigns a lex(icographic) rank a text position.[1] Since \$ occurs exactly once in S, no two prev-encoded conjugates of S are equal and consequently RA_S represents a well-defined permutation.

pBWT. Following the definition of Kim and Cho's [6], the pBWT of the p-string S consists of two strings F'_S and L'_S, both of length n over the alphabet $\Sigma_\mathrm{s} \cup [1..\sigma_\mathrm{p}]$. Both strings are defined by $\mathrm{F}'_S[i] = [\![\mathrm{Rot}(S, \mathrm{RA}_S[i]-1)]\!][1]$ and $\mathrm{L}'_S[i] = [\![\mathrm{Rot}(S, \mathrm{RA}_S[i]-1)]\!][n]$, for each $1 \leq i \leq n$. We can restore the original input from the pBWT by using a technique called *backward search* on L'_S and F'_S, which we assume the reader is familiar with as we defer the exact definition for the pBWT due to the lack of space. The backward search also allows us to perform count queries for a pattern in that it gives us a range in L'_S whose length is the number of occurrences of the pattern.

c-matching. In what follows, we study a generalization of the indexing problem for p-matching, which we call the c-matching problem. The *c-matching problem* is to count or locate all conjugates of text p-strings $T_1, ..., T_d$ that have a prefix p-matching a pattern p-string P. The *indexing problem for c-matching to* $\mathcal{T}$ is to reversibly preprocess a set $\mathcal{T}$ of non-empty text p-strings $T_1, ..., T_d$ so that we can efficiently count and locate the occurrences of conjugates of $T_1, ..., T_d$ that have a prefix p-matching a pattern p-string P. To keep the time complexity independent of d, it is desirable to have a single index solving the problem.

Our Contribution. In what follows, we provide an index solving the c-matching problem. A straightforward solution (which we present in this section) would index each input text twice $T_i \cdot T_i, i \in [1..d]$, so that the index is capable of reporting occurrences at the borders between the two T_i's. In a postprocessing, it is sufficient to discard those occurrences that start within the second T_i. Unfortunately, given n is the sum of the lengths of the input strings, we need $2n$ characters to index. We elaborate on a circular representation (in Sect. 3) that allows us to consider only n input characters, and thus is half of the size of that straightforward solution. Additionally, our solution is based on the eBWT dropping the requirement for an artificial delimiter in the input.

Theorem 2.2. *There exists a $2n \lg \sigma + O(n)$-bits index supporting c-matching. Given a pattern p-string of length m, the index answers a count query with (a) $O(m)$ rank/select on strings of length n with alphabet size σ, and (b) $O(m)$ RMQ queries on a conceptual string*

[1] Readers familiar with suffix arrays may observe that the semantics of RA_S resemble the suffix array.

of length n with alphabet size n. If the pattern is larger than the shortest input text, the index additionally needs time linear in the number of pattern occurrences. This additional step is also required for locate queries having the same time complexities.

Explicit time bounds depend on the data structure used for the queries introduced in Sect. 2. A data structure supporting all queries is the wavelet tree taking $O(\lg \sigma)$ time per query, combined with some RMQ data structure taking constant time per query.

Straightforward Solution. A straightforward solution is to index the concatenation of multiple p-strings, interpreted as a single p-string, with the pBWT, which we study as a starter. Our input $\mathcal{T} = \{T_1, T_2, ..., T_d\}$ is a finite set of non-empty text p-strings over $\Sigma_\mathrm{s} \cup \Sigma_\mathrm{p} - \{\$\}$. Let $n = |T_1 \cdots T_d|$ and $n_k = |T_k|$ for each $1 \leq k \leq d$. Our running example is $\mathcal{T} = \{\mathtt{AC}, \mathtt{AbC}, \mathtt{Aab}, \mathtt{ABBA}\}$. Let $S = T_1^2 T_2^2 \cdots T_d T_d[..n_d - 1]\$$. Then S satisfies the preconditions to be indexed with the pBWT. As each conjugate of each text p-string considered in $\mathcal{T}$ appears once as a prefix of distinct suffixes of S, the index built on the pBWT of S counts at least as many conjugates that have a prefix matching a pattern p-string P, but there might be occurrences which are double-counted and consequently the result requires further treatment. We need to check each element of the range returned by the backward search for duplicates. Legal occurrences can be located if we maintain a bit string B_T of length $2n$ indicating the beginning of a text and the beginning of its repeat, in text-range. If the rank of 1's at position $\mathrm{RA}_S[i]$ is even, the occurrence at lex-range i is to be discarded. If the rank is uneven, we derive the text p-string and the respective conjugate. B_T also allows for the reversibility of the transform. To avoid probing RA_S for count queries, we maintain a bit string B_L of length $2n$ to indicate legality of each entry in lex-range. While a count query is answered by the pBWT via backward searching a pattern, yielding a range $[\ell..r]$ within L'_S whose length is the number of occurrences of the sought pattern, the number of legal occurrences is $\mathrm{rank}_1(B_\mathrm{L}, r) - \mathrm{rank}_1(B_\mathrm{L}, \ell - 1)$. We also need to be wary of pattern p-strings exceeding the length of any text p-string.

For our example in Fig. 2, the backward search for the pattern $P = \mathtt{a}$ returns the range $[2..3]$ of length 2, but there is only the conjugate $\mathrm{Rot}(T_3, 1)$ starting with $\mathtt{a}$. We locate both occurrences by probing RA_S and get $\mathrm{RA}_S[2] = 12$ and $\mathrm{RA}_S[3] = 15$. We need to check their legality, which we can do with B_T depicted in Fig. 3; we observe that the entry 3 of our range is illegal.

In what follows, we define a new order for strings, which allows us to pave the way for our epBWT, which has, like the eBWT, no need for the artificial dollar symbols.

2.2 ω-Order of p-Strings

Inspired by Mantaci et al. [7], we introduce a new order $\preceq_\omega$ on $\Sigma_\mathrm{s} \cup \Sigma_\mathrm{p}$. For any p-strings U and W, let $U \preceq_\omega W$ if and only if $\langle U^\omega \rangle < \langle W^\omega \rangle$ or $\mathrm{root}(\llbracket U \rrbracket) = \mathrm{root}(\llbracket W \rrbracket)$. Then $\preceq_\omega$ defines a total order on $(\Sigma_\mathrm{s} \cup \Sigma_\mathrm{p})^*$. We write $U =_\omega W$ if and only if $U \preceq_\omega W$ and $W \preceq_\omega U$. We write $U \prec_\omega W$ if and only if $U \preceq_\omega W$ and $U \neq_\omega W$. For example, $\mathtt{BBAA} \prec_\omega \mathtt{ABBA}$, $\mathtt{aBBaB} \prec_\omega \mathtt{aBB}$ and $\mathtt{CA} =_\omega \mathtt{BCBC} =_\omega \mathtt{ABABAB}$. Note that our definition coincides with that of Mantaci et al. [7] if we only consider p-strings over Σ_s. We give a convenient way to

i	$\mathrm{RA}_S[i]$	$\mathrm{F}'_S[i]$	$\mathrm{L}'_S[i]$	$\mathrm{LF}'_S(i)$	i	$\mathrm{RA}_S[i]$	$\mathrm{F}'_S[i]$	$\mathrm{L}'_S[i]$	$\mathrm{LF}'_S(i)$
1	24	\$	3	8	13	22	1	2	19
2	12	a	1	9	14	20	1	2	20
3	15	a	1	10	15	18	1	2	21
4	13	b	a	2	16	10	3	b	5
5	9	b	2	11	17	7	2	b	6
6	6	b	2	12	18	4	2	2	22
7	16	b	a	3	19	21	2	1	14
8	23	3	1	13	20	19	2	1	15
9	11	1	3	16	21	17	2	b	7
10	14	1	b	4	22	3	2	2	23
11	8	2	2	17	23	2	2	2	24
12	5	2	2	18	24	1	2	\$	1

Figure 2: The pBWT of $S = \texttt{ACACAbCAbCAabAabABBAABB\$}$.

	$\vert T_1 \vert$	$\vert T_1 \vert$	T_2	$\vert$	T_2	$\vert$	T_3	$\vert$	T_3	$\vert$	T_4					$\vert$
$S[i] =$	A	C	A	C	A	b	C	A	b	C	A	a	b	A	a	b	A	B	B	A	A	B	B	\$
$B_{\mathrm{T}}[i] =$	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0
$i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Figure 3: Concatenation S of the input texts. The last factor of S is $T_4[..|T_4|-1] \cdot \$$.

compare two p-strings according to ω-order by making use of periodicity.

Lemma 2.3 (Lemma 5 by Hon et al. [4])**.** *Let U and W denote p-strings and $z = \max\{|U|, |W|\}$. Then $U =_\omega W$ if and only if $\langle U^\omega\rangle[..2z] = \langle W^\omega\rangle[..2z]$.*

Corollary 2.4. *Let U and W denote p-strings and $z = \max\{|U|, |W|\}$. Then $U \prec_\omega W$ if and only if $\langle U^\omega\rangle[..2z] < \langle W^\omega\rangle[..2z]$.*

We give an adaptation to the ω-order of the findings on p-strings presented by Iseri et al. [5]. For any two p-strings U, W with $z = \max\{|U|, |W|\}$, let

$$\mathrm{lcp}_\omega(U,W) = \begin{cases} \mathrm{lcp}(\langle U^\omega\rangle[..2z], \langle W^\omega\rangle[..2z]) & \text{if } U \neq_\omega W, \text{ or} \\ \infty & \text{otherwise.} \end{cases}$$

Let $\mathrm{lcp}^\infty_\omega(U,W)$ return the number of occurrences of ∞ in $\langle W^\omega\rangle[..\mathrm{lcp}_\omega(U,W)]$ if $U \neq_\omega W$, otherwise $|W|_{\mathrm{p}}$. For every non-empty p-string V, let $\pi(V) = [\![V]\!][1]$.

Lemma 2.5 (Lemma 4 by Iseri et al. [5] and Cor. 2.4)**.** *For p-strings $U, W \neq \varepsilon$, let $\lambda = \mathrm{lcp}_\omega(\mathrm{Rot}(U,1), \mathrm{Rot}(W,1))$ and $e = \mathrm{lcp}^\infty_\omega(\mathrm{Rot}(U,1), \mathrm{Rot}(W,1))$. If $\mathrm{Rot}(U,1) \prec_\omega \mathrm{Rot}(W,1)$ holds, then Fig. 4 shows a complete list of cases for $\lambda' = \mathrm{lcp}_\omega(U,W)$, $e' = \mathrm{lcp}^\infty_\omega(U,W)$ and the ω-order of U and W, where a case starting with the letter* A *is under the condition that at least one of $\pi(U)$ or $\pi(W)$ is an element of Σ_{s} and in a case starting with the letter* B *neither $\pi(U)$ nor $\pi(W)$ is an element of Σ_{s}.*

Corollary 2.6. *For any two p-strings U and W satisfying $\pi(U) = \pi(W)$, $\mathrm{Rot}(U,1) \prec_\omega \mathrm{Rot}(W,1)$ if and only if $U \prec_\omega W$.*

cases	additional conditions	λ'	e'	ω-order
(A1)	$\pi(U) \neq \pi(W)$	0	0	$U \prec_\omega W$ $\Leftrightarrow \pi(U) < \pi(W)$
(A2)	$\pi(U) = \pi(W)$	$\lambda + 1$	e	$U \prec_\omega W$
(B1)	$\pi(U) = \pi(W) \leq e$	$\lambda + 1$	e	$U \prec_\omega W$
(B2)	$\pi(U) \leq e$ and $\pi(U) < \pi(W)$	h	$\pi(U)$	$U \prec_\omega W$
(B3)	$\pi(W) \leq e$ and $\pi(W) < \pi(U)$	g	$\pi(W)$	$W \prec_\omega U$
(B4)	$e < \min\{\pi(U), \pi(W)\}$	$\lambda + 1$	$e + 1$	$U \prec_\omega W$

Figure 4: Cases of Lemma 2.5, $h = \text{select}_{U[1]}(U[2..], 1)$ and $g = \text{select}_{W[1]}(W[2..], 1)$.

3 An Extension of the pBWT

We define the epBWT of $\mathcal{T} = \{T_1, ..., T_d\}$ by sorting all the conjugates considered in $\mathcal{T}$ according to the ω-order. More precisely, we sort $\text{Rot}(\llbracket T_k \rrbracket, j)$ for $1 \leq k \leq d$ and $1 \leq j \leq n_k$ using $\langle \text{Rot}(T_k, j)^\omega \rangle[..2z]$ with $z = \max\{n_h \mid 1 \leq h \leq d\}$ as keys, where ties are broken w.r.t. the index of the corresponding text p-string first, and then w.r.t. the index within the corresponding text p-string.

Let $T = T_1 \cdots T_d$, i.e. $T = T_1 \cdot T_2 \cdot T_3 \cdot T_4 = \texttt{AC} \cdot \texttt{AbC} \cdot \texttt{Aab} \cdot \texttt{ABBA}$ for our running example. We imitate Hon et al. [4] to keep track of the starting position of each text in T by defining a bit string LEN_T that represents the length of each text in their order. A length k is encoded in unary by $\texttt{10}^{k-1}$ and we append an extra $\texttt{1}$ at the end of LEN_T as a sentinel. For our example, $\text{LEN}_T = \texttt{1010010010001}$.

We store the $\llbracket .. \rrbracket$-encoding of all the texts in the string $\text{ENC}_T = \llbracket T_1 \rrbracket \cdots \llbracket T_d \rrbracket$ over $\Sigma_\text{s} \cup [1..\sigma_\text{p}]$. By definition, $\text{ENC}_T = \text{root}(\llbracket T_1 \rrbracket)^{\exp(\llbracket T_1 \rrbracket)} \cdots \text{root}(\llbracket T_d \rrbracket)^{\exp(\llbracket T_d \rrbracket)}$. Thus, ENC_T factorizes into text roots. We define a bit string PRV_T marking the positions of ENC_T with a 1 if the position is the starting position of such a factor (i.e, a text root), in text-range. The definition is similar to that of LEN_T. In particular, $\text{LEN}_T = \text{PRV}_T$ if and only if $\llbracket T_k \rrbracket$ is primitive for each $1 \leq k \leq d$.

For our running example, $\llbracket T_1 \rrbracket = \texttt{22}$, $\llbracket T_2 \rrbracket = \texttt{2b2}$, $\llbracket T_3 \rrbracket = \texttt{1ab}$, and $\llbracket T_4 \rrbracket = \texttt{2121}$. Thus, $\text{ENC}_T = \texttt{222b21ab2121}$ and $\text{PRV}_T = \texttt{1110010010101}$. Having LEN_T and ENC_T allows us to restore the $\llbracket .. \rrbracket$-encodings of the text p-strings.

Let $\varphi_T(i) = \text{select}_1(\text{PRV}_T, \text{rank}_1(\text{PRV}_T, i) + 1) - 1$ if $\text{PRV}_T[i] = 1$, or $\varphi_T(i) = i - 1$ otherwise, for every $1 \leq i \leq n$. The idea is that φ_T moves circularly to the previous position inside a root of an encoded input text. For our running example, $\varphi_T(1) = 1$, $\varphi_T(3) = 5$ and $\varphi_T(8) = 7$.

For each $i \in [1..n]$, let $C(i) = \text{Rot}\left(T_{\text{rank}_1(\text{LEN}_T, \text{FPQ}_1(\text{LEN}_T, i))}, i - \text{FPQ}_1\left(\text{LEN}_T, i\right)\right)$. For our example, $C(1) = \texttt{AC}$, $C(2) = \texttt{CA}$, $C(10) = \texttt{BBAA}$ and $C(11) = \texttt{BAAB}$. The function $\text{R}_T : [1..n] \to [1..n]$ is defined by $\text{R}_T(i) < \text{R}_T(j) \Leftrightarrow C(i) \prec_\omega C(j) \vee (C(i) =_\omega C(j) \wedge i < j)$ for each $1 \leq i, j \leq n$ with $i \neq j$. R_T is a permutation[2] and the inverse function, denoted by R_T^{-1}, satisfies $\left(C(\text{R}_T^{-1}(i)) \prec_\omega C(\text{R}_T^{-1}(j))\right) \vee \left(C(\text{R}_T^{-1}(i)) =_\omega C(\text{R}_T^{-1}(j)) \wedge \text{R}_T^{-1}(i) < \text{R}_T^{-1}(j)\right)$

[2] R_T maps from text-range to lex-range, and thus has semantics similar to RA^{-1} for the pBWT or the inverse suffix array for the BWT.

i	$C(i)$	$\mathrm{ENC}_T[i]$	$\langle C(i)^\omega\rangle[..8]$	$\mathrm{R}_T^{-1}(i)$	$\varphi_T(\mathrm{R}_T^{-1}(i))$	$\mathrm{L}_T[i]$	$\langle C(\mathrm{R}_T^{-1}(i))^\omega\rangle[..8]$
1	AC	2	ɑɑ222222	7	6	1	abɑab3ab
2	CA	2	ɑɑ222222	8	7	a	bɑab3ab3
3	AbC	2	ɑbɑ3b33b	4	3	2	bɑɑb33b3
4	bCA	b	bɑɑb33b3	6	8	b	ɑab3ab3a
5	CAb	2	ɑɑb33b33	3	5	2	ɑbɑ3b33b
6	Aab	1	ɑab3ab3a	10	9	2	ɑ1ɑ13131
7	abA	a	abɑab3ab	12	11	2	ɑ1ɑ13131
8	bAa	b	bɑab3ab3	5	4	b	ɑɑb33b33
9	ABBA	2	ɑɑ131313	9	10	1	ɑɑ131313
10	BBAA	1	ɑ1ɑ13131	11	12	1	ɑɑ131313
11	BAAB	2	ɑɑ131313	1	1	2	ɑɑ222222
12	AABB	1	ɑ1ɑ13131	2	2	2	ɑɑ222222

Figure 5: The epBWT and auxiliary data structures of our running example.

for any $1 \leq i < j \leq n$.

Then the epBWT of $\mathcal{T}$, denoted by L_T, is a string of length n over $\Sigma_\mathrm{s} \cup [1..\sigma_\mathrm{p}]$ such that $\mathrm{L}_T[i] = \mathrm{ENC}_T[\varphi_T(\mathrm{R}_T^{-1}(i))]$ for each $1 \leq i \leq n$. Fig. 5 shows the epBWT of our running example.

3.1 LF-Mapping of the epBWT

The *LF-mapping* $\mathrm{LF}_T : [1..n] \to [1..n]$ is defined by $\mathrm{LF}_T(i) = \mathrm{R}_T(\varphi_T(\mathrm{R}_T^{-1}(i)))$. LF_T is a permutation that decomposes into $\mathrm{rank}_1(\mathrm{PRV}_T, n+1) - 1$ cycles. We denote its inverse *FL-mapping* by FL_T. Finally, the string F_T of length n over $\Sigma_\mathrm{s} \cup [1..\sigma_\mathrm{p}]$ is defined by $\mathrm{F}_T[i] = \mathrm{ENC}_T[\mathrm{R}_T^{-1}(i)]$ satisfying $\mathrm{F}_T[\mathrm{LF}_T(i)] = \mathrm{L}_T[i]$ for every $1 \leq i \leq n$.

Lemma 3.1. *Let* $1 \leq i < j \leq n$. *If* $\mathrm{L}_T[i] = \mathrm{L}_T[j]$, *then* $\mathrm{LF}_T(i) < \mathrm{LF}_T(j)$.

Proof. Let $1 \leq i < j \leq n$ and $\mathrm{L}_T[i] = \mathrm{L}_T[j]$. The definition of φ_T and Corollary 2.6 imply $C\left(\varphi_T\left(\mathrm{R}_T^{-1}(i)\right)\right) \preceq_\omega C\left(\varphi_T\left(\mathrm{R}_T^{-1}(j)\right)\right) \Leftrightarrow C\left(\mathrm{R}_T^{-1}(i)\right) \preceq_\omega C\left(\mathrm{R}_T^{-1}(j)\right)$. Since $i < j$, the right side is true per definition of R_T. Then $C(\varphi_T(\mathrm{R}_T^{-1}(i))) \preceq_\omega C(\varphi_T(\mathrm{R}_T^{-1}(j)))$ implies $\mathrm{R}_T(\varphi_T(\mathrm{R}_T^{-1}(i))) < \mathrm{R}_T(\varphi_T(\mathrm{R}_T^{-1}(j)))$. By definition, $\mathrm{LF}_T(i) < \mathrm{LF}_T(j)$. □

The previous result does not necessarily hold if we use LEN_T instead of PRV_T in the definition of φ_T. As a counterexample, consider $\mathrm{L}_{T_1} = \mathtt{22}$ of $\{T_1\}$. An LF-mapping based on LEN_{T_1} would map 1 to 2 and 2 to 1.

Corollary 3.2. *Let* $1 \leq i \leq n$. *Then* $\mathrm{LF}_T(i) = \mathrm{select}_{\mathrm{L}_T[i]}(\mathrm{F}_T, \mathrm{rank}_{\mathrm{L}_T[i]}(\mathrm{L}_T, i))$ *and* $\mathrm{FL}_T(i) = \mathrm{select}_{\mathrm{F}_T[i]}(\mathrm{L}_T, \mathrm{rank}_{\mathrm{F}_T[i]}(\mathrm{F}_T, i))$.

Corollary 3.3. *Given* F_T, L_T, *a representation of* R_T^{-1} *and* LEN_T, *we can restore the input text p-strings up to p-match.*

i	$C(i)$	$\mathrm{R}_T(i)$	$\mathrm{R}_T^{-1}(i)$	$\mathrm{F}_T[i]$	$\mathrm{L}_T[i]$	$\mathrm{LF}_T(i)$	$\mathrm{FL}_T(i)$
1	AC	11	7	a	1	4	2
2	CA	12	8	b	a	1	4
3	AbC	5	4	b	2	5	8
4	bCA	3	6	1	b	2	1
5	CAb	8	3	2	2	8	3
6	Aab	4	10	1	2	9	9
7	abA	1	12	1	2	10	10
8	bAa	2	5	2	b	3	5
9	ABBA	9	9	2	1	6	6
10	BBAA	6	11	2	1	7	7
11	BAAB	10	1	2	2	11	11
12	AABB	7	2	2	2	12	12

Figure 6: LF- and FL-mapping of our running example.

The original position of an entry $\mathrm{F}_T[i]$ of F_T is given by $\mathrm{R}_T^{-1}(i)$ for $1 \leq i \leq n$ via $\mathrm{ENC}_T[\mathrm{R}_T^{-1}(i)] = \mathrm{F}_T[i]$. In the particular case that $[\![T_k]\!]$ is primitive for every $1 \leq k \leq d$, reversibility up to order, conjugate and p-match of the input text p-strings is given by simply examining the decomposition of the FL-mapping.

3.2 Circular, Parameterized Matching

Adapting the techniques of Kim and Cho [6], we propose an index solving the c-matching problem for the p-string texts $T_1, ..., T_d$ and a p-string pattern P by leveraging backward search. To this end, for any p-string V, let the *V-interval* be the maximal interval $[\ell..r]$ such that $\langle V \rangle$ is a prefix of $\langle C(\mathrm{R}_T^{-1}(i))^\omega \rangle$ for each $\ell \leq i \leq r$. Fix some pattern p-string P and let $m = |P|$. The ε-interval of T is $[1..n]$. A single step of the backward search algorithm updates the $P[j+1..]$-interval to the $P[j..]$-interval for some $1 \leq j \leq m$, with $P[m+1..] = \varepsilon$.

Lemma 3.4. *Given L_T, F_T and some $c \in \Sigma_\mathrm{s} \cup \Sigma_\mathrm{p}$, we can update the V-interval $[\ell..r]$ of a p-string V to the cV-interval $[\ell'..r']$ of the p-string cV.*

Proof. We treat three different cases like in Lemma 14 by Iseri et al. [5]. Keep the identity $\varphi_T(\mathrm{R}_T^{-1}(i)) = \mathrm{R}_T^{-1}(\mathrm{LF}_T(i))$ for each $1 \leq i \leq n$ in mind.
Case 1. Let $c \in \Sigma_\mathrm{s}$. For any $1 \leq i \leq n$, $\langle C(\mathrm{R}_T^{-1}(\mathrm{LF}_T(i)))^\omega \rangle$ is prefixed by $\langle cV \rangle$ if and only if $\langle C(\mathrm{R}_T^{-1}(i))^\omega \rangle$ is prefixed by $\langle V \rangle$ and $\mathrm{L}_T[i] = c$. Hence, $\mathrm{LF}_T(i) \in [\ell'..r']$ if and only if $i \in [\ell..r]$ and $\mathrm{L}_T[i] = c$. As a consequence of Lemma 3.1, $\ell' = \mathrm{LF}_T(\mathrm{FNQ}_c(\mathrm{L}_T, \ell))$ and $r' = \mathrm{LF}_T(\mathrm{FPQ}_c(\mathrm{L}_T, r))$.
Case 2. Let $c \in \Sigma_\mathrm{p}$ and $V[k] = c$ for some $1 \leq k \leq |V|$. For any $1 \leq i \leq n$, $\langle C(\mathrm{R}_T^{-1}(\mathrm{LF}_T(i)))^\omega \rangle$ is prefixed by $\langle cV \rangle$ if and only if $\langle C(\mathrm{R}_T^{-1}(i))^\omega \rangle$ is prefixed by $\langle V \rangle$ and $\mathrm{L}_T[i] = \pi(cV)$. Thus, $\mathrm{LF}_T(i) \in [\ell'..r']$ if and only if $i \in [\ell..r]$ and $\mathrm{L}_T[i] = \pi(cV)$. By Lemma 3.1, $\ell' = \mathrm{LF}_T(\mathrm{FNQ}_{\pi(cV)}(\mathrm{L}_T, \ell))$ and $r' = \mathrm{LF}_T(\mathrm{FPQ}_{\pi(cV)}(\mathrm{L}_T, r))$.
Case 3. Let $c \in \Sigma_\mathrm{p}$ and $V[j] \neq c$ for each $1 \leq j \leq |V|$. Let $e = |V|_\mathrm{p} = |cV|_\mathrm{p} - 1$. For any $1 \leq i \leq n$, $\langle C(\mathrm{R}_T^{-1}[\mathrm{LF}_T(i)])^\omega \rangle$ is prefixed by $\langle cV \rangle$ if and only if $\langle C(\mathrm{R}_T^{-1}[i])^\omega \rangle$

is prefixed by $\langle V \rangle$ and $\mathrm{L}_T[i] > e$. Then $\mathrm{LF}_T(i) \in [\ell'..r']$ if and only if $i \in [\ell..r]$ and $\mathrm{L}_T[i] > e$ and consequently the length of $[\ell'..r']$ is exactly the number of entries in $[\ell..r]$ with $\mathrm{L}_T[i] > e$. Moreover, $C(\mathrm{R}_T^{-1}[\mathrm{LF}_T(i)]) \prec_\omega C(\mathrm{R}_T^{-1}[\mathrm{LF}_T(j)])$ for every $i, j \in [\ell..r]$ with $\mathrm{L}_T[i] \leq e$ and $\mathrm{L}_T[j] > e$ by Cases (A1), (B2) and (B3) of Lemma 2.5. Thus, it suffices to compute the largest entry in $\{\mathrm{LF}_T(i) \mid \ell \leq i \leq r\}$ to obtain r'. Then $\ell' = r' - \mathrm{rangecount}_{\mathrm{L}_T}(\ell, r, e+1, \sigma_{\mathrm{p}}) + 1$. □

For a pattern p-string P with $m \leq \min\{n_k \mid 1 \leq k \leq d\}$, the index supports count queries by reporting the length of the P-interval (computed by the backward search steps of Lemma 3.4) and locate queries with the addition of a representation of R_T^{-1} and LEN_T. If $m > \min\{n_k \mid 1 \leq k \leq d\}$, we need to locate all occurrences and discard those from texts with lengths less than m for accurate results. Thus, our proposed index augmented with representations of R_T^{-1} and LEN_T solves the indexing problem for c-matching, and we obtain Thm. 2.2, where the conceptional string in the statement (b) is LF_T represented as stated in Cor. 3.2.

Example. Finally, we give an example for the matching with epBWT. Let $P = \mathtt{CAA}$. We start out with the initial ε-interval $[1..12]$ of $P[4..]$ and update it to the $\mathtt{A}$-interval of $P[3..]$. Since $\mathtt{A}$ is a p-symbol that does not appear in $P[4..] = \varepsilon$, we face the third case described in Lemma 3.4. We have $e = 0$, $r' = 12$ and $\ell' = 12 - \mathrm{rangecount}_{\mathrm{L}_T}(1, 12, 1, 3) + 1 = 12 - 9 + 1 = 4$. Next is the update of the $\mathtt{A}$-interval of $P[3..]$ to the $\mathtt{AA}$-interval of $P[2..]$. As $\mathtt{A}$ is a p-symbol that does appear in $P[3..] = \mathtt{A}$, we face the second case of Lemma 3.4. We compute $\ell' = \mathrm{LF}_T(\mathrm{FNQ}_1(\mathrm{L}_T, 4)) = \mathrm{LF}_T(9) = 6$ and $r' = \mathrm{LF}_T(\mathrm{FPQ}_1(\mathrm{L}_T, 12)) = \mathrm{LF}_T(10) = 7$. Last, the update of the $\mathtt{AA}$-interval of $P[2..]$ to the $\mathtt{CAA}$-interval of $P[1..]$. Since $\mathtt{C}$ is a p-symbol that does not appear in $P[2..] = \mathtt{AA}$, we face the third case again. We have $e = 1$, $r' = 10$ and $\ell' = 10 - \mathrm{rangecount}_{\mathrm{L}_T}(6, 7, 2, 3) + 1 = 10 - 2 + 1 = 9$. Since $m > 2$, we need to discard all results from T_1. The pattern p-string P occurs at positions 1 and 3 of T_4, i.e. $\mathrm{Rot}(T_4, 0)[..3]$ and $\mathrm{Rot}(T_4, 2)[..3]$ p-match P. Hence, the count remains 2. For comparison, the backward search on the index utilizing the pBWT of S in Fig. 2 returns the lex-range interval $[19..21]$ of length 3 and locating each occurrence results in discarding one occurrence at lex-range position 19.

Acknowledgements. This research was supported by JSPS KAKENHI with grant numbers `JP21K17701` and `JP23H04378`.

References

[1] B. S. Baker. A Theory of Parameterized Pattern Matching: Algorithms and Applications. In *Proceedings of ACM*, pages 71–80. Association for Computing Machinery, 1993.

[2] A. Ganguly, R. Shah, and S. V. Thankachan. pBWT: Achieving Succinct Data Structures for Parameterized Pattern Matching and Related Problems. In *Proceedings of SODA*, pages 397–407, 2017.

[3] D. Hashimoto, D. Hendrian, D. Köppl, R. Yoshinaka, and A. Shinohara. Computing the Parameterized Burrows–Wheeler Transform Online. In *Proceedings of CPM*, volume 13617 of *LNCS*, pages 70–85. Springer, 2022.

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

[5] K. Iseri, T. I, D. Hendrian, D. Köppl, R. Yoshinaka, and A. Shinohara. Breaking a Barrier in Constructing Compact Indexes for Parameterized Pattern Matching. *CoRR*, abs/2308.05977, 2023.

[6] S.-H. Kim and H.-G. Cho. Simpler FM-index for parameterized string matching. *Information Processing Letters*, 165:106026, 2021.

[7] S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. An extension of the Burrows–Wheeler Transform. *Theoretical Computer Science*, 387(3):298–312, 2007.

[8] J. Mendivelso, S. V. Thankachan, and Y. Pinzón. A brief history of parameterized matching problems. *Discrete Applied Mathematics*, 274:103–115, 2020.

[9] T. Shibuya. Generalization of a Suffix Tree for RNA Structural Pattern Matching. *Algorithmica*, 39(1):1–19, 2004.