Structural Pattern Matching – Succinctly*

Arnab Ganguly[1], Rahul Shah[2], and Sharma V. Thankachan[3]

1. **University of Wisconsin - Whitewater, Whitewater, USA**
   gangulya@uww.edu
2. **Louisiana State University, Baton Rouge, USA and NSF, Arlington, USA**
   rahul@csc.lsu.edu, rahul@nsf.gov
3. **University of Central Florida, Orlando, USA**
   sharma.thankachan@ucf.edu

Abstract

Let T be a text of length n containing characters from an alphabet Σ, which is the union of two disjoint sets: Σ_s containing static characters (s-characters) and Σ_p containing parameterized characters (p-characters). Each character in Σ_p has an associated complementary character from Σ_p. A pattern P (also over Σ) matches an equal-length substring S of T iff the s-characters match exactly, there exists a one-to-one function that renames the p-characters in S to the p-characters in P, and if a p-character x is renamed to another p-character y then the complement of x is renamed to the complement of y. The task is to find the starting positions (occurrences) of all such substrings S. Previous indexing solution [Shibuya, SWAT 2000], known as *Structural Suffix Tree*, requires $\Theta(n \log n)$ bits of space, and can find all *occ* occurrences in time $O(|P| \log \sigma + occ)$, where $\sigma = |\Sigma|$. In this paper, we present the first succinct index for this problem, which occupies $n \log \sigma + O(n)$ bits and offers $O(|P| \log \sigma + occ \cdot \log n \log \sigma)$ query time.

1998 ACM Subject Classification F.2.2 Pattern Matching

Keywords and phrases Parameterized Pattern Matching, Suffix tree, Burrows-Wheeler Transform, Wavelet Tree, Fully-functional succinct tree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.35

1 Introduction

Text Indexing is a classical problem defined as: pre-process a text T of length n containing characters from an alphabet Σ of size $\sigma \le n$ and then build a data structure, such that given a pattern P (also over Σ) as a query, we can report all the *occ* starting positions (or simply, occurrences) of P in T. *Suffix Tree* is the ubiquitous data structure for this purpose [14]. Unfortunately, it requires $\Theta(n \log n)$ bits of space, which is too large for most practical purposes (15-50 times the text). Grossi and Vitter [13], and Ferragina and Manzini [6] addressed this problem by introducing space-efficient indexes, namely *Compressed Suffix Arrays* (CSA) and *FM-Index* respectively. Subsequently, a lot of progress has been made either in improving these initial breakthroughs [2, 7, 8, 18, 20], or to achieve space-efficient indexes for other problems which require suffix trees as a component [16, 23].

The key concept behind the FM-Index and the CSA is the *suffix link*: the suffix link of a node u points to a node v iff the string from root to v is the same as the string from root to u with the first character truncated. Suffix links have the following so called *rank-preserving* property: the leaves obtained by following suffix links from the leaves in u's

* Work was partially supported by NSF Grant CCF–1527435.

© Arnab Ganguly, Rahul Shah and Sharma V. Thankachan;
licensed under Creative Commons License CC-BY
28th International Symposium on Algorithms and Computation (ISAAC 2017).
Editors: Yoshio Okamoto and Takeshi Tokuyama; Article No. 35; pp. 35:1–35:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

subtree appear in the same relative lexicographic order in the subtree of v. However, in many important variants [1, 4, 5, 10, 15, 21, 22] of the suffix tree, such as the parameterized suffix tree, the 2D suffix tree, and the structural suffix tree, this rank-preserving property of suffix links does not hold. Consequently, there has been very little progress in designing compressed representations of these suffix tree variants. Only recently, Ganguly et al. [9] designed the first succinct index for parameterized pattern matching [1]. We consider its generalization [21], which has applications in RNA structural matching.

Throughout this paper, we use the following terminologies: Σ is an alphabet of size $\sigma \geq 2$, which is the union of two disjoint sets – Σ_s having σ_s static characters (s-characters) and Σ_p having σ_p parameterized characters (p-characters). For each p-character, we associate a p-character, called the complement character. For a string S, $|S|$ is its length, $S[i], 1 \leq i \leq |S|$, is its ith character and $S[i,j]$ is its substring from i to j. If $i > j$, $S[i,j]$ denotes an empty string. Also S_i denotes the circular suffix starting at position i. Specifically, S_i is S if $i = 1$ and is $S[i,|S|] \circ S[1,i-1]$ otherwise, where $\circ$ denotes the *concatenation*.

▶ **Definition 1.** Two equal-length strings S and S' are a *structural-match* (s-match) iff

- $S[i] \in \Sigma_s \iff S'[i] \in \Sigma_s$,
- $S[i] = S'[i]$ when $S[i] \in \Sigma_s$,
- there exists a one-to-one matching-function f that renames the p-characters in S to the p-characters in S', i.e., $S'[i] = f(S[i])$ when $S[i] \in \Sigma_p$, and
- if a p-character x in S is renamed to y in S', then the complement (if exists) of x in S is renamed to the complement of y in S'.

Consider the following examples. Let $\Sigma_s = \{A, B, C\}$ and $\Sigma_p = \{w, x, y, z\}$, where the complement pairs are w-x and y-z. Then $AxByCx$ is an s-match with $AyBxCy$; in this case, there are no complementary requirements. Also, $AxBwCx$ is an s-match with $AzByCz$; here, x is paired with z, and w (complement of x) is paired with y (complement of z). However, $AxBwCx$ is not an s-match with $AzBxCz$ (even though the one-to-one criterion is satisfied); this is because as x is paired with z, w should have been paired with y. Lastly, $AxBwCx$ is not an s-match with $AzBxCy$ because x has to be renamed to both z and y, which violates the one-to-one criterion.

We consider the following indexing problem introduced by Shibuya [21].

▶ **Problem 2.** *Let T be a text of length n over Σ. We assume T terminates in a uniquely appearing s-character \$. Index T, such that given a pattern P (also over Σ), we can report all starting positions (occurrences) of the substrings of T that are an s-match with P.*

Shibuya presented a $\Theta(n \log n)$-bit and $O(|P| \log \sigma + occ)$-time index for this problem. We present the following new result.

▶ **Theorem 3.** *By using an $n \log \sigma + O(n)$-bit index of T, we can count the number of s-matches of a pattern P in $O(|P| \log \sigma)$ time. Subsequently, each match can be reported in $O(\log \sigma \log n)$ time.*

1.1 Overview of Techniques

We start with the closely related parameterized matching (p-matching) problem of Baker [1]. Two strings are a p-match if they satisfy the first three criteria in Definition 1. Thus if two strings are an s-match, they are definitely also a p-match, but may not be true the other way around. To create an index for the p-matching problem (i.e., replace s-match by p-match in Problem 2), Baker [1] introduced an encoding scheme such that two strings are a p-match

iff their encoded strings are the same. Using this encoding scheme, Baker obtained a linear space index for the p-matching problem. Similarly, the key to obtain a linear-space index for Problem 2 is an encoding scheme such that two strings are an s-match if their encoded strings are the same. Luckily, we already have such an encoding scheme. Specifically, using the encoding scheme of Shibuya [21], we can construct the *structural suffix tree* (s-suffix tree) as follows: first encode each suffix of T and then create a compact trie of these encoded suffixes. To report the occurrences of a pattern, we first find the highest node u in the s-suffix tree such that the string obtained by concatenating the edge labels from root to u is prefixed by the encoded pattern. Then, we report the starting positions of the encoded suffixes corresponding to the leaves in the subtree of u. However, Shibuya's encoding scheme (as well as Baker's scheme) has the following drawback: on prepending the preceding character of a suffix, the encoding of the original suffix changes. Consequently, FM-Index [6] and CSA [13] no longer work for these definitions of pattern matching.

Since the p-matching problem of Baker [1] is similar to Problem 2, one may be tempted to think that we can simply re-use (with minor adjustments) the succinct data structure of Ganguly et al. [9] for the p-matching problem. Although, this is true, the extension is not trivial. This is because, in contrast to the encoding scheme [1] used for p-matching, Shibuya's encoding scheme has a caveat: when we prepend the previous character of a suffix, the change in the encoding of the original suffix can occur at two positions. Hence, the index of Ganguly et al. [9] will no longer directly work. The first step, therefore, is a new encoding scheme which alleviates this problem, and a version of the s-suffix tree based on this encoding scheme; Section 2 presents the details.

Since we have now restricted the number of points of change (on prepending) to at most one, we use techniques similar to that employed by Ganguly et al. [9]. We store the number of distinct p-characters up to this point of change (from the start of the suffix) in $\approx \log \sigma$ bits per suffix. However, we make a distinction between the cases when the change is due to the complement of the prepended p-character versus the change due to the same p-character. This forms the backbone of our data structure, and we call it the *Structural Burrows-Wheeler Transform* (sBWT); the details are in Section 3.

The next step is to compute the starting positions of the lexicographically arranged encoded (with our new encoding scheme) suffixes. We implement the *Structural LF mapping* (sLF mapping), using which we can decode the starting positions without explicitly storing them. Summarizing our discussions thus far, we can see that the key is to compute sLF mapping. To this end, we use the sBWT and the topology of the s-suffix tree; the crucial insight is provided in Lemma 9. Based on this lemma, we implement sLF mapping in Section 4; space and time complexities are described in Lemma 14.

The last piece of the puzzle is to compute the suffix range of the encoded pattern (i.e., find the range of leaves under the node u defined at the beginning of this section). We again use sLF mapping, the s-suffix tree topology, and sBWT to implement a backward search procedure (like that in the FM Index [6] and succinct index for the p-matching problem [9]). The details of the backward search procedure for s-matching are in Section 5.

2 Linear-Space Index

We first consider the encoding scheme by Shibuya [21]. A string S is encoded into an equal-length string $\mathsf{sencode}(S)$ by replacing the first occurrence of every p-character in S by 0 and any other occurrence of a p-character by the difference in text position from its previous occurrence. Specifically, for any $i \in [1, |S|]$, $\mathsf{sencode}(S)[i] = S[i]$ if $S[i]$ is an s-character;

otherwise, $\mathsf{sencode}(S)[i] = (i - j)$, where $j < i$ is the last occurrence of $S[i]$ before i. If j does not exist, then $j = i$.

Now, for every p-character $S[i]$, where $\mathsf{sencode}(S)[i] = 0$, we find the rightmost $j < i$ such $S[j]$ is the complement of $S[i]$. If j exists, then replace $\mathsf{sencode}(S)[i]$ by $-(i - j)$. For e.g., $\mathsf{sencode}(AxBwAwCxAx) = A0B(-2)A2C6A2$, where the first step yields the string $A0B0A2C6A2$. Here, $\Sigma_s = \{A, B, C\}$ and $\Sigma_p = \{w, x\}$; additionally, w and x are complement of each other.

► **Fact 4** ([21]). *Two strings S and S' are an s-match iff $\mathsf{sencode}(S) = \mathsf{sencode}(S')$. Also S and a prefix of S' are an s-match iff $\mathsf{sencode}(S)$ is a prefix of $\mathsf{sencode}(S')$.*

2.1 New Encoding Scheme

Unfortunately, for our purposes, the encoding scheme defined in the previous sub-section suffers from a drawback. Specifically, let S be a string and x be a p-character. Then $\mathsf{sencode}(xS)[2, |S| + 1]$ can differ from $\mathsf{sencode}(S)$ at two distinct positions. For example, consider the string $S = wAwBxAx$. Here, $\Sigma_s = \{A, B\}$ and $\Sigma_p = \{w, x\}$; additionally, w and x are complement of each other. Then, $\mathsf{sencode}(S) = 0A2B(-2)A2$ and $\mathsf{sencode}(xS) = \mathsf{sencode}(xwAwBxAx) = 0(-1)A2B5A2$. We want to avoid such an encoding scheme as it will prevent us from using the techniques of Ganguly et al. [9]. To this end, we present the following new encoding scheme.

We encode a string S as $\Phi(S)$ as follows. If $S[i]$ is static, then $\Phi(S)[i] = S[i]$. Consider a p-character $S[i]$ and let $j^+ < i$ and $j^- < i$ be the rightmost occurrence of $S[i]$ and the complement of $S[i]$ in $S[1, i - 1]$. If there is no occurrence j^+ (resp. j^-), we let $j^+ = -1$ (resp. $j^- = -1$). If $j^+ = j^- = -1$, then replace $S[i]$ by 0. Otherwise, if $j^+ > j^-$, then $\Phi(S)[i] = (i - j^+)$. Otherwise, if $j^- > j^+$, then $\Phi(S)[i] = -(i - j^-)$. For example, $\Phi(AxByCx) = A0B0C4$ and $\Phi(AxBwAwCxAx) = A0B(-2)A2C(-2)A2$. Here, $\Sigma_s = \{A, B, C\}$ and $\Sigma_p = \{w, x\}$; additionally, w and x are complement of each other.

Importantly, note that we alleviate the problem of Shibuya's encoding. Specifically, $\mathsf{sencode}(xS)[2, |S| + 1]$ can differ from $\mathsf{sencode}(S)$ at most at one position, which is easily illustrated by choosing $S = wAwBxAx$. All we are left to do is show that our encoding scheme still guarantees that two strings are an s-match iff the corresponding encoded strings are the same, which is handled by the following lemma.

► **Lemma 5.** *Two strings S and S' are an s-match iff $\Phi(S) = \Phi(S')$. Also S and a prefix of S' are a p-match iff $\Phi(S)$ is a prefix of $\Phi(S')$.*

Proof. If S and S' are an s-match, then $\Phi(S) = \Phi(S')$ as S can be renamed to S' by applying the necessary one-to-one function. Therefore, it suffices to show that $\Phi(S) = \Phi(S')$ implies S and S' are an s-match. We note that the ith zero in $\Phi(S)$ (resp. in $\Phi(S')$) corresponds to the ith distinct p-character, say c_i (resp. c'_i), in S (resp. in S') such that neither c_i (resp. c'_i) nor its complement appear before. Thus, we establish the one-to-one mapping $c_i \to c'_i$. Let p be the position of an occurrence of c_i in S. Let $q > p$ be the minimum position (if any) where c_i (or, its complement) occurs in $S[p + 1, |S|]$. Since $\Phi(S') = \Phi(S)$, q is also the minimum position where c'_i (or, its complement) occurs in $S'[p + 1, |S'|]$. Therefore, if any position p is the occurrence of c_i (resp. its complement) in S, then p is the occurrence of c'_i (resp. its complement) in S'. ◄

► **Convention 6.** *The integer characters (corresponding to p-characters) are lexicographically smaller than s-characters. An integer character i comes before another integer character j iff $i < j$. Also, \$ is the largest character.*

2.2 Structural Suffix Tree

Structural Suffix Tree (sST) is the compacted trie of all strings in $\mathcal{P} = \{\Phi(T[k,n]) \mid 1 \leq k \leq n\}$. Each edge is labeled with a string over $\Sigma' = \Sigma_s \cup \{0, 1, \ldots n-1\}$. We use $\mathsf{str}(u)$ to denote the concatenation of edge labels on the path from root to node u. The path of each leaf node corresponds to the encoding of a unique suffix of T, and leaves are ordered in the lexicographic order of the corresponding encoded suffix. Clearly, sST consists of n leaves (one per each encoded suffix) and at most $n-1$ internal nodes. We also store the structural suffix array $\mathsf{sSA}[1,n]$ i.e., $\mathsf{sSA}[i] = j$ and $\mathsf{sSA}^{-1}[j] = i$ iff $\Phi(T[j,n])$ is the ith lexicographically smallest string in $\mathcal{P}$. Note that $\mathsf{str}(\ell_i) = \Phi(T[\mathsf{sSA}[i],n])$, where ℓ_i is the ith leftmost leaf in sST. The total space required is $\Theta(n \log n)$ bits.

To find all occurrences of P, traverse sST from root by following the edges labels and find the highest node u (called *locus*) such that $\mathsf{str}(u)$ is prefixed by $\Phi(P)$. Then find the range $[sp, ep]$ (called *suffix range* of $\Phi(P)$) of leaves in the subtree of u and report $\{\mathsf{sSA}[i] \mid sp \leq i \leq ep\}$ as the output. The query time is $O(|P| \log \sigma + occ)$, where occ is the number of occurrences of P in T.

We remark that the structural suffix tree described here varies from that by Shibuya [21]. Their tree is based on sencode and can be constructed in $O(n \log \sigma)$ time using $\Theta(n \log n)$ bits of working space. Based on Fact 4 and Lemma 5, we observe that the longest common prefix (LCP) of any two encoded suffix is the same whether we use sencode or Φ as the encoding function. Therefore, given Shibuya's tree, we can easily create sST by relabeling the edges, and then sorting them based on their first character and Convention 6. The additional time needed is $O(n)$ using any linear-time sorting algorithm. Summarizing, we can create sST in $O(n \log \sigma)$ time using $\Theta(n \log n)$ bits of working space.

3 Structural Burrows-Wheeler Transform

We use a similar transform to that of the Burrows and Wheeler [3], which we call as the Structural Burrows-Wheeler Transform (sBWT). Sort the circular suffixes T_x, $1 \leq x \leq n$, based on their $\Phi(\cdot)$ encoding, where character precedence is determined by Convention 6. Then, obtain the last character $L[i]$ of the ith lexicographically smallest circular suffix. Denote by f_i^+ (resp. f_i^-) the first occurrence of $L[i]$ (resp. the complement of $L[i]$) in the circular suffix T_i. In case, there is no occurrence of $L[i]$'s complement, we take $f_i^- = n+1$.

The sBWT is defined as $\mathsf{sBWT}[i] =$

$$\begin{cases} L[i], & \text{if } L[i] \in \Sigma_s \\ \text{number of distinct p-characters in } T_{\mathsf{sSA}[i]}[1, f_i^+], & \text{if } L[i] \in \Sigma_p \text{ and } f_i^+ < f_i^- \\ -\text{number of distinct p-characters in } T_{\mathsf{sSA}[i]}[1, f_i^-], & \text{if } L[i] \in \Sigma_p \text{ and } f_i^+ > f_i^- \end{cases}$$

▶ **Observation 7.** *For any $1 \leq i \leq n$, let $c = \mathsf{sBWT}[i]$. Then, $\Phi(T_{\mathsf{sSA}[i]-1}) =$*

$$\begin{cases} c \circ \Phi(T_{\mathsf{sSA}[i]})[1, n-1], & \text{if } c \in \Sigma_s \\ 0 \circ \Phi(T_{\mathsf{sSA}[i]})[1, f_i^+ - 1] \circ f_i^+ \circ \Phi(T_{\mathsf{sSA}[i]})[f_i^+ + 1, n-1], & \text{if } c \in [1, \sigma_p] \\ 0 \circ \Phi(T_{\mathsf{sSA}[i]})[1, f_i^- - 1] \circ -f_i^- \circ \Phi(T_{\mathsf{sSA}[i]})[f_i^- + 1, n-1], & \text{if } c \in [-\sigma_p, -1] \end{cases}$$

The structural last-to-first column (sLF) mapping of i is the position at which the character at $L[i]$ lies in the first column of the sorted encoded suffixes. Specifically, $\mathsf{sLF}(i) = \mathsf{sSA}^{-1}[\mathsf{sSA}[i] - 1]$, where $\mathsf{sSA}^{-1}[0] = \mathsf{sSA}^{-1}[n]$. The following lemma is a straightforward adaptation of Theorem 3 in [9].

▶ **Lemma 8.** *Assume* $\mathsf{sLF}(\cdot)$ *can be computed in* t_{sLF} *time. By using an additional* $O(n)$*-bit data structure, we can compute* $\mathsf{sSA}[\cdot]$ *in* $O(t_{\mathsf{sLF}} \cdot \log n)$ *time.*

4 Implementing Structural LF Mapping

As highlighted by Lemma 8, the objective is to compute sLF. In this section, we show that $\mathsf{sLF}(i)$ can be computed in $O(\log \sigma)$ time using $n \log \sigma + O(n)$ bits.

▶ **Lemma 9.** *Consider two suffixes* i *and* j *corresponding to the leaves* ℓ_i *and* ℓ_j *in* sST*.*

(a) *If* $L[i]$ *is parameterized and* $L[j]$ *is static, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$*.*

(b) *If both* $L[i]$ *and* $L[j]$ *are static, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$ *iff either* $\mathsf{sBWT}[i] < \mathsf{sBWT}[j]$*, or* $\mathsf{sBWT}[i] = \mathsf{sBWT}[j]$ *and* $i < j$*.*

(c) *Assume* $i < j$ *and both* $L[i]$ *and* $L[j]$ *are parameterized. Let* u *be the lowest common ancestor of* ℓ_i *and* ℓ_j *in* sST*, and* z *be the number of* 0*'s in the string* $\mathsf{str}(u)$*. Then,*

1. *If* $|\mathsf{sBWT}[i]|, |\mathsf{sBWT}[j]| \leq z$*, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$ *iff*
   - *either* $\mathsf{sBWT}[i], \mathsf{sBWT}[j] > 0$ *and* $\mathsf{sBWT}[i] \geq \mathsf{sBWT}[j]$*,*
   - *or* $\mathsf{sBWT}[i] < 0 < \mathsf{sBWT}[j]$*,*
   - *or* $\mathsf{sBWT}[i], \mathsf{sBWT}[j] < 0$ *and* $|\mathsf{sBWT}[i]| \leq |\mathsf{sBWT}[j]|$
2. *If* $|\mathsf{sBWT}[i]| \leq z < |\mathsf{sBWT}[j]|$*, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$ *iff* $\mathsf{sBWT}[i] < 0$
3. *If* $|\mathsf{sBWT}[i]| > z \geq |\mathsf{sBWT}[j]|$*, then* $\mathsf{sLF}(i) < \mathsf{sLF}(j)$ *iff* $\mathsf{sBWT}[j] > 0$
4. *If* $|\mathsf{sBWT}[i]|, |\mathsf{sBWT}[j]| > z$*, then* $\mathsf{sLF}(i) > \mathsf{sLF}(j)$ *iff*
   - *either* $\mathsf{sBWT}[i] = z + 1$*, the first character on the* u *to* ℓ_i *path is* 0*, and the first character on the* u *to* ℓ_j *path is not an s-character,*
   - *or* $\mathsf{sBWT}[j] = -(z + 1)$*, and the first character on the* u *to* ℓ_j *path is* 0*.*

Proof. (a) and (b): Follows immediately from Convention 6 and Observation 7. (c) Let $d = |\mathsf{str}(u)|$. Define f_i to be smaller of the two values f_i^+ or f_i^-. Similarly, f_j is defined. Clearly, the conditions (1)-(4) can be written as: (1) Both $f_i, f_j \leq d$, (2) $f_i \leq d$ and $f_j > d$, (3) $f_i > d$ and $f_j \leq d$, and (4) Both $f_i, f_j > d$. Then the claims (1)-(3) follow from Observation 7 and Convention 6. To prove (4), observe that if the suffixes i and j swap order on being prepended by their previous characters then at least either f_i or f_j equals $d + 1$. The claim follows from Observation 7 and Convention 6. ◀

4.1 Wavelet Tree (WT) over sBWT

Let $A[1, m]$ be an array over an alphabet of size σ. There exists a data structure of size $m \log \sigma + o(m)$ bits, using which the following queries can be answered in $O(\log \sigma / \log \log m)$ time [6, 11, 12, 17]:

- $A[i]$,
- $\mathsf{rank}_A(i, x)$ = the number of occurrences of x in $A[1, i]$,
- $\mathsf{select}_A(i, x)$ = the ith occurrence of x in $A[1, m]$, and
- $\mathsf{count}_A(i, j, x, y)$ = number of elements in $A[i, j]$ that are at least x and at most y.

We drop the subscript A when the context is clear. Recall that the sBWT is a string of length n over the alphabet set $\Sigma_s \cup \{1, 2, \ldots, \sigma_p\} \cup \{-1, -2, \ldots, -\sigma_p\}$ of size $\sigma' = \sigma_s + 2\sigma_p \leq 2\sigma$. By using a WT over sBWT in $n \log \sigma' + o(n) = n \log \sigma + O(n)$ bits, we can support the above operations over sBWT in time $O(1 + \log \sigma / \log \log n)$.

4.2 Succinct representation of sST

A tree having m nodes can be represented in $2m + o(m)$ bits, such that if each node is labeled by its pre-order rank, the following operations can be supported in $O(1)$ time (note that $m < 2n$ in our case) [19]:

- $\mathsf{pre\text{-}order}(u)/\mathsf{post\text{-}order}(u)$ = the pre/post-order rank of node u,
- $\mathsf{parent}(u)$ = the parent of node u,
- $\mathsf{nodeDepth}(u)$ = the number of edges on the path from root to u,
- $\mathsf{child}(u, q)$ = the qth leftmost child of node u,
- $\mathsf{lca}(u, v)$ = the lowest common ancestor (LCA) of two nodes u and v,
- $\mathsf{L}(u)/\mathsf{R}(u)$ = the leftmost/rightmost leaf of the subtree rooted at u, and
- $\mathsf{levelAncestor}(u, D)$ = the ancestor of u such that $\mathsf{nodeDepth}(u) = D$.

Additionally, we can find the pre-order rank of the ith leftmost leaf in $O(1)$ time. Moving forward, we use ℓ_i to denote the ith leftmost leaf in sST.

4.3 ZeroDepth and ZeroNode

For a node u, $\mathsf{zeroDepth}(u)$ is the number of 0's in $\mathsf{str}(u)$. For a leaf ℓ_i, $\mathsf{sBWT}[i] \in [1, \sigma_p]$ (resp. $\mathsf{sBWT}[i] < 0$), we define $\mathsf{zeroNode}(\ell_i)$ to be the locus (if exists) of $\mathsf{str}(\ell_i)[1, f_i^+]$ (resp. the locus of $\mathsf{str}(\ell_i)[1, f_i^-]$). Equivalently, $\mathsf{zeroNode}(\ell_i)$ is the highest node (if exists) z on the root to ℓ_i path such that $\mathsf{zeroDepth}(w) \geq |\mathsf{sBWT}[i]|$. Moving forward, whenever we refer to $\mathsf{zeroNode}(\ell_i)$, we assume $\mathsf{sBWT}[i] \in [-\sigma_p, \sigma_p]$. We present the following lemma.

▶ **Lemma 10.** *By using the Wavelet Tree over* sBWT *and an additional $O(n)$-bit data structure, we can find* $\mathsf{zeroNode}(\ell_i)$ *in $O(\log \sigma)$ time.*

Proof. We begin with the following definitions. For any node x on the root to ℓ_i path π, define $\alpha(x)$ = the number of leaves ℓ_j, $j \in [\mathsf{L}(x), \mathsf{R}(x)]$ such that $L[j] \in \Sigma_p$ and $f_j \leq |\mathsf{str}(x)|$, and $\beta(x) = \mathsf{count}(\mathsf{L}(x), \mathsf{R}(x), -c, c)$, where $c = |\mathsf{sBWT}[i]|$. Consider a node u_k on π. Now, $\mathsf{zeroNode}(\ell_i)$ is below u_k iff $\beta(u_k) > \alpha(u_k)$. Therefore, $\mathsf{zeroNode}(\ell_i)$ is the shallowest node $u_{k'}$ on this path that satisfies $\beta(u_{k'}) \leq \alpha(u_{k'})$. Equipped with this knowledge, now we can binary search on π (using nodeDepth and levelAncestor operations) to find the exact location. The first question is to compute $\alpha(x)$, which is handled by Lemma 11. A normal binary search will have to consider n nodes on the path in the worst case. Lemma 12 shows how to reduce this to $\lceil \log \sigma \rceil$. Thus, the binary search has at most $\lceil \log \log \sigma \rceil$ steps, and the total time is $\log \log \sigma \times \lceil \frac{\log \sigma}{\log \log n} \rceil = O(\log \sigma)$, as required. ◀

The following are our helper lemmas for proving Lemma 10. The proofs are similar to those of Lemmas 4 and 5 in [9] respectively. We omit the proofs due to space limitations.

▶ **Lemma 11.** *We can compute $\alpha(x)$ in $O(1)$ time using an $O(n)$-bit data structure.*

▶ **Lemma 12.** *By using the Wavelet Tree over* sBWT *and an additional $O(n)$-bit data structure, in $O(\log \sigma)$ time, we can find an ancestor w_i of ℓ_i such that* $\mathsf{zeroDepth}(w_i) <$ $|\mathsf{sBWT}[i]|$ *and w_i is at most $\lceil \log \sigma \rceil$ nodes above* $\mathsf{zeroNode}(\ell_i)$.

4.4 Additional Components

Define f_j to be the smaller of f_j^+ and f_j^-, where $L[j] \in \Sigma_p$. Let $\mathsf{leafLeadChar}(j)$ be a boolean variable, which is 0 iff $f_j = (|\mathsf{str}(v)| + 1)$, where $v = \mathsf{parent}(\mathsf{zeroNode}(j))$.

For a node u, $\mathsf{pCount}(v)$ is the rightmost child w of v such that the first character on the edge (v, w) is a p-character. Since $\sum_v \mathsf{pCount}(v) = O(n)$, we can compute $\mathsf{pCount}(v)$

in $O(1)$ by using an $O(n)$-bit data structure. Let $\mathsf{fCount}^+(x)$ (resp. $\mathsf{fCount}^-(x)$) be the number of leaves ℓ_j in x's subtree, such that $\mathsf{sBWT}[j] \in [1, \sigma_p]$ (resp. $\mathsf{sBWT}[j] \in [-\sigma_p, -1]$) and $|\mathsf{str}(y)| + 2 \leq f_j \leq |\mathsf{str}(x)| + 1$, where $y = \mathsf{parent}(x)$. Additionally, for any leaf ℓ_j, assign $\mathsf{fCount}^+(\ell_j) = 1$ (resp. $\mathsf{fCount}^-(\ell_j) = 1$) if $f_j > |\mathsf{str}(\ell_j)|$ and $\mathsf{sBWT}[j] \in [1, \sigma_p]$ (resp. $\mathsf{sBWT}[j] < 0$). Let $\mathsf{fSum}^+(x)$ be the sum of $\mathsf{fCount}^+(y)$ of all nodes y which come before x in pre-order and are not ancestors of x. Let $\overleftarrow{\mathsf{fSum}}^-(x)$ be the sum of $\mathsf{fCount}^-(y)$ of all nodes y which come after $\mathsf{R}(x)$ in pre-order. Let $\mathsf{fAncestor}^+(x)$ be the number of leaves ℓ_j such that $\mathsf{pre\text{-}order}(\ell_j) < \mathsf{pre\text{-}order}(x)$, $f_j^+ = |\mathsf{str}(\mathsf{lca}(\ell_j, x))| + 1$, $\mathsf{sBWT}[j] \in [1, \sigma_p]$, and the first character on the path from $\mathsf{lca}(\ell_j, x)$ to x is an s-character.

We present the following important lemma (proof is similar to that of Lemma 3 in [9] and is omitted due space restriction).

▶ **Lemma 13.** *By using an $O(n)$-bit data structure, for any node x, we can compute the following in $O(1)$ time:* $\mathsf{fSum}^+(x)$*,* $\overleftarrow{\mathsf{fSum}}^-(x)$*, and* $\mathsf{fAncestor}^+(x)$*.*

4.5 Computing $\mathsf{sLF}(i)$ when $\mathsf{sBWT}[i] \in [\sigma_p + 1, \sigma]$

Using Lemma 9, $\mathsf{sLF}(i) > \mathsf{sLF}(j)$ iff either $j \in [1, n]$ and $\mathsf{sBWT}[j] < \mathsf{sBWT}[i]$, or $j \in [1, i-1]$ and $\mathsf{sBWT}[i] = \mathsf{sBWT}[j]$. Then,

$$\mathsf{sLF}(i) = 1 + \mathsf{count}(1, n, 1, \mathsf{sBWT}[i] - 1) + \mathsf{count}(1, i-1, \mathsf{sBWT}[i], \mathsf{sBWT}[i])$$

4.6 Computing $\mathsf{sLF}(i)$ when $\mathsf{sBWT}[i] \in [1, \sigma_p]$

We first assume that $\mathsf{zeroNode}(\ell_i)$ is defined, i.e., $f_i \leq |\mathsf{str}(\ell_i)|$. This can be easily checked in $O(1)$ time by maintaining a bit-vector. First find $z = \mathsf{zeroNode}(\ell_i)$ and locate the node $v = \mathsf{parent}(z)$. Depending on whether $\mathsf{leafLeadChar}(i) = 0$, or not, we find the ranges $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and if required $\mathcal{S}_4$ and $\mathcal{S}_5$, as illustrated in Figure 1.

Sub-case 1 ($f_i = |\mathsf{str}(v)| + 1$). Let w be the parent of v. We partition the leaves into 4 sets: (a) $\mathcal{S}_1$: leaves to the left of v's subtree, (b) $\mathcal{S}_2$: leaves in z's subtree, (c) $\mathcal{S}_3$: leaves to the right of v's subtree, and (d) $\mathcal{S}_4$ (resp. $\mathcal{S}_5$): leaves in v's subtree, and to the left (resp. right) of that of z's subtree. In case, v is the root node r, we take $w = r$; consequently, $\mathcal{S}_1 = \mathcal{S}_3 = \varnothing$.

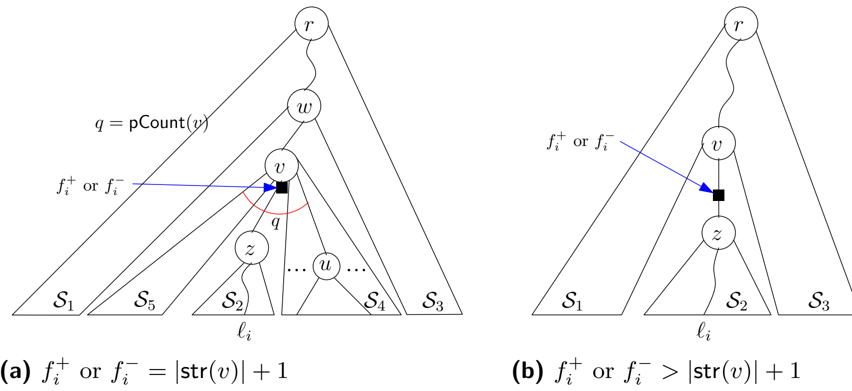
Sub-case 2 ($f_i > |\mathsf{str}(v)| + 1$). We partition the leaves into 3 sets: (a) $\mathcal{S}_1$ (resp. $\mathcal{S}_3$): leaves to the left (resp. right) of z's subtree. (b) $\mathcal{S}_2$: leaves in z's subtree.

Let $c = \mathsf{sBWT}[i]$. Define N_k to be the number of leaves ℓ_j in $\mathcal{S}_k$ such that $\mathsf{sLF}(j) \leq \mathsf{sLF}(i)$. Then, $\mathsf{sLF}(i) = N_1 + N_2 + N_3 + N_4 + N_5$ is computed as follows.

Computing N_1. For any $\ell_j \in \mathcal{S}_1$, $\mathsf{sLF}(j) < \mathsf{sLF}(i)$ iff one of the following holds: (1) $\mathsf{sBWT}[j] \in [1, \sigma_p]$ and $f_j^+ > 1 + |\mathsf{str}(\mathsf{lca}(\ell_i, \ell_j))|$, or (2) $\mathsf{sBWT}[j] \in [1, \sigma_p]$, $f_j^+ = 1 + |\mathsf{str}(\mathsf{lca}(\ell_i, \ell_j))|$, and the leading character on the path from $\mathsf{lca}(\ell_i, \ell_j)$ to ℓ_i is an s-character, or (3) $\mathsf{sBWT}[j] < 0$. Then,

$$N_1 = \begin{cases} \mathsf{fSum}^+(v) + \mathsf{fAncestor}^+(v) + \mathsf{count}(1, \mathsf{L}(v) - 1, -\sigma_p, -1), & \text{if } \mathsf{leafLeadChar}(i) = 0 \\ \mathsf{fSum}^+(z) + \mathsf{fAncestor}^+(z) + \mathsf{count}(1, \mathsf{L}(z) - 1, -\sigma_p, -1), & \text{otherwise} \end{cases}$$

Computing N_2. If $c > 0$, then for any leaf $\ell_j \in \mathcal{S}_2$, $\mathsf{sLF}(j) \leq \mathsf{sLF}(i)$ iff one of the following holds: (1) either $\mathsf{sBWT}[j] > c$ or $\mathsf{sBWT}[j] = c$ and $j \leq i$, (2) $\mathsf{sBWT}[j] < 0$. If $c < 0$, then



(a) f_i^+ or $f_i^- = |\mathsf{str}(v)| + 1$ **(b)** f_i^+ or $f_i^- > |\mathsf{str}(v)| + 1$

Figure 1 Illustration of various ranges when $T_{\mathsf{sSA}[i]}$ is preceded by a p-character

for any leaf $\ell_j \in \mathcal{S}_2$, $\mathsf{sLF}(j) \le \mathsf{sLF}(i)$ iff one of the following holds: (1) $-1 \ge \mathsf{sBWT}[j] > c$, (2) $\mathsf{sBWT}[j] = c$ and $j \le i$. Therefore,

$$N_2 = \begin{cases} \mathsf{count}(\mathsf{L}(z), \mathsf{R}(z), c+1, \sigma_p) + \mathsf{count}(\mathsf{L}(z), i, c, c) & \\ \qquad\qquad + \mathsf{count}(\mathsf{L}(z), \mathsf{R}(z), -\sigma_p, -1), & \text{if } c \in [1, \sigma_p] \\ \mathsf{count}(\mathsf{L}(z), \mathsf{R}(z), c+1, -1) + \mathsf{count}(\mathsf{L}(z), i, c, c), & \text{if } c < 0 \end{cases}$$

Computing N_3. For any leaf $\ell_j \in \mathcal{S}_3$, $\mathsf{sLF}(j) < \mathsf{sLF}(i)$ iff $\mathsf{sBWT}[j] < 0$ and $f_j^- \le 1 + |\mathsf{str}(\mathsf{lca}(z, \ell_j))|$. Therefore,

$$N_3 = \begin{cases} \mathsf{count}(\mathsf{R}(v)+1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(v), & \text{if } \mathsf{leafLeadChar}(i) = 0 \\ \mathsf{count}(\mathsf{R}(z)+1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(z), & \text{otherwise} \end{cases}$$

Computing N_4. Let u be the $\mathsf{pCount}(v)$th child of v. For any leaf $\ell_j \in \mathcal{S}_4$ such that $\mathsf{sBWT}[j] \in [1, \sigma_p]$, $f_j^+ \ne |\mathsf{str}(v)| + 1$; otherwise, the suffix j should not have deviated from i at the node v. Likewise, if $\mathsf{sBWT}[j] < 0$, then $f_j^- \ne |\mathsf{str}(v)| + 1$.
If $c > 0$, then N_4 is the number of leaves ℓ_j in $\mathcal{S}_4$ such that $j \le \mathsf{R}(u)$ and either (1) $\sigma_p \ge \mathsf{sBWT}[j] \ge c$, or (2) $\mathsf{sBWT}[j] < 0$. If $c < 0$, then N_4 is the number of leaves ℓ_j in $\mathcal{S}_4$ such that $j \le \mathsf{R}(u)$ and $-1 \ge \mathsf{sBWT}[j] > c$. Therefore,

$$N_4 = \begin{cases} \mathsf{count}(\mathsf{R}(z)+1, \mathsf{R}(u), c, \sigma_p) + \mathsf{count}(\mathsf{R}(z)+1, \mathsf{R}(u), -\sigma_p, -1), & \text{if } c \in [1, \sigma_p] \\ \mathsf{count}(\mathsf{R}(z)+1, \mathsf{R}(u), c+1, -1), & \text{if } c < 0 \end{cases}$$

Computing N_5. Note that for any leaf $\ell_j \in \mathcal{S}_5$ such that $\mathsf{sBWT}[j] \in [1, \sigma_p]$, $f_j^+ \ne |\mathsf{str}(v)|+1$; otherwise, the suffix j should not have deviated from i at the node v. Likewise, if $\mathsf{sBWT}[j] < 0$, then $f_j^- \ne |\mathsf{str}(v)| + 1$. Also, the leading character of the path from v to ℓ_j is negative.
If $c > 0$, then N_5 is the number of leaves ℓ_j in $\mathcal{S}_5$ that satisfies one of the following: (1) $\sigma_p \ge \mathsf{sBWT}[j] \ge c$, or (2) $\mathsf{sBWT}[j] < 0$. If $c < 0$, then N_5 is the number of leaves ℓ_j in $\mathcal{S}_5$ such that $-1 \ge \mathsf{sBWT}[j] > c$. Therefore,

$$N_5 = \begin{cases} \mathsf{count}(\mathsf{L}(v), \mathsf{L}(z)-1, c, \sigma_p) + \mathsf{count}(\mathsf{L}(v), \mathsf{L}(z)-1, -\sigma_p, -1), & \text{if } c \in [1, \sigma_p] \\ \mathsf{count}(\mathsf{L}(v), \mathsf{L}(z)-1, c+1, -1), & \text{if } c < 0 \end{cases}$$

Algorithm 1 computes $\mathsf{sLF}(i)$

1: $c \leftarrow \mathsf{sBWT}[i]$
2: **if** $(c > \sigma_p)$ **then**
3: $\quad \mathsf{sLF}(i) \leftarrow 1 + \mathsf{count}(1, n, -\sigma_p, c-1) + \mathsf{count}(1, i-1, c, c)$
4: **else**
5: $\quad z \leftarrow \mathsf{zeroNode}(\ell_i)$, $L_z \leftarrow \mathsf{L}(z)$, $R_z \leftarrow \mathsf{R}(z)$
6: $\quad$ **if** $(\mathsf{leafLeadChar}(i)$ is $0)$ **then**
7: $\quad\quad v \leftarrow \mathsf{parent}(z)$, $L_v \leftarrow \mathsf{L}(v)$, $R_v \leftarrow \mathsf{R}(v)$
8: $\quad\quad u \leftarrow \mathsf{child}(v, \mathsf{pCount}(v))$, $R_u \leftarrow \mathsf{R}(u)$
9: $\quad\quad N_1 \leftarrow \mathsf{fSum}^+(v) + \mathsf{count}(1, L_v - 1, -\sigma_p, -1)$
10: $\quad\quad N_3 \leftarrow \mathsf{count}(R_v + 1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(v)$
11: $\quad\quad$ **if** $(c > 0)$ **then**
12: $\quad\quad\quad N_4 \leftarrow \mathsf{count}(R_z + 1, R_u, c, \sigma_p) + \mathsf{count}(R_z + 1, R_u, -\sigma_p, -1)$
13: $\quad\quad\quad N_5 \leftarrow \mathsf{count}(L_v, L_z - 1, c, \sigma_p) + \mathsf{count}(L_v, L_z - 1, -\sigma_p, -1)$
14: $\quad\quad$ **else**
15: $\quad\quad\quad N_4 \leftarrow \mathsf{count}(R_z + 1, R_u, c + 1, -1)$
16: $\quad\quad\quad N_5 \leftarrow \mathsf{count}(L_v, L_z - 1, c + 1, -1)$
17: $\quad$ **else**
18: $\quad\quad N_1 \leftarrow \mathsf{fSum}^+(z) + \mathsf{count}(1, L_z - 1, -\sigma_p, -1)$
19: $\quad\quad N_3 \leftarrow \mathsf{count}(R_z + 1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(z)$
20: $\quad$ **if** $(c > 0)$ **then**
21: $\quad\quad N_2 \leftarrow \mathsf{count}(L_z, R_z, c + 1, \sigma_p) + \mathsf{count}(L_z, i, c, c) + \mathsf{count}(L_z, R_z, -\sigma_p, -1)$
22: $\quad$ **else**
23: $\quad\quad N_2 \leftarrow \mathsf{count}(L_z, R_z, c + 1, -1) + \mathsf{count}(L_z, i, c, c)$
24: $\quad \mathsf{sLF}(i) \leftarrow N_1 + N_2 + N_3 + N_4 + N_5$

Now, we arrive at the scenario when $\mathsf{zeroNode}(\ell_i)$ is not defined, i.e., $f_i > |\mathsf{str}(\ell_i)|$. Following the arguments in this section, it is easy to arrive at the following:

$$\mathsf{sLF}(i) = 1 + \mathsf{fSum}^+(\ell_i) + \mathsf{fAncestor}^+(\ell_i) + \mathsf{count}(1, i-1, -\sigma_p, -1) \\ + \mathsf{count}(i+1, n, -\sigma_p, -1) - \mathsf{fSum}^-(\ell_i), \text{ when } f_i > |\mathsf{str}(\ell_i)|$$

Summarizing the discussions in this section, we have proved the following.

▶ **Lemma 14.** *We can compute* $\mathsf{sLF}(i)$ *in* $O(\log \sigma)$ *time using the Wavelet Tree over* sBWT *and an additional* $O(n)$*-bit data structure.*

5 Finding Suffix Range via Backward Search

We use an adaptation of the backward search algorithm in the FM-index [6]. In particular, given a proper suffix Q of P, assume that we know the suffix range $[sp_1, ep_1]$ of $\Phi(Q)$. Our task is to find the suffix range $[sp_2, ep_2]$ of $\Phi(c \circ Q)$, where c is the character previous to Q in P. If c is a static character, then

$$sp_2 = 1 + \mathsf{count}(1, n, -\sigma_p, c-1) + \mathsf{count}(1, sp_1 - 1, c, c)$$
$$ep_2 = \mathsf{count}(1, n, -\sigma_p, c-1) + \mathsf{count}(1, ep_1, c, c)$$

Now, we consider the scenario when c is a p-character.

5.1 Case 1 (Neither c nor its complement appears in Q)

Let d be the number of distinct p-characters in Q, which can be computed in $O(1)$ time after pre-processing P in $O(|P| \log \sigma)$ time. Note that $\mathsf{sLF}(i) \in [sp_2, ep_2]$ iff $i \in [sp_1, ep_1]$, $\mathsf{sBWT}[i] \in [-\sigma_p, \sigma_p]$ and $f_i > |Q|$. Then,

$$(ep_2 - sp_2 + 1) = \mathsf{count}(sp_1, ep_1, d+1, \sigma_p) + \mathsf{count}(sp_1, ep_1, -\sigma_p, -d-1)$$

Let $u = \mathsf{lca}(\ell_{sp_1}, \ell_{ep_1})$. For any i, $\mathsf{sLF}(i) < sp_2$ iff (1) $i < sp_1$, $\mathsf{sBWT}[i] \in [1, \sigma_p]$ and $f_i^+ > 1 + |\mathsf{str}(\mathsf{lca}(u, \ell_i))|$, or (2) $i < sp_1$, $\mathsf{sBWT}[i] \in [1, \sigma_p]$, $f_i^+ = 1 + |\mathsf{str}(\mathsf{lca}(u, \ell_i))|$, and the leading character on the path from $\mathsf{lca}(u, \ell_i)$ to u is an s-character, or (3) $i \in [sp_1, ep_1]$, $\mathsf{sBWT}[i] < 0$ and $f_i^- \leq |Q|$, or (4) $i < sp_1$ and $\mathsf{sBWT}[i] < 0$, or (5) $i > ep_1$, $\mathsf{sBWT}[i] < 0$ and $f_i^- \leq 1 + |\mathsf{str}(\mathsf{lca}(u, \ell_i))|$. Therefore,

$$\begin{aligned} sp_2 = 1 &+ \mathsf{fSum}^+(u) + \mathsf{fAncestor}^+(u) + \mathsf{count}(sp_1, ep_1, -d, -1) \\ &+ \mathsf{count}(1, sp_1 - 1, -\sigma_p, -1) + \mathsf{count}(ep_1 + 1, n, -\sigma_p, -1) - \overleftarrow{\mathsf{fSum}}^-(u) \end{aligned}$$

5.2 Case 2 (c or its complement appears in Q)

Assume that the number of characters until the first occurrence of c (resp. c's complement) in Q is f^+ (resp. f^-). If f^+ or f^- does not exist, we take it to be $|Q| + 1$. Let d^+ and d^- be respectively the number of distinct p-characters in $Q[1, f^+]$ and $Q[1, f^-]$ respectively. After an initial $O(|P| \log \sigma)$ time pre-processing, d^+ and d^- can retrieved in $O(1)$ time.

Case when $f^+ < f^-$: Note that $\mathsf{sLF}(i) \in [sp_2, ep_2]$ iff $i \in [sp_1, ep_1]$, $\mathsf{sBWT}[i] \in [1, \sigma_p]$ and $f_i^+ = f^+$. Consider any $i, j \in [sp_1, ep_1]$ such that $i < j$, both $\mathsf{sLF}(i), \mathsf{sLF}(j) \in [sp_2, ep_2]$, and both $\mathsf{sBWT}[i], \mathsf{sBWT}[j] \in [1, \sigma_p]$. Now, $f_i^+ = f_j^+ = f^+$, and $\mathsf{sLF}(i) < \mathsf{sLF}(j)$. Therefore,

$$\begin{aligned} (ep_2 - sp_2 + 1) &= \mathsf{count}(sp_1, ep_1, d^+, d^+), \text{ and} \\ sp_2 &= \mathsf{sLF}(\min\{j \mid j \in [sp_1, ep_1] \text{ and } \mathsf{sBWT}[j] = d^+\}) \\ &= \mathsf{sLF}(\mathsf{select}(1 + \mathsf{rank}(sp_1 - 1, d^+), d^+)) \end{aligned}$$

Case when $f^+ > f^-$: Based on the above arguments, we can derive the following.

$$\begin{aligned} (ep_2 - sp_2 + 1) &= \mathsf{count}(sp_1, ep_1, -d^-, -d^-), \text{ and} \\ sp_2 &= \mathsf{sLF}(\min\{j \mid j \in [sp_1, ep_1] \text{ and } \mathsf{sBWT}[j] = -d^-\}) \\ &= \mathsf{sLF}(\mathsf{select}(1 + \mathsf{rank}(sp_1 - 1, d^-), d^-)) \end{aligned}$$

Thus, the suffix range of $\Phi(P)$ is computed in $O(|P| \log \sigma)$ time. Applying Lemmas 8 and 14, we arrive at Theorem 3.

References

1 Brenda S. Baker. A theory of parameterized pattern matching: algorithms and applications. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 71–80. ACM, 1993. doi:10.1145/167088.167115.

2 Djamal Belazzougui and Gonzalo Navarro. Alphabet-independent compressed text indexing. *ACM Trans. Algorithms*, 10(4):23:1–23:19, 2014. doi:10.1145/2635816.

3 M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation (now part of Hewlett-Packard, Palo Alto, CA), 1994.

4 Richard Cole and Ramesh Hariharan. Faster suffix tree construction with missing suffix links. *SIAM J. Comput.*, 33(1):26–42, 2003. doi:10.1137/S0097539701424465.

5 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Order-preserving incomplete suffix trees and order-preserving indexes. In Oren Kurland, Moshe Lewenstein, and Ely Porat, editors, *String Processing and Information Retrieval - 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings*, volume 8214 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2013. doi:10.1007/978-3-319-02432-5_13.

6 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

7 Johannes Fischer. Wee LCP. *Inf. Process. Lett.*, 110(8-9):317–320, 2010. doi:10.1016/j.ipl.2010.02.010.

8 Johannes Fischer and Volker Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, First International Symposium, ESCAPE 2007, Hangzhou, China, April 7-9, 2007, Revised Selected Papers*, volume 4614 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2007. doi:10.1007/978-3-540-74450-4_41.

9 Arnab Ganguly, Rahul Shah, and Sharma V Thankachan. pBWT: Achieving succinct data structures for parameterized pattern matching and related problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 397–407. Society for Industrial and Applied Mathematics, 2017.

10 Raffaele Giancarlo. The suffix of a square matrix, with applications. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas.*, pages 402–411. ACM/SIAM, 1993.

11 Alexander Golynski, Roberto Grossi, Ankur Gupta, Rajeev Raman, and S. Srinivasa Rao. On the size of succinct indices. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, volume 4698 of *Lecture Notes in Computer Science*, pages 371–382. Springer, 2007. doi:10.1007/978-3-540-75520-3_34.

12 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 841–850. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644250.

13 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

14 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997.

15 Dong Kyue Kim, Yoo Ah Kim, and Kunsoo Park. Generalizations of suffix arrays to multi-dimensional matrices. *Theor. Comput. Sci.*, 302(1-3):223–238, 2003. doi:10.1016/S0304-3975(02)00828-9.

16 Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4):52:1–52:47, 2013. doi:10.1145/2535933.

17 Gonzalo Navarro. Wavelet trees for all. *J. Discrete Algorithms*, 25:2–20, 2014. `doi:10.1016/j.jda.2013.07.004`.

18 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1), 2007. `doi:10.1145/1216370.1216372`.

19 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014. `doi:10.1145/2601073`.

20 Luís M. S. Russo, Gonzalo Navarro, and Arlindo L. Oliveira. Fully compressed suffix trees. *ACM Transactions on Algorithms*, 7(4):53, 2011. `doi:10.1145/2000807.2000821`.

21 Tetsuo Shibuya. Generalization of a suffix tree for RNA structural pattern matching. In Magnús M. Halldórsson, editor, *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*, pages 393–406. Springer, 2000. `doi:10.1007/3-540-44985-X_34`.

22 Tetsuo Shibuya. Geometric suffix tree: Indexing protein 3-d structures. *J. ACM*, 57(3):15:1–15:17, 2010. `doi:10.1145/1706591.1706595`.

23 Dekel Tsur. Top-k document retrieval in optimal space. *Inf. Process. Lett.*, 113(12):440–443, 2013. `doi:10.1016/j.ipl.2013.03.012`.