# Resolution of the Burrows-Wheeler Transform Conjecture

Dominik Kempa<sup>\*1</sup> and Tomasz Kociumaka<sup>†2,1</sup>

<sup>1</sup>University of California, Berkeley, USA kempa@berkeley.edu, kociumaka@berkeley.edu

<sup>2</sup>Bar-Ilan University, Ramat Gan, Israel

#### Abstract

The Burrows–Wheeler Transform (BWT) is an invertible text transformation that permutes symbols of a text according to the lexicographical order of its suffixes. BWT is the main component of popular lossless compression programs (such as bzip2) as well as recent powerful compressed indexes (such as r-index [Gagie et al., J. ACM, 2020]), central in modern bioinformatics. The compression ratio of BWT is quantified by the number r of equal-letter runs. Despite the practical significance of BWT, no non-trivial bound on the value of r is known. This is in contrast to nearly all other known compression methods, whose sizes have been shown to be either always within a polylog n factor (where n is the length of text) from z, the size of Lempel–Ziv (LZ77) parsing of the text, or significantly larger in the worst case (by a  $n^{\varepsilon}$  factor for  $\varepsilon > 0$ ).

In this paper, we show that  $r = \mathcal{O}(z \log^2 n)$  holds for every text. This result has numerous implications for text indexing and data compression; for example: (1) it proves that many results related to BWT automatically apply to methods based on LZ77, e.g., it is possible to obtain functionality of the suffix tree in  $\mathcal{O}(z \operatorname{polylog} n)$  space; (2) it shows that many text processing tasks can be solved in the optimal time assuming the text is compressible using LZ77 by a sufficiently large polylog n factor; (3) it implies the first non-trivial relation between the number of runs in the BWT of the text and its reverse.

In addition, we provide an  $\mathcal{O}(z \operatorname{polylog} n)$ -time algorithm converting the LZ77 parsing into the run-length compressed BWT. To achieve this, we develop a number of new data structures and techniques of independent interest. In particular, we introduce a notion of compressed string synchronizing sets (generalizing the recently introduced powerful technique of string synchronizing sets [STOC 2019]) and show how to efficiently construct them. Next, we propose a new variant of wavelet trees for sequences of long strings, establish a non-trivial bound on their size, and describe efficient construction algorithms. Finally, we describe new indexes that can be constructed directly from the LZ77-compressed text and efficiently support pattern matching queries on substrings of the text.

<sup>\*</sup>Supported by NSF grants no. 1652303 and 1934846, and an Alfred P. Sloan Fellowship grant.

<sup>&</sup>lt;sup>†</sup>Supported by ISF grants no. 1278/16 and 1926/19, by a BSF grant no. 2018364, and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (agreement no. 683064).

# 1 Introduction

Lossless data compression aims to exploit redundancy in the input data to represent it in a small space. Despite the abundance of compression methods, nearly every existing tool falls into one of the few general frameworks, among which the three most popular are: Lempel–Ziv compression (where the nominal and most commonly used is the LZ77 variant [84]), statistical compression (this includes, for example, context mixing [57], prediction by partial matching (PPM) [21], and dynamic Markov coding [22]), and Burrows–Wheeler transform (BWT) [17]. As seen in the Large Text Compression Benchmark [56], these three frameworks underlie most existing compressors.

One of the features that best differentiates these algorithms is whether they better remove the redundancy caused by skewed symbols frequencies or by repeated fragments. The idea in LZ77 (which underlies, for example, 7-zip [69] and gzip [31] compressors) is to partition the input text into long substrings, each having an earlier occurrence in the text. Every substring is then encoded as a pointer to the previous occurrence using a pair of integers. This method natively handles long repeated substrings and can achieve an exponential compression ratio given sufficiently repetitive input. Statistical compressors, on the other hand, are based on representing (predicting) symbols in the input based on their frequencies. This is formally captured by the notion of the *kth order empirical entropy*  $H_k(T)$  [23]. For any sufficiently long text T, symbol frequencies (taking context into account) in any power of T (the concatenation of several copies of T) do not change significantly [52, Lemma 2.6]. Therefore,  $|T^t|H_k(T^t) \approx t \cdot |T|H_k(T)$  for any t > 1, i.e., entropy is not sensitive to long repetitions, and hence statistical compressors are outperformed by LZ77, when the goal is to capture long repetitions [25, 30, 46, 52, 79].

The above analysis raises the question about the nature of compressibility of the Burrows– Wheeler transform. The compression of BWT-based compressors, such as  $\mathtt{bzip2}$  [77], is quantified by the number r of equal-letter blocks in the BWT. The clear picture described above no longer applies to the measure r. On one hand, Manzini [61] proved that r can be upper-bounded in terms of the kth order empirical entropy of the input string. On the other hand, already in 2008, Sirén et al. [79] observed that BWT achieves excellent compression (superior to statistical methods) on highly repetitive collections and provided probabilistic analysis exhibiting cases when ris small. Yet, after more than a decade, no upper bound on r in terms of z was discovered.

This lack of understanding is particularly frustrating due to numerous applications of BWT in the field of bioinformatics and compressed computation. One of the most successful applications of BWT is in *compressed indexing*, which aims to store a compressed string, simultaneously supporting various queries (such as random access, pattern matching, or even suffix array queries) on the uncompressed version. While classical (uncompressed) indexes, such as suffix trees [83] and suffix arrays [59], have been successful in many applications, they are not suitable for storing and searching big highly repetitive databases. Such datasets are virtually impossible to search without preprocessing: Github databases, for example, take more than 20 terabytes, and the recently finished 100000 Human Genome Project [34] produced 75 terabytes of DNA [63]. These databases are, however, highly compressible: Github averages 20 versions per project [63], and two human genomes are 99.9% similar [75]. This area has witnessed a remarkable surge of interest in recent years [5, 10, 11, 12, 20, 26, 28, 44, 64, 74, 78]. BWT-based indexes, such as *r*-index [30], are among the most powerful [28], and their space usage is up to  $\mathcal{O}(\text{polylog }n)$ factors away from the value *r*. For a comprehensive overview of this field, we refer the reader to a survey by Navarro [63].

In addition to text indexing, BWT has many applications in compressed computation. For example, BWT is the main component of the popular read aligners such as Bowtie [53], BWA [54], and Soap2 [55]. Modern textbooks spend dozens of pages describing applications of BWT [1, 58, 62, 66]. The richness of these applications has even spawned a dedicated seminar [27]. Given the

importance and practical significance of BWT, one of the biggest open problems that emerged in the field of lossless data compression and compressed computation asks:

What is the upper bound on the output size of the Burrows-Wheeler transform?

With the exception of BWT, essentially every other known compression method has been proven [29, 46] to produce output whose size is always within an  $\mathcal{O}(\text{polylog } n)$  factor from z, the output size of the LZ77 algorithm (e.g., grammar compression [18, 76], collage systems [47], and macro schemes [81]), or larger by a polynomial factor ( $n^{\varepsilon}$  for some  $\varepsilon > 0$ ) in the worst case (e.g., LZ78 [85], compressed word graphs (CDAWGs) [13]).<sup>1</sup> BWT is known to never compress much better than LZ77, i.e.,  $z = \mathcal{O}(r \log n)$  [29]. The opposite relation (that  $r = \mathcal{O}(z \operatorname{polylog} n)$ ) was generally conjectured to be false. For example, after presenting how to support suffix array and suffix tree queries in  $\mathcal{O}(r \operatorname{polylog} n)$  space, Gagie et al. [28] speculate that "(...) it seems unlikely that one can provide suffix array or tree functionality within space related to g, z, or  $\gamma$ , since these measures are not related to the structure of the suffix array: this is likely to be a specific advantage of measure r".

**Our Contribution** We prove that  $r = O(z \log^2 n)$  holds for all strings, resolving the BWT conjecture in the more surprising way than anticipated, and solving an open problem by Prezza [73] and Gagie et al. [28, 29]. This result alone has multiple implications for indexing and compression:

- 1. It is possible to support suffix array and suffix tree functionality in  $\mathcal{O}(z \operatorname{polylog} n)$  space [30].
- 2. It was shown in [44] that many string processing tasks (including BWT and LZ77 construction) can be solved in  $\mathcal{O}(n/\log_{\sigma} n + r \operatorname{polylog} n)$  time (where  $\sigma$  is the alphabet size), i.e., if the text is sufficiently compressible by BWT (formally, when  $n/r = \Omega(\operatorname{polylog} n)$ ), these tasks can be solved in optimal time (which is unlikely to be possible for general texts [45]). Our result loosens this assumption to  $n/z = \Omega(\operatorname{polylog} n)$ .
- 3. Until now, methods based on the Burrows–Wheeler transform were thought to be neither statistical nor dictionary (LZ-like) compression algorithms [28, 79]. Our result challenges the notion that the BWT forms its own compression type: In view of our bound, BWT is much closer to LZ compressors than was previously thought.

Our slightly stronger bound  $r = \mathcal{O}(\delta \log^2 n)$ , where  $\delta \leq z$  is a symmetric (insensitive to string reversal) repetitiveness measure recently introduced in [49], further shows that:

4. The number  $\bar{r}$  of BWT runs in the reverse of the text satisfies  $\bar{r} = \mathcal{O}(r \log^2 n)$ , which is the first non-trivial bound in terms of r. This result is of practical importance due to many algorithms whose efficiency depends on  $\bar{r}$  [6, 8, 9, 67, 68, 71, 72].

After proving  $r = \mathcal{O}(z \log^2 n)$  and  $r = \mathcal{O}(\delta \log^2 n)$ , by a tighter analysis, we obtain  $r = \mathcal{O}(z \log z \max(1, \log \frac{n}{z \log z}))$  and  $r = \mathcal{O}(\delta \log \delta \max(1, \frac{n}{\delta \log \delta}))$ , and we show that the latter is asymptotically tight for the full spectrum of values of n and  $\delta$ . As a side-result, we obtain a tight upper bound  $\mathcal{O}(n \log \delta)$  on the sum of irreducible LCP values. This improves upon the previously known bound  $\mathcal{O}(n \log r)$  [41].

We then describe an  $\mathcal{O}(z \log^8 n)$ -time algorithm converting the LZ77 parsing into run-length compressed BWT (the polylog *n* factor has not been optimized). This offers up to exponential speedup over the previously fastest space-efficient algorithms, which need  $\Omega(n \log z)$  time [68, 71]. To achieve this, we develop new data structures and techniques of independent interest. In particular, we introduce a notion of *compressed string synchronizing sets*, generalizing the powerful

<sup>&</sup>lt;sup>1</sup>The choice for LZ77 as a representative in this class follows from the fact that most of the other methods are NP-hard to optimize [18, 32], while LZ77 admits a simple linear-time compression algorithm (see, e.g., [42]).

technique introduced in [45]. We also describe a new variant of wavelet trees [35], designed to work for sequences of long strings. Finally, we describe new indexes that can be built directly from the LZ77-compressed text and support fast pattern matching queries on text substrings.

**Organization of the Paper** Section 2 introduces the basic notation. We present our upper and lower bounds in Sections 3 and 4, respectively. Finally, in Section 5, we develop our algorithm converting LZ77 to run-length compressed BWT, with the description of our LZ77-based text indexes deferred to Section 6.

#### $\mathbf{2}$ Preliminaries

For any string S, we write  $S[i \dots j]$ , where  $1 \le i, j \le |S|$ , to denote a substring of S. If i > j, we assume  $S[i \dots j]$  to be the empty string  $\varepsilon$ . By  $\overline{S}$  we denote the *reverse* of S.

An integer  $p \in [1 \dots |S|]$  is a *period* of a string S if S[i] = S[i + p] holds for every  $i \in [1 \dots |S|]$ [1..|S|-p]. The classic periodicity lemma [24] states that if a string S has periods p, q such that  $p + q - \gcd(p, q) \le |S|$ , then  $\gcd(p, q)$  is also a period of S.

The shortest period of S is denoted as per(S). A string S is called *periodic* if  $per(S) \leq \frac{1}{2}|S|$ . The following fact is a folklore consequence of the periodicity lemma.

**Fact 2.1** (see [4, Fact 1]). Any two distinct periodic strings of the same length differ on at least two positions.

i

Throughout the paper, we consider a string (text) T[1..n] of  $n \ge 1$  symbols from an ordered alphabet  $\Sigma$  of size  $\sigma$ . We assume T[n] =\$, where  $\$ \in \Sigma$  is the lexicographically smallest symbol in  $\Sigma$ , and that \$ does not occur anywhere else in T.

The suffix array [59] of T is an array SA[1..n]containing a permutation of the integers  $[1 \dots n]$ such that  $T[SA[1] \dots n] \prec T[SA[2] \dots n] \prec \dots \prec$  $T[SA[n] \dots n]$ , where  $\prec$  denotes the lexicographic order. The closely related Burrows-Wheeler transform [17] BWT[1...n] of T is defined by BWT[i] = T[SA[i]-1] if SA[i] > 1 and BWT[i] = T[n] otherwise. The BWT is invertible: given BWT[1..n], the text T can be restored in  $\mathcal{O}(n)$  time.

For any string  $S = c_1^{\ell_1} c_2^{\ell_2} \cdots c_h^{\ell_h}$ , where  $c_i \in \Sigma$ and  $\ell_i > 0$  for  $i \in [1..h]$ , and  $c_i \neq c_{i+1}$  for  $i \in [1..h)$ , we define the run-length encoding of S as a sequence  $\operatorname{RL}(S) = ((c_1, \lambda_1), \dots, (c_h, \lambda_h)),$ where  $\lambda_i = \ell_1 + \ldots + \ell_i$  for  $i \in [1 \ldots h]$ . Throughout, we let r = |RL(BWT)| denote the number of runs in the BWT of T. E.g., for the text T = bbabaabababababababababa in Fig. 1, we haveBWT =  $a^1b^6a^1b^2a^6b^1a^2$ , and hence r = 8.

LCP[i]	SA[i]	BWT[	i T	[SA]	[i].	$\cdot n$
--------	-------	------	-----	------	------	-----------

1	<u>0</u>	20	a	\$
2	<u>0</u>	19	b	a\$
3	1	14	b	aababa\$
4	6	5	b	aababababaababa\$
5	1	17	b	aba\$
6	3	12	b	abaababa\$
7	8	3	b	abaababababaababa\$
8	<u>3</u>	15	a	ababa\$
9	$\underline{5}$	10	b	ababaababa\$
10	5	8	b	abababaababa\$
11	$\overline{7}$	6	a	ababababaababa\$
12	0	18	a	ba\$
13	2	13	a	baababa\$
14	7	4	a	baababababaababa\$
15	2	16	a	baba\$
16	4	11	a	babaababa\$
17	9	2	b	babaababababaababa\$
18	<b>4</b>	9	a	bababaababa\$
19	6	7	a	babababaababa\$
20	<u>1</u>	1	\$	bbabaababababababa\$

Figure 1: A list of lexicographically sorted suffixes of the string T = bbabaabababababababababaalong with the BWT, SA, and LCP tables. The irreducible LCP values are bold and underlined.

By  $lcp(S_1, S_2)$  we denote the length of the longest common prefix of strings  $S_1$  and  $S_2$ . For  $j_1, j_2 \in [1 ... n]$ , we let LCE $(j_1, j_2) = lcp(T[j_1 ... n], T[j_2 ... n])$ . The LCP array (see [59, 43]),  $LCP[1 \dots n]$ , is defined as LCP[i] = LCE(SA[i], SA[i-1]) for  $i \in [2 \dots n]$  and LCP[1] = 0. We say that the value LCP[i] is reducible if BWT[i] = BWT[i-1] and irreducible otherwise (including i = 1). Note that there are exactly r irreducible LCP values.

**Theorem 2.2** (Kärkkäinen et al. [41]). The sum of all irreducible LCP values is at most  $n \log r$ .

We say that a fragment  $T[i \dots i + \ell)$  is a previous factor if it has an earlier occurrence in T, i.e.,  $\text{LCE}(i, i') \ge \ell$  holds for some  $i' \in [1 \dots i)$ . An LZ77-like factorization of T is a factorization  $T = F_1 \cdots F_f$  into non-empty phrases such that each phrase  $F_j$  with  $|F_j| > 1$  is a previous factor. In the underlying LZ77-like representation, every phrase  $F_j = T[i \dots i + \ell)$  that is a previous factor is encoded as  $(i', \ell)$ , where  $i' \in [1 \dots i)$  satisfies  $\text{LCE}(i, i') \ge \ell$  (and is chosen arbitrarily in case of multiple possibilities); if  $F_j = T[i]$  is not a previous factor, it is encoded as (T[i], 0).

The following relation between z and r is known.

**Theorem 2.3** (Gagie et al. [29]). Every string of length n satisfies  $z = O(r \log n)$ .

# 3 Upper Bounds

# 3.1 Basic Upper Bound

To illustrate the main idea of our proof technique, we first prove the upper bound in its simplest form  $r = \mathcal{O}(z \log^2 n)$ . The following lemma stands at the heart of our proof.

**Lemma 3.1.** For every  $\ell \in [1 ... n]$ , the number of irreducible LCP values in  $[\ell ... 2\ell)$  is  $\mathcal{O}(z \log n)$ .

*Proof.* Let  $T^{\infty}$  be an infinite string defined so that  $T^{\infty}[i] = T[1 + (i - 1) \mod n]$  for  $i \in \mathbb{Z}$ ; in particular,  $T^{\infty}[1 \dots n] = T[1 \dots n]$ . Due to T[n] = \$, we have  $T^{\infty}[SA[1] \dots] \prec \cdots \prec T^{\infty}[SA[n] \dots]$  and  $BWT[i] = T^{\infty}[SA[i] - 1]$  for  $i \in [1 \dots n]$ .

Denote  $S_m = \{S \in \Sigma^m : S \text{ is a substring of } T^\infty\}$  for  $m \ge 1$ . Observe that  $|S_m| \le mz$  since every length-*m* substring of  $T^\infty$  has an occurrence crossing or beginning at a phrase boundary of the LZ77 parsing of *T*. This includes substrings overlapping two copies of *T*, which cross the boundary between the last and the first phrase.

The idea of the proof is as follows. With each irreducible value  $\text{LCP}[i] \in [\ell ... 2\ell)$ , we associate a cost of  $\ell$  units, which are charged to individual characters of strings in  $S_{3\ell}$ . We then show that each of the strings in  $S_{3\ell}$  is charged at most  $2 \log n$  times. The number of irreducible LCP values in  $[\ell ... 2\ell)$  equals  $\frac{1}{\ell}$  times the total cost, which is at most

$$|\mathcal{S}_{3\ell}| \cdot 2\log n \le 6\ell z\log n$$

To devise the announced assignment of cost to characters of strings in  $S_{3\ell}$ , consider the trie  $\mathcal{T}$  of all reversed strings in  $S_{\ell}$  (see Fig. 2 for an example). By  $v_X$  denote the node of  $\mathcal{T}$  whose path from the root of  $\mathcal{T}$  is labelled by a string X.

Let  $LCP[i] \in [\ell ... 2\ell)$  be an irreducible LCP value; note that i > 1 due to  $LCP[i] \ge \ell > 0$ . Let  $j_0 = SA[i-1]$  and  $j_1 = SA[i]$  so that LCP[i] = $LCE(j_0, j_1)$ . Since LCP[i] is irreducible, we have



 $T^{\infty}[j_0 - 1] = \text{BWT}[i - 1] \neq \text{BWT}[i] = T^{\infty}[j_1 - 1]$ . For  $k \in [1 \dots \ell]$ , the *k*th unit of the cost associated with LCP[*i*] is charged to the *k*th character  $(T^{\infty}[j_t - 1])$  of the string  $T^{\infty}[j_t - k \dots j_t - k + 3\ell) \in S_{3\ell}$ , where  $t \in \{0, 1\}$  is such that the subtree of  $\mathcal{T}$  rooted at  $v_{\overline{T^{\infty}[j_1-1\dots j_t-k+\ell)}}$  contains less leaves than the subtree rooted at  $v_{\overline{T^{\infty}[j_1-1\dots j_1-t-k+\ell)}}$  (we choose t = 0 in case of ties).

Note that at most log *n* characters of each  $S \in S_{3\ell}$  can be charged during the above procedure: whenever S[k], with  $k \in [1 \dots \ell]$ , is charged, the subtree of  $\mathcal{T}$  rooted at  $v_{\overline{S[k+1\dots \ell]}}$  has at least twice as many leaves as the subtree rooted at  $v_{\overline{S[k\dots \ell]}}$ , and this can happen for at most log  $|S_{\ell}| \leq \log n$ nodes  $v_{\overline{S[k\dots \ell]}}$  on the path from the root of  $\mathcal{T}$  to the leaf  $v_{\overline{S[1\dots \ell]}}$ .

It remains to show that, for every  $S \in S_{3\ell}$ , a single position S[k], with  $k \in [1 \dots \ell]$ , can be charged at most twice. For this, observe that the characters charged for a single irreducible value LCP[i] are at different positions (of strings in  $S_{3\ell}$ ). Hence, to analyze the total charge assigned to S[k], we only need to bound the number of possible candidate positions i. Let  $[b \dots e]$  be the set of indices i' such that  $T^{\infty}[SA[i'] \dots]$  starts with  $S[k + 1 \dots 3\ell]$ . In the above procedure, if a character S[k] is charged a unit of cost corresponding to LCP[i], then  $S[k + 1 \dots 3\ell]$  is a prefix of either  $T^{\infty}[SA[i-1] \dots] = T^{\infty}[j_0 \dots]$  or  $T^{\infty}[SA[i] \dots] = T^{\infty}[j_1 \dots]$ . Hence,  $\{i - 1, i\} \cap [b \dots e] \neq \emptyset$ . At the same time,  $LCE(SA[i-1], SA[i]) < 2\ell$  and all strings  $T^{\infty}[SA[i'] \dots]$  with  $i' \in [b \dots e]$  share a common prefix  $S[k + 1 \dots 3\ell]$  of length  $3\ell - k \ge 2\ell$ . Consequently, i = b or i = e + 1.

**Theorem 3.2.** Every string of length n satisfies  $r = O(z \log^2 n)$ .

*Proof.* Recall that r is the total number of irreducible LCP values. Thus, the claim follows by applying Lemma 3.1 for  $\ell_i = 2^i$ , with  $i \in [0 \dots \lfloor \log n \rfloor]$ , and observing that the number of LCP values 0 is exactly  $\sigma \leq z$ .

# 3.2 Tighter Upper Bound

To obtain a tighter bound, we refine the ideas from Section 3.1, starting with a counterpart of Lemma 3.1.

**Lemma 3.3.** For every  $\ell \in [1..n]$ , the number of irreducible LCP values in  $[\ell ..2\ell)$  is  $\mathcal{O}(z \log z)$ .

*Proof.* The proof follows closely that of Lemma 3.1. However, with each irreducible value  $\text{LCP}[i] \in [\ell ... 2\ell)$ , we associate cost  $\lceil \frac{1}{2}\ell \rceil$  rather  $\ell$ . We then show that each of the strings in  $S_{3\ell}$  is charged at most  $2 \cdot (3 + \log z)$  times (rather than  $2 \log n$  times). Then, the number of irreducible LCP values in the range  $\lceil \ell ... 2\ell \rangle$  does not exceed  $\frac{2}{\ell}$  times the total cost, which is bounded by

$$|\mathcal{S}_{3\ell}| \cdot 2 \cdot (3 + \log z) \le 6\ell z (3 + \log z).$$

Recall the trie  $\mathcal{T}$  of all reversed strings in  $\mathcal{S}_{\ell}$ . For a node v of  $\mathcal{T}$ , by size(v) we denote the number of leaves in the subtree of  $\mathcal{T}$  rooted in v. An edge connecting  $v \neq \operatorname{root}(\mathcal{T})$  to its parent in  $\mathcal{T}$  is called *light* if v has a sibling v' satisfying size $(v') \geq \operatorname{size}(v)$  (see Fig. 2). In the proof of Lemma 3.1, we observed that the characters S[k] of  $S \in \mathcal{S}_{3\ell}$  that can be charged correspond to light edges on the path from the root of  $\mathcal{T}$  to the leaf  $v_{\overline{S[1..\ell]}}$ : whenever S[k], with  $k \in [1..\ell]$ , is charged, the edge connecting  $v_{\overline{S[k..\ell]}}$  to its parent  $v_{\overline{S[k+1..\ell]}}$  is light. We then noted that there are at most  $\log |\mathcal{S}_{\ell}| \leq \log n$  light edges on each root-to-leaf path in  $\mathcal{T}$ . Here, we perform the same assignment of cost to the characters of strings in  $\mathcal{S}_{3\ell}$  as in Lemma 3.1, but only for units  $k \in [1..\lfloor \frac{1}{2}\ell \rceil]$ . This implies that only characters S[k] of  $S \in \mathcal{S}_{3\ell}$  with  $k \leq \lfloor \frac{1}{2}\ell \rfloor$  are charged. It remains to show that any root-to-leaf path in  $\mathcal{T}$  contains at most  $3 + \log z$  light edges between a node at depth at least  $\lfloor \frac{1}{2}\ell \rfloor$  and its child.

Consider a light edge from a node v to its parent u at depth at least  $\lfloor \frac{1}{2}\ell \rfloor$ . Let v' be a sibling of v satisfying size $(v') \geq \text{size}(v)$ , and let  $S_v, S_{v'}$  be the labels of the paths from the root to v

and v', respectively. These labels differ on the last position only so, by Fact 2.1, they cannot be both periodic. Let  $\tilde{v} \in \{v, v'\}$  be such that  $S_{\tilde{v}}$  is not periodic, and let  $\tilde{m} = \text{size}(\tilde{v})$ .

Consider the set S of length- $\ell$  strings corresponding to the leaves in the subtree of  $\mathcal{T}$  rooted at  $\tilde{v}$  (i.e., the labels of the root-to-leaf paths passing through  $\tilde{v}$ ). Define  $\overline{S} := \{\overline{P} : P \in S\}$  and note that  $\overline{S} \subseteq S_{\ell}$  because  $\mathcal{T}$  is the trie of *reversed* strings from  $S_{\ell}$ . Let  $e_1 < \cdots < e_{\tilde{m}}$  denote the ending positions of the leftmost occurrences in  $T^{\infty}[1..)$  of strings in  $\overline{S}$ . By definition, we have an occurrence of  $\overline{S_{\tilde{v}}}$  ending in  $T^{\infty}$  at every position  $e_i$  with  $i \in [1..\tilde{m}]$ . Now,  $\operatorname{per}(S_{\tilde{v}}) > \frac{1}{2}|S_{\tilde{v}}| \geq \frac{1}{4}\ell$ implies that  $e_{i+1} - e_i > \frac{1}{4}\ell$  for every  $i \in [1..\tilde{m} - 1]$  (otherwise, the two close occurrences of  $\overline{S_{\tilde{v}}}$  would yield  $\operatorname{per}(\overline{S_{\tilde{v}}}) = \operatorname{per}(S_{\tilde{v}}) \leq \frac{1}{4}\ell$ ). Consequently, at least  $\frac{1}{4}|S| = \frac{1}{4}\tilde{m}$  length- $\ell$  substrings of  $T^{\infty}[1..)$  have disjoint leftmost occurrences. Since each leftmost occurrence crosses or begins at a phrase boundary of the LZ77 parsing of T, we conclude that  $z \geq \frac{1}{4}\tilde{m}$ , and therefore  $\operatorname{size}(v) \leq \operatorname{size}(\tilde{v}) = \tilde{m} \leq 4z$ .

The reasoning above shows that once a root-to-leaf path encounters a light edge connecting a node u at depth at least  $\lfloor \frac{1}{2}\ell \rfloor$  to its child v, we have  $\operatorname{size}(v) \leq 4z$ . The number of the remaining light edges on the path is at most  $\log(\operatorname{size}(v)) \leq 2 + \log z$  by the standard bound applied to the subtree of  $\mathcal{T}$  rooted at v.

**Theorem 3.4.** Every string of length n satisfies  $r = \mathcal{O}(z \log z \max(1, \log \frac{n}{z \log z}))$ .

*Proof.* To obtain tighter bounds on the number of irreducible LCP values in  $[\ell ... 2\ell)$ , we consider three cases:

- 1.  $\ell \leq \log z$ . We repeat the proof of Lemma 3.1, except that we observe that the number of light edges on each root-to-leaf path in  $\mathcal{T}$  is bounded by  $\ell$ . Thus, the number of irreducible LCP values in  $[\ell . . 2\ell)$  is  $\mathcal{O}(z\ell)$ .
- 2.  $\log z < \ell \leq \frac{n}{z}$ . We use the bound  $\mathcal{O}(z \log z)$  of Lemma 3.3.
- 3.  $\frac{n}{z} < \ell$ . We repeat the proof of Lemma 3.3, except that we observe that  $|\mathcal{S}_{3\ell}| \le n$ . Thus, the number of irreducible LCP values in  $[\ell \dots 2\ell)$  is  $\mathcal{O}(\frac{n \log z}{\ell})$ .

The above upper bounds, applied for every  $\ell = 2^i$  with  $i \in [0 \dots \log n]$ , yield

$$r \leq \sigma + \sum_{i=0}^{\lfloor \log n \rfloor} \left| \left\{ j \in [2 \dots n] : \operatorname{BWT}[j-1] \neq \operatorname{BWT}[j] \text{ and } \operatorname{LCP}[j] \in \left[2^i \dots 2^{i+1}\right) \right\} \right|$$
$$= \sigma + \sum_{i=0}^{\lfloor \log \log z \rfloor} \mathcal{O}\left(z2^i\right) + \sum_{i=\lfloor \log \log z \rfloor+1}^{\lfloor \log \frac{n}{z} \rfloor} \mathcal{O}\left(z\log z\right) + \sum_{i=\lfloor \log \frac{n}{z} \rfloor+1}^{\lfloor \log n \rfloor} \mathcal{O}\left(\frac{n\log z}{2^i}\right)$$
$$= \sigma + \mathcal{O}\left(z\log z\right) + \mathcal{O}\left(z\log z\max\left(1,\log\frac{n}{z\log z}\right)\right) + \mathcal{O}\left(z\log z\right)$$
$$= \mathcal{O}\left(z\log z\max\left(1,\log\frac{n}{z\log z}\right)\right).$$

# **3.3** Upper Bound in Terms of $\delta$

Let  $\delta = \max_{m=1}^{n} \frac{1}{m} |\mathcal{S}_{m}|$  denote the substring complexity of T [49]. Note that letting  $\delta = \sup_{m=1}^{\infty} \frac{1}{m} |\mathcal{S}_{m}|$  is equivalent because  $|\mathcal{S}_{m}| \leq n$  holds for  $m \geq 1$ , which implies  $\frac{1}{m} |\mathcal{S}_{m}| \leq 1 \leq |\mathcal{S}_{1}|$  for  $m \geq n$ . We start by noting that  $\delta \leq z$  since  $|\mathcal{S}_{m}| \leq mz$  holds for every  $m \geq 1$ , as observed in the proof of Lemma 3.1. Furthermore,  $|\mathcal{S}_{m}| \leq m\delta$  holds by definition of  $\delta$ , so  $\delta$  can replace z in the proof of Lemma 3.1.

To adapt the proof Lemma 3.3, we need to generalize the observation that at most z substrings from  $S_{\ell}$  may have disjoint leftmost occurrences in  $T^{\infty}[1..)$ . This observation is easy since the LZ77 parsing naturally yields a set of z positions (phrase boundaries) in T. The substring complexity  $\delta$  does not provide such structure, but as the lemma below implies, we can replace z by  $3\delta$  in the aforementioned observation. The proof of Lemma 3.5 is a straightforward modification of the argument used in [49, Lemma 6]. For completeness, below we write down the full reasoning, with technical details tailored to our notation (e.g.,  $S_{\ell}$  defined in terms of  $T^{\infty}$  rather than T).

**Lemma 3.5** (based on [49, Lemma 6]). For any positive integer  $\ell$ , the total number of positions in  $T^{\infty}[1..)$  covered by the leftmost occurrences of strings from  $S_{\ell}$  is at most  $3\delta\ell$ .

Proof. Let C denote the set of positions in  $T^{\infty}[1..)$  covered by the leftmost occurrences of strings from  $S_{\ell}$ , and let  $C' = C \setminus [1..\ell)$ . For any  $i \in C'$  denote  $S_i = T^{\infty}[i-\ell+1..i+\ell]$ , and let  $S = \{S_i : i \in C'\} \subseteq S_{2\ell}$ . We will show that |S| = |C'|. Let  $i \in C'$ . First, observe that, due to  $i \geq \ell$ , the fragment  $S_i$  is entirely contained in  $T^{\infty}[1..)$ . Furthermore, by definition,  $S_i$  contains the leftmost occurrence of some  $S \in S_{\ell}$ . Thus, this occurrence of  $S_i$  in  $T^{\infty}[1..)$  must also be the leftmost one in  $T^{\infty}[1..)$ . Consequently, the substrings  $S_i$  for  $i \in C'$  are distinct.

We have thus shown that  $|C'| = |S| \le |S_{2\ell}|$ . Since  $|S_{2\ell}| \le 2\delta\ell$  holds by definition of  $\delta$ , we obtain  $|C| < |C'| + \ell \le |S_{2\ell}| + \ell \le (2\delta + 1)\ell \le 3\delta\ell$ .

**Lemma 3.6.** For every  $\ell \in [1..n]$ , the number of irreducible LCP values in  $[\ell .. 2\ell)$  is  $\mathcal{O}(\delta \log \delta)$ .

*Proof.* Compared to the proof of Lemma 3.3, we use the bound  $|S_{3\ell}| \leq 3\ell\delta$  instead of  $|S_{3\ell}| \leq 3\ell z$ . The only other modification needed is that, for every light edge connecting a node u of  $\mathcal{T}$  at depth at least  $\lfloor \frac{1}{2}\ell \rfloor$  to its child v, we need to prove size $(v) = \mathcal{O}(\delta)$ .

Let  $\widetilde{m} \geq \text{size}(v)$  be defined as in the proof of Lemma 3.3. Recall that we have identified at least  $\frac{\widetilde{m}}{4}$  strings in  $S_{\ell}$  whose leftmost occurrences in  $T^{\infty}[1..)$  are disjoint. By Lemma 3.5, there are at most  $3\delta$  such substrings. Thus,  $\text{size}(v) \leq \widetilde{m} \leq 12\delta$ .

By replacing the thresholds  $\log z$  and  $\frac{n}{z}$  with  $\log \delta$  and  $\frac{n}{\delta}$ , respectively, in the proof of Theorem 3.4, we immediately obtain a bound in terms of  $\delta$ .

**Theorem 3.7.** Every string of length n satisfies  $r = \mathcal{O}(\delta \log \delta \max(1, \log \frac{n}{\delta \log \delta}))$ .

Note that the trivial upper bound  $r = \mathcal{O}(n)$  is tighter if  $\delta \log \delta > n$ . In Section 4, we show that a combination of these two upper bounds is asymptotically optimal. For this, we construct tight examples in which the values  $\delta$  cover the whole spectrum between  $\mathcal{O}(1)$  and  $\Omega(n)$ .

# 3.4 Further Upper Bounds

By combining Theorem 3.7 with known properties of the substring complexity  $\delta$ , we obtain the first bound relating the number of BWT runs in the string and its reverse. No such bounds (even polynomial in  $r \log n$ ) were known before.

**Corollary 3.8.** If r and  $\bar{r}$  denote the number of runs in the BWT of a length-n text and its reverse, respectively, then  $\bar{r} = \mathcal{O}(r \log r \max(1, \frac{n}{r \log r})).$ 

*Proof.* Since the value of  $\delta$  is the same for the text and its reverse, Theorem 3.7 yields  $\bar{r} = \mathcal{O}(\delta \log \delta \max(1, \log \frac{n}{\delta \log \delta}))$ . Combining [46, Theorem 3.9] and [49, Lemma 2] gives  $\delta \leq r$ . Consequently, we obtain  $\bar{r} = \mathcal{O}(r \log r \max(1, \frac{n}{r \log r}))$ .

Our technique also lets us strengthen the bound of Theorem 2.2 on the sum of irreducible LCP values.

#### **Theorem 3.9.** For every string of length n, the sum of all irreducible LCP values is $\mathcal{O}(n \log \delta)$ .

*Proof.* As for the irreducible LCP values not exceeding  $\frac{n}{\delta}$ , Lemma 3.6 immediately yields

$$\sum_{\substack{\mathrm{LCP}[j] \le n/\delta \\ \mathrm{BWT}[j-1] \neq \mathrm{BWT}[j]}} \mathrm{LCP}[j] < \sum_{i=0}^{\lfloor \log \frac{n}{\delta} \rfloor} 2^{i+1} \cdot \left| \left\{ j : \mathrm{BWT}[j-1] \neq \mathrm{BWT}[j] \text{ and } \mathrm{LCP}[j] \in [2^i \dots 2^{i+1}) \right\} \right|$$
$$= \sum_{i=0}^{\lfloor \log \frac{n}{\delta} \rfloor} 2^{i+1} \cdot \mathcal{O}\left(\delta \log \delta\right) = \mathcal{O}\left(n \log \delta\right).$$

For irreducible LCP values larger than  $\frac{n}{\delta}$ , a simple approach would be to separately consider  $\Theta(\log \delta)$  ranges of LCP values,  $[2^i \dots 2^{i+1})$  for  $i \in [\lfloor \log \frac{n}{\delta} \rfloor \dots \lfloor \log n \rfloor]$ , and bound the number of irreducible LCP values in each range by  $\mathcal{O}(\frac{n \log \delta}{2^i})$ , as in the proof of Theorem 3.7. Unfortunately, this only gives an overall bound of  $\mathcal{O}(n \log^2 \delta)$  on the sum of irreducible LCP values.

Instead, we employ a similar scoring as in the proof of Lemma 3.6, except that we handle all the irreducible values  $\text{LCP}[i] > \frac{n}{\delta}$  together. With each such value, we associate  $\text{LCP}[i] - \lfloor \frac{n}{2\delta} \rfloor \ge \frac{1}{2}\text{LCP}[i]$  units of cost, and we charge them to individual characters of strings in  $S_n$ . We then show that each of the strings in  $S_n$  is charged at most  $6 + 2\log\delta$  times. Consequently, the sum of irreducible LCP values larger than  $\frac{n}{\delta}$  does not exceed twice the total cost, which is bounded by

$$2|\mathcal{S}_n| \cdot (6 + 2\log \delta) = 4n(3 + \log \delta).$$

The cost assignment is based on the trie  $\mathcal{T}$  of all reversed strings in  $\mathcal{S}_n$ . Let  $\mathrm{LCP}[i] > \frac{n}{\delta}$  be an irreducible LCP value, and let  $j_0 = \mathrm{SA}[i-1]$  and  $j_1 = \mathrm{SA}[i]$  so that  $\mathrm{LCP}[i] = \mathrm{LCE}(j_0, j_1)$ . Since  $\mathrm{LCP}[i]$  is irreducible, we have  $T^{\infty}[j_0-1] = \mathrm{BWT}[i-1] \neq \mathrm{BWT}[i] = T^{\infty}[j_1-1]$ . For  $k \in (n - \mathrm{LCP}[i] \dots n - \lfloor \frac{n}{2\delta} \rfloor]$ , the kth unit of the cost associated with  $\mathrm{LCP}[i]$  is charged to the kth character  $(T^{\infty}[j_t-1])$  of the string  $T^{\infty}[j_t-k\dots j_t-k+n) \in \mathcal{S}_n$ , where  $t \in \{0,1\}$  is such that  $\mathrm{size}(v_{\overline{T^{\infty}[j_t-1\dots j_t-k+n)}}) \leq \mathrm{size}(v_{\overline{T^{\infty}[j_{1-t}-1\dots j_{1-t}-k+n)}})$ . Note that  $T^{\infty}[j_0\dots j_0-k+n) =$  $T^{\infty}[j_1\dots j_1-k+n)$  holds due to  $\mathrm{LCE}(j_0, j_1) > n-k$ , so the edge from  $v_{\overline{T^{\infty}[j_t-1\dots j_t-k+n)}}$  to its parent  $v_{\overline{T^{\infty}[j_t\dots j_t-k+n)}}$  is light.

For every  $S \in S_n$ , a single position S[k], with  $k \in [1 \dots n - \lfloor \frac{n}{2\delta} \rfloor]$ , can be charged at most twice. This is because there is a unique position  $j \in [1 \dots n]$  such that  $S = T^{\infty}[j - k \dots j - k + n)$ , and S[k] can be charged for LCP[i] only if j = SA[i-1] or j = SA[i].

It remains to prove that at most  $3 + \log \delta$  characters of each  $S \in S_n$  can be charged. For this, we note that every charged character S[k], with  $k \in [1 \dots n - \lfloor \frac{n}{2\delta} \rfloor]$ , corresponds to a light edge on the path from the root of  $\mathcal{T}$  to the leaf  $v_{\overline{S}}$  connecting a node u at depth at least  $\lfloor \frac{n}{2\delta} \rfloor$  to its child v. Let us fix the highest such pair (u, v). It is not difficult to see that the argument from the proof Lemma 3.3 yields at least  $\frac{\operatorname{size}(v)}{4\delta}$  strings in  $S_n$  with disjoint leftmost occurrences in  $T^{\infty}[1 \dots)$ . However, all such occurrences overlap, so  $\operatorname{size}(v) \leq 4\delta$  and, consequently, the number of the remaining light edges on the path from the root of  $\mathcal{T}$  to  $v_{\overline{S}}$  is at most  $\log(\operatorname{size}(v)) \leq 2 + \log \delta$ . Including the edge from u to v, we obtain a bound of  $3 + \log \delta$  chargeable characters in total.  $\Box$ 

Due to  $\delta \leq r$ , the presented upper bound is always (asymptotically) at least as strong as the bound of Theorem 2.2; furthermore, it can be strictly stronger since  $\log \delta = o(\log r)$  is possible when  $\delta = \log^{o(1)} n$ . In Section 4, we construct strings proving tightness of the new bound for the values  $\delta$  ranging from  $\mathcal{O}(1)$  to  $\Omega(n)$ .

# 4 Lower Bounds

In this section, we present examples showing asymptotic tightness of the upper bounds in Section 3.3.

# 4.1 Lower Bound for the Number of BWT Runs

We give two constructions, corresponding to the bound of Theorem 3.7 and the trivial bound  $r = \mathcal{O}(n)$ , respectively.

For  $\ell \geq 1$ , let  $\operatorname{bin}_{\ell}(x) \in \{0, 1\}^{\ell}$  be the binary representation of  $x \in [0 \dots 2^{\ell})$ , and let  $\operatorname{bin}_{\ell}^{-1}$  be the inverse mapping.

**Lemma 4.1.** For all integers  $\ell \geq 2$  and  $K \geq 1$ , the length n, the substring complexity  $\delta$ , and the number of runs r in the BWT of a string  $T_{\ell,K} \in \{\$, 0, 1, 2\}^+$ , defined with

$$T_{\ell,K} = \left( \bigotimes_{k=0}^{K-1} \bigotimes_{i=0}^{2^{\ell}-1} \left( 2^{2^{k}\ell} \cdot \operatorname{bin}_{\ell}(i) \right) \right) \cdot \$,$$

satisfy  $n = \Theta(2^{K+\ell}\ell)$ ,  $\delta = \Theta(2^{\ell})$ , and  $r = \Omega(2^{\ell}\ell K)$ .

*Proof.* Let  $B_{\ell,k} := \bigoplus_{i=0}^{2^{\ell}-1} \left( 2^{2^{k_{\ell}}} \cdot \operatorname{bin}_{\ell}(i) \right)$  so that  $T_{\ell,K}$  is the concatenation of  $B_{\ell,k}$  for  $k \in [0 \dots K)$ , followed by a . Note that  $|B_{\ell,k}| = 2^{\ell} \cdot (2^{k_{\ell}} + \ell) = \Theta(2^{k+\ell}\ell)$ , so the length of  $T_{\ell,K}$  satisfies

$$n = 1 + \sum_{k=0}^{K-1} |B_{\ell,k}| = 1 + \sum_{k=0}^{K-1} \Theta\left(2^{k+\ell}\ell\right) = \Theta(2^{K+\ell}\ell).$$

To show that  $\frac{1}{m}|\mathcal{S}_m| = \mathcal{O}(2^\ell)$  holds for every  $m \in [1 \dots n]$ , we consider three cases:

- $m \leq \ell$ . Observe that any occurrence of  $S \in \mathcal{S}_m \cap \{0, 1, 2\}^*$  in  $T_{\ell,K}$  overlaps at most one maximal block of 2s. It is easy to see that there are  $\mathcal{O}(2^m)$  such strings S. Adding m substrings containing the symbol \$ thus yields  $|\mathcal{S}_m| = m + \mathcal{O}(2^m) = \mathcal{O}(2^\ell)$ .
- $\ell < m \leq 2^{K-1}\ell$ . Observe that any length-*m* substring of  $B_{\ell,k+1}$  with  $m \leq 2^k\ell$  is also a substring of  $B_{\ell,k}$ . More generally, if a length-*m* substring of  $T_{\ell,K}$  does not contain a \$, then its leftmost occurrence starts within  $B_{\ell,k}$  for some  $k \in [0., \lceil \log \frac{m}{\ell} \rceil]$ . Hence,

$$|\mathcal{S}_m| \le m + \sum_{k=0}^{\lceil \log \frac{m}{\ell} \rceil} |B_{\ell,k}| = m + \sum_{k=0}^{\lceil \log \frac{m}{\ell} \rceil} \mathcal{O}\left(2^{k+\ell}\ell\right) = \mathcal{O}(\frac{m}{\ell} \cdot 2^{\ell} \cdot \ell) = \mathcal{O}(m \cdot 2^{\ell}).$$

•  $2^{K-1}\ell < m$ . Then,  $\frac{1}{m}|\mathcal{S}_m| \le \frac{n}{m} = \mathcal{O}\left(\frac{2^{K+\ell}\ell}{2^{K}\ell}\right) = \mathcal{O}(2^\ell).$ 

To show  $\delta = \Omega(2^{\ell})$ , observe that, for any  $i \in [0 \dots 2^{\ell} - 1)$  and  $t \in [0 \dots \ell)$ , the string  $2^{t} \dots \delta_{\ell}(i) \cdot 2^{\ell-t} \in \{0, 1, 2\}^{2\ell}$  is a substring of  $T_{\ell,K}$ . Since all these substrings are different, we have  $|\mathcal{S}_{2\ell}| \geq (2^{\ell} - 1) \cdot \ell$  and  $\delta \geq \frac{1}{2\ell} |\mathcal{S}_{2\ell}| \geq 2^{\ell-1} - \frac{1}{2} = \Omega(2^{\ell})$ . As for the lower bound on r, we start by observing that, for any strings  $S, P \in \{0, 1\}^+$  such

As for the lower bound on r, we start by observing that, for any strings  $S, P \in \{0, 1\}^+$  such that  $|S| + |P| = \ell$  and any integer  $k \in [0 \dots K)$ , the string  $S2^{2^k\ell}P$  is a substring of  $B_{\ell,k}^{\infty}$ . Let us define  $s = \lim_{|S|}^{-1}(S)$  and  $p = \lim_{|P|}^{-1}(P)$ , and consider three cases:

•  $S \neq \mathbf{1}^{|S|}$ . Let  $x = p2^{|S|} + s < 2^{\ell} - 1$  and observe that  $S2^{2^{k}\ell}P$  is a substring of  $\operatorname{bin}_{\ell}(x) \cdot 2^{2^{k}\ell} \cdot \operatorname{bin}_{\ell}(x+1) = \operatorname{bin}_{|P|}(p) \cdot \operatorname{bin}_{|S|}(s) \cdot 2^{2^{k}\ell} \cdot \operatorname{bin}_{|P|}(p) \cdot \operatorname{bin}_{|S|}(s+1)$ .

- $S = 1^{|S|}$  and  $P \neq 0^{|P|}$ . Let  $x = (p-1)2^{|S|} + s < 2^{\ell} 1$  and observe that  $S2^{2^{k}\ell}P$  is a substring of  $\operatorname{bin}_{\ell}(x) \cdot 2^{2^{k}\ell} \cdot \operatorname{bin}_{\ell}(x+1) = \operatorname{bin}_{|P|}(p-1) \cdot \operatorname{bin}_{|S|}(s) \cdot 2^{2^{k}\ell} \cdot \operatorname{bin}_{|P|}(p) \cdot \operatorname{bin}_{|S|}(0)$ .
- $S = \mathbf{1}^{|S|}$  and  $P = \mathbf{0}^{|P|}$ . Then,  $S\mathbf{2}^{2^k\ell}P$  is a substring of  $\operatorname{bin}_{\ell}(2^{\ell} 1) \cdot 2^{2^k\ell} \cdot \operatorname{bin}_{\ell}(0)$ .

Note that in all but the last case,  $S2^{2^k\ell}P$  is a substring of  $B_{\ell,k}$ . However, since every  $B_{\ell,k}$  ends with  $bin_{\ell}(2^{\ell}-1)$ ,  $S2^{2^k\ell}P$  also occurs in  $T_{\ell,K}$  unless  $S = 1^{|S|}$ ,  $P = 0^{|P|}$ , and k = 0. The number of remaining triples (S, P, k) is equal to  $\ell - 1$  times the number of maximal blocks of 2s in  $T_{\ell,K}$  excluding the prefix  $2^{\ell}$  of  $T_{\ell,K}$ . As distinct triples yield distinct substrings, no substring  $S2^{2^k\ell}P$  occurs in  $T_{\ell,K}$  twice.

We will focus on triples satisfying  $S \neq \mathbf{1}^{|S|}$  and  $P \neq \mathbf{1}^{|P|}$ . By the discussion above, for each such triple,  $S2^{2^k\ell}P$  occurs exactly once in  $T_{\ell,K}$ . We will now show that every such triple (S, P, k) corresponds to a different run in the BWT of  $T_{\ell,K}$ . Thus, computing the number of considered triples yields

$$r \ge K \cdot \sum_{t=1}^{\ell-1} (2^t - 1)(2^{\ell-t} - 1) = K \left( 2^\ell (\ell - 3) + \ell + 3 \right) = \Omega \left( 2^\ell \ell K \right).$$

Let *i* be the rank of the suffix of  $T_{\ell,K}$  having  $S2^{2^k\ell}P$  as a prefix. We will show that i+1 is the rank of the suffix prefixed with  $S2^{2^k\ell}P'$ , where  $P' = bin_{|P|}(p+1)$ . Suppose that these suffixes are not adjacent in SA. Any suffix that is in between them must start with  $S2^{2^k\ell}c$  for some  $c \in \{0, 1\}$ . Furthermore, since every maximal block of 2s in  $T_{\ell,K}$  is followed by some  $\hat{P} \in \{0, 1\}^{|P|}$ , and there is exactly one occurrence of  $S2^{2^k\ell}\hat{P}$  in  $T_{\ell,K}$  for every possible  $\hat{P}$ , there would exist  $\hat{P}$  such that  $P \prec \hat{P} \prec P'$ . This is clearly impossible since it implies  $p < bin_{|P|}^{-1}(\hat{P}) < p+1$ .

Observe now that, by  $S \neq 1^{|S|}$ , the substrings  $S2^{2^k\ell}P$  and  $S2^{2^k\ell}P'$  are preceded in  $T_{\ell,K}$  by  $P = bin_{|P|}(p)$  and  $P' = bin_{|P|}(p+1)$ , respectively. Therefore,  $BWT[i+1] = (p+1) \mod 2 \neq p \mod 2 = BWT[i]$ .

**Lemma 4.2.** For all integers  $\ell \geq 2$  and  $\Delta \in \Omega(2^{\ell}) \cap \mathcal{O}(2^{\ell}\ell)$ , the length *n*, the substring complexity  $\delta$ , and the number of runs *r* in the BWT of a string  $T'_{\ell,\Delta} \in \{\$_1, \ldots, \$_{\Delta}, 0, 1, 2\}^+$ , defined with

$$T'_{\ell,\Delta} = \left( \bigotimes_{i=0}^{2^{\ell}-1} \left( 2^{\ell} \cdot \operatorname{bin}_{\ell}(i) \right) \right) \cdot \$_1 \$_2 \cdots \$_{\Delta},$$

satisfy  $n = \Theta(2^{\ell}\ell)$ ,  $\delta = \Theta(\Delta)$ , and  $r = \Omega(2^{\ell}\ell)$ .

*Proof.* The length satisfies  $n = 2^{\ell} \cdot 2\ell + \Delta = \Theta(2^{\ell}\ell)$ .

To show that  $\frac{1}{m}|\mathcal{S}_m| = \mathcal{O}(\Delta)$  holds for every  $m \in [1 \dots n]$ , we consider two cases:

•  $m \leq \ell \log_3 2$ . Trivially,  $|\mathcal{S}_m \cap \{0, 1, 2\}^*| \leq 3^m$ . Adding  $m + \Delta - 1$  substrings containing the symbols  $\hat{s}_i$  yields

$$\frac{1}{m}|\mathcal{S}_m| = \mathcal{O}\left(\frac{3^m + \Delta}{m}\right) = \mathcal{O}\left(\frac{2^\ell + \Delta}{m}\right) = \mathcal{O}\left(\frac{\Delta}{m}\right) = \mathcal{O}(\Delta).$$

•  $m > \ell \log_3 2$ . Then,  $\frac{1}{m} |\mathcal{S}_m| \le \frac{n}{m} = \mathcal{O}(\frac{2^\ell \ell}{m}) = \mathcal{O}(2^\ell) = \mathcal{O}(\Delta)$ .

To show  $\delta = \Omega(\Delta)$ , we observe that  $|\mathcal{S}_1| = \Delta + 3$ .

To bound r from below, as in the proof of Lemma 4.1, we observe that, for any strings  $S, P \in \{0, 1\}^+$  such that  $|S| + |P| = \ell$ , the string  $S2^{\ell}P$  occurs exactly once in  $T'_{\ell,\Delta}$  unless  $S = \mathbf{1}^{|S|}$  and  $P = \mathbf{0}^{|P|}$ . Moreover, every string  $S2^{\ell}P$  with  $S \neq \mathbf{1}^{|S|}$  and  $P \neq \mathbf{1}^{|P|}$  corresponds to a different run in the BWT. Hence, counting such strings yields  $r \geq 2^{\ell} \cdot (\ell-3) + \ell + 3 = \Omega(2^{\ell}\ell)$ .  $\Box$ 

Combining Lemmas 4.1 and 4.2, we obtain the following lower bound.

**Theorem 4.3.** For every  $N \ge 1$  and  $\Delta \in [1 .. N]$ , there exists a string T whose length n, substring complexity  $\delta$ , and number of runs r in the BWT satisfy  $n = \Theta(N)$ ,  $\delta = \Theta(\Delta)$ , and  $r = \Theta(\min(n, \delta \log \delta \max(1, \log \frac{n}{\delta \log \delta}))).$ 

*Proof.* If  $\Delta \log \Delta < \frac{1}{2}N$ , we set  $T = T_{\ell,K}$  with  $\ell = \max(2, \lceil \log \Delta \rceil)$  and  $K = \lceil \log \frac{N}{\Delta \log \Delta} \rceil$ . By Lemma 4.1, we thus have

$$\begin{split} n &= \Theta(2^{K+\ell}\ell) = \Theta\left(\frac{N}{\Delta \log \Delta} \Delta \log \Delta\right) = \Theta(N),\\ \delta &= \Theta(2^{\ell}) = \Theta(\Delta),\\ r &= \Omega(2^{\ell}\ell K) = \Omega\left(\Delta \log \Delta \log \frac{N}{\Delta \log \Delta}\right) = \Omega\left(\delta \log \delta \max(1, \log \frac{n}{\delta \log \delta})\right). \end{split}$$

If  $\Delta \log \Delta \geq \frac{1}{2}N$ , we set  $T = T'_{\ell,\Delta}$  with  $\ell = \max(2, \lceil \log \frac{N}{\log N} \rceil)$ . Note that  $\Delta = \Omega(\frac{N}{\log N}) = \Omega(2^{\ell})$  and  $\Delta = \mathcal{O}(N) = \mathcal{O}(2^{\ell}\ell)$ , so the assumptions of Lemma 4.2 are satisfied. We thus have

$$n = \Theta(2^{\ell}\ell) = \Theta\left(\frac{N}{\log N}\log\frac{N}{\log N}\right) = \Theta(N),$$
  
$$\delta = \Theta(\Delta),$$
  
$$r = \Omega(2^{\ell}\ell) = \Omega\left(\frac{N}{\log N}\log\frac{N}{\log N}\right) = \Omega(N) = \Omega(n)$$

In both cases, either the upper bound of Theorem 3.7 or the bound  $r = \mathcal{O}(n)$  is tight.  $\Box$ 

### 4.2 Lower Bound for the Sum of Irreducible LCP Values

The same strings show that our upper bound  $\mathcal{O}(n \log \delta)$  on the sum of irreducible LCP values is also tight.

**Lemma 4.4.** For all integers  $\ell \geq 2$  and  $K \geq 1$ , the sum of irreducible LCP values of  $T_{\ell,K}$  is  $r_{\Sigma} = \Omega(2^{K+\ell}\ell^2)$ .

Proof. In the proof of Lemma 4.1, we showed that, for every  $k \in [0..K)$  and  $S, P \in \{0,1\}^+ \setminus \{1\}^+$ such that  $|S| + |P| = \ell$ , the symbols preceding suffixes starting with  $S2^{2^k\ell}P \in \{0,1,2\}^+$  and  $S2^{2^k\ell}P' \in \{0,1,2\}^+$ , where  $P' = \lim_{|P|}(\lim_{|P|}(P) + 1)$ , are distinct, and that these suffixes are adjacent lexicographically. This implies that the corresponding irreducible LCP value is at least  $2^k\ell$ . With k = K - 1, noting that there are  $\Omega(2^\ell\ell)$  choices for S and P, we obtain  $r_{\Sigma} = 2^{K-1}\ell \cdot \Omega(2^\ell\ell) = \Omega(2^{K+\ell}\ell^2)$ .

Analogously to Lemma 4.2 and Theorem 4.3, we also get the following results:

**Lemma 4.5.** For all integers  $\ell \geq 2$  and  $\Delta \in \Omega(2^{\ell}) \cap \mathcal{O}(2^{\ell}\ell)$ , the sum of irreducible LCP values of  $T'_{\ell,\Delta}$  is  $r_{\Sigma} = \Omega(2^{\ell}\ell^2)$ .

**Theorem 4.6.** For every  $N \ge 1$  and  $\Delta \in [1..N]$ , there exists a string T whose length n, substring complexity  $\delta$ , and sum  $r_{\Sigma}$  of irreducible LCP values satisfy  $n = \Theta(N)$ ,  $\delta = \Theta(\Delta)$ , and  $r_{\Sigma} = \Theta(n \log \delta)$ .

# 5 Converting LZ77 to Run Length BWT

In this section, we describe an algorithm that, given the LZ77 parsing of a text  $T \in \Sigma^n$ , computes its run-length compressed BWT in  $\mathcal{O}(z \operatorname{polylog} n)$  time. We start with an overview that explains the key concepts. Next, we present two new data structures utilized in our algorithm: the compressed string synchronizing set (Section 5.1) and the compressed wavelet tree (Section 5.2). The conversion algorithm is then developed in Section 5.3.

For any substring Y of  $T^{\infty}$ , we define  $\operatorname{lpos}(Y) = \min\{i \in [1 \dots n] : T^{\infty}[i \dots i + |Y|] = Y\}$ . If Y is a substring of  $\overline{T}^{\infty}$ , we define  $\overline{\operatorname{lpos}}(Y)$  by replacing  $T^{\infty}$  in the definition with  $\overline{T}^{\infty}$ . We say that a substring Y of  $T^{\infty}$  is *left-maximal* if there exist distinct symbols  $a, b \in \Sigma$  such that the strings aY and bY are also substrings of  $T^{\infty}$ . The following definition, assuming  $\Sigma \cap \mathbb{N} = \emptyset$ , plays a key role in our construction.

**Definition 5.1** (BWT modulo  $\ell$ ). Let  $T \in \Sigma^n$ ,  $\ell \ge 1$  be an integer, and  $Y_i = T^{\infty}[SA[i] ... SA[i] + \ell)$ for  $i \in [1...n]$ . We define the string  $BWT_{\ell} \in (\Sigma \cup \mathbb{N})^n$ , called the BWT modulo  $\ell$  (of T), as follows. For  $i \in [1...n]$ ,

$$BWT_{\ell}[i] = \begin{cases} lpos(Y_i) & if Y_i \text{ is left-maximal,} \\ BWT[i] & otherwise. \end{cases}$$

The algorithm runs in  $k = \lceil \log n \rceil$  rounds. For  $q \in [0..k)$ , the input to the *q*th round is  $\operatorname{RL}(\operatorname{BWT}_{\ell})$ , where  $\ell = 2^q$ , and the output is  $\operatorname{RL}(\operatorname{BWT}_{2\ell})$ . At the end of the algorithm, we have  $\operatorname{RL}(\operatorname{BWT}_{2^k}) = \operatorname{RL}(\operatorname{BWT})$  because  $X \in \mathcal{S}_{2^k}$  is never left-maximal for  $2^k \ge n$ .

Informally, in round q, we are given a (run-length compressed) subsequence of BWT that can be determined based on sorting the suffixes only up to their prefixes of length  $2^q$ . BWT<sub> $\ell$ </sub>[b ...e]  $\in$  $\Sigma^+$  implies BWT<sub> $\ell+1$ </sub>[b ...e]  $\in \Sigma^+$  (because a prefix of a left-maximal substring is left-maximal). Hence, these subsequences need not be modified until the end of the algorithm (except possibly merging their runs with adjacent runs). For the remaining positions, BWT<sub> $\ell$ </sub> identifies the (leftmost occurrences of) substrings to be inspected in the qth round with the aim of replacing their corresponding runs in BWT<sub> $\ell$ </sub> with previously unknown BWT symbols (as defined in BWT<sub> $2\ell$ </sub>).

We call a block BWT[b..e] uniform if all symbols in BWT[b..e] are equal, and non-uniform otherwise. The following lemma ensures feasibility of the above construction.

# **Lemma 5.2.** For any integer $\ell \geq 1$ , it holds $|\operatorname{RL}(BWT_{\ell})| < 2r$ .

*Proof.* Denote  $\operatorname{RL}(\operatorname{BWT}_{\ell}) = ((c_1, \lambda_1), \ldots, (c_h, \lambda_h))$ , letting  $\lambda_0 = 0$ . By definition of  $\operatorname{BWT}_{\ell}$ , if  $c_i \in \mathbb{N}$ , then the block  $\operatorname{BWT}(\lambda_{i-1} \ldots \lambda_i]$  is non-uniform. Thus, there are at most r-1 runs of symbols from  $\mathbb{N}$  in  $\operatorname{BWT}_{\ell}$ .

On the other hand,  $c_i \in \Sigma$  and  $c_j \in \Sigma$ , with i < j, cannot both belong to the same run in BWT. If this was true, then either  $c_{i+1} \in \Sigma$  (which implies  $c_{i+1} = c_i$ , contradicting the definition of  $\operatorname{RL}(\operatorname{BWT}_{\ell})$ ), or  $c_{i+1} \in \mathbb{N}$ , which is impossible since then  $\operatorname{BWT}(\lambda_i \dots \lambda_{i+1}]$  is nonuniform. Thus, there are at most r runs of symbols from  $\Sigma$  in  $\operatorname{BWT}_{\ell}$ .

# 5.1 Compressed String Synchronizing Sets

Our algorithm builds on the notion of *string synchronizing sets*, recently introduced in [45]. Synchronizing sets are one of the most powerful techniques for sampling suffixes. As demonstrated in [48], in the uncompressed setting, they are the key in obtaining time-optimal solutions to many problems, and their further applications are still being discovered [3, 45]. In this section, we introduce a notion of *compressed string synchronizing sets*. Our construction is the first implementation of synchronizing sets in the compressed setting and thus of independent interest.

We start with the definition of basic synchronizing sets.

**Definition 5.3** ( $\tau$ -synchronizing set [45]). Let T be a string of length n, and let  $\tau \in [1 ... \lfloor \frac{n}{2} \rfloor]$ . A set  $S \subseteq [1 ... n - 2\tau + 1]$  is called a  $\tau$ -synchronizing set of T if it satisfies the following consistency and density conditions:

1. If  $T[i ... i + 2\tau) = T[j ... j + 2\tau)$ , then  $i \in S$  if and only if  $j \in S$  (for  $i, j \in [1 ... n - 2\tau + 1]$ ), 2.  $S \cap [i ... i + \tau) = \emptyset$  if and only if  $per(T[i ... i + 3\tau - 2]) \leq \frac{1}{3}\tau$  (for  $i \in [1 ... n - 3\tau + 2]$ ).

In most applications, we want to minimize |S|. Observe that the Thue–Morse sequence  $T_{\text{TM}}$  [82] does not contain any *cube* (substring of the form  $W^3$ ). Thus, by density condition, any synchronizing set S of the length-*n* prefix of  $T_{\text{TM}}$  satisfies  $|S| = \Omega\left(\frac{n}{\tau}\right)$  unless  $n < 3\tau$ . Therefore, we cannot hope to achieve an upper bound improving in the worst case upon the following one.

**Theorem 5.4** ([45]). For any string T of length n and parameter  $\tau \in [1 \dots \lfloor \frac{n}{2} \rfloor]$ , there exists a  $\tau$ -synchronizing set S of size  $|S| = O(\frac{n}{\tau})$ . Moreover, such S can be (deterministically) constructed in O(n) time.

Storing S for compressible strings presents the following challenge: As shown in [60], a lengthn prefix of  $T_{\text{TM}}$  satisfies  $z = \mathcal{O}(\log n)$  and yet, as discussed above, every  $\tau$ -synchronizing set of  $T_{\text{TM}}$  satisfies  $|\mathsf{S}| = \Omega\left(\frac{n}{\tau}\right)$ . Thus, although  $|\mathsf{S}|$  can be smaller than  $\frac{n}{\tau}$ , the assumption  $z \ll n$ does not imply  $|\mathsf{S}| \ll n$ , preventing us from keeping plain S when  $\tau = o(\frac{n}{z})$ .

We thus exploit a different property of compressible strings: their substrings Y satisfy  $lpos(Y) \in \bigcup_{j=1}^{z} (e_j - |Y| \dots e_j]$ , where  $e_j$  is the last position of the *j*th phrase in the LZ77 parsing of T. By consistency of S, it suffices to store  $\bigcup_{j=1}^{z} S \cap (e_j - 2\tau \dots e_j]$ . To check if  $i \in S$ , we then locate  $i' = lpos(T[i \dots i + 2\tau))$  and check if  $i' \in \bigcup_{j=1}^{z} S \cap (e_j - 2\tau \dots e_j]$ . This motivates the following (more general) definition.

**Definition 5.5** (Compressed  $\tau$ -synchronizing set). Let S be a  $\tau$ -synchronizing set of string T[1 ... n] for some  $\tau \in [1 ... \lfloor \frac{n}{2} \rfloor]$ , and, for every  $j \in [1 ... z]$ , let  $e_j$  denote the last position of the *j*th phrase in the LZ77 parsing of T. For  $k \in \mathbb{N}_{\geq 2}$ , we define the compressed representation of S as

$$\operatorname{comp}_k(\mathsf{S}) := \bigcup_{j=1}^{z} \mathsf{S} \cap \Big( e_j - k\tau \dots e_j + k\tau \Big].$$

Next, we prove that every text T has a synchronizing set S with a small compressed representation, and we show how to efficiently compute such S from the LZ77 parsing of T.

#### 5.1.1 The Nonperiodic Case

We initially assume that  $per(T[i \dots i+\tau)) > \frac{1}{3}\tau$  holds for all  $i \in [1 \dots n-\tau+1]$ .

**Theorem 5.6.** Let T be a string of length n and let  $\tau \in [1 \dots \lfloor \frac{n}{2} \rfloor]$ . Assume that  $per(T[i \dots i+\tau)) > \frac{1}{3}\tau$  holds for all  $i \in [1 \dots n - \tau + 1]$ . Then, for every  $k \in \mathbb{N}_{\geq 2}$ , there exists a  $\tau$ -synchronizing set S of T with  $comp_k(S) \leq 12kz$ .

Proof. Let  $h: S_{\tau} \to [0,1]$  be a function mapping strings to real values in [0,1] independently and uniformly at random. Note that h is collision-free almost surely (with probability 1). Let us define id :  $[1 \dots n - \tau + 1] \to [0,1]$  with  $id(i) = h(T[i \dots i + \tau))$ . Observe that (almost surely) id is an *identifier function*, that is, id(i) = id(j) holds if and only if  $T[i \dots i + \tau) = T[j \dots j + \tau)$ . In [45, Lemma 8.2], it is proved that then

$$\mathsf{S} := \{i \in [1 \dots n - 2\tau + 1] : \min\{\mathrm{id}(j) : j \in [i \dots i + \tau]\} \in \{\mathrm{id}(i), \mathrm{id}(i + \tau)\}\}$$

is a  $\tau$ -synchronizing set of T. Moreover,  $\mathbb{E}[|\mathsf{S}|] = \mathcal{O}(\frac{n}{\tau})$ . To see this, observe that, for  $j, j' \in [i \dots i + \tau]$ ,  $\mathrm{id}(j) = \mathrm{id}(j')$  implies  $|j' - j| > \frac{1}{3}\tau$  (otherwise, assuming j < j', we have  $\mathrm{per}(T[j \dots j' + j]) = \frac{1}{3}\tau$ ).

 $(\tau)$   $(\tau) \leq \frac{1}{3}\tau$ , which contradicts  $\operatorname{per}(T[j \dots j + \tau)) > \frac{1}{3}\tau)$ . Thus,  $T[i \dots i + 2\tau - 1]$  contains  $d \geq \frac{\tau}{3}$  distinct length- $\tau$  substrings. Since each has equal chance of having the smallest id, we have  $\mathbb{P}[i \in \mathsf{S}] \leq \frac{2}{d} \leq \frac{6}{\tau}$ , and consequently, by linearity of expectation,  $\mathbb{E}[|\mathsf{S}|] \leq \frac{6n}{\tau}$ . More generally,  $\mathbb{E}[|\mathsf{S} \cap (i \dots i + \ell)|] \leq \frac{6\ell}{\tau}$ , and therefore

$$\mathbb{E}\big[\left|\operatorname{comp}_{k}(\mathsf{S})\right|\big] = \mathbb{E}\left[\left|\bigcup_{j=1}^{z}\mathsf{S}\cap(e_{j}-k\tau\ldots e_{j}+k\tau)\right|\right] \le \sum_{j=1}^{z}\mathbb{E}\big[\left|\mathsf{S}\cap(e_{j}-k\tau\ldots e_{j}+k\tau)\right|\big] \le 12kz.$$

In particular,  $|\operatorname{comp}_k(\mathsf{S})| \leq 12kz$  holds for some h.

The above proof does not lead to an efficient algorithm for constructing S as it relies on the random assignment of unique names to *all* substrings in  $S_{\tau}$  and, since  $|S_{\tau}| = \Theta(z\tau)$  holds in the worst case, we cannot hope to achieve  $\mathcal{O}(z \text{ polylog } n)$  time this way. Next, we prove that assigning unique names to all elements of  $S_{\tau}$  is in fact not necessary.

**Lemma 5.7.** Let T be a string of length n and let  $\tau \in [1 ... \lfloor \frac{n}{2} \rfloor]$ . Assume that  $\operatorname{per}(T[i ... i + \tau)) > \frac{1}{3}\tau$  holds for all  $i \in [1 ... n - \tau + 1]$ . For  $i \in [1 ... n - \tau + 1]$ , let  $\operatorname{id}(i) := h(T[i ... i + \tau))$ , where  $h: S_{\tau} \to [0, 1]$  assigns independent and uniformly random values.

If  $\kappa = \max(1, \frac{\tau}{3c \ln n})$  for c > 1, then, with probability at least  $1 - n^{1-c}$ , all positions  $i \in [1 \dots n - 2\tau + 1]$  satisfy  $\min\{\operatorname{id}(j) : j \in [i \dots i + \tau]\} \leq \frac{1}{\kappa}$ .

*Proof.* Recall from the proof of Theorem 5.6 that  $T[i ... i + 2\tau - 1]$  contains  $d \ge \frac{\tau}{3}$  distinct length- $\tau$  substrings. Since the values of h are independent and uniformly distributed, we have  $\mathbb{P}\left[\min\{\mathrm{id}(j): j \in [i ... i + \tau]\} > \frac{1}{\kappa}\right] = \left(1 - \frac{1}{\kappa}\right)^d \le \exp(-\frac{d}{\kappa}) \le \exp(-\frac{\tau}{3\kappa})$ . The probability above is trivially 0 if  $\kappa = 1$ , so we can bound it by  $n^{-c}$  if  $\kappa = \max(1, \frac{\tau}{3c \ln n})$ . Taking the union bound across all positions i, we derive the final claim.

If  $\kappa$  is set as in the above lemma for a sufficiently large constant c, then, with high probability, each window contains at least one substring with a "small" identifier  $\leq \frac{1}{\kappa}$ . The "large" identifiers of other substrings are never used in the construction of the synchronizing set S and hence need not be specified. Consequently, to carry out the randomized construction of S using Theorem 5.6, rather than choosing a random function  $h: S_{\tau} \to [0, 1]$ , it suffices to select a random subset  $S_{\text{sample}} \subseteq S_{\tau}$  with rate  $\frac{1}{\kappa}$  (each string in  $S_{\tau}$  is included in  $S_{\text{sample}}$  independently with probability  $\frac{1}{\kappa}$ ) and then construct a uniformly random function  $h_{\text{sample}}: S_{\text{sample}} \to [0, \frac{1}{\kappa}]$  (mapping strings in  $S_{\text{sample}}$  to real values in  $[0, \frac{1}{\kappa}]$  independently and uniformly at random).

Clearly, the element-wise sampling of  $S_{\tau}$  is equivalent to sampling the set  $P_{\text{left}}$  containing the starting positions of the leftmost occurrences of strings in  $S_{\tau}$ . Sampling  $P_{\text{left}}$  directly is still hard, though. The key observation is that instead of  $P_{\text{left}}$  (which is difficult to compute), we can sample (at the same rate) elements of its superset  $P_{\text{close}} := \bigcup_{j=1}^{z} (e_j - \tau \dots e_j]$ , which is readily available, and yet still sufficiently small. Let  $P'_{\text{sample}} \subseteq P_{\text{close}}$  be a resulting sample. We then define the desired sample with  $P_{\text{sample}} := P'_{\text{sample}} \cap P_{\text{left}}$ . Crucially, however, we have

$$\mathbb{E}\left[|P'_{\text{sample}}|\right] = \frac{1}{\kappa}|P_{\text{close}}| \le \frac{3c\ln n}{\tau} \cdot z\tau = \mathcal{O}(z\log n).$$

To finish the construction, it suffices to pick a random function  $h_{\text{sample}} : S_{\text{sample}} \to [0, \frac{1}{\kappa}]$ to obtain  $\operatorname{id}(i) := h_{\text{sample}}(T[i \dots i + \tau))$  (letting  $\operatorname{id}(i) = 1$  if  $T[i \dots i + \tau) \notin S_{\text{sample}}$ ). Then, by Lemma 5.7 and the discussion above, using  $h_{\text{sample}}$  is with high probability equivalent to using a uniformly random function  $h : S_{\tau} \to [0, 1]$  during the construction behind Theorem 5.6. Moreover, we can also detect failures (that min { $\operatorname{id}(j) : j \in [i \dots i + \tau]$ } = 1 for some i), so the algorithm is Las-Vegas randomized. **Theorem 5.8.** Let T be a string of length n and let  $\tau \in [1 ... \lfloor \frac{n}{2} \rfloor]$ . Assume that  $\operatorname{per}(T[i ... i+\tau)) > \frac{1}{3}\tau$  holds for all  $i \in [1 ... n - \tau + 1]$ . There exists a Las-Vegas randomized algorithm that, for any constant  $k \in \mathbb{N}_{\geq 2}$ , given the LZ77 parsing of T, constructs in  $\mathcal{O}(z \log^5 n)$  time a compressed representation  $\operatorname{comp}_k(\mathsf{S})$  of a  $\tau$ -synchronizing set  $\mathsf{S}$  of T satisfying  $\operatorname{comp}_k(\mathsf{S}) \leq 24kz$ .

*Proof.* We first compute comp'(S) :=  $\bigcup_{j=1}^{z} S \cap (e_j - 3\tau + 2 \dots e_j + \tau)$ . The algorithm consists of three steps.

1. We start by computing the sample  $P_{\text{sample}} \subseteq P_{\text{left}}$  for  $\kappa = \frac{\tau}{3c \ln n}$ . As discussed above, for this we first compute  $P'_{\text{sample}} \subseteq P_{\text{close}}$  using the same rate  $\kappa$ . Using the algorithm from [15, 16], this takes  $\mathcal{O}\left(\frac{1}{\kappa}|P_{\text{close}}| + \log n\right) = \mathcal{O}(z \log n)$  time with high probability.<sup>2</sup> We then discard every  $i \in P'_{\text{sample}} \setminus P_{\text{left}}$ . By Theorem 6.11, this takes  $\mathcal{O}(\log^4 n)$  time per position. Overall, by taking Theorem 6.11 into account, computing  $P_{\text{sample}} = P'_{\text{sample}} \cap P_{\text{left}}$ , we spend  $\mathcal{O}(z \log^5 n)$  time. 2. Let  $\mathcal{S}_{\text{sample}} = \{T[i \dots i + \tau) : i \in P_{\text{sample}}\}$ . By consistency of S, whether  $i \in S$  or not,

2. Let  $S_{\text{sample}} = \{T[i \dots i + \tau) : i \in P_{\text{sample}}\}$ . By consistency of S, whether  $i \in S$  or not, depends only on  $T[i \dots i + 2\tau)$ . Thus, to determine comp'(S) using the construction in Theorem 5.6, it suffices to find all occurrences of strings from  $S_{\text{sample}}$  inside length- $(6\tau - 4)$  substrings of T centered at the boundaries of LZ77 phrases. Let  $P_{\text{occ}} = \{i \in [1 \dots n] : T[i \dots i + \tau) \in S_{\text{sample}}\}$ . Formally, we compute

$$P_{\text{nearocc}} := P_{\text{occ}} \cap \left( \bigcup_{j=1}^{z} [e_j - 3\tau + 3 \dots e_j + 2\tau - 1] \right).$$

As discussed above, for every  $i \in P_{\text{left}}$ , we have  $\mathbb{P}[i \in P_{\text{sample}}] = \frac{1}{\kappa}$ , or equivalently,  $\mathbb{P}[X \in \mathcal{S}_{\text{sample}}] = \frac{1}{\kappa}$  for  $X \in \mathcal{S}_{\tau}$ . Thus,  $\mathbb{E}[P_{\text{occ}} \cap [i \dots i + \ell)] = \frac{\ell}{\kappa}$  for  $i, \ell \in [1 \dots n]$  and hence

$$\mathbb{E}\Big[|P_{\text{nearocc}}|\Big] \le \frac{z(5\tau-3)}{\kappa} = \mathcal{O}(z\log n).$$

Let  $\ell = 3\tau - 2$ ,  $e_0 = 0$ , and  $e_{z+1} = n+1$ . We call the *j*th,  $j \in [1 \dots z]$ , phrase  $T[e_{j-1} + 1 \dots e_j]$ in the LZ77 parsing of T short if  $e_j - e_{j-1} \leq 2\ell - \tau$ , and long otherwise. We define the sentinel phrases 0 and z + 1 to be long. We create a text  $T_{\text{near}} \in (\Sigma \cup \{\#\})^*$ , where  $\# \notin \Sigma$ , as follows. Consider listing all long phrases left to right. Let  $T[e_{j-1} + 1 \dots e_j]$ ,  $j \in [1 \dots z + 1]$ , be the current long phrase and let  $e_{j'}$  be the last position of the preceding long phrase. Append  $T[\max(e_{j'} - \ell + 1, 1) \dots \min(e_{j-1} + \ell, n)]\#$  to  $T_{\text{near}}$ . It is easy to check that there is a bijection between  $P_{\text{nearocc}}$  and occurrences of strings from  $S_{\text{sample}}$  in  $T_{\text{near}}$ , and every string from  $S_{\text{sample}}$ has at least one occurrence in  $T_{\text{near}}$ .

To compute  $P_{\text{nearocc}}$ , we observe that excluding all # symbols,  $T_{\text{near}}$  consists of not more than 2z substrings, each with an earlier occurrence in  $T_{\text{near}}$ . In the construction of  $T_{\text{near}}$ , these substrings are:  $T[\max(e_{j'} - \ell + 1, 1) \dots e_{j'}], T[e_{j'} + 1 \dots \min(e_{j'} + \ell, e_{j'+1})], T[\min(e_{j'} + \ell, e_{j'+1}) + 1 \dots e_{j'+1}], \dots, T[e_{j-1} + 1 \dots \min(e_{j-1} + \ell, n)]$ . Thus, after accounting for the # symbols,  $T_{\text{near}}$ has an LZ77-like parsing with at most 3z phrases. The phrase boundaries of this parsing can be obtained immediately from the LZ77 parsing of T. To guarantee that their sources are in  $T_{\text{near}}$ we need to find their leftmost occurrences in T. Using Theorem 6.11 (applied to the LZ77 parsing of T), this takes  $\mathcal{O}(z \log^4 n)$  time. Then, to compute  $P_{\text{nearocc}}$ , we use the reporting index from Theorem 6.10 (constructed from the parsing of  $T_{\text{near}}$ ). Starting positions of example occurrences of strings from  $S_{\text{sample}}$  can be easily mapped to  $T_{\text{near}}$  via  $P_{\text{sample}}$ . Therefore, computing  $P_{\text{nearocc}}$ takes  $\mathcal{O}(z \log^4 n + |P_{\text{sample}}| \log^3 n + |P_{\text{nearocc}}| \log n) = \mathcal{O}(z \log^4 n)$  time in total.

3. We now compute comp'(S) from  $P_{\text{nearocc}}$ . Assume  $|P_{\text{nearocc}}| = \mathcal{O}(z \log n)$ . We start by constructing  $h_{\text{sample}} : S_{\text{sample}} \to [0, \frac{1}{\kappa}]$  that independently assigns uniformly random values.

<sup>&</sup>lt;sup>2</sup>The algorithm technically samples  $[0..|P_{\text{close}}|)$  rather than  $P_{\text{close}}$ , but the desired subset  $P'_{\text{sample}} \subseteq P_{\text{close}}$  is easy to obtain by exploiting the fact that  $P_{\text{close}}$  consists of at most z contiguous integer ranges.

With high probability,  $\mathcal{O}(\log n)$ -bit precision is sufficient to guarantee that there are no ties. We implicitly assign  $h_{\text{sample}}(X) = 1$  for  $X \notin S_{\text{sample}}$ . We keep a hash table mapping  $i \in P_{\text{sample}}$  to  $h_{\text{sample}}(T[i \dots i + \tau))$ . Moreover, with each  $i \in P_{\text{nearocc}}$  we store  $i' \in P_{\text{sample}}$  such that  $T[i \dots i + \tau) = T[i' \dots i' + \tau)$ . All i' can be computed during the construction of  $P_{\text{nearocc}}$ . Then, given  $i \in P_{\text{nearocc}}$ , we obtain  $h_{\text{sample}}(T[i \dots i + \tau))$  in  $\mathcal{O}(1)$  time.

Let  $P_{\text{nearocc}} = \{p_1, \ldots, p_k\}$  denote its elements in ascending order. Let  $j \in [1 \dots z]$ , and let range  $[p_b, \ldots, p_e]$  be such that  $P_{\text{nearocc}} \cap [e_j - 3\tau + 3 \dots e_j + 2\tau - 1] = \{p_b, \ldots, p_e\}$ . Denote  $\mathcal{I} = [e_j - 3\tau + 3 \dots e_j + \tau - 1]$  and consider the computation of  $S \cap \mathcal{I}$  according to Theorem 5.6. By Lemma 5.7, with high probability, for every  $i \in \mathcal{I}$ , we have  $[i \dots i + \tau] \cap \{p_b, \dots, p_e\} \neq \emptyset$ . Assume this is the case. Our goal is to find all  $i \in \mathcal{I}$ , for which

$$m_i := \min\{h_{\text{sample}}(T[t \dots t + \tau)) : t \in [i \dots i + \tau] \cap \{p_b, \dots, p_e\}\}$$

satisfies  $m_i \in \{h_{\text{sample}}(T[i \dots i + \tau)), h_{\text{sample}}(T[i + \tau \dots i + 2\tau))\}$ . This can only happen if  $\{i, i + \tau\} \cap \{p_b, \dots, p_e\} \neq \emptyset$  and hence it suffices to inspect  $\leq 2(e - b + 1)$  values of i. Using balanced BST to maintain  $[i \dots i + \tau] \cap \{p_b, \dots, p_e\}$ , the search can be implemented in  $\mathcal{O}((e - b) \log n)$  time. It is not difficult to modify this approach so that in total for all  $j \in [1 \dots z]$ , it takes  $\mathcal{O}(|P_{\text{nearocc}}|\log n) = \mathcal{O}(z \log^2 n)$  time. Note that during this algorithm we can detect whether for some  $i \in \mathcal{I}, [i \dots i + \tau] \cap \{p_b, \dots, p_e\} = \emptyset$ . If this happens, we restart the algorithm.

Let us return to the construction of  $\operatorname{comp}_k(\mathsf{S})$ . Observe that  $\mathsf{S}$  is entirely determined by  $\operatorname{comp}'(\mathsf{S})$ . Furthermore, it allows computing  $\mathsf{S} \cap [i \ldots i + \tau)$  in  $\mathcal{O}(\log^4 n + |\mathsf{S} \cap [i \ldots i + \tau)|)$  time. Namely, first locate the leftmost occurrence  $T[i_{\text{left}} \ldots i_{\text{left}} + 3\tau - 1)$  of  $T[i \ldots i + 3\tau - 1)$  using Theorem 6.11, and then copy the positions using the observation

$$i + \Delta \in (\mathsf{S} \cap [i \dots i + \tau))$$
 if and only if  $i_{\text{left}} + \Delta \in (\mathsf{S} \cap [i_{\text{left}} \dots i_{\text{left}} + \tau))$ 

Thus,  $\operatorname{comp}_k(\mathsf{S})$  is easily computed in  $\mathcal{O}(kz \log^4 n + |\operatorname{comp}_k(\mathsf{S})|)$  time. During the construction, we keep track of the number of positions in  $\operatorname{comp}_k(\mathsf{S})$  and restart the algorithm if their number gets too large. This concludes the construction of  $\operatorname{comp}_k(\mathsf{S})$ .

#### 5.1.2 The General Case

Periodic fragments are handled similarly as in [45]. This yields the following two results, which constitute the main outcome of this section.

**Theorem 5.9.** Let T be a string of length n and let  $\tau \in [1 \dots \lfloor \frac{n}{2} \rfloor]$ . For any  $k \in \mathbb{N}_{\geq 2}$ , there exists a  $\tau$ -synchronizing set  $\mathsf{S}$  of T satisfying  $\operatorname{comp}_k(\mathsf{S}) \leq 36kz$ .

*Proof.* The key difference, compared to Theorem 5.6, is that we can no longer assume that  $\operatorname{per}(T[i \dots i+\tau)) > \frac{1}{3}\tau$  holds for all *i*. Let us denote the set of positions that violate this assumption  $\mathbf{Q} := \{i \in [1 \dots n - \tau + 1] : \operatorname{per}(T[i \dots i + \tau)) \leq \frac{1}{3}\tau\}$ . We can then modify the construction as follows. Letting again id :  $[1 \dots n - \tau + 1] \rightarrow [0, 1]$  be any identifier function, we now define:

$$\mathsf{S} := \{i \in [1 \dots n - 2\tau + 1] : \min\{\mathrm{id}(j) : j \in [i \dots i + \tau] \setminus \mathsf{Q}\} \in \{\mathrm{id}(i), \mathrm{id}(i + \tau)\}\}$$

It is proved in [45] that: (1) Such construction yields a correct synchronizing set of T (Lemma 8.2). (2) If  $\mathsf{B} := \{i \in [1 \dots n - \tau + 1] \setminus \mathsf{Q} : \operatorname{per}(T[i \dots i + \tau - 1)) \leq \frac{1}{3}\tau \text{ or } \operatorname{per}(T[i + 1 \dots i + \tau)) \leq \frac{1}{3}\tau\}$  and  $\mathcal{B} := \{T[i \dots i + \tau) : i \in \mathsf{B}\}$ , then  $\operatorname{id}(i) := h(T[i \dots i + \tau))$ , where  $h : \mathcal{S}_{\tau} \to [0 \dots |\mathcal{S}_{\tau}|)$  is uniformly random function such that h(X) < h(Y) holds for all  $X \in \mathcal{B}, Y \notin \mathcal{B}$ , satisfies  $\mathbb{E}[|\mathsf{S}|] = \mathcal{O}\left(\frac{n}{\tau}\right)$ .

To see (2), let  $\mathsf{B}_{\text{near}} := \bigcup_{i \in \mathsf{B}} [i - \tau \dots i]$ . Then  $\mathbb{E}[|\mathsf{S}|] = \mathbb{E}[|\mathsf{S} \cap \mathsf{B}_{\text{near}}|] + \mathbb{E}[|\mathsf{S} \cap ([1 \dots n] \setminus \mathsf{B}_{\text{near}})|]$ .

• By the property of h, if  $i \in S \cap B_{\text{near}}$  then  $\{i, i + \tau\} \cap B \neq \emptyset$ , since then id achieves the minimum value on the position in B. Moreover, since by periodicity lemma, for any i, we have  $|[i \dots i + \lfloor \frac{\tau}{3} \rfloor) \cap B| \leq 2$ , then altogether we obtain  $|S \cap B_{\text{near}}| \leq 2|B| \leq \frac{12n}{\tau}$ .

• It is easy to see that if  $[i ... i + \tau] \cap \mathbb{Q} \neq \emptyset$  and  $[i ... i + \tau] \not\subseteq \mathbb{Q}$  then  $[i ... i + \tau] \cap \mathbb{B} \neq \emptyset$ . Thus, by contraposition, if  $i \in [1 ... n] \setminus \mathbb{B}_{near}$  then, as in the analysis in Theorem 5.6,  $\mathbb{P}[i \in \mathsf{S}] \leq \frac{6}{\tau}$ . Consequently,  $\mathbb{E}[|\mathsf{S} \cap ([1 ... n] \setminus \mathbb{B}_{near})|] \leq \frac{6n}{\tau}$ .

More generally,  $|\mathbf{S} \cap (i \dots i + \ell] \cap \mathbf{B}_{near}| \le |(i \dots i + \ell] \cap \mathbf{B}| + |(i + \tau \dots i + \ell + \tau] \cap \mathbf{B}| \le 4 \lceil \ell / \lceil \frac{\tau}{3} \rceil \rceil \le 4 \lceil \frac{3\ell}{\tau} \rceil$  and  $\mathbb{E}[|\mathbf{S} \cap ((i \dots i + \ell] \setminus \mathbf{B}_{near})|] \le \frac{6\ell}{\tau}$ . Thus, denoting  $\mathcal{I}_j := (e_j - k\tau \dots e_j + k\tau]$ , we have

$$\mathbb{E}\left[\left|\operatorname{comp}_{k}(\mathsf{S})\right|\right] = \mathbb{E}\left[\left|\bigcup_{j=1}^{z}\mathsf{S}\cap\mathcal{I}_{j}\right|\right] \leq \sum_{j=1}^{z}\mathbb{E}\left[\left|\mathsf{S}\cap\mathcal{I}_{j}\right|\right]$$
$$\leq \sum_{j=1}^{z}\left|\mathsf{S}\cap\mathcal{I}_{j}\cap\mathsf{B}_{\operatorname{near}}\right| + \sum_{j=1}^{z}\mathbb{E}\left[\left|\mathsf{S}\cap\left(\mathcal{I}_{j}\setminus\mathsf{B}_{\operatorname{near}}\right)\right|\right] \leq 36kz.$$

In particular,  $|\operatorname{comp}_k(\mathsf{S})| \leq 36kz$  holds for some h.

**Theorem 5.10.** Let T be a string of length n and let  $\tau \in [1 \dots \lfloor \frac{n}{2} \rfloor]$ . There exists a Las-Vegas randomized algorithm that, for any constant  $k \in \mathbb{N}_{\geq 2}$ , given the LZ77 parsing of T, constructs in  $\mathcal{O}(z \log^5 n)$  time a compressed representation  $\operatorname{comp}_k(\mathsf{S})$  of a  $\tau$ -synchronizing set  $\mathsf{S}$  of T satisfying  $|\operatorname{comp}_k(\mathsf{S})| \leq 72kz$ .

*Proof.* The algorithm is a modified construction from Theorem 5.8. The problematic part of adapting the randomized construction is enforcing that our sampling of substrings is equivalent to using a uniformly random function  $h : S_{\tau} \to [0, 1]$  among those satisfying h(X) < h(Y) for all  $X \in \mathcal{B}, Y \notin \mathcal{B}$ .

The key observation is that the bound  $|S \cap (i \dots i + \ell] \cap B_{\text{near}}| \leq 4 \lfloor \frac{3\ell}{\tau} \rfloor$  holds (for all *i*) in the *worst case*, and not only in expectation. We can thus explicitly set  $S \cap B_{\text{near}} := \bigcup_{i \in B} \{i - \tau, i\}$ , and then what remains is to determine S for positions  $i \notin B_{\text{near}}$ . For all such *i*, as discussed above, we either have  $[i \dots i + \tau] \cap Q = \emptyset$  (in which case the randomized construction remains unchanged), or  $[i \dots i + \tau] \subseteq Q$  (in which case there is nothing to do). To implement this modification, we need to be able to efficiently represent and compute sets

$$\mathsf{Q}_{\mathrm{nearocc}} := \mathsf{Q} \cap \left( \cup_{j=1}^{z} \mathcal{I}_{j} \right) \quad \text{ and } \quad \mathsf{B}_{\mathrm{nearocc}} := \mathsf{B} \cap \left( \cup_{j=1}^{z} \mathcal{I}_{j} \right),$$

where  $\mathcal{I}_{j} = [e_{j} - 3\tau + 2 \dots e_{j} + 2\tau - 1].$ 

Towards efficient representation, observe that if  $i+1 \in \mathbb{Q}$  and  $i \notin \mathbb{Q}$  then  $i \in \mathbb{B}$ . Analogously, if  $i-1 \in \mathbb{Q}$  and  $i \notin \mathbb{Q}$  then  $i \in \mathbb{B}$ . Thus,  $\mathbb{B}$  forms a boundary between  $\mathbb{Q}$  and  $[1 \dots n] \setminus \mathbb{Q}$ . We may also have  $\{i, i+1\} \subseteq \mathbb{B}$ , i.e., the enclosed interval of positions is empty. Nevertheless, augmenting every  $i \in \mathbb{B}_{\text{nearocc}}$  with a single bit yields a representation of  $\bigcup \{\mathbb{Q} \cap \mathcal{I}_j : j \in$  $[1 \dots z]$  and  $\mathbb{B} \cap \mathcal{I}_j \neq \emptyset \} \subseteq \mathbb{Q}_{\text{nearocc}}$ . To represent remaining elements of  $\mathbb{Q}_{\text{nearocc}}$ , it suffices to store a sequence of z bits, with the jth bit indicating whether  $\mathbb{Q} \cap \mathcal{I}_j$  is nonempty. It remains to see that since (as noticed in the proof of Theorem 5.9) for any i we have  $|[i \dots i + \lceil \frac{\tau}{3} \rceil) \cap \mathbb{B}| \leq 2$ , the set  $\mathbb{B}_{\text{nearocc}}$  satisfies  $|\mathbb{B}_{\text{nearocc}}| = \mathcal{O}(z)$ . Thus, we obtain an  $\mathcal{O}(z)$ -space representation of  $\mathbb{B}_{\text{nearocc}}$  and  $\mathbb{Q}_{\text{nearocc}}$ . This also implies that we can efficiently store (as  $\mathcal{O}(z)$  runs of consecutive positions) and random-access the set  $P_{\text{close}} \setminus (\mathbb{B}_{\text{nearocc}} \cup \mathbb{Q}_{\text{nearocc}})$ , during the sampling phase.

To finish the construction, it remains to show how to find  $\mathsf{B}_{\text{nearocc}}$ . By [45, Lemma 8.8], for any *i*, computation of  $[i \, ..\, i + b) \cap \mathsf{B}$ , where  $b \leq \left\lceil \frac{\tau}{3} \right\rceil$ , can be reduced to two LCE queries and the computation of the shortest period of some substring. More precisely, letting  $X = T[i + b \, ..\, i + \tau - 1)$ , we first determine  $p := \operatorname{per}(X)$ . If  $p > \frac{1}{3}\tau$  we conclude  $[i \, ..\, i + b) \cap \mathsf{B} = \emptyset$ . Otherwise, we compute the longest superstring  $Y = T[y \, ..\, y']$  of X that has period p and is a substring of  $T[i \, ..\, i + b + \tau - 1)$ . If  $|Y| \ge \tau - 1$  we have  $[i \, ..\, i + b) \cap \mathsf{B} = \{y - 1, y' - \tau + 2\}$  and otherwise we again have  $[i \, ..\, i + b) \cap \mathsf{B} = \emptyset$ . The set  $[i \, ..\, i + b) \cap \mathsf{Q}$  is deduced similarly.

If  $\tau > 3$ , we set  $b = \lfloor \frac{\tau-1}{3} \rfloor$  and use the above method to determine  $\mathsf{B}_{\text{nearocc}}$  and  $\mathsf{Q}_{\text{nearocc}}$ . To find p, we observe that we only need its exact value if  $p \leq \lfloor \frac{1}{3}\tau \rfloor$ . Otherwise, the knowledge that  $p > \lfloor \frac{1}{3}\tau \rfloor$  holds is sufficient. Since for our choice of b it holds  $\lfloor \frac{1}{3}\tau \rfloor \leq \frac{|X|}{2}$ , we can reduce the computation of p to the so-called 2-period query that given a string X asks to return  $\operatorname{per}(X)$  or to report that X is not periodic, that is  $\operatorname{per}(X) > \frac{1}{2}|X|$ . Using Theorem 6.7, we can answer 2period queries for substrings of T in  $\mathcal{O}(\log^3 n)$  time after a  $\mathcal{O}(z \log^2 n)$ -time preprocessing of the LZ77-compressed T. Computing y, y' on the other hand, is done using LCE queries in  $\mathcal{O}(\log n)$ time after the  $\mathcal{O}(z \log n)$ -time preprocessing (Theorem 6.3). Since  $\bigcup_{j=1}^{z} \mathcal{I}_{j}$  can be decomposed into  $\mathcal{O}(z)$  length-b intervals, overall we execute  $\mathcal{O}(z)$  queries, and hence the computation of  $\mathsf{B}_{\text{nearocc}}$  and  $\mathsf{Q}_{\text{nearocc}}$  (including preprocessing) takes  $\mathcal{O}(z \log^3 n)$  time.

To handle  $\tau \leq 3$ , we observe that if  $\tau \leq 2$ , then  $\mathsf{Q} = \mathsf{B} = \emptyset$ . For  $\tau = 3$ , we compute  $\mathsf{Q}_{\text{nearocc}}$  and  $\mathsf{B}_{\text{nearocc}}$  by checking consecutive  $i \in \bigcup_{j=1}^{z} \mathcal{I}_{j}$  using the definition of  $\mathsf{Q}$  and  $\mathsf{B}$ .

# 5.2 Compressed Wavelet Trees

Along with string synchronizing sets, wavelet trees [35], originally invented for text indexing, play a central role in our algorithm. Unlike virtually all prior applications of wavelet trees, ours uses a sequence of very long strings (up to  $\Theta(n)$  symbols). This approach is feasible since all strings are substrings of the text, which is stored in the LZ77-compressed form. In this section, we describe this novel variant of wavelet trees, dubbed here *compressed wavelet trees*. In particular, we prove the upper bound on their size, describe an efficient construction from the LZ77-compressed text, and show how to augment them to support some fundamental queries.

Let  $\Sigma$  be an alphabet of size  $\sigma \geq 1$ . Consider a string  $W[1 \dots m]$  over the alphabet  $\Sigma^{\ell}$  so that W is a sequence of  $m \geq 0$  strings of length  $\ell \geq 0$  over the alphabet  $\Sigma$ . The wavelet tree of W is defined as follows. Let  $\mathcal{T}$  be a perfect  $\sigma$ -ary rooted tree of height  $\ell$  with edges labelled by symbols of  $\Sigma$  such that, for every  $Y \in \Sigma^{\ell}$ , there exists a root-to-leaf path in  $\mathcal{T}$  whose edges are labelled  $Y[1], \dots, Y[\ell]$ . We define the label of a node as the concatenation of the edge labels on the path from the root. For  $X \in \Sigma^d$ , where  $d \in [0 \dots \ell]$ , by  $v_X$  we denote the node of  $\mathcal{T}$  labelled X. We let  $V(\mathcal{T}) = \bigcup_{d=0}^{\ell} \{v_X : X \in \Sigma^d\}$  denote the node set of  $\mathcal{T}$ .

With each node  $v_X \in V(\mathcal{T})$  we associate an increasing sequence  $I_X[1..h]$  of primary indices such that

$$\{I_X[i]: i \in [1 \dots h]\} = \{j \in [1 \dots m]: W[j][1 \dots |X|] = X\}.$$

Based on  $I_X$ , we define  $B_X \in \Sigma^*$  such that, for  $i \in [1 ... h]$ ,

$$B_X[i] = W[I_X[i]][|X| + 1],$$

if  $|X| < \ell$  and  $B_X = \varepsilon$  if  $|X| = \ell$ . In other words,  $B_X$  is a string containing the symbol at position |X| + 1 for each string of W that is prefixed by X. Importantly, the symbols in  $B_X$  occur in the same order as these strings occur in W.

As typically done in the applications of wavelet trees, we only explicitly store the strings  $B_X$ . The values of primary indices  $I_X$  are retrieved using additional data structures, based on the following observation.

**Lemma 5.11** ([35]). Let  $X \in \Sigma^d$ , where  $d \in [0 \dots \ell)$ . For every  $c \in \Sigma$  and  $j \in [1 \dots |I_{Xc}|]$ , we have  $I_{Xc}[j] = I_X[i]$ , where  $B_X[i]$  is the *j*th occurrence of *c* in  $B_X$ .

We define the compressed wavelet tree  $\mathcal{T}_c$  of W as the wavelet tree of W in which all strings  $B_X$  have been run-length compressed and, with the exception of  $\{v_{\varepsilon}\} \cup \{v_{W[i]}\}_{i=1}^m$ , all nodes  $v_X$  satisfying  $|\operatorname{RL}(B_X)| \leq 1$ , have been removed (the unary paths are collapsed into single edges). The shape and edge labels of the resulting tree are identical to the compact trie of strings  $W[1], \ldots, W[m]$ .

We store edge labels of  $\mathcal{T}_c$  as pointers to substrings in W. We assume that values of  $\ell$  and m fit into a single machine word so that each edge of  $\mathcal{T}_c$  and each element of  $\operatorname{RL}(B_X)$  can be encoded in  $\mathcal{O}(1)$  space. Since  $|\operatorname{RL}(B_Y)| \geq 1$  holds for every internal node  $v_Y \in V(\mathcal{T}_c)$ , and unless  $|V(\mathcal{T}_c)| = 1$ , each leaf  $v_Z$  in  $\mathcal{T}_c$  can be injectively mapped to an element of  $\operatorname{RL}(B_{Z'})$  for the parent  $v_{Z'}$  of  $v_Z$ , the space to store  $\mathcal{T}_c$  is dominated by the run-length compressed strings  $B_X$ , i.e.,  $\mathcal{T}_c$  needs  $\mathcal{O}(1 + \sum_{v_X \in V(\mathcal{T}_c)} |\operatorname{RL}(B_X)|)$  space.

**Theorem 5.12.** Let W be a non-empty sequence of equal-length strings and let  $\mathcal{T}_c$  be its compressed wavelet tree. Then,  $\sum_{v_X \in V(\mathcal{T}_c)} |\operatorname{RL}(B_X)| = \mathcal{O}(1 + |\operatorname{RL}(W)| \log |\operatorname{RL}(W)|).$ 

*Proof.* Let m = |W|,  $k = |\operatorname{RL}(W)| \leq m$ , and  $k' = |\{W[i] : i \in [1..m]\}| \leq k$ . Due to  $|V(\mathcal{T}_c)| \leq 2k' = \mathcal{O}(k)$ , we can focus on nodes  $v_X \in V(\mathcal{T}_c)$  such that  $|\operatorname{RL}(B_X)| \geq 2$ .

The proof resembles that of Lemma 3.1. With each  $X \in \Sigma^*$  such that  $|\operatorname{RL}(B_X)| \geq 2$ , we associate  $|\operatorname{RL}(B_X)| - 1$  units of cost and charge them to individual elements of W. We then show that each run in  $\operatorname{RL}(W)$  is in total charged at most  $2 \log k'$  units of cost. Consequently,

$$\sum_{\substack{v_X \in V(\mathcal{T}_c) \\ |\mathrm{RL}(B_X)| \ge 2}} |\mathrm{RL}(B_X)| \le 4k \log k' = \mathcal{O}(k \log k).$$

Consider  $X \in \Sigma^d$  with  $|\operatorname{RL}(B_X)| \geq 2$ ; note that  $d < \ell$ . Let  $\operatorname{RL}(B_X) = ((c_1, \lambda_1), \dots, (c_h, \lambda_h))$ . Observe that if we let  $p_0 = I_X[\lambda_i]$  and  $p_1 = I_X[\lambda_i + 1]$  for some  $i \in [1 \dots h)$ , then  $W[p_0][d + 1] = c_i \neq c_{i+1} = W[p_1][d + 1]$ . Moreover,  $B_X[\lambda_i] \neq B_X[\lambda_i + 1]$  implies  $W[p_0 + 1] \neq W[p_0]$  and  $W[p_1 - 1] \neq W[p_1]$ . The *i*th unit of cost is charged to  $W[p_t]$ , where  $t \in \{0, 1\}$  is chosen depending on the sizes of subtrees of  $\mathcal{T}_c$  rooted at the children of  $v_X$ , so that the subtree containing  $v_{W[p_t]}$  has at most as many leaves as the subtree containing  $v_{W[p_1-t]}$ .

Now, consider a run  $W[b \, . \, b'] = Y^{\delta}$  in  $\operatorname{RL}(W)$ . For a single depth d, the run could be charged at most twice, with at most one unit assigned to W[b] due to  $p_1 = b$  and at most one unit assigned to W[b'] due to  $p_0 = b'$ , both for  $X = Y[1 \, . \, d]$ . Moreover, note that the subtree size on the path from  $v_Y$  to the root  $v_{\varepsilon}$  of  $\mathcal{T}_c$  doubles for every depth d for which the run was charged. Thus, the total charge of the run is at most  $2 \log k'$  units.

Let  $W[1 \dots m]$  be a sequence of substrings of  $T^{\infty}$  of the same length  $\ell$ . Observe that if we have access to T, then the sequence W can be compactly encoded in  $\mathcal{O}(1 + |\operatorname{RL}(W)|)$  space. Namely, it suffices to store the length  $\ell$  and the sequence  $\operatorname{RL}((\operatorname{lpos}(W[i]))_{i \in [1\dots m]})$ . The following theorem shows that given such compact encoding of W and the LZ77 parsing of T, the compressed wavelet tree of W can be constructed efficiently.

**Theorem 5.13.** Given the LZ77 parsing of text T[1..n], and a sequence W[1..m] of  $m \le n$ substrings of  $T^{\infty}$  of length  $\ell \le n$ , represented as  $\operatorname{RL}((\operatorname{lpos}(W[i]))_{i \in [1..m]})$ , the compressed wavelet tree of W can be constructed in  $\mathcal{O}((z + |\operatorname{RL}(W)|) \log^2 n)$  time.

Proof. We start by constructing the compact trie  $\mathcal{T}_c$  of W. Let  $k = |\operatorname{RL}(W)| \leq m, \mathcal{W} = \{W[i] : i \in [1..m]\}$  and  $k' = |\mathcal{W}| \leq k$ . We first construct  $\mathsf{P} = \{\operatorname{lpos}(W[i]) : i \in [1..m]\}$  to represent  $\mathcal{W}$ . Given  $\operatorname{RL}((\operatorname{lpos}(W[i]))_{i\in[1..m]})$ ,  $\mathsf{P}$  is easy to compute in  $\mathcal{O}(k \log k)$  time. We now observe that an LCE query on T suffices to determine the lexicographical order between any substrings of  $T^{\infty}$ . We construct the data structure for LCE queries on T using Theorem 6.3 in  $\mathcal{O}(z \log^2 n)$  time. We then lexicographically sort all length- $\ell$  substrings of  $T^{\infty}$  starting at positions in  $\mathsf{P}$  in  $\mathcal{O}(k' \log k' \log n) = \mathcal{O}(k \log^2 n)$  time (e.g., using mergesort). Finally, we construct  $\mathcal{T}_c$  by inserting elements of  $\mathcal{W}$  in the order given by  $\mathsf{P}$ . We maintain the stack containing the internal nodes on the rightmost path, with the deepest node on top. Adding each string first removes some

elements from the stack, and then adds at most one new element. The total number of steps is thus  $\mathcal{O}(k)$ .

In the second step of the algorithm, we compute  $\operatorname{RL}(B_X)$  for all  $v_X \in V(\mathcal{T}_c)$ . Let  $\Sigma = \{c_0, \ldots, c_{\sigma-1}\}$  be the alphabet of T. If  $k \leq 1$  or  $\ell = 0$ , then  $|V(\mathcal{T}_c)| \leq 2$  and the step takes  $\mathcal{O}(1)$  time. Let us thus assume  $k \geq 2$  and  $\ell \geq 1$ . This implies  $\sigma \geq 2$ . Denote  $\operatorname{RL}(W) = ((R_1, \lambda_1), \ldots, (R_k, \lambda_k))$ , letting  $\lambda_0 = 0$ . Let also  $\delta_i = \lambda_i - \lambda_{i-1}$  for  $i \in [1 \ldots k]$ . For any  $i \in [1 \ldots |I_X|]$ , let  $J_X[i]$  be the index  $j \in [1 \ldots k]$  satisfying  $I_X[i] \in [\lambda_{j-1} + 1 \ldots \lambda_j]$ . It holds

$$W[I_X[i]] = R_{J_X[i]}.$$

Recall, that for  $|X| < \ell$ ,  $B_X[i] = W[I_X[i]][|X| + 1] = R_{J_X[i]}[|X| + 1]$ . Thus, when computing  $B_X$ , we can use the sequence  $J_X$  instead of  $I_X$ . Note that letting  $J_X[1 \dots |J_X|] = p_1^{t_1} p_2^{t_2} \dots p_h^{t_h}$ , where  $p_i \neq p_{i+1}$  for  $i \in [1 \dots h)$ , it holds  $t_j = \delta_{p_j}$  for  $j \in [1 \dots h]$ . Moreover,  $p_1 < \dots < p_h$ . We can thus store each  $J_X$  simply as a set  $J'_X = \{p_1, \dots, p_h\}$ .

We process all  $v_X \in V(\mathcal{T}_c)$  in the order of nonincreasing |X|. During the algorithm, we maintain sets  $J'_X$  for a subset of nodes  $v_X \in V(\mathcal{T}_c)$ . More precisely, at any point, we keep a data structure representing the set  $J'_X$  in sorted order for (only) the highest processed node on each leaf-to-root path.

Let us first assume  $\sigma = 2$ . To start, observe that the trie construction is easily augmented to compute, for any  $i \in [1 ... k]$ , the pointer to  $v_{R_i}$ . Thus, to initialize  $J'_X$  for all leafs  $v_X$ , iterate through  $\operatorname{RL}(W)$  and add i to  $J'_{R_i}$ . Consider now  $v_X \in V(\mathcal{T}_c)$  that is not a leaf. If  $v_X$  has one child  $v_{XY}$ , then it holds  $X = \varepsilon$  and  $B_{\varepsilon} = c^m$  where c is the first symbol in Y. Thus,  $\operatorname{RL}(B_X)$ is easy to compute. Assume thus that  $v_{XY_0}$  and  $v_{XY_1}$  are children of  $v_X$ , and for  $i \in \{0, 1\}$ ,  $Y_i$ starts with  $c_i \in \Sigma$ . Clearly,  $J'_X = J'_{XY_0} \cup J'_{XY_1}$ . Moreover, the sorted set  $J'_X$  is obtained by merging  $|\operatorname{RL}(B_X)|$  contiguous subsequences from either  $J'_{XY_0}$  or  $J'_{XY_1}$  (both regarded as sorted sequences). Suppose that during the *i*th step of the merging, we append to  $J'_X$  the elements  $\{p_b, \ldots, p_e\} \subseteq J'_{XY_h}$ , where  $p_b < \ldots < p_e$  and  $h \in \{0, 1\}$ . Note, that this implies that the *i*th run in  $B_X$  is  $c_h^{\delta}$ , where

$$\delta = \sum_{p \in J'_{XY_b} \cap [p_b \dots p_e]} \delta_p$$

To implement the merging efficiently, each set  $J'_X$  is represented using an AVL tree augmented (using standard techniques, i.e., each node stores the sum of elements in its subtree) to support computing, for any p', p'' the value of  $\sum_{p \in J'_X \cap [p'..p'']} \delta_p$  in  $\mathcal{O}(\log k)$  time. Since AVL trees also support split/join in  $\mathcal{O}(\log k)$  time, each subsequence  $\{p_b, \ldots, p_e\}$  can be appended to  $J'_X$  in  $\mathcal{O}(\log k)$  time. During each append, we compute the next element of  $\operatorname{RL}(B_X)$ . Consequently, the AVL tree for  $J'_X$ , together with  $\operatorname{RL}(B_X)$ , can be computed in  $\mathcal{O}(|\operatorname{RL}(B_X)|\log k)$  time. By Theorem 5.12, this yield the claim.

To generalize the merging to any  $\sigma \geq 2$ , during the computation of  $J'_X$ , we maintain a priority queue containing, for  $h \in [0..\sigma)$ , the smallest unextracted element of  $J'_{XY_h}$ . After extracting (and removing) the minimum element p' corresponding to  $h \in [0..\sigma)$  from the queue, the new minimum element p'' specifies the range [p'..p''] of elements to be extracted from  $J'_{XY_h}$ . The queue contains at most  $\sigma$  elements. We perform  $\mathcal{O}(1)$  queue operations for each run. Thus, the complexity of the construction is not affected.

The above Theorem holds also if W is the sequence of substrings of  $\overline{T}^{\infty}$ .

**Theorem 5.14.** Given the LZ77 parsing of text T[1..n], and a sequence W[1..m] of  $m \le n$  substrings of  $\overline{T}^{\infty}$  of length  $\ell \le n$ , represented as  $\operatorname{RL}((\overline{\operatorname{Ipos}}(W[i]))_{i\in[1..m]})$ , the compressed wavelet tree of W can be constructed in  $\mathcal{O}((z + |\operatorname{RL}(W)|) \log^2 n)$  time.

*Proof.* It suffices to observe that Theorem 6.3 supports also LCE queries on  $\overline{T}$ . Thus, the construction is identical as in the proof of Theorem 5.13.

**Computing Primary Indices** The key operation that we want to support on  $\mathcal{T}_c$  is, given a pointer to  $v_X \in V(\mathcal{T}_c)$  and an integer  $q \in [1 . . |I_X|]$ , compute the value  $I_X[q]$ .

Let us first consider a simpler problem. Given a pointer to  $v_X \in V(\mathcal{T}_c)$  different from the root, and  $q \in [1 . . |I_X|]$ , compute q' such that  $I_X[q] = I_{X_p}[q']$ , where  $v_{X_p}$  is the parent of  $v_X$ .

**Proposition 5.15.** Let  $\mathcal{T}_c$  be the compressed wavelet tree of W[1..m]. There exists a data structure of size  $\mathcal{O}(1 + |\operatorname{RL}(B_X)|)$  that, given a pointer to  $v_X \in V(\mathcal{T}_c)$  and an integer  $q \in [1..|I_X|]$ , in  $\mathcal{O}(\log m)$  time returns q' such that  $I_X[q] = I_{X_p}[q']$ , where  $v_{X_p}$  is the parent of  $v_X$ .

*Proof.* Observe that by Lemma 5.11, q' is the position of the qth occurrence of  $X[|X_p| + 1]$  in  $B_{X'}$ .

Denote  $\operatorname{RL}(B_Y) = ((c_1, \lambda_1), \dots, (c_k, \lambda_k))$ , where  $v_Y \in V(\mathcal{T}_c)$ . For any  $i \in [1 \dots k]$ , define  $n_i = |\{i \in [1 \dots \lambda_i] : B_Y[i] = c_i\}|$ . For  $c \in \Sigma$ , let  $L_{Y,c}$  be the sequence containing all pairs in  $\{(\lambda_i, n_i) : i \in [1 \dots k] \text{ and } c_i = c\}$ , sorted by  $\lambda_i$ .

Observe, that for any  $q \geq 1$ , given  $L_{Y,c}$ , the position of the qth occurrence of c in  $B_Y$ , if it exists, can be computed in  $\mathcal{O}(\log m)$  time using binary search. Thus, the data structure consists simply of sequences  $L_{X,c}$  for all pairs (X,c), such that c occurs in  $B_X$ . The total length of all sequences is  $\mathcal{O}(|\mathrm{RL}(B_X)|)$ . All sequences are concatenated and stored in one array, and we store a balanced BST containing the size and a pointer to the beginning of  $L_{Y,c}$ , for every c occurring in  $B_Y$ .

Let us now consider a more general problem. Let  $X_{up}, X_{down} \in \Sigma^*$  be such that  $X_{up}$  is a prefix of  $X_{down}$ , and  $v_{X_{up}}, v_{X_{down}} \in V(\mathcal{T}_c)$ . Let  $\mathcal{X} \subseteq \Sigma^*$  be the set of labels of nodes of  $\mathcal{T}_c$  on the path connecting (and including)  $v_{X_{up}}$  and  $v_{X_{down}}$ . Given a pointer to any node  $v_X$  of  $\mathcal{T}_c$  such that  $X \in \mathcal{X}$ , and an integer  $q \in [1 ... |I_X|]$ , we aim to compute q' such that  $I_X[q] = I_{X_{up}}[q']$ .

Let  $s = |I_{X_{up}}|$  and

$$\ell_i = \operatorname{lcp}\left(W[I_{X_{\operatorname{up}}}[i]], X_{\operatorname{down}}\right),$$

where  $i \in [1..s]$ . Consider  $\mathcal{P} = \{(i, \ell_i)\}_{i \in [1..s]}$  as a set of points on a plane. Let us fix some  $X \in \mathcal{X}$ , and let  $m_i = |\{j \leq i : \ell_j \geq |X|\}|$  for  $i \in [1..s]$ . The value of  $m_i$  is the number of points of  $\mathcal{P}$  inside the upper left "quadrant" defined by i and |X|. Clearly,  $m_1 \leq m_2 \leq \ldots \leq m_s$ .

**Lemma 5.16.** Let  $q' = \min\{i \in [1 \dots s] : m_i \ge q\}$ . Then,  $I_X[q] = I_{X_{up}}[q']$ .

Proof. By  $X \in \mathcal{X}$ , the string  $X_{up}$  is a prefix of X. By Lemma 5.11,  $I_X$  is a subsequence of  $I_{X_{up}}$ , and q' is the position of the qth index  $i \in [1 \dots s]$  satisfying  $lcp(W[I_{X_{up}}[i]], X) \ge |X|$ . Since X is a prefix of  $X_{down}$ , we obtain that for  $i \in [1 \dots s]$ ,  $lcp(W[I_{X_{up}}[i]], X) \ge |X|$  if and only if  $\ell_i \ge |X|$ . Therefore, q' is the qth index  $i \in [1 \dots s]$  for which it holds  $\ell_i \ge |X|$ , or equivalently, the smallest  $i \in [1 \dots s]$  satisfying  $m_i \ge q$ .

It thus suffices to store a data structure answering orthogonal range counting queries on the set of points  $\mathcal{P}$ . Then, by the above lemma, the value q' can be found using binary search. Using the data structure of [19] for range queries, we obtain the solution to our problem using  $\mathcal{O}(s)$  space and answering queries in  $\mathcal{O}(\log^2 s)$  time. To reduce the space usage, we observe that the sequence  $(\ell_i)_{i \in [1..s]}$  is compressible.

Lemma 5.17.

$$\left| \operatorname{RL} \left( (\ell_i)_{i \in [1..s]} \right) \right| \le 1 + \sum_{X \in \mathcal{X} \setminus \{X_{\operatorname{down}}\}} \left| \operatorname{RL}(B_X) \right|.$$

*Proof.* The proof is by induction on  $|\mathcal{X}|$ . If  $|\mathcal{X}| = 1$ , then  $X_{up} = X_{down}$  and hence  $\ell_i = |X_{down}|$ holds for all  $i \in [1 \dots s]$ . Consequently  $|\operatorname{RL}((\ell_i)_{i \in [1 \dots s]})| = 1$ .

Let us thus assume  $|\mathcal{X}| \geq 2$ . Let k denote the number of  $i \in [1 \dots s)$  satisfying  $\ell_i \neq \ell_{i+1}$ . Let  $i \in [1 \dots s)$  be any such position. Let  $X'_{up}$  be the shortest string in  $\mathcal{X}' = \mathcal{X} \setminus \{X_{up}\}$ , and denote  $s' = |I_{X'_{up}}|$ , and  $\ell'_j = \operatorname{lcp}(W[I_{X'_{up}}[j]], X'_{down})$ , for  $j \in [1 \dots s']$ . Consider two cases:

- 1.  $B_{X_{up}}[i] = B_{X_{up}}[i+1]$ . Denote  $c = X_{down}[|X_{up}|+1]$ . We observe that it must hold  $B_{X_{up}}[i] = c$ , since otherwise  $\ell_i = lcp(W[I_{X_{up}}[i]], X_{down}) = |X_{up}|$  and analogously  $\ell_{i+1} =$  $|X_{up}|$ , contradicting  $\ell_i \neq \ell_{i+1}$ . Let  $i' = |\{j \leq i : B_{X_{up}}[j] = c\}|$ . By Lemma 5.11, we have  $I_{X'_{up}}[i'] = I_{X_{up}}[i]$  and  $I_{X'_{up}}[i'+1] = I_{X_{up}}[i+1]$ . Thus, we have  $\ell'_{i'} \neq \ell'_{i'+1}$ . It is easy to see that this mapping of i to i' is injective. Thus, this case can happen at most  $|\operatorname{RL}((\ell'_j)_{j \in [1, s']})| - 1$  times. By the inductive assumption, it holds  $|\operatorname{RL}((\ell'_j)_{j \in [1, s']})| - 1 \leq 1$  $\sum_{X \in \mathcal{X}' \setminus \{X_{\text{down}}\}} |\text{RL}(B_X)|.$ 2.  $B_{X_{\text{up}}}[i] \neq B_{X_{\text{up}}}[i+1]$ . This case can happen at most  $|\text{RL}(B_{X_{\text{up}}})| - 1$  times.

Thus,  $k \leq |\operatorname{RL}(B_{X_{\operatorname{up}}})| - 1 + \sum_{X \in \mathcal{X}' \setminus \{X_{\operatorname{down}}\}} |\operatorname{RL}(B_X)| < \sum_{X \in \mathcal{X} \setminus \{X_{\operatorname{down}}\}} |\operatorname{RL}(B_X)|$ . It remains to note that  $|\operatorname{RL}((\ell_i)_{i \in [1..s]})| = 1 + k$ .

Denote  $\operatorname{RL}((\ell_i)_{i \in [1, s]}) = ((t_1, \lambda_1), \dots, (t_{s'}, \lambda_{s'}))$ , letting  $\lambda_0 = 0$ , and  $\delta_i = \lambda_i - \lambda_{i-1}$  for  $i \in [1 \dots s']$ . If we now let  $m'_i = \sum_{j \leq i, t_j \geq |X|} \delta_j$  for  $i \in [1 \dots s']$ , then  $m'_1 \leq \dots \leq m'_{s'}$  and Lemma 5.16 generalizes as follows.

**Lemma 5.18.** Let  $q'' = \min\{i \in [1 \dots s'] : m'_i \ge q\}$  and  $q' = \lambda_{q''-1} - m'_{q''-1} + q$ . Then,  $I_X[q] = I_{X_{\rm up}}[q'].$ 

*Proof.* As shown in the proof of Lemma 5.16, it suffices to prove that q' is the qth index  $i \in$  $[1 \dots s]$  satisfying  $\ell_i \geq |X|$ . It is easy to see, by definition of  $(m'_i)_{i \in [1 \dots s']}$ , that such i satisfies  $i \in (\lambda_{q''-1} \dots \lambda_{q''}]$ , where q'' is the smallest index  $i \in [1 \dots s']$  for which it holds  $m'_i \geq q$ . More precisely, it is the  $(q - m'_{a''-1})$ th position inside this interval. This yields  $q' = \lambda_{q''-1} - m'_{a''-1} + q$ , as claimed. 

Thus, rather than s points in the range counting structure, we can instead store a set of only  $s' \leq 1 + \sum_{X \in \mathcal{X} \setminus \{X_{\text{down}}\}} |\text{RL}(B_X)|$  weighted points  $\mathcal{P}' = \{(i, t_i)\}_{i \in [1..s']}$ , where the weight of the *i*th point is  $\delta_i$ . The range counting query is now replaced with the query that returns the total weight of points in a given range. Using [19], the data structure takes  $\mathcal{O}(s' \log s')$  space and answers range sum queries in  $\mathcal{O}(\log^2 s')$  time. Accounting for the binary search, we obtain the solution to our problem running in  $\mathcal{O}(\log^3 s')$  time per query. We have thus proved the following result.

**Proposition 5.19.** Let  $\mathcal{T}_c$  be the compressed wavelet tree of  $W[1 \dots m]$ . Let  $v_{X_{up}}, v_{X_{down}} \in V(\mathcal{T}_c)$ be such that  $X_{up}$  is a prefix of  $X_{down}$ . Let  $\mathcal{X}$  be the set of labels of nodes on the path between  $v_{X_{up}}$  and  $v_{X_{down}}$ . There exists a data structure of size  $\mathcal{O}(1 + \sum_{X \in \mathcal{X}} |\operatorname{RL}(B_X)| \log m)$  that, given a pointer to any  $v_X \in V(\mathcal{T}_c)$  with  $X \in \mathcal{X}$ , and  $q \in [1 \dots |I_X|]$ , returns in  $\mathcal{O}(\log^3 m)$  time q' such that  $I_X[q] = I_{X_{up}}[q'].$ 

We now show how to combine the above data structures to solve the general problem of computing primary index queries.

**Theorem 5.20.** Let  $\mathcal{T}_c$  be the compressed wavelet tree of  $W[1 \dots m]$ . There exists a data structure of size  $\mathcal{O}(1 + |\mathrm{RL}(W)|\log^2 m)$  that, given a pointer to  $v_X \in V(\mathcal{T}_c)$  and an integer  $q \in [1 \dots |I_X|]$ , in  $\mathcal{O}(\log^4 m)$  time returns  $I_X[q]$ .

Proof. We apply the heavy path decomposition [80, 36] to  $\mathcal{T}_c$ . Let u be a node of  $\mathcal{T}_c$  other than its root. An edge connecting u to its parent is called *light* if u has a sibling u' satisfying  $\operatorname{size}(u) \leq \operatorname{size}(u')$ ; otherwise, the edge is called *heavy* (see also Lemma 3.3). Then, every node has at most one child connected via a heavy edge, and there is at most log m light edges on any leaf-to-root path.

Let  $v_{X_{up}}, v_{X_{down}} \in V(\mathcal{T}_c)$  be such that  $X_{up}$  is a proper prefix of  $X_{down}$ . A path between  $v_{X_{up}}$  and  $v_{X_{down}}$  is called *heavy* if all edges on the path are heavy, and each of  $v_{X_{up}}, v_{X_{down}}$  is incident with exactly one heavy edge.

The data structure consists of two components. First, for every heavy path with endpoints  $v_{X_{up}}$  and  $v_{X_{down}}$ , we store the data structure from Proposition 5.19. Since every node in  $V(\mathcal{T}_c)$  belongs to at most one heavy path, the total size of these data structures, by Theorem 5.12, is  $\mathcal{O}(1 + |\mathrm{RL}(W)| \log^2 m)$ . The second component is the data structure from Proposition 5.15 for every node of  $\mathcal{T}_c$ .

Given a pointer to any  $v_X \in V(\mathcal{T}_c)$ , and an integer  $q \in [1 ... |I_X|]$ , the algorithm first checks if  $X = \varepsilon$ . If yes, then it holds  $I_X[q] = q$  and hence it returns q as the answer. Otherwise, we distinguish between two cases. If the edge to the parent  $v_{X_p}$  of  $v_X$  is light then, by Proposition 5.15, we first compute  $q' \in [1 ... |I_{X_p}|]$  such that  $I_{X_p}[q'] = I_X[q]$ , and then recursively compute  $I_{X_p}[q']$ . Otherwise (i.e., if the edge to  $v_{X_p}$  is heavy), by Proposition 5.19, we first compute  $q' \in [1 ... |I_{X_{up}}|]$  such that  $I_{X_{up}}[q'] = I_X[q]$  (where  $v_{X_{up}}$  is the highest node on the heavy path containing  $v_X$ ; each node stores the pointer to such node), and then recursively compute  $I_{X_{up}}[q']$ .

Every two steps of the recursion, the number of light edges on path to the root of  $\mathcal{T}_c$  decreases by at least one. Thus, the query takes  $\mathcal{O}(\log^4 m)$  time.

Finally, we prove that the data structure described above can be constructed efficiently, given the compact representation of W, and the LZ77 parsing of text T[1..n].

**Theorem 5.21.** Given the LZ77 parsing of a string T[1..n], and a sequence W[1..m] of  $m \leq n$  substrings of  $T^{\infty}$  of length  $\ell \leq n$ , represented as  $\operatorname{RL}((\operatorname{lpos}(W[i]))_{i \in [1..m]})$ , the compressed wavelet tree of W, supporting primary index queries in  $\mathcal{O}(\log^4 n)$  time, can be constructed in  $\mathcal{O}((z + |\operatorname{RL}(W)|) \log^2 n)$  time.

*Proof.* The algorithm extends the construction of the compressed wavelet tree presented in Theorem 5.13.

As in the basic algorithm, after constructing the compact trie  $\mathcal{T}_c$  of  $W[1 \dots m]$ , we process all nodes  $v_X \in V(\mathcal{T}_c)$  bottom-up. During the traversal, we compute  $\operatorname{size}(v_X)$ , i.e., the number of leaves in the subtree rooted in  $v_X$ , for all  $v_X \in V(\mathcal{T}_c)$ . This lets us identify the heavy edges. The algorithm maintains the invariant, that after the construction of  $\operatorname{RL}(B_X)$  is complete, it stores a representation of the sequence  $\operatorname{RL}((\ell_i)_{i \in [1..|I_X|]})$ , defined by  $\ell_i = \operatorname{lcp}(W[I_X[i]], X_{\operatorname{down}})$ , where  $X_{\operatorname{down}}$  is the longest string having X as a prefix, and for which it holds that  $v_{X_{\operatorname{down}}} \in V(\mathcal{T}_c)$ , and all edges on the path from  $v_X$  to  $v_{X_{\operatorname{down}}}$  are heavy. Then, if the path connecting  $v_X$  and  $v_{X_{\operatorname{down}}}$  is heavy, i.e.,  $X_{\operatorname{down}} \neq X$ , and  $v_X$  is the root of  $\mathcal{T}_c$  or the edge to the parent of  $v_X$  is light (see also the proof of Theorem 5.20), the algorithm uses this representation to initialize the data structure from Proposition 5.19. By Theorem 5.12 and [19, Table I], over all heavy paths, this takes  $\mathcal{O}(1 + |\operatorname{RL}(W)|\log^2 n)$  time.

Consider any  $v_X \in V(\mathcal{T}_c)$  and let  $s = |I_X|$ . The first challenge is representing the sequence  $\operatorname{RL}((\ell_i)_{i \in [1..s]})$  to support efficient queries and updates. Let us first focus on representing simply  $(\ell_i)_{i \in [1..s]}$ . We start by observing that the set of pairs  $\{(I_X[i], \ell_i)\}_{i \in [1..s]}$  is a valid representation of  $(\ell_i)_{i \in [1..s]}$ , since for any  $i, i' \in [1..s]$  it holds i < i' if and only if  $I_X[i] < I_X[i']$ . To see the advantage of this representation, compared to  $\{(i, \ell_i)\}_{i \in [1..s]}$ , let  $v_{X'} \in V(\mathcal{T}_c)$  be child of  $v_X$  connected by a heavy edge, and let  $(\ell'_i)_{j \in [1..p]}$  be the sequence defined by  $\ell'_i = \operatorname{lcp}(W[I_{X'}[j]], X_{\operatorname{down}})$ 

for  $j \in [1 . . p]$ , where  $p = |I_{X'}|$ . By Lemma 5.11,  $(\ell'_j)_{j \in [1 . . p]}$  is a subsequence of  $(\ell_i)_{i \in [1 . . s]}$ . Moreover, the "position"  $I_{X'}[j]$  of  $\ell'_j$  in the representation of  $(\ell'_j)_{j \in [1 . . p]}$  is automatically the correct position of  $\ell'_j$  in the representation of  $(\ell_i)_{i \in [1 . . s]}$ . More precisely, it holds

$$\{(I_{X'}[j], \ell'_j)\}_{j \in [1..p]} \subseteq \{(I_X[i], \ell_i)\}_{i \in [1..s]}.$$

The remaining elements of  $\{(I_X[i], \ell_i)\}_{i \in [1..s]}$  correspond to children of  $v_X$  other than  $v_{X'}$ . Specifically, letting c = X'[|X| + 1], we have

$$\{(I_X[i], \ell_i)\}_{i \in [1..s]} \setminus \{(I_{X'}[j], \ell'_j)\}_{j \in [1..p]} = \{(I_X[i], |X|) : i \in [1..s] \text{ and } B_X[i] \neq c\}.$$

We now note that the sequence  $J_X$ , used as a replacement of  $I_X$  in the proof of Theorem 5.13, satisfies the sufficient conditions to replace  $I_X$  also in this case. Namely, it holds  $J_X[1] \leq \ldots \leq J_X[s]$ , and for  $i \in [1 \ldots s)$ ,  $\ell_i \neq \ell_{i+1}$  implies  $J_X[i] < J_X[i+1]$ . Thus,  $\{(J_X[i], \ell_i)\}_{i \in [1 \ldots s]}$  is also a valid representation of  $(\ell_i)_{i \in [1 \ldots s]}$ , and moreover, since  $I_X[i] = I_{X'}[j]$  implies  $J_X[i] = J_{X'}[j]$ ,  $\{(J_X[i], \ell_i)\}_{i \in [1 \ldots s]}$  is also a disjoint union of  $\{(J_{X'}[j], \ell'_j)\}_{j \in [1 \ldots p]}$  and  $\{(J_X[i], |X|) : i \in [1 \ldots s] \text{ and } B_X[i] \neq c\}$ . The advantage of using the sequence  $J_X$  is that the needed values are easy to compute during the construction in Theorem 5.13. Therefore, letting  $\operatorname{RL}((\ell_i)_{i \in [1 \ldots s]}) = ((t_1, \lambda_1), \ldots, (t_{s'}, \lambda_{s'}))$ , where  $\lambda_0 = 0$ , and  $\delta_j = \lambda_j - \lambda_{j-1}$  for  $j \in [1 \ldots s']$ , we represent the sequence  $\operatorname{RL}((\ell_i)_{i \in [1 \ldots s]})$  as a set of pairs

$$Q_X = \{ (J_X[\lambda_{j-1}+1], (t_j, \delta_j)) \}_{j \in [1..s']}$$

Note, that  $J_X[\lambda_{j-1} + 1] < J_X[\lambda_j + 1]$  for  $j \in [1 ... s')$ .

During the algorithm, each set  $Q_X$  is stored in an AVL tree, with the first element of each pair as the key. In addition, for each set of pairs Q, we augment the tree that stores it (using the same technique as in the proof of Theorem 5.13) to compute, for any k', the value of

$$\sum_{\substack{(k,(t,\delta))\in Q\\k < k'}} \delta$$

Assume that  $v_X$  is the node currently processed by the algorithm. Denote  $\operatorname{RL}(B_X) = ((c_1, \kappa_1), \ldots, (c_{s''}, \kappa_{s''}))$ , letting  $\kappa_0 = 0$  and  $c_0 = c_{s''+1} = c$  (recall that c = X'[|X|+1]). Observe that during the computation of  $\operatorname{RL}(B_X)$ , at no extra cost, the algorithm in the proof of Theorem 5.13 can also return all values  $J_X[\kappa_{i-1}+1]$ , where  $i \in [1 \dots s'']$ . If  $X_{\text{down}} = X$ , i.e.,  $v_X$  is a leaf or all its children are connected using light edges, then it holds  $\ell_i = |X|$  for every  $i \in [1 \dots s]$ . Thus, the set representing  $\operatorname{RL}((\ell_i)_{i \in [1 \dots s]})$  consists of a single pair  $Q_X = \{(J_X[\kappa_0+1], (|X|, s))\}$ . Let us thus assume that  $v_{X'}$  is the child of  $v_X$  connected using a heavy edge. Assume also that we are given the representation  $Q_{X'}$  of  $\operatorname{RL}((\ell'_j)_{j \in [1 \dots p]})$ . By the above characterization, the set  $Q_X$  contains the pair  $(J_X[\kappa_{j_b-1}+1], (|X|, \kappa_{j_e} - \kappa_{j_b-1}))$ , for all subsets  $\{j_b, j_e\} \subseteq [1 \dots s'']$  that satisfy  $j_b \leq j_e, c \notin \{c_{j_b}, c_{j_b+1}, \dots, c_{j_e}\}$ , and  $c_{j_b-1} = c_{j_e+1} = c$ . Simply adding these pairs into  $Q_{X'}$ , however, does not produce the correct  $Q_X$ .

Let us initialize  $Q := Q_{X'}$ . Consider all subsets  $\{j_b, j_e\} \subseteq [1 \dots s'']$  satisfying the above conditions in the order of increasing  $j_b$ . The complication, following from the fact that  $\ell_i = \ell_{i+1}$ does not imply  $J_X[i] = J_X[i+1]$ , is to ensure that after inserting the pair corresponding to the subset  $\{j_b, j_e\}$ , into the current set Q, the value

$$\delta_{\text{prev}} = \sum_{\substack{(k,(t,\delta)) \in Q \\ k < J_X[\kappa_{j_b-1}+1]}} \delta$$

satisfies  $\delta_{\text{prev}} = \kappa_{j_b-1}$ . To achieve this, before inserting the pair, we first compute (using the augmented AVL tree) the current value  $\delta_{\text{prev}}$  and the pointer to the node storing the pair  $(k, (t, \delta)) \in Q$  with the largest k smaller than  $J_X[\kappa_{j_b-1}+1]$ . If  $\delta_{\text{prev}} > \kappa_{j_b-1}$  (note that this implies  $j_e < s''$ ), we replace the pair  $(k, (t, \delta))$  in Q with two pairs  $(k, (t, \delta'))$  and  $(J_X[\kappa_{j_e} + 1], (t, \delta''))$ , such that  $\delta' + \delta'' = \delta$  and  $\delta'' = \delta_{\text{prev}} - \kappa_{j_b-1}$ . Only then, we insert the pair  $(J_X[\kappa_{j_b-1} + 1], (|X|, \kappa_{j_e} - \kappa_{j_b-1}))$  into Q.

Each operation on Q takes  $\mathcal{O}(\log m)$  time. Thus, computing the set  $Q_X$  from  $Q_{X'}$  takes  $\mathcal{O}(|\mathrm{RL}(B_X)|\log m)$  time. By Theorem 5.12, over all  $v_X \in V(\mathcal{T}_c)$ , this amounts to  $\mathcal{O}(1 + |\mathrm{RL}(W)|\log^2 n)$  time.

To complete the construction, we observe that given  $\operatorname{RL}(B_Y)$ , all sequences  $L_{Y,c}$  (with c occurring in  $B_Y$ ) in the proof of Proposition 5.15 are easy to compute in  $\mathcal{O}(1+|\operatorname{RL}(B_Y)|\log|\operatorname{RL}(B_Y)|)$  time. Thus, by Theorem 5.12, initializing the structure from Proposition 5.15 takes  $\mathcal{O}(1+|\operatorname{RL}(W)|\log^2 n)$  time.

Similarly, as for Theorem 5.14, the above result applies also when W is a sequence of substrings of  $\overline{T}^{\infty}$ .

**Theorem 5.22.** Given the LZ77 parsing of a string T[1..n], and a sequence W[1..m] of  $m \leq n$  substrings of  $\overline{T}^{\infty}$  of length  $\ell \leq n$ , represented as  $\operatorname{RL}((\overline{\operatorname{lpos}}(W[i]))_{i \in [1..m]})$ , the compressed wavelet tree of W, supporting primary index queries in  $\mathcal{O}(\log^4 n)$  time, can be constructed in  $\mathcal{O}((z + |\operatorname{RL}(W)|) \log^2 n)$  time.

## 5.3 The Algorithm

We are now ready to show how to construct the sequences  $RL(BWT_{\ell})$ , where  $\ell = 2^q$  for  $q \in [0., \lceil \log n \rceil]$ .

For small  $\ell$ , constructing  $\operatorname{RL}(\operatorname{BWT}_{\ell})$  reduces to sorting and computing frequencies of length- $\Theta(\ell)$  substrings of  $T^{\infty}$ .

**Proposition 5.23.** Let  $\ell = \mathcal{O}(1)$ . Given the LZ77 parsing of T[1 ... n], the sequence  $RL(BWT_{\ell})$  can be constructed in  $\mathcal{O}(z \log^4 n)$  time.

Proof. First, we compute  $S_{\ell}$  with each string  $X \in S_{\ell}$  represented by  $p_X := \operatorname{lpos}(X)$ . For this, we exploit the fact that  $p_X \in (e_j - \ell \dots e_j]$  for some  $j \in [1 \dots z]$ . Thus, it suffices to list the  $\mathcal{O}(\ell z) = \mathcal{O}(z)$  candidate positions p, group them according to  $T^{\infty}[p \dots p + \ell)$  (by sorting), and keep the leftmost position in each group. Next, for each  $X \in S_{\ell}$ , we count the number  $c_X$  of positions  $i \in [1 \dots n]$  such that  $T^{\infty}[i \dots i + \ell) = X$ . For this, we use Theorem 6.21 to compute the number of occurrences of X in  $T^{\infty}[1 \dots n + \ell)$ . Similarly, we count the number  $c'_X$  of positions  $i \in [1 \dots n]$  such that  $T^{\infty}[i - 1 \dots i + \ell) = T^{\infty}[p_X - 1 \dots p_X + \ell)$ ; observe that X is left-maximal if and only if  $c'_X < c_X$ . Finally, we construct  $\operatorname{BWT}_{\ell}$  from left to right processing the strings  $X \in S_{\ell}$  in lexicographic order and appending  $p_X^{c_X}$  if  $c'_X < c_X$ , and  $(T^{\infty}[p_X - 1])^{c_X}$  otherwise. The overall running time  $\mathcal{O}(z \log^4 n)$  is dominated by the use of Theorem 6.21.

Let  $q \ge 4$ . We show how to compute  $\operatorname{RL}(\operatorname{BWT}_{2\ell})$ , given the LZ77 parsing of T and  $\operatorname{RL}(\operatorname{BWT}_{\ell})$ . The main idea of the algorithm is as follows.

Let S be a  $\tau$ -synchronizing set of T, where  $\tau = \lfloor \frac{\ell}{3} \rfloor$ . As noted earlier,  $\text{BWT}_{\ell}[j] \in \Sigma$  implies  $\text{BWT}_{2\ell}[j] \in \Sigma$ . Let  $\text{BWT}_{\ell}[y \dots y'] \in \mathbb{N}^+$  be a run in  $\text{BWT}_{\ell}$ . By definition of  $\text{BWT}_{\ell}$ , the suffixes of  $T^{\infty}$  starting at positions  $i \in \text{SA}[y \dots y']$  share a common prefix of length  $\ell \geq 3\tau$ . Thus, assuming that  $S \cap [i \dots i + \tau) \neq \emptyset$  holds for all  $i \in \text{SA}[y \dots y']$  (the periodic case is handled separately), by the consistency of S, all text positions  $i \in \text{SA}[y \dots y']$  share a common offset  $\Delta$ with  $i + \Delta = \min(S \cap [i \dots i + \tau))$ . This lets us deduce the order of length- $2\ell$  prefixes  $T[i \dots i + 2\ell)$  based on the order of strings  $T[i + \Delta ... i + 2\ell)$  starting at synchronizing positions. For this, from the sorted list of fragments  $T[s...s+2\ell-\Delta)$  across  $s \in S$ , we extract, using a wavelet tree, those preceded by  $T[i...i + \Delta)$  (a prefix common to  $T^{\infty}[i...)$  for  $i \in SA[y...y']$ ). Importantly, the synchronizing positions s sharing  $T[s-\ell...s+2\ell)$  can be processed together; hence, by Theorem 5.9, it suffices to use  $\mathcal{O}(z)$  distinct substrings.

We formalize these ideas as follows. Let

$$\mathsf{R} = \left\{ i \in [1 \dots n - 3\tau + 2] : \text{per}\left(T[i \dots i + 3\tau - 2]\right) \le \frac{1}{3}\tau \right\}.$$

The description of the algorithm is divided into the nonperiodic case (when  $R = \emptyset$ ) and the general case.

#### 5.3.1 The Nonperiodic Case

Let  $(s'_i)_{i \in [1, |S|]}$  be the sequence containing all positions in S such that i < j holds if

•  $T^{\infty}[s'_i \dots s'_i + 7\tau) \prec T^{\infty}[s'_j \dots s'_j + 7\tau)$ , or •  $T^{\infty}[s'_i \dots s'_i + 7\tau) = T^{\infty}[s'_j \dots s'_j + 7\tau)$  and  $\overline{T^{\infty}[s'_i - \tau \dots s'_i)} \prec \overline{T^{\infty}[s'_j - \tau \dots s'_j)}$ .

Based on  $(s'_i)_{i \in [1, |S|]}$ , we define three length-|S| sequences. For  $i \in [1, |S|]$ , we set

$$\widetilde{W}[i] = T^{\infty}[s'_i - \tau \dots s'_i + 7\tau),$$
  

$$W[i] = \overline{T^{\infty}[s'_i - \tau \dots s'_i)},$$
  

$$W'[i] = T^{\infty}[s'_i \dots s'_i + 7\tau).$$

Recall that we can compactly represent the sequence  $\widetilde{W}$  in  $\mathcal{O}(1 + |\operatorname{RL}(\widetilde{W})|)$  space using  $\operatorname{RL}((\operatorname{lpos}(\widetilde{W}[j]))_{j \in [1..|S|]})$ . The sequences W and W' can be represented analogously, except that we use  $\operatorname{RL}((\operatorname{lpos}(W[j]))_{j \in [1..|S|]})$  for W.

**Lemma 5.24.** The sequences  $\widetilde{W}$ , W, and W' defined above satisfy  $|\operatorname{RL}(W)|$ ,  $|\operatorname{RL}(W')| \leq |\operatorname{RL}(\widetilde{W})| \leq |\operatorname{comp}_7(\mathsf{S})|$ .

*Proof.* For the first inequality, note that  $\widetilde{W}[i] = \widetilde{W}[i+1]$  implies W[i] = W[i+1] and W'[i] = W'[i+1].

Let  $\operatorname{RL}(\widetilde{W}) = ((R_1, \lambda_1), \dots, (R_h, \lambda_h))$ . Observe that  $i \neq j$  implies  $R_i \neq R_j$ . For  $i \in [1 \dots h]$ , let  $T^{\infty}[p - \tau \dots p + 7\tau)$  be the occurrence of  $R_i$  in  $T^{\infty}$  that minimizes  $p \in [1 \dots n]$ . Then, there exists  $j \in [1 \dots z]$  such that  $e_j - 8\tau . By the consistency of <math>\mathsf{S}$ , we conclude that  $p \in \mathsf{S} \cap (e_j - 7\tau \dots e_j + \tau] \subseteq \operatorname{comp}_7(\mathsf{S})$ . The claim follows, since this map is injective.

Importantly, the compact representations of  $\widetilde{W}$ , W, and W' can be computed efficiently.

**Lemma 5.25.** Given  $\operatorname{comp}_7(\mathsf{S})$  and the LZ77 parsing of T, the compact representations of W, W, and W' can be constructed in  $\mathcal{O}(z \log^4 n + |\operatorname{comp}_7(\mathsf{S})| \log^3 n)$  time.

Proof. Denote  $\operatorname{RL}(W) = ((R_1, \lambda_1), \ldots, (R_k, \lambda_k))$ , letting  $\lambda_0 = 0$ , and  $\delta_i = \lambda_i - \lambda_{i-1}$  for  $i \in [1 \dots k]$ . As observed in the proof of Lemma 5.24, it holds  $\{R_i\}_{i \in [1 \dots k]} = \{T^{\infty}[i - \tau \dots i + 7\tau) : i \in \operatorname{comp}_7(S)\}$ . Using LCE queries on T we can sort any set of substrings of  $T^{\infty}$ . Thus, using Theorems 6.3 and 6.11, we first compute the sequence  $(\operatorname{lpos}(R_i))_{i \in [1 \dots k]}$ . We then observe that by the consistency of S, the value  $\delta_i$  is the number of occurrences of  $R_i$  in  $T^{\infty}$  starting at a position j satisfying  $j + \tau \in [1 ... n]$ . Thus, we obtain  $(\lambda_i)_{i \in [1...k]}$  using Theorem 6.21. In total, this takes  $\mathcal{O}(z \log^4 n + |\operatorname{comp}_7(\mathsf{S})| \log^3 n)$  time.

To obtain the compact representation of W (resp. W'), it now suffices to merge the adjacent equal runs obtained by discarding the length- $7\tau$  suffix (resp. length- $\tau$  prefix) of strings in  $\widetilde{W}$ . Importantly, the counts for the runs in  $\operatorname{RL}(W)$  and  $\operatorname{RL}(W')$  are computed from the counts of  $\operatorname{RL}(\widetilde{W})$ . The merging is performed using Theorem 6.3. We then compute the leftmost occurrences using Theorems 6.11 and 6.12. In total, we spend  $\mathcal{O}(z \log^4 n + |\operatorname{comp}_7(\mathsf{S})| \log^3 n)$ time.

Next, we recall the notion of *distinguishing prefixes*, originally introduced in [45], that allows mapping each suffix  $T^{\infty}[i..)$  to the corresponding node of the wavelet tree of W.

**Definition 5.26** (Distinguishing prefix). For any position  $i \in [1 \dots \max(\mathsf{S} \cup \{0\})]$ , let  $i_{\text{succ}} = \min\{j \in \mathsf{S} : j \ge i\}$ . The distinguishing prefix of  $T[i \dots n]$  is  $D_i = T[i \dots i_{\text{succ}} + 2\tau)$ .

Let  $\mathcal{D} = \{D_j : j \in [1 \dots \max(\mathsf{S} \cup \{0\})]\}$ . Note that if Y starts with  $D \in \mathcal{D}$ , then, for every occurrence  $T^{\infty}[i \dots i+|Y|) = Y$  with  $i \in [1 \dots n]$ , the distinguishing prefix  $D_i$  is defined and satisfies  $D_i = D$ .<sup>3</sup> Thus, for any such Y, we define  $D_Y = D$ . We denote  $D'_Y = D_Y[1 \dots |D_Y| - 2\tau]$ .

We now present the key lemma used in our algorithm. Assume that we have constructed a wavelet tree of W.

**Lemma 5.27.** Let Y be a string starting with an element of  $\mathcal{D}$ . Denote Y = XX', where  $X = D'_Y$ , and assume that  $|X| < \tau$  and  $|X'| \leq 7\tau$ . Let  $[y \, .. \, y']$  be the range of all indices i such that  $T^{\infty}[SA[i]..]$  starts with Y for  $i \in [y \, .. \, y']$ .

Let  $W'[f \dots f']$  be the range containing all elements of W' prefixed with the string X', and let  $[b \dots b'] = \{i \in [1 \dots |I_{\overline{X}}|] : I_{\overline{X}}[i] \in [f \dots f']\}$ . Then

- 1.  $B_{\overline{X}}[b \dots b']$  and  $BWT[y \dots y']$  are equal as multisets.
- 2.  $|\operatorname{RL}(B_{\overline{X}}[b \, . \, b'])| \leq 3|\operatorname{RL}(\operatorname{BWT}[y \, . \, y'])|.$

*Proof.* 1. Due to T[n] = \$, by the consistency of S, there is a one-to-one correspondence between the occurrences of Y in  $T^{\infty}$  starting in [1 ... n], and positions  $s \in \mathsf{S}$  satisfying (a)  $T^{\infty}[s ... s + |X'|) = X'$ , and (b)  $T^{\infty}[s - |X| ... s) = X$ . Let us interpret the process of identifying the subsequence of  $(s'_i)_{i \in [1..]\mathsf{S}]}$  containing all such s as a two-step search.

First, we note that  $s_i^{f} \in S$  satisfies condition (a) if and only if  $i \in [f \dots f']$ . We refer to the process of identifying the range  $[f \dots f']$  as the *forward search*. Then, to additionally satisfy (b), we select a subsequence of  $W[f \dots f']$  containing only strings ending with X (*backward search*). By definition of  $[b \dots b']$ , such subsequence is given by  $I_{\overline{X}}[b \dots b']$ , and moreover,  $B_{\overline{X}}[b \dots b']$  contains symbols preceding suffix X in all  $W[f \dots f']$  having X as a suffix. This yields the claim.

2. Let  $\widetilde{Y}$  be any substring of  $T^{\infty}$  such that Y is a prefix of  $\widetilde{Y}$  and  $|\widetilde{Y}| = |X| + 7\tau$ . Let  $[\widetilde{y} \dots \widetilde{y}']$ ,  $[\widetilde{f} \dots \widetilde{f}']$ , and  $[\widetilde{b} \dots \widetilde{b}']$  be the ranges (as in the lemma statement) for  $\widetilde{Y}$ . Since Y is a prefix of  $\widetilde{Y}$  and  $D_{\widetilde{Y}} = D_Y$ , we obtain  $[\widetilde{y} \dots \widetilde{y}'] \subseteq [y \dots y']$ ,  $[\widetilde{f} \dots \widetilde{f}'] \subseteq [f \dots f']$ , and  $[\widetilde{b} \dots \widetilde{b}'] \subseteq [b \dots b']$ . Moreover, by definition of  $I_{\overline{X}}$ , the range  $[b \dots b']$  is a disjoint union of ranges  $[\widetilde{b} \dots \widetilde{b}']$  corresponding to all choices of  $\widetilde{Y}$ .

Since  $|\tilde{Y}| - |X| = 7\tau$  implies  $|\operatorname{RL}(W'[\tilde{f} \dots \tilde{f}'])| = 1$ , the symbols in  $B_{\overline{X}}[\tilde{b} \dots \tilde{b}']$  appear in the nondecreasing order. Consequently,  $B_{\overline{X}}[b \dots b']$  can be obtained by partitioning  $\operatorname{BWT}[y \dots y']$  into blocks corresponding to all  $\tilde{Y}$ , and sorting the symbols in each block. If  $\operatorname{BWT}[y \dots y']$  initially contains k runs, this adds at most 2(k-1) new runs.

<sup>&</sup>lt;sup>3</sup>Here, we utilize the assumption that T[n] =\$. For this reason, if Y contains \$\$, then  $T^{\infty}[i \dots i+|Y|) = Y$  for at most one index  $i \in [1 \dots n]$ .

Let  $\operatorname{BWT}_{\ell}[y \, . \, y'] = c^{\delta} \in \mathbb{N}^+$  be a run in  $\operatorname{BWT}_{\ell}$ . Since  $T^{\infty}[c \, . \, c + \ell)$  is left-maximal, we have  $c + \ell \leq n$ . Let  $Y = T[c \, . \, c + \ell)$ . By  $3\tau \leq \ell$  and  $\mathsf{R} = \emptyset$ , we obtain  $[c \, . \, c + \tau) \cap \mathsf{S} \neq \emptyset$ . Moreover, if Y = XX' is such that  $|X| = \Delta$ , where

$$c + \Delta = \min(\mathsf{S} \cap [c \dots c + \tau)),$$

then  $D'_Y = X$ . Since we also have  $|X'| \leq 2\ell \leq 7\tau$  (due to  $q \geq 4$ ), Lemma 5.27 holds for Y. Let  $[b \, . \, b']$  be the range of  $B_{\overline{X}}$  corresponding to Y through Lemma 5.27. Then,  $|\operatorname{RL}(B_{\overline{X}})| > 1$ . Moreover:

- For every  $j \in [0..\delta)$  such that  $\operatorname{BWT}_{2\ell}[y+j] \in \Sigma$ , it holds:  $\operatorname{BWT}_{2\ell}[y+j] = B_{\overline{X}}[b+j]$ . To see this, apply Lemma 5.27 to all strings  $\widetilde{Y} \in \mathcal{Y} := \{T^{\infty}[\operatorname{SA}[j] .. \operatorname{SA}[j] + 2\ell) : j \in [y .. y']\}$ ordered lexicographically. Since  $D_{\widetilde{Y}} = D_Y$ , the corresponding ranges  $[\widetilde{b} .. \widetilde{b}']$  form a left-toright partition of [b .. b']. Thus, if  $\widetilde{Y}$  is not left-maximal, its BWT block is  $\operatorname{BWT}[\widetilde{y} .. \widetilde{y}'] = B_{\overline{X}}[\widetilde{b} .. \widetilde{b}']$ .
- On the other hand, if  $c' = BWT_{2\ell}[y+j] \in \mathbb{N}$  holds for some  $j \in [0..\delta)$ , then the string  $B_{\overline{X}}[\tilde{b}..\tilde{b}']$  corresponding (through Lemma 5.27) to  $\widetilde{Y} = T^{\infty}[c'..c'+2\ell) \in \mathcal{Y}$  is not unary, i.e., there exists an index  $\hat{b} \in [\tilde{b}..\tilde{b}')$  satisfying  $B_{\overline{X}}[\hat{b}] \neq B_{\overline{X}}[\hat{b}+1]$ , and  $lcp(W'[I_{\overline{X}}[\hat{b}]])$ ,  $W'[I_{\overline{X}}[\hat{b}+1]]) \geq 2\ell |X|$ . The converse is also true: if  $\hat{b} \in [b..b')$  satisfies the two conditions, then  $BWT_{2\ell}[y+(\hat{b}-b)] \in \mathbb{N}$ . Consequently, the set of left-maximal strings in  $\mathcal{Y}$  is

$$\begin{aligned} \{X \cdot W'[I_{\overline{X}}[\hat{b}]][1 \dots 2\ell - |X|] : \\ \hat{b} \in [b \dots b'), \ B_{\overline{X}}[\hat{b}] \neq B_{\overline{X}}[\hat{b} + 1], \text{ and} \\ \log(W'[I_{\overline{X}}[\hat{b}]], W'[I_{\overline{X}}[\hat{b} + 1]]) \ge 2\ell - |X|\}. \end{aligned}$$

Moreover, letting  $\widetilde{Y} = X \cdot W'[I_{\overline{X}}[\hat{b}]][1 \dots 2\ell - |X|]$  for any  $\hat{b}$  satisfying the above conditions, the range  $[\widetilde{y} \dots \widetilde{y}'] = \{j \in [1 \dots n] : T^{\infty}[\mathrm{SA}[j] \dots \mathrm{SA}[j] + 2\ell) = \widetilde{Y}\}$  satisfies  $[\widetilde{y} \dots \widetilde{y}'] = y - b + [\widetilde{b} \dots \widetilde{b}']$ , where  $B_{\overline{X}}[\widetilde{b} \dots \widetilde{b}']$  corresponds to  $\widetilde{Y}$  through Lemma 5.27.

The algorithm processing a run  $\operatorname{BWT}_{\ell}[y \, . \, y'] = c^{\delta} \in \mathbb{N}^+$  is thus as follows. Letting  $Y = T[c \, . \, c + \ell)$  and  $X = D'_Y$ , we first compute |X| and the pointer to  $v_{\overline{X}}$ . We then perform a single forward and backward search to find the ranges  $[f \, . \, f']$  and  $[b \, . \, b']$  for Y. Given these, the computation of  $\operatorname{BWT}_{2\ell}[y \, . \, y']$  is achieved by a series of forward and backward searches (at most one per run of  $B_{\overline{X}}[b \, . \, b']$ ).

The length |X| is computed using  $\operatorname{comp}_7(\mathsf{S})$ . The pointers to  $v_{\overline{X}}$  are precomputed since  $|\operatorname{RL}(B_{\overline{X}})| > 1$  implies  $v_{\overline{X}} \in V(\mathcal{T}_c)$ . To implement a forward search, we use LCE queries on T. A backward search is implemented using primary index queries on the wavelet tree of W. We thus obtain:

**Proposition 5.28.** Let T be a string of length n, and let  $\ell = 2^q$  be such that  $q \in [4 \dots \lceil \log n \rceil)$ . If  $\mathsf{R} = \emptyset$ , then, given  $\operatorname{RL}(\operatorname{BWT}_{\ell})$  and the LZ77 parsing of T, the sequence  $\operatorname{RL}(\operatorname{BWT}_{2\ell})$  can be constructed in  $\mathcal{O}((r+z)\log^5 n)$  time.

Proof. Let  $\tau = \lfloor \frac{\ell}{3} \rfloor$ . We start by constructing the compressed representation  $\operatorname{comp}_7(\mathsf{S})$  of a  $\tau$ -synchronizing set  $\mathsf{S}$  of T satisfying  $|\operatorname{comp}_7(\mathsf{S})| = \mathcal{O}(z)$ . By Theorem 5.10, this takes  $\mathcal{O}(z \log^5 n)$  time. Next we construct the compact representations of  $\widetilde{W}$ , W, and W'. By Lemmas 5.24 and 5.25, they need  $\mathcal{O}(z)$  space and can be built in  $\mathcal{O}(z \log^4 n)$  time. Lastly, we construct the compressed wavelet tree  $\mathcal{T}_c$  for W, augmented with the data structure supporting primary index queries. By Theorem 5.22, this takes  $\mathcal{O}(z \log^2 n)$  time.

Let  $\operatorname{BWT}_{\ell}[y \ldots y'] = c^{\delta} \in \mathbb{N}^+$  be one of the runs in  $\operatorname{BWT}_{\ell}$ . As noted earlier, this implies  $c+\ell \leq n$ . By  $3\tau \leq \ell$  and  $\mathsf{R} = \emptyset$ , we then obtain  $\mathsf{S} \cap [c \ldots c+\tau) \neq \emptyset$ . Denote  $Y = T[c \ldots c+\ell) = XX'$ , with  $X = D'_Y$ . Then:

- 1. By  $c = \operatorname{lpos}(Y)$ , we have  $\mathsf{S} \cap [c \dots c + \tau) \subseteq \operatorname{comp}_7(\mathsf{S})$  and get  $|X| = \min(\mathsf{S} \cap [c \dots c + \tau)) c$ in  $\mathcal{O}(\log n)$  time.
- 2. To compute the pointer to  $v_{\overline{X}}$ , we store a hash table than maps  $(|Z|, \overline{\text{lpos}}(Z))$  to  $v_Z$  for every  $v_Z \in V(\mathcal{T}_c)$ . Such table is easily initialized in  $\mathcal{O}(z \log^5 n)$  time during the construction of  $\mathcal{T}_c$ . Thus, using Theorem 6.12, we obtain a pointer to  $v_{\overline{X}}$  in  $\mathcal{O}(\log^4 n)$  time.
- 3. To find the range  $[f \dots f']$  (i.e., the forward search), we use the fact that W' is lexicographically sorted. Thus, by Theorem 6.3, we only need  $\mathcal{O}(\log^2 n)$  time.
- 4. Lastly, to find the range  $[b \, . \, b']$  (i.e., the backward search), we use primary index queries on  $\mathcal{T}_c$ . Using binary search and the data structure from Theorem 5.22, computing  $[b \, . \, b']$  takes  $\mathcal{O}(\log^5 n)$  time.

We then proceed to the construction of  $\operatorname{RL}(\operatorname{BWT}_{2\ell}[y \dots y'])$ . We begin by initializing the output to  $\operatorname{RL}(B_{\overline{X}}[b \dots b'])$ . By the above discussion, it remains to identify all left-maximal  $\widetilde{Y} \in \mathcal{Y}$ and replace (in the output) the range  $[\widetilde{y} \dots \widetilde{y}'] \subseteq [y \dots y']$  corresponding to each such  $\widetilde{Y}$  with  $(\operatorname{lpos}(\widetilde{Y}))^{\widetilde{y}'-\widetilde{y}+1}$ .

Using  $\operatorname{RL}(B_{\overline{X}}[b \dots b'])$ , and LCE and primary index queries we find all  $\hat{b} \in [b \dots b')$  satisfying  $B_{\overline{X}}[\hat{b}] \neq B_{\overline{X}}[\hat{b}+1]$  and  $\operatorname{lcp}(W'[I_{\overline{X}}[\hat{b}]], W'[I_{\overline{X}}[\hat{b}+1]]) \geq 2\ell - |X|$ . For every such  $\hat{b}$ , we find the ranges  $[\tilde{f} \dots \tilde{f}']$  and  $[\tilde{b} \dots \tilde{b}']$ , corresponding to the left-maximal  $\tilde{Y} = X \cdot W'[I_{\overline{X}}[\hat{b}]][1 \dots 2\ell - |X|]$  as above (using LCE and primary index queries). To compute  $\operatorname{lpos}(\tilde{Y})$  we use Theorem 6.11, noting that  $\tilde{Y}$  occurs in  $\widetilde{W}[I_{\overline{X}}[\hat{b}]]$ .

Initializing the auxiliary indexes (Theorems 6.3, 6.11 and 6.12) takes  $\mathcal{O}(z \log^4 n)$  time. By Lemma 5.27 we spend at most  $\mathcal{O}(\log^5 n)$  time per run of BWT<sub>2l</sub>. Thus, by Lemma 5.2, we spend  $\mathcal{O}(r \log^5 n)$  time overall.

#### 5.3.2 The General Case

In this section, we show how to extend the algorithm from the previous section to allow  $\mathsf{R} \neq \emptyset$ .

We start by observing, that the only place where we used the assumption  $\mathsf{R} = \emptyset$  in Proposition 5.28 is to deduce that for any run  $\mathrm{BWT}_{\ell}[y \, . \, y'] = c^{\delta}$  of  $\mathrm{BWT}_{\ell}$  that satisfies  $c^{\delta} \in \mathbb{N}^+$ , it holds  $\mathsf{S} \cap [c \, . \, c + \tau) \neq \emptyset$ . All lemmas in the previous section hold, however, even when  $\mathsf{R} \neq \emptyset$ . The general algorithm is therefore an extension of the procedure in Proposition 5.28, i.e., all runs  $\mathrm{BWT}_{\ell}[y \, . \, y'] = c^{\delta} \in \mathbb{N}^+$  satisfying  $\mathsf{S} \cap [c \, . \, c + \tau) \neq \emptyset$  are processed as in Proposition 5.28, except rather than computing  $\mathsf{S}$  satisfying  $|\mathrm{comp}_7(\mathsf{S})| = \mathcal{O}(z)$ , we instead compute  $\mathsf{S}$  satisfying  $|\mathrm{comp}_7(\mathsf{S})| = \mathcal{O}(z)$ .

The remaining runs are handled as follows. Let  $\mathsf{R}' := \{j \in \mathsf{R} : j - 1 \notin \mathsf{R}\}$  be a subset of  $\mathsf{R}$ . For  $k \in [1 \dots n]$ , let  $\mathcal{F}_k = \{T^{\infty}[i \dots i + k) : i \in \mathsf{R}\}$  and  $\mathcal{F}'_k := \{T^{\infty}[i \dots i + k) : i \in \mathsf{R}'\} \subseteq \mathcal{F}_k$ .

Lemma 5.29. It holds:

1. 
$$|\mathcal{F}'_{2\ell}| \leq |\text{comp}_7(\mathsf{S})|.$$
  
2. If  $T^{\infty}[i \dots i + 2\ell) = \widetilde{Y} \in \mathcal{F}_{2\ell} \setminus \mathcal{F}'_{2\ell}$  then  $T^{\infty}[i-1] = \widetilde{Y}[\text{per}(\widetilde{Y}[1 \dots 3\tau - 1])]$ 

Proof. 1. Let  $\widetilde{Y} \in \mathcal{F}'_{2\ell}$ . Then, there exists  $j \in \mathsf{R}'$  such that  $T^{\infty}[j \dots j + 2\ell) = \widetilde{Y}$ . Let  $j' = \operatorname{lpos}(T^{\infty}[j-1\dots j+2\ell))$ . Since  $j-1 \in \mathsf{S}$ , by the consistency condition, we have  $j' \in \mathsf{S}$ . Moreover, since for  $\ell \geq 16$  we have  $2\ell + 1 \leq 7\tau$  (recall, that  $\tau = \lfloor \frac{\ell}{3} \rfloor$ ), it also holds  $j' \in \operatorname{comp}_7(\mathsf{S})$ . This proves the claim as this mapping from  $\mathcal{F}'_{2\ell}$  to  $\operatorname{comp}_7(\mathsf{S})$  is injective.

2. By definition of  $\mathcal{F}'_{2\ell}$ ,  $\widetilde{Y} \in \mathcal{F}_{2\ell} \setminus \mathcal{F}'_{2\ell}$  implies i > 1 and  $i - 1 \in \mathsf{R}$ . We also have  $i \in \mathsf{R}$ . This implies that  $p := \operatorname{per}(T^{\infty}[i-1..i+3\tau-2))$  and  $p' := \operatorname{per}(T^{\infty}[i..i+3\tau-1)) = \operatorname{per}(\widetilde{Y}[1..3\tau-1])$  satisfy  $p, p' \leq \frac{1}{3}\tau$ . By periodicity lemma, p = p' and hence  $T^{\infty}[i-1] = T^{\infty}[i-1+p'] = \widetilde{Y}[p']$ .  $\Box$ 

Let  $\operatorname{BWT}_{\ell}[y \dots y'] = c^{\delta} \in \mathbb{N}^+$  be a run in  $\operatorname{RL}(\operatorname{BWT}_{\ell})$ . Let  $Y = T[c \dots c + \ell)$  and assume  $\mathsf{S} \cap [c \dots c + \tau) = \emptyset$ . Consider  $\widetilde{Y} \in \mathcal{Y} := \{T^{\infty}[\operatorname{SA}[i] \dots \operatorname{SA}[i] + 2\ell) : i \in [y \dots y']\}$ . Let moreover  $[\widetilde{y} \dots \widetilde{y}'] \subseteq [y \dots y']$  be the range of all indices i such that  $T^{\infty}[\operatorname{SA}[i] \dots]$  starts with  $\widetilde{Y}$  for  $i \in [\widetilde{y} \dots \widetilde{y}']$ . Observe, that  $\widetilde{Y} \in \mathcal{F}_{2\ell}$  and thus, by the above lemma, whenever  $\widetilde{Y} \in \mathcal{F}_{2\ell} \setminus \mathcal{F}'_{2\ell}$ , we have  $\operatorname{BWT}_{2\ell}[\widetilde{y} \dots \widetilde{y}'] = \widetilde{Y}[p]^{\widetilde{y}' - \widetilde{y} + 1} = Y[p]^{\widetilde{y}' - \widetilde{y} + 1}$ , where  $p = \operatorname{per}(\widetilde{Y}[1 \dots 3\tau - 1]) = \operatorname{per}(Y[1 \dots 3\tau - 1])$ , i.e.,  $\operatorname{BWT}_{2\ell}[\widetilde{y} \dots \widetilde{y}']$  is a unary string consisting of the symbol that depends only on Y.

With this in mind, whenever during the processing of  $\operatorname{RL}(\operatorname{BWT}_{\ell})$  in the algorithm of Section 5.3.1 we encounter a run  $\operatorname{BWT}_{\ell}[y \, . \, y'] = c^{\delta} \in \mathbb{N}^+$  such that  $\mathsf{S} \cap [c \, . \, c + \tau) = \emptyset$  (to recognize such runs, we can simply check if  $\mathsf{S} \cap [c \, . \, c + \tau) = \emptyset$ , since by definition of  $\operatorname{BWT}_{\ell}$ , the occurrence  $T[c \, . \, c + \ell)$  is the leftmost in T), we set (letting  $Y = T[c \, . \, c + \ell)$ )

$$BWT_{2\ell}[y ... y'] = Y[per(Y[1 ... 3\tau - 1])]^{y'-y+1}.$$

The resulting BWT<sub>2ℓ</sub> is correct, except for SA ranges corresponding to  $\widetilde{Y} \in \mathcal{F}'_{2\ell}$ . Since there are at most  $|\text{comp}_7(\mathsf{S})| = \mathcal{O}(z)$  such  $\widetilde{Y}$ , we can individually find each such range and correct the corresponding symbols of BWT<sub>2ℓ</sub>.

In the rest of the section we focus on implementing the correction algorithm. Observe first that to compute  $\mathcal{F}'_{2\ell}$ , it suffices to iterate through positions in  $\operatorname{comp}_9(\mathsf{S})$ . By the density condition, whenever for adjacent elements  $s_i < s_{i+1}$  we have  $s_{i+1} - s_i > \tau$ , then  $s_i + 1 \in \mathsf{R}'$ . For each such  $s_i$ , we collect a symbol-string pair  $(T[s_i], T^{\infty}[s_i + 1 \dots s_i + 1 + 2\ell))$ . The resulting collection contains  $\mathcal{O}(z)$  pairs. If we now sort it according to the strings, we can easily tell whether the range BWT $[\tilde{y} \dots \tilde{y}']$  corresponding to each  $\tilde{Y} \in \mathcal{F}'_{2\ell}$  is uniform or not. If so, we know the preceding symbol and the frequency of  $\tilde{Y}$  is obtained using Theorem 6.21. Otherwise we only need the frequency. The main challenge in the correction phrase is to compute the starting index  $\tilde{y}$  of the range in SA corresponding to each  $\tilde{Y} \in \mathcal{F}'_{2\ell}$ . More precisely, for each  $X \in \mathcal{F}'_{2\ell}$  it suffices to compute a *local rank*  $r_X = |\operatorname{pos}(\tilde{Y})|$ , where<sup>4</sup>

$$\operatorname{pos}(X) := \Big\{ j \in \mathsf{R} : T^{\infty}[j \dots j + 2\ell) \prec X \text{ and } T^{\infty}[j \dots j + \ell) = X[1 \dots \ell] \Big\}.$$

The outline of the algorithm for computing all  $r_X$  is similar to [45, Section 6.1.2]. However, nearly every step needs to be redesigned to use only  $\mathcal{O}(z \operatorname{polylog} n)$  space. Motivated by Lemma 5.29, we focus on the properties of runs of consecutive positions in R. We first partition such runs into classes where the computation can be done independently.

**Equivalent Runs** We say that two positions  $i, j \in \mathbb{R}$  are  $\mathbb{R}$ -equivalent if there exists a primitive string H of length  $|H| \leq \frac{1}{3}\tau$  such that both  $T[i \dots i + \tau)$  and  $T[j \dots j + \tau)$  are substrings of  $H^{\infty}$ . It is easy to see that if i and j are  $\mathbb{R}$ -equivalent, then there is exactly  $\operatorname{per}(T[i \dots i + \tau)) = \operatorname{per}(T[j \dots j + \tau))$  choices for H. For each equivalence class of this relation, we designate a unique  $H \in \Sigma^+$  and call it the  $\mathbb{R}$ -root of  $j \in [i]$ , denoting  $\mathbb{R}$ -root(j) = H. Then, i is  $\mathbb{R}$ -equivalent to j if and only if  $\mathbb{R}$ -root $(i) = \mathbb{R}$ -root(j).

Let us now explain how we choose R-roots. Let  $j \in \mathsf{R}$  and let  $p = \operatorname{per}(T[j \dots j + \tau))$ . We define  $\mathsf{R}\operatorname{-root}(j) := T[j' \dots j' + p)$ , where  $j' = \min\{j'' \in \mathsf{R} : j'' \text{ is } \mathsf{R}\operatorname{-equivalent to } j\}$ . Crucially, in the above definition it always holds  $j' \in \mathsf{R}'$ . This allows us to efficiently compute the R-root

<sup>&</sup>lt;sup>4</sup>To ease the notation, in the rest of this section we use X to denote a generic string from  $\mathcal{F}'_{2\ell}$ , but remark that such X corresponds to  $\tilde{Y}$  in Section 5.3.1.

for any  $j \in \mathsf{R}$  as follows. Let  $(F_1, \ldots, F_k)$  be the sequence such that  $\{F_1, \ldots, F_k\} = \mathcal{F}'_{2\tau}$  and  $\operatorname{lpos}(F_1) < \ldots < \operatorname{lpos}(F_k)$ . Consider the string  $T_{\text{root}}$  defined as follows:

$$T_{\text{root}} := T \cdot \left( \bigotimes_{i=0}^{k-1} F_{k-i} \cdot \#_i \right),$$

where  $\#_i$  are distinct sentinel symbols not occurring in T.

Let  $j \in \mathbb{R}$  and assume we computed  $p = \operatorname{per}(T[j \dots j + \tau))$  using Theorem 6.7 (note that here we are guaranteed that  $p \leq \frac{\tau}{3}$  thus computing the exact period is equivalent to a 2-period query). To compute  $\mathbb{R}$ -root(j), we first find the rightmost occurrence  $T_{\operatorname{root}}[j' \dots j' + \tau)$  of  $T[j \dots j + \tau)$  in  $T_{\operatorname{root}}$ . Clearly, we must have j' > |T|. We then binary search b and e such that  $T[j' \dots j' + \tau)$ occurs inside  $T_{\operatorname{root}}[b \dots e] = F_t$  for some  $t \in [1 \dots k]$  and obtain  $\mathbb{R}$ -root $(j) = T_{\operatorname{root}}[b \dots b + p)$ . Moreover,  $\Delta := (p + b - i_{\operatorname{left}}) \mod p < p$  satisfies  $\mathbb{R}$ -root $(j) = T[j + \Delta \dots j + \Delta + p)$ .

**Lemma 5.30.** Given the LZ77 parsing and a string synchronizing set S of text T satisfying  $|\operatorname{comp}_9(S)| = \mathcal{O}(z)$ , we can in  $\mathcal{O}(z \log^4 n)$  time construct a data structure, than given  $j \in \mathbb{R}$ , returns in  $\mathcal{O}(\log^4 n)$  time a value  $\Delta satisfying <math>\mathbb{R}$ -root $(j) = T[j + \Delta \dots j + \Delta + p)$ .

*Proof.* The sequence  $(F_1, \ldots, F_k)$  is easily obtained using  $\operatorname{comp}_9(\mathsf{S})$  and Theorems 6.3 and 6.11. We then construct the LZ77 parsing of  $T_{\text{root}}$  by taking the parsing of T and appending 2k phrases. We then feed the resulting parsing to Theorem 6.12.

The usefulness of R-roots is due to the following two properties. First, if  $j \in \mathbb{R} \setminus \mathbb{R}'$  then R-root $(j) = \mathbb{R}$ -root(j-1). Second, if  $X = T^{\infty}[j \dots j + 2\ell)$  where  $j \in \mathbb{R}'$ , then every  $j' \in \text{pos}(X)$ satisfies  $\mathbb{R}$ -root $(j') = \mathbb{R}$ -root(j). Thus, we can first partition the set of runs in  $\mathbb{R}$  according to  $\mathbb{R}$ -roots and then, to compute pos(X), it suffices to search through j' with the same  $\mathbb{R}$ -root as j.

**Equivalent Positions** For  $j \in \mathbb{R}$ , let  $\alpha_j = \min\{j' \ge j : j' \notin \mathbb{R}\} + 3\tau - 2$ . Recall [45, Section 6.1.2] that runs of consecutive positions in  $\mathbb{R}$  (and hence in particular positions in  $\mathbb{R}'$ ) are easily identified as follows. Denote  $S = \{s_1, \ldots, s_{n'}\}$ , where  $s_i < s_j$  if i < j, and let  $s_0 = 0$ ,  $s_{n'+1} = n - 2\tau + 2$ . Then, by density condition, it holds:

$$\mathsf{R}' = \{s_i + 1 : i \in [0 \dots n'] \text{ and } s_{i+1} - s_i > \tau\}.$$

Furthermore, whenever  $j-1 = s_i$  for  $j \in \mathsf{R}$ , then  $\alpha_j = s_{i+1} + 2\tau - 1$ . Thus, by [45, Fact 3.2], for any  $j \in \mathsf{R}$ ,  $T[j \dots \alpha_j)$  is the longest prefix of  $T[j \dots n]$  having period  $p = \text{per}(T[j \dots j + 3\tau - 1))$ .

For  $j \in \mathbb{R}$  we define type(j) = +1 if  $T[\alpha_j] \succ T[\alpha_j - p]$  and type(j) = -1 otherwise, where  $p = \operatorname{per}(T[j \dots \alpha_j))$ . Similarly as for R-root, if  $j \in \mathbb{R} \setminus \mathbb{R}'$ , then type $(j) = \operatorname{type}(j-1)$ . Moreover, if  $X = T^{\infty}[j \dots j + 2\ell] \subseteq \mathcal{F}'_{2\ell}$  (where  $j \in \mathbb{R}'$ ) is such that  $\alpha_j - j < 2\ell$  and type(j) = -1 then type(j') = -1 holds for all  $j' \in \operatorname{pos}(X)$ . With the above observations in mind, let  $\mathbb{R}^- = \{j \in \mathbb{R} : \operatorname{type}(j) = -1\}$ ,  $\mathbb{R}^+ = \mathbb{R} \setminus \mathbb{R}^-$ ,  $\mathbb{R}'^- = \mathbb{R}' \cap \mathbb{R}^-$  and  $\mathbb{R}'^+ = \mathbb{R}' \cap \mathbb{R}^+$ . Moreover, let  $\mathcal{F}'_k^- := \{T^{\infty}[j \dots j + k) : j \in \mathbb{R}'^-$  and  $\alpha_j - j < k\}$ . The set  $\mathcal{F}'_{2\ell}^+$  is defined analogously. The above observation can then be stated as follows: if  $X \in \mathcal{F}'_{2\ell}^-$  then any  $j \in \operatorname{pos}(X)$  satisfies  $j \in \mathbb{R}^-$ . Note also that any such j satisfies  $\alpha_j - j < 2\ell$ . We can therefore focus on computing  $r_X$  only for  $X \in \mathcal{F}'_{2\ell}^-$ . The set  $\mathcal{F}'_{2\ell}^+$  is processed separately.

To efficiently use R-roots as a mean of comparing equivalent runs in R, we classify individual positions within a run. Let H = R-root(j) for  $j \in \text{R}$ . Observe that the following R-decomposition  $T[j \dots \alpha_j) = H'H^kH''$  (where  $k \ge 1$ , H' is a proper prefix of H, and H'' is a proper suffix of H) is unique. We call the triple R-sig(j) := (|H'|, k, |H''|) the R-signature of  $j \in \text{R}$  and the value  $\operatorname{R-exp}(j) = k$  its  $\operatorname{R-exponent}$ . Note that |H'| in the R-signature is the value  $\Delta$  computed in Lemma 5.30. It is hence easy to use Lemma 5.30 and Theorem 6.3 to compute  $\alpha_j$  and the R-signature for each  $j \in \operatorname{R}$  in  $\mathcal{O}(\log^4 n)$  time. Moreover, if  $X = T^{\infty}[j \dots j + 2\ell)$  for  $j \in \operatorname{R'}^-$ , then it holds  $\operatorname{R-exp}(j') \leq \operatorname{R-exp}(j)$  for all  $j' \in \operatorname{pos}(X)$ . Thus, letting

$$r_X^{=} := |\{j' \in \operatorname{pos}(X) : \operatorname{\mathsf{R-exp}}(j') = \operatorname{\mathsf{R-exp}}(j)\}|,$$
  
$$r_X^{<} := |\{j' \in \operatorname{pos}(X) : \operatorname{\mathsf{R-exp}}(j') < \operatorname{\mathsf{R-exp}}(j)\}|,$$

we have  $r_X = r_X^{=} + r_X^{<}$ . We will compute the terms separately.

**Computing**  $r_X^{=}$  Denote by  $(z_i)_{i \in [1..|\mathsf{R}'^-|]}$  a sequence containing all  $j \in \mathsf{R}'^-$  sorted first according to their R-root, second (in case of ties) according to |H''| in their R-signature, and third (in case there are still ties) according to suffix  $T[\alpha_j \dots n]$ . We can use this sequence to compute  $r_X^{=}$ , where  $X = T^{\infty}[j \dots j + 2\ell) \in \mathcal{F}_{2\ell}'^{-}$  for  $j \in \mathsf{R}'^-$ , as follows.

Let X = X'X'' where  $|X'| = \alpha_j - j < 2\ell$  (by the consistency of S, the decomposition does not depend on the choice of j). Then, let  $i' \in [1 ... |\mathsf{R}'^-|]$  be the smallest index such that  $j' = z_{i'}$ satisfies  $\mathsf{R}$ -root $(j') = \mathsf{R}$ -root(j) and X'' is a prefix of  $T[\alpha_{j'} ...n]$  (at least one such i' exists since j occurs in  $(z_i)_{i \in [1 ... |\mathsf{R}'^-|]}$ ). By definition of the sequence  $(z_i)_{i \in [1 ... |\mathsf{R}'^-|]}$ , i' can be computed using binary search and LCE queries (Theorem 6.3). The value  $r_X^{=}$  is then equal to the number of indices  $i'' \in [1 ... i')$  such that  $j' = z_{i''}$  satisfies  $\mathsf{R}$ -root $(j') = \mathsf{R}$ -root(j) and the factor  $H'H^k$  in the  $\mathsf{R}$ -decomposition of j' is at least as long as for j. To compute the number of such i' we store a collection of 2D points containing a point for each  $j \in \mathsf{R}'^-$  with its position in  $(z_i)_{i \in [1 ... |\mathsf{R}'^-|]}$  as an x-coordinate and  $|H'H^k|$  in its  $\mathsf{R}$ -decomposition as its y-coordinate. Computing  $r_X^{=}$  then (as explained above) corresponds to a 3-sided orthogonal range counting query [19].

The above algorithm requires storing and searching sets of size  $|\mathsf{R}'^-|$  which can be as large as  $\Omega(\frac{n}{\tau})$ . We observe, however, that the used sequences can be compressed, and furthermore, a compressed representation of this (or more precisely, sufficiently similar) sequence can be computed quickly, given  $\operatorname{comp}_9(\mathsf{S})$  and the LZ77 parsing of T. The key observation is that the relevant for the algorithm information about each  $j \in \mathsf{R}'^-$  is the length- $\Theta(\ell)$  substring of T located around position  $\alpha_j$ . More precisely, we first note that since  $|X''| \leq \ell + 3 \leq 2\ell$ , to compute i' we only access substrings  $T^{\infty}[\alpha_i \dots \alpha_i + 2\ell)$  where  $i \in \mathsf{R}'^-$ . On the other hand, since  $|X'| < 2\ell$ , to determine the R-root and select all  $j' \in \mathsf{R}'^-$  for which the factor  $H'H^k$  in the Rdecomposition is at least as long as for j, it suffices to know  $T[\alpha_i - 2\ell \dots \alpha_i), i \in \mathsf{R}'^-$ . Thus, rather than  $(z_i)_{i\in[1..|\mathsf{R}'^-|]}$ , the above algorithm can be executed on the sequence of strings  $Z[1..|\mathsf{R}'^-|]$ defined as follows. For  $i \in [1..|\mathsf{R}'^-|]$ ,

$$Z[i] := T^{\infty}[\alpha_{z_i} - 2\ell \dots \alpha_{z_i} + 2\ell).$$

Using Z directly is problematic, since  $\operatorname{RL}(Z)$  may have multiple runs of the same string and hence we may have  $|\operatorname{RL}(Z)| = \omega(z \operatorname{polylog} n)$ . We note, however, that  $|\mathcal{Z}| = \mathcal{O}(z)$  holds, where  $\mathcal{Z} := \{Z[i] : i \in [1 ... |\mathsf{R}'^-|]\}$ . To see this, note that if  $T^{\infty}[i ... i + |Z|) = Z$  for some  $Z \in \mathcal{Z}$ , then by consistency condition  $i + 2\ell = \alpha_{i'}$  for some  $i' \in \mathsf{R}$ . Similarly, if  $j \in \mathsf{R}$ , then  $\alpha_j - 2\tau + 1 \in \mathsf{S}$ . Thus,  $i + 2\ell - 2\tau + 1 \in \mathsf{S}$ . In particular,  $j' = \operatorname{lpos}(Z)$  satisfies  $j' + 2\ell - 2\tau + 1 \in \mathsf{S}$ . This mapping is injective, and moreover, since  $|Z| = 4\ell$  and  $2\ell \leq 7\tau$ , we have  $j' + 2\ell - 2\tau + 1 \in \operatorname{comp}_9(\mathsf{S})$ .

Combined with the observation that two equal elements of Z are always either both included or both excluded, when computing  $r_X^{\equiv}$ , we thus instead consider the string sequence Z' defined by  $(z'_i)_{i \in [1..|\mathsf{R}'^-|]}$  containing all elements  $j \in \mathsf{R}'^-$  sorted first according to their R-root, second according to |H''|, third according to the substring  $T^{\infty}[\alpha_j \dots \alpha_j + 2\ell)$ , and finally according to the reversed substring  $T^{\infty}[\alpha_j - 2\ell \dots \alpha_j)$ . Then,  $|\mathrm{RL}(Z')| = \mathcal{O}(z)$  holds, and  $\mathrm{RL}(Z')$  can be computed using LZ77 of T,  $\mathrm{comp}_9(\mathsf{S})$ , and Theorems 6.3 and 6.11. The query algorithm remains unchanged, except each 2D point now represents a run of strings, and hence is augmented with the weight corresponding to its frequency.

**Lemma 5.31.** Let T be a length-n text. Given the LZ77 parsing and a string synchronizing set  $\mathsf{S}$  of T satisfying  $|\operatorname{comp}_9(\mathsf{S})| = \mathcal{O}(z)$ , the set  $\{r_X^{=}\}_{X \in \mathcal{F}_{2\ell}^{'-}}$  can be computed in  $\mathcal{O}(z \log^4 n)$  time.

**Computing**  $r_X^{\leq}$  Let us start with the following inefficient algorithm. Consider sorting all  $j \in \mathsf{R}'^-$  first by  $\mathsf{R}$ -root(j) and then by  $\mathsf{R}$ -exp(j). Let

$$\mathcal{H} := \{\mathsf{R}\operatorname{-root}(j) : j \in \mathsf{R}\}.$$

For  $H \in \mathcal{H}$ , let  $\mathsf{R}_{H}^{-} := \{j \in \mathsf{R}^{-} : \mathsf{R}\text{-}\mathrm{root}(j) = H\}$  and  $\mathsf{R}_{H}^{\prime -} := \mathsf{R}^{\prime} \cap \mathsf{R}_{H}^{-}$ .

Let us fix some  $H \in \mathcal{H}$ . The algorithm processes all elements of  $\mathsf{R}'_H$  in rounds. In round i, we consider all positions in the set  $Q_i := \{j \in \mathsf{R}'_H : \mathsf{R}\text{-}\exp(j) = i\}$ . We execute rounds in increasing order, skipping i for which  $Q_i = \emptyset$ . The algorithm maintains an array  $C[0 \dots |H|)$  that satisfies the following invariant: at the end of round i,  $C[t] = |\{j \in \mathsf{R}^-_{H,t} : \mathsf{R}\text{-}\exp(j) \leq i\}|$ , where  $\mathsf{R}^-_{H,t} := \{j \in \mathsf{R}^-_H : \mathsf{R}\text{-}\operatorname{sig}(j) \in \{t\} \times \mathbb{N}^2\}$ . Round i proceeds as follows.

- 1. First, we look at  $|Q_{i-1}|$  and if  $Q_{i-1} = \emptyset$ , we increment all C[0..|H|) by  $(i i_{\text{prev}} 1) \cdot \sum_{i' \ge i} |Q_{i'}|$ , where  $i_{\text{prev}} = \max\{i' < i : Q_{i'} \neq \emptyset\}$ . This accounts for skipped exponents. After this, it holds  $C[t] = |\{j \in \mathsf{R}^-_{H,t} : \mathsf{R}\text{-}\exp(j) \le i 1\}|$  for all  $t \in [0..|H|)$ .
- 2. Now, iterate through  $X = T^{\infty}[j \dots j+2\ell) \in \mathcal{F}_{2\ell}^{\prime-}$  satisfying  $\mathsf{R}\operatorname{-root}(j) = H$  and  $\mathsf{R}\operatorname{-exp}(j) = i$ . For each such j, we answer  $r_X^{\prime} = C[t] + \dots + C[|H| - 1]$ , where  $\mathsf{R}\operatorname{-sig}(j) \in \{t\} \times \mathbb{N}^2$  (note that  $\mathsf{R}\operatorname{-sig}(j)$  does not depend on the exact choice of j). Since all considered j satisfy  $j \in Q_i$ , there is nothing to do in this step when  $Q_i = \emptyset$ . Hence, skipping such  $Q_i$  is correct.
- 3. Then, for  $j \in Q_i$  add one to all counters in the range to  $C[0 \dots t]$ , where  $\mathsf{R}\text{-sig}(j) \in \{t\} \times \mathbb{N}^2$ . This accounts for  $j \in \mathsf{R}'_H$  having  $\mathsf{R}\text{-exp}(j) = i$ .
- 4. Finally, increment C[0..|H|) by  $\sum_{i'>i} |Q_{i'}|$ , accounting  $j \in \mathsf{R}_H^- \setminus \mathsf{R}_H'^-$  having  $\mathsf{R}\text{-exp}(j) = i$ .

If we represent C using a balanced BST, each update in the algorithm takes  $\mathcal{O}(\log n)$  time.

The above algorithm processes each element  $j \in \mathsf{R}'_H$  separately. To turn it into an efficient algorithm for compressible strings (obtaining a runtime of the form  $\mathcal{O}(z \operatorname{polylog} n)$ ), we first note that the only used information for  $j \in \mathsf{R}'_H$  is  $\operatorname{R-sig}(j)$ . Moreover, two elements with equal  $\operatorname{R-signatures}$  are processed in the same way. We can thus process them together, i.e., in step 3, rather than by one, we increment C[0..t] by the frequency of a given signature. Finally, we note that all  $X = T^{\infty}[j .. j + 2\ell)$  considered in step 2 satisfy  $\alpha_j - j < 2\ell$ , thus we only need to execute rounds up to  $i_{\max} := \lceil 2\ell/|H| \rceil$ . Consequently, the algorithm remains correct if instead of using  $\operatorname{R-sig}(j)$  for all  $j \in \mathsf{R}'^-$ , we use truncated  $\operatorname{R-signatures}$  defined as:

$$\mathsf{R}\text{-}\mathrm{sig}'(j) := \begin{cases} \left(|H'|, k, |H''|\right) & \text{if } k < \left\lceil \frac{2\ell}{|H|} \right\rceil \\ \left(0, \left\lceil \frac{2\ell}{|H|} \right\rceil, |H''|\right) & \text{otherwise,} \end{cases}$$

where  $\mathsf{R}\text{-sig}(j) = (|H'|, k, |H''|)$ . The following lemma ensures that this (assuming truncating signatures is combined with group processing of equal signatures) significantly improves the runtime. Note that we need to count separately for each  $H \in \mathcal{H}$ , as (truncated) R-signatures for different R-roots can be equal.

Lemma 5.32.

$$\sum_{H \in \mathcal{H}} \left| \left\{ \mathsf{R}\text{-sig}'(j) : j \in \mathsf{R}'^{-}_{H} \right\} \right| = \mathcal{O}(z).$$

*Proof.* Let us fix  $H \in \mathcal{H}$ .

Consider first  $j \in \mathsf{R}'_H$  such that  $\mathsf{R}\text{-sig}(j) \in \mathbb{N} \times \{k\} \times \mathbb{N}$ , where  $k < \lceil 2\ell/|H| \rceil$ . We injectively assign  $\mathsf{R}\text{-sig}'(j) = \mathsf{R}\text{-sig}(j)$  to an element of  $\operatorname{comp}_8(\mathsf{S})$ . Let  $i_{\text{left}} = \operatorname{lpos}(T[j-1..\alpha_j+1))$  (taking one symbol past the periodic region ensures that the mapping is injective). By the consistency of  $\mathsf{S}$ , it holds  $i_{\text{left}} \in \mathsf{S}$ . Furthermore, by  $k < \lceil 2\ell/|H| \rceil$ , it holds  $\alpha_j - j + 2 \leq (k+2)|H| \leq 2\ell + 2|H| \leq 8\tau$  (since  $2\ell \leq 7\tau$  and  $|H| \leq \frac{1}{3}\tau$ ). Thus, for some  $t \in [1..z]$ ,  $i_{\text{left}} \in \mathsf{S} \cap (e_t - 8\tau \dots e_t] \subseteq \operatorname{comp}_8(\mathsf{S})$ . This map remains injective even considering all  $H \in \mathcal{H}$ .

Consider now  $j \in \mathsf{R}'_H$  such that  $\mathsf{R}\text{-sig}(j) \in \mathbb{N} \times \{k\} \times \{|H''|\}$ , where  $k \geq \lceil 2\ell/|H| \rceil$ . We again construct an injective map of  $\mathsf{R}\text{-sig}'(j)$  to  $\operatorname{comp}_8(\mathsf{S})$ . Denote  $\ell_{\operatorname{tail}} = |H| \cdot \lceil 2\ell/|H| \rceil + |H''|$ . Let  $i_{\operatorname{left}} = \operatorname{lpos}(T[\alpha_j - \ell_{\operatorname{tail}} \dots \alpha_j + 1))$ . By the consistency condition of  $\mathsf{S}$ , for every  $j \in \mathsf{R}$  it holds  $\alpha_j - 2\tau + 1 \in \mathsf{S}$ . Thus, since  $i_{\operatorname{left}} \in \mathsf{R}$  and  $\alpha_{i_{\operatorname{left}}} = i_{\operatorname{left}} + \ell_{\operatorname{tail}}$ , it holds  $i_{\operatorname{left}} + \ell_{\operatorname{tail}} - 2\tau + 1 \in \mathsf{S}$ . We have  $6\tau \leq 2\ell \leq \ell_{\operatorname{tail}} + 1 \leq 2\ell + 2|H| \leq 8\tau$ . Thus, for some  $t \in [1 \dots z]$ , we have  $e_t - 8\tau < i_{\operatorname{left}} \leq e_t$  and so  $i_{\operatorname{left}} + \ell_{\operatorname{tail}} - 2\tau + 1 \in \mathsf{S} \cap (e_t - 4\tau \dots e_t + 6\tau] \subseteq \operatorname{comp}_8(\mathsf{S})$ . The map remains injective considering all  $H \in \mathcal{H}$ .

Using the above lemma, the set of distinct truncated R-signatures, along with their frequencies, can be constructed from the LZ77 parsing and  $\operatorname{comp}_9(S)$  using Theorems 6.3 and 6.21.

**Lemma 5.33.** Let T be a length-n text. Given the LZ77 parsing and a string synchronizing set  $\mathsf{S}$  of T satisfying  $|\operatorname{comp}_9(\mathsf{S})| = \mathcal{O}(z)$ , the set  $\{r_X^{\leq}\}_{X \in \mathcal{F}_{2e}'}$  can be computed in  $\mathcal{O}(z \log^4 n)$  time.

**Putting Things Together** It remains to explain how to compute  $r_X$  for  $X = T^{\infty}[j \dots j+2\ell) \in \mathcal{F}'_{2\ell}$  where  $j \in \mathsf{R}'$  and  $\alpha_j - j \geq 2\ell$ . We observe that for such X,  $r_X^{=}$  and  $r_X^{<}$  can be obtained during the computation of  $\{r_X^{=}: X \in \mathcal{F}'_{2\ell}\}$  and  $\{r_X^{<}: X \in \mathcal{F}'_{2\ell}\}$ .

More precisely, after completing the last round during the processing of some contiguous subsequence of all  $j' \in \mathsf{R}'_{H}$ , we can set, for every  $X = T^{\infty}[j \dots j+2\ell) \in \mathcal{F}'_{2\ell}$ , such that  $\mathsf{R}\operatorname{-root}(j) = H$  and  $\alpha_j - j \ge 2\ell$ ,  $r_X^{<} := C[t] + \dots + C[|H| - 1]$ , where  $\mathsf{R}\operatorname{-sig}(j) \in \{t\} \times \mathbb{N}^2$ . Similarly, computing  $r_X^{=}$  for each  $X = T^{\infty}[j \dots j+2\ell) \in \mathcal{F}'_{2\ell}$  with  $e_j - j \ge 2\ell$  only requires one query on the 2D orthogonal range counting data structure. Thus, handling all  $X \in \mathcal{F}'_{2\ell} \setminus \left(\mathcal{F}'_{2\ell} \cup \mathcal{F}'_{2\ell}\right)$  is not more expensive than handling  $\mathcal{F}'_{2\ell}$  and  $\mathcal{F}'_{2\ell}^{+}$ .

By combining Proposition 5.28 with Lemmas 5.31 and 5.33, we therefore obtain the following result and consequently the main result of this section.

**Proposition 5.34.** Let T be a string of length n, and let  $\ell = 2^q$  be such that  $q \in [4 \dots \lceil \log n \rceil)$ . Then, given  $\operatorname{RL}(\operatorname{BWT}_{\ell})$  and the LZ77 parsing of T, the sequence  $\operatorname{RL}(\operatorname{BWT}_{2\ell})$  can be constructed in  $\mathcal{O}((r+z)\log^5 n)$  time.

By Proposition 5.23 we can compute  $\text{BWT}_{\ell}$  for  $\ell = 2^q$  and q < 4 in  $\mathcal{O}(z \log^4 n)$  time. For  $q \geq 4$ , we use Proposition 5.34. Thus, by the upper bound  $r = \mathcal{O}(z \log^2 n)$  from Theorem 3.2, we obtain the main result of this section.

**Theorem 5.35.** There exists a Las-Vegas randomized algorithm that, given the LZ77 parsing of a text T of length n, computes its run-length compressed Burrows–Wheeler transform in  $\mathcal{O}((r + z)\log^6 n) = \mathcal{O}(z\log^8 n)$  time.

# 6 Auxiliary Recompression-Based Data Structures

For a context-free grammar  $\mathcal{G}$ , we denote by  $\Sigma_{\mathcal{G}}$  and  $\mathcal{N}_{\mathcal{G}}$ , the set of non-terminals and the set of terminals, respectively. The set of symbols is  $\mathcal{S}_{\mathcal{G}} := \Sigma_{\mathcal{G}} \cup \mathcal{N}_{\mathcal{G}}$ . If the grammar  $\mathcal{G}$  is clear from context, the subscripts  $\mathcal{G}$  might be omitted.

A straight-line grammar (SLG) is a context-free grammar  $\mathcal{G}$  such that:

- each non-terminal  $A \in \mathcal{N}$  has a unique production  $A \to \mathsf{rhs}(A)$ , where  $\mathsf{rhs}(A) \in \mathcal{S}^*$ ,
- the set of symbols S admits a partial order  $\prec$  such that  $B \prec A$  if B appears in  $\mathsf{rhs}(A)$ .

A simple inductive argument shows that, for each symbol  $A \in S$ , the language L(A) consists of a unique string, which we call the *expansion* of A and denote  $\exp(A)$ . In particular, the expansion  $\exp(S)$  of a starting symbol  $S \in S$  is the unique string *represented* by  $\mathcal{G}$ .

The parse tree  $\mathcal{T}(A)$  of a symbol  $A \in \mathcal{S}$  is a rooted ordered tree with each node  $\nu$  associated to a symbol  $s(\nu) \in \mathcal{S}$ . The root of  $\mathcal{T}(A)$  is a node  $\rho$  with  $s(\rho) = A$ . If  $A \in \Sigma$ , then  $\rho$  has no children. If  $A \in \mathcal{N}$  and  $\mathsf{rhs}(A) = B_1 \cdots B_k$ , then  $\rho$  has k children, and the subtree rooted at the *i*th child is (a copy of)  $\mathcal{T}(B_i)$ . The *height*  $\mathsf{height}(A)$  of a symbol  $A \in \mathcal{S}$  is defined as the height of its parse tree  $\mathcal{T}(A)$ . In other words,  $\mathsf{height}(A) = 0$  if  $A \in \Sigma$  and  $\mathsf{height}(A) = 1 + \max_{i=1}^k \mathsf{height}(B_i)$ if  $\mathsf{rhs}(A) = B_1 \cdots B_k$ . The parse tree  $\mathcal{T}_{\mathcal{G}}$  of an SLG  $\mathcal{G}$  is defined as the parse tree  $\mathcal{T}(S)$  of the starting symbol S, and the height of  $\mathcal{G}$  is defined as the height of S.

Each node  $\nu$  of  $\mathcal{T}(A)$  is associated with a fragment  $\exp(\nu)$  of  $\exp(A)$  matching  $\exp(s(\nu))$ . For the root  $\rho$ , we define  $\exp(\rho) = \exp(A)[1..|\exp(A)|]$  to be the whole  $\exp(A)$ . Moreover, if  $\exp(\nu) = \exp(A)[\ell..r)$ ,  $\mathsf{rhs}(s(\nu)) = B_1 \cdots B_k$ , and  $\nu_1, \ldots, \nu_k$  are the children of  $\nu$ , then  $\exp(\nu_i) = \exp(A)[r_{i-1} \ldots r_i)$ , where  $r_i = \sum_{j=1}^i |\exp(B_j)|$  for  $0 \le i \le k$ . This way, the fragments  $\exp(\nu_i)$  form a partition of  $\exp(\nu)$ , and  $\exp(\nu_i)$  matches  $\exp(s(\nu_i))$  (as claimed).

Without loss of generality, we assume that each symbol  $A \in S$  appears as  $s(\nu)$  for a node  $\nu$  of  $\mathcal{T}_{\mathcal{G}}$ ; the remaining symbols can be removed from  $\mathcal{G}$  without affecting the string generated by  $\mathcal{G}$ .

**Straight-Line Programs** We say that a straight-line grammar  $\mathcal{G}$  is in *Chomsky normal form* (CNF) if  $|\mathsf{rhs}(A)| = 2$  holds for each  $A \in \mathcal{N}$ . A straight-line grammar in CNF is called a *straight-line program* (SLP). An SLP  $\mathcal{G}$  of size g (with g symbols) representing a text T of length n can be stored  $\mathcal{O}(g)$  space ( $\mathcal{O}(g \log n)$  bits) with each non-terminal  $A \in \mathcal{N}$  storing  $\mathsf{rhs}(A)$  and  $|\exp(A)|$ . This representation allows for efficiently traversing the parse tree  $\mathcal{T}_{\mathcal{G}}$ : given a node  $\nu$  represented as a pair ( $s(\nu), \exp(\nu)$ ), it is possible to retrieve in constant time an analogous representation of a child  $\nu_i$  of  $\nu$  given its index  $i \in \{1, 2\}$  (among the children of  $\nu$ ) or an arbitrary position T[j] contained in  $\exp(\nu_i)$ .

Rytter [76] provided an efficient algorithm that converts the LZ77 representation of a string into an SLP generating it. Unfortunately, he assumed a weaker *non-self-referential* variant of LZ77, where  $T[i ... i + \ell)$  is a previous factor only if there exists  $i' \in [1 ... i - \ell]$  with LCE $(i, i') \ge \ell$ . In the following theorem, we adapt his methods to the self-referential variant allowing  $i' \in [1 ... i)$ .

**Theorem 6.1.** Given an LZ77-like parsing of a string T[1..n] into f phrases, an SLP  $\mathcal{G}$  of size  $\mathcal{O}(f \log n)$  and height  $\mathcal{O}(\log n)$  generating T can be constructed in  $\mathcal{O}(f \log n)$  time.

*Proof.* Rytter [76, Section 3] defined AVL grammars as SLPs satisfying the following extra condition: if  $\mathsf{rhs}(A) = BC$  for  $A \in \mathcal{N}$ , then  $|\mathsf{height}(B) - \mathsf{height}(C)| \leq 1$ . This guarantees [76, Lemma 1] that  $\mathsf{height}(A) = \mathcal{O}(\log |\exp(A)|)$  holds for every  $A \in \mathcal{S}$ .

The algorithm of [76] builds  $\mathcal{G}$  incrementally: each step involves adding a symbol A with a desired expansion  $\exp(A)$ , as well as a bounded number of auxiliary symbols. In the last step, the starting symbol S with  $\exp(S) = T$  is added. Final post-processing includes pruning the grammar to remove symbols not occurring in  $\mathcal{T}_{\mathcal{G}}$ . Each step is of one of three kinds:

- (a) A new terminal symbol can be added to  $\mathcal{G}$  in  $\mathcal{O}(1)$  time (along with no auxiliary symbols).
- (b) Given two symbols B, C ∈ S, a new symbol A with exp(A) = exp(B) exp(C) can be added to G in O(1+|height(B) - height(C)|) time along with O(|height(B) - height(C)|) auxiliary symbols [76, Lemma 2].

(c) Given a symbol  $A \in \mathcal{S}$  and two positions  $1 \leq i \leq j \leq |\exp(A)|$ , a new symbol B with  $\exp(B) = \exp(A)[i \dots j]$  can be added to  $\mathcal{G}$  in  $\mathcal{O}(1 + \log |\exp(A)|)$  time along with  $\mathcal{O}(\log |\exp(A)|)$  auxiliary symbols [76, Lemma 3 and Theorem 2].

Now, a non-self-referential LZ77-like parsing  $T = F_1 \cdots F_f$  can be processed by iteratively constructing symbols  $A_j$  with  $\exp(A_j) = F_1 \cdots F_j$  for  $j \in [1 \dots f]$ . At each iteration j, a symbol  $B_j$  with  $\exp(B_j) = F_j$  is first constructed. If  $F_j = T[i \dots i + \ell)$  is a previous fragment represented by  $(i', \ell)$  such that  $i' \in [1 \dots i - \ell]$  and  $\operatorname{LCE}(i, i') \geq \ell$ , then  $F_j = (F_1 \cdots F_{j-1})[i' \dots i' + \ell) =$  $\exp(A_{j-1})[i' \dots i' + \ell - 1]$ , so  $B_j$  can be constructed using operation (c). Otherwise  $F_j = T[i]$  is a single character (not occurring in  $F_1 \cdots F_{j-1}$ ), so  $B_j$  can be constructed using operation (a). Finally,  $A_j$  is obtained based on  $A_{j-1}$  and  $B_j$  using operation (b). Consequently iteration jinvolves  $\mathcal{O}(\log |F_1 \cdots F_j|) = \mathcal{O}(\log n)$  new symbols and can be implemented in  $\mathcal{O}(\log n)$  time, for a total of  $\mathcal{O}(f \log n)$  symbols and  $\mathcal{O}(f \log n)$  time across all iterations.

In order to apply the same scheme for a (potentially) self-referential LZ77-like parsing, we need to specify the construction of  $B_j$  for a previous fragment  $F_j = T[i \dots i + \ell)$  represented with  $i' \in (i - \ell \dots i)$  such that  $\text{LCE}(i, i') \geq \ell$ . Note that  $P := T[i' \dots i)$  is then a string period of  $F_j$  and, consequently,  $F_j = P^{\infty}[1 \dots \ell]$ . Thus, we first use operation (c) to construct a symbol  $P_{j,0}$  representing P; this costs  $\mathcal{O}(1 + \log |F_1 \dots F_{j-1}|) = \mathcal{O}(\log n)$  time and auxiliary symbols. Next, for  $k \in [1 \dots \lceil \log \frac{\ell}{|P|} \rceil]$ , we use operation (b) to construct symbols  $P_{j,k}$  representing  $P^{2^k}$ . Note that each application of operation (b) involves  $\mathcal{O}(1)$  time and no auxiliary symbols, for a total of  $\mathcal{O}(\log \frac{\ell}{|P|}) = \mathcal{O}(\log \ell) = \mathcal{O}(\log n)$  time and symbols. Finally, we derive  $B_j$  using operation (c) based on  $F_j = P^{\infty}[1 \dots \ell] = \exp(P_{j,k'})[1 \dots \ell]$ , where  $k' = \lceil \log \frac{\ell}{|P|} \rceil$ ; this costs  $\mathcal{O}(\log |P^{2^{k'}}|) = \mathcal{O}(k' + \log |P|) = \mathcal{O}(\log \ell) = \mathcal{O}(\log n)$  time and symbols. Overall, iteration j still costs  $\mathcal{O}(\log n)$  time and symbols.

**Run-Length Straight-Line Programs** A run-length straight-line program (RLSLP) is a straight-line grammar  $\mathcal{G}$  whose non-terminals can be classified into pairs  $A \to BC$ , where  $B, C \in \mathcal{S}$  and  $B \neq C$ , and powers  $A \to B^k$ , where  $B \in \mathcal{S}$  and  $k \geq 2$  is an integer. Analogously to an SLP, an RLSLP of size g (with g symbols) representing a text T of length n can be stored in  $\mathcal{O}(g)$  space ( $\mathcal{O}(g \log n)$  bits) allowing efficient traversal of the parse tree  $\mathcal{T}_{\mathcal{G}}$ .

**Recompression** Recompression [40] is a technique of computing a small and locally consistent RLSLP  $\mathcal{G}$  representing a given text T. Recompression is based on a sequence of strings  $T_1, \ldots, T_r \in \mathcal{S}^*$  such that  $T_r = S$ , where S is the starting symbol, the string  $T_{j-1}$  for  $2 \leq j \leq r$  can be obtained from  $T_j$  by replacing some non-terminals A in  $T_j$  with  $\mathsf{rhs}(A)$ , and  $T_1 = T$ .

Recompression proceeds iteratively starting from  $T_1 = T$ . As long as  $|T_j| > 1$ , the algorithm partitions  $T_j$  into blocks using one of the following two schemes:

- Run-length encoding: If j is odd, then  $T_j$  is partitioned into maximal blocks consisting of equal symbols (runs).
- Alphabet partitioning: If j is even, then the set of symbols occurring in  $T_j$  is decomposed into *left* symbols and *right* symbols, respectively (implementation of this process differs between variants of recompression). Whenever  $T_j[i]$  is a left symbol and  $T_j[i+1]$  is a right symbol,  $T_j[i \dots i+1]$  forms a length-2 block. The remaining positions form length-1 blocks.

Then, for blocks  $T_j[\ell ...r]$  of length at least 2, new non-terminals A with  $\mathsf{rhs}(A) = T_j[\ell ...r]$  are created. This process is implemented so that matching blocks get the same non-terminal. Finally,  $T_{j+1}$  is obtained from  $T_j$  by *collapsing* each block  $T_j[\ell ...r]$  of length at least 2 to the corresponding non-terminal. When  $|T_j| = 1$ , the procedure terminates (resulting in r := j), and the only symbol of  $T_i$  is declared to be the starting symbol of  $\mathcal{G}$ . We then say that  $\mathcal{G}$  is an *r*-round recompression RLSLP, noting that the height of  $\mathcal{G}$  does not exceed r-1.

A recompression RLSLP can be constructed efficiently based on any LZ77-like representation.

**Theorem 6.2.** There exists an algorithm that, given an LZ77-like parsing of a string T[1..n]into f phrases, constructs in  $\mathcal{O}(f \log^2 n)$  time an  $\mathcal{O}(\log n)$ -round recompression RLSLP  $\mathcal{G}$  of size  $\mathcal{O}(f \log^2 n)$  generating T.

*Proof.* Given the LZ77-like parsing of  $\mathcal{T}$ , the algorithm first uses Theorem 6.1 to represent T as an SLP  $\hat{\mathcal{G}}$  of size  $|\mathcal{S}_{\hat{\mathcal{G}}}| = \mathcal{O}(f \log n)$ . Then, using the algorithm of Jeż [38, 39] (see also [37, Section 4]),  $\hat{\mathcal{G}}$  is converted into an  $\mathcal{O}(\log n)$ -round recompression RLSLP  $\mathcal{G}$  of size  $\mathcal{O}(|\mathcal{S}_{\hat{\mathcal{C}}}|\log n) =$  $\mathcal{O}(f \log^2 n)$ . The running time of the two steps is  $\mathcal{O}(f \log n)$  and  $\mathcal{O}(f \log^2 n)$ , respectively. 

**LCE Queries** I [37] showed that LCE queries can be answered in  $\mathcal{O}(r)$  time in a string T represented with an r-round recompression RLSLP. Since a recompression RLSLP generating the reversed string  $\overline{T}$  can be obtained by reversing  $\mathsf{rhs}(A)$  for each non-terminal A, the same holds for LCE queries in T. Due to Theorem 6.2, this yields the following result.

**Theorem 6.3.** Given an LZ77-like parsing of a string T[1..n] into f phrases, we can in  $\mathcal{O}(f \log^2 n)$  time construct a data structure that supports LCE queries in T and in its reverse  $\overline{T}$ in  $\mathcal{O}(\log n)$  time.

#### Pattern Matching in RLSLPs 6.1

Given a pattern P and a text T, a fragment  $T[j \dots j + |P|)$  matching P is called an occurrence of P in T. We denote the set of starting positions of occurrences of P in T by  $Occ(P,T) = \{j \in \{j \in I\}\}$  $[1 \dots |T| - |P| + 1] : P = T[j \dots j + |P|]$ . In this section, we characterize the occurrences of P in T based on the parse tree  $\mathcal{T}$  of an RLSLP  $\mathcal{G}$  representing T. The underlying concepts originate from multiple works on pattern matching in compressed and dynamic strings [2, 33, 65, 20].

Let  $x = T[\ell \dots r]$  be a non-empty fragment of T. The hook of x, denoted hook(x), is the lowest node  $\nu$  in  $\mathcal{T}$  such that  $\exp(\nu)$  contains x. Equivalently,  $\mathsf{hook}(x)$  is the lowest common ancestor of the leaf representing  $T[\ell]$  and the leaf representing T[r-1] in  $\mathcal{T}$ . For |x| > 1, the anchor of x, denoted  $\operatorname{anch}(x)$ , is the length a of the longest prefix y of x such that  $\operatorname{hook}(y) \neq \operatorname{hook}(x)$ . For |x| = 1, we define  $\operatorname{anch}(x) = 0$ . For a node  $\nu$  of  $\mathcal{T}$  and an integer  $a \in [0..|P|)$ , let  $Occ(P, \nu, a) = \{j \in Occ(P, T) : hook(T[j . . j + |P|)) = \nu \text{ and } anch(T[j . . j + |P|)) = a\}.$  The following observation characterizes this set.

**Observation 6.4.** Let  $\nu$  be node of  $\mathcal{T}$  with  $s(\nu) = A$  and  $\exp(\nu) = T[\ell \dots r)$ , and let  $P = P_L \cdot P_R$ be a non-empty pattern. Then  $Occ(P, \nu, |P_L|) \neq \emptyset$  if and only if one of the following holds:

- 1.  $A \in \Sigma$ ,  $P_L = \varepsilon$ , and  $P_R = A$ . Then  $Occ(P, \nu, |P_L|) = \{\ell\}$ . 2.  $A \to BC$ ,  $P_L \neq \varepsilon$  is a suffix of exp(B), and  $P_R \neq \varepsilon$  is a prefix of exp(C). Then  $Occ(P, \nu, |P_L|) = \{\ell + |\exp(B)| - |P_L|\}.$
- 3.  $A \to B^k$ ,  $P_L \neq \varepsilon$  is a suffix of  $\exp(B)$ , and  $P_R \neq \varepsilon$  is a prefix of  $\exp(B)^{k-1}$ . Then  $\operatorname{Occ}(P,\nu,|P_L|) = \left\{ \ell + i | \exp(B)| |P_L| : i \in \left[1 \dots k \left\lceil \frac{|P_R|}{|\exp(B)|} \right\rceil \right] \right\}.$

Local consistency of recompression RLSLPs implies that the occurrences of any pattern have just a few different anchors and that a set of few *potential anchors* can be computed efficiently.

Lemma 6.5 ([33, Corollary 10.4], [37, Lemma 10]). For a pattern P and a text T represented by an RLSLP  $\mathcal{G}$ , define  $\operatorname{anch}(P,T) = {\operatorname{anch}(T[j \dots j + |P|)) : j \in \operatorname{Occ}(P,T)}$ . If  $\mathcal{G}$  is an r-round recompression RLSLP, then  $|\operatorname{anch}(P,T)| = \mathcal{O}(r)$ . Moreover, given any  $j \in \operatorname{Occ}(P,T)$ , a superset  $\operatorname{Anch}(P) \supseteq \operatorname{anch}(P,T)$  of size  $|\operatorname{Anch}(P)| = \mathcal{O}(r)$  can be computed in  $\mathcal{O}(r)$  time.

Internal Pattern Matching Queries in LZ77-Compressed Strings For a (static) text T, internal pattern matching queries (IPM queries) [51] given two fragments x, y of T with |y| < 2|x|, ask for the occurrences of x contained within y. Due to the assumption |y| < 2|x|, the output Occ(x, y) can be represented using a single arithmetic progression [51, Lemma 2.1]; see also [14, 70]. If T is uncompressed, then IPM queries can be answered in  $\mathcal{O}(1)$  time after  $\mathcal{O}(n)$ -time preprocessing [51, 48]. Combining Theorems 6.2 and 6.3, Observation 6.4, and Lemma 6.5, we can answer IPM queries in LZ77-compressed strings in  $\mathcal{O}(\log^3 n)$  time.

**Theorem 6.6.** Given an LZ77-like parsing of a string T[1..n] into f phrases, we can in  $\mathcal{O}(f \log^2 n)$  time construct a data structure that supports IPM queries in T in  $\mathcal{O}(\log^3 n)$  time.

*Proof.* Our data structure consists of a recompression RLSLP  $\mathcal{G}$  generating T (constructed using Theorem 6.2), and a component for LCE queries in T and in its reverse  $\overline{T}$  (built using Theorem 6.3). Thus, the construction time is  $\mathcal{O}(f \log^2 n)$ .

Given  $x = T[\ell_x \dots r_x)$  and  $y = T[\ell_y \dots r_y)$ , the query algorithm works as follows. First, the algorithm applies Lemma 6.5 to obtain a set  $\operatorname{Anch}(x)$  of potential anchors. Since  $\mathcal{G}$  is an  $\mathcal{O}(\log n)$ -round recompression RLSLP, we have  $|\operatorname{Anch}(x)| = \mathcal{O}(\log n)$ , and this step takes  $\mathcal{O}(\log n)$  time. Next, the algorithm identifies all nodes  $\nu$  in the parse tree  $\mathcal{T}$  for which  $\exp(\nu)$ intersects  $y = T[\ell_y \dots r_y)$  on at least |x| positions. Due to |y| < 2|x|, for each of these nodes  $\nu$ , the fragment  $\exp(\nu)$  contains position  $T[\ell_y + |x| - 1]$ . Therefore, it suffices to traverse the path from the root of T towards the leaf representing  $T[\ell_y + |x| - 1]$  as long as the intersection is at least |x| positions. Consequently, this step takes  $\mathcal{O}(\log n)$  time, and it results in  $\mathcal{O}(\log n)$  nodes  $\nu$ . We call them *potential hooks* because the hook any occurrence of x contained in y must be one of these nodes.

For each potential anchor  $a \in \operatorname{Anch}(x)$  and for each potential hook  $\nu$ , the algorithm constructs  $\operatorname{Occ}(x,\nu,a)$  using Observation 6.4. Each of the three cases depending on  $A := s(\nu)$  is easy to implement since  $\exp(\nu) = T[\ell \dots r)$ ,  $P_L = T[\ell_x \dots \ell_x + a)$ , and  $P_R = T[\ell_x + a \dots r_x)$  are all given as fragments of T. The case of  $A \in \Sigma$  is particularly simple and costs  $\mathcal{O}(1)$  time. The case of  $A \to BC$  requires computing the longest common suffix of  $P_L$  and  $\exp(B) = T[\ell \dots \ell + |\exp(B)|)$ , and the longest common prefix of  $P_R$  and  $\exp(C) = T[\ell + |\exp(B)| \dots r)$ , which reduces to an LCE query in  $\overline{T}$  and T, respectively. Similarly, the case of  $A \to B^k$  requires computing the longest common suffix of  $P_L$  and  $\exp(B) = T[\ell \dots \ell + |\exp(B)|)$ , and the longest common prefix of  $P_R$  and  $\exp(B)^{k-1} = T[\ell + |\exp(B)| \dots r)$ . Summing up, retrieving  $\operatorname{Occ}(x,\nu,a)$  for a potential anchor  $a \in \operatorname{Anch}$  and potential hook  $\nu$  costs  $\mathcal{O}(\log n)$  time.

Finally, the algorithm filters occurrences contained in y, that is, starting in  $[\ell_y ... r_y - |x|]$ , shifts the starting positions by  $\ell_y - 1$  (so that they become relative to y), and takes the union across potential anchors  $a \in \operatorname{Anch}(x)$  and potential hooks  $\nu$ . This post-processing takes  $\mathcal{O}(\log^2 n)$  time; the resulting set  $\operatorname{Occ}(x, y)$  must form an arithmetic progression [51, Lemma 2.1].

Next, we show that 2-period queries [50] can be answered efficiently in grammar-compressed strings. A 2-period query, given a fragment x of T, asks to return per(x) or to report that x is not periodic, that is  $per(x) > \frac{1}{2}|x|$ . If T is uncompressed, then 2-period queries can be answered in  $\mathcal{O}(1)$  time after  $\mathcal{O}(n)$ -time preprocessing [51, 48, 7] using relatively simple tools. Nevertheless, in grammar-compressed strings, it is convenient to derive 2-period queries from IPM queries.

**Theorem 6.7.** Given an LZ77-like parsing of a string T[1..n] into f phrases, we can in  $\mathcal{O}(f \log^2 n)$  time construct a data structure that supports 2-period queries in T in  $\mathcal{O}(\log^3 n)$  time.

*Proof.* The data structure consists of components for LCE queries (Theorem 6.3) and IPM queries (Theorem 6.6), both of which take  $\mathcal{O}(f \log^2 n)$  to construct. The query algorithm, given

 $x = T[\ell ... r)$ , makes an IPM query asking for the occurrences of  $x[1 ... \lceil \frac{1}{2} |x| \rceil] = T[\ell ... \lceil \frac{\ell+r}{2} \rceil)$  in  $x[2 ... |x|] = T[\ell + 1 ... r)$ . If there are no occurrences, then the algorithm reports that x is not periodic. Otherwise, it retrieves the starting position p of the leftmost occurrence and checks if p is a period of x, that is whether x[1 + p ... |x|] = x[1 ... |x| - p], using an LCE $(\ell, \ell + p)$  query in T. If this test succeeds, then the algorithm returns p; if the test fails, the algorithm reports that x is not periodic. This procedure costs  $\mathcal{O}(\log^3 n)$  time, dominated by the IPM query.

The correctness follows from the fact that if  $per(x) \leq \frac{1}{2}|x|$ , then the first two occurrences of  $x[1..\lceil \frac{1}{2}|x\rceil]$  in x start at positions 1 and per(x) + 1. The lack of the intermediate occurrences is due to the primitivity of x[1..per(x)], which occurs in x[1..2per(x)] just twice, as a prefix and as a suffix, by the synchronization property of primitive strings.

# 6.2 Indexing LZ77-Compressed Texts

In this section, we describe an efficient index that, after preprocessing an LZ77-compressed text T, given a pattern P, represented by its arbitrary occurrence in T, efficiently reports all the occurrences of P in T (returns Occ(P,T)), finds the leftmost occurrence (returns min Occ(P,T)), or counts the occurrences (returns |Occ(P,T)|). While numerous indexes for dictionary-compressed strings have been developed (see [30] for a recent overview), ours satisfies two rare properties: it can be constructed efficiently without decompressing T, and it lets the pattern P to be given by its occurrence in T (rather than in the plain representation). In particular, we are not aware of any index answering counting queries and satisfying either property. On the other hand, in this version of the manuscript we do not optimize the polylog n factors in the running times.

We start with two auxiliary results, and then we proceed to describing our index. For an RLSLP  $\mathcal{G}$  with parse tree  $\mathcal{T}$  and a symbol  $A \in \mathcal{S}$ , let  $\mathsf{nodes}(A)$  denote the set of nodes  $\nu$  in T with  $s(\nu) = A$ . Moreover,  $\mathsf{depth}(\nu)$  denotes the number of nodes on the path from  $\nu$  to the root.

**Lemma 6.8.** Given an RLSLP  $\mathcal{G}$ , we can in  $\mathcal{O}(|\mathcal{S}|)$  time compute, for any symbol  $A \in \mathcal{S}$ , the leftmost node first(A)  $\in$  nodes(A) and the number of nodes count(A) =  $|\mathsf{nodes}(A)|$ . Moreover, we can in  $\mathcal{O}(|\mathcal{S}|)$  time construct a data structure that, for any symbol  $A \in \mathcal{S}$ , enumerates  $\mathsf{nodes}(A)$  in  $\mathcal{O}(\mathsf{depth}(\nu))$  time per each  $\nu \in \mathsf{nodes}(A)$ .

*Proof.* To compute first(A) for each  $A \in S$ , the algorithm performs a pre-order traversal of the parse tree  $\mathcal{T}$ . Upon entering a node  $\nu$ , the algorithm retrieves  $A = s(\nu)$  and checks if first(A) has already been computed. If so, then the algorithm ignores the subtree of  $\nu$ . Otherwise, first(A) is set to  $\nu$ , and the algorithm continues traversing the subtree of  $\nu$ . If  $A \to B^k$ , the algorithm visit only the first child of  $\nu$ , so that the total running time is  $\mathcal{O}(|\mathcal{S}|)$ .

To compute  $\operatorname{count}(A)$  for each  $A \in S$ , the algorithm processes symbols in the  $\succ$  order (if B appears in  $\operatorname{rhs}(A)$ , then B is processed after A). At the beginning,  $\operatorname{count}(A)$  is initialized to 1 if A is the starting symbol and to 0 otherwise. Processing a non-terminal  $A \in \Sigma$  is void. Processing a terminal  $A \to BC$ , the algorithm adds  $\operatorname{count}(A)$  to both  $\operatorname{count}(B)$  and  $\operatorname{count}(C)$ . Processing a terminal  $A \to B^k$ , the algorithm adds  $k \cdot \operatorname{count}(A)$  to  $\operatorname{count}(B)$ . This procedure guarantees that the value  $\operatorname{count}(A)$  becomes correct before A is processed. The running time is  $\mathcal{O}(|\mathcal{S}|)$ .

Preprocessing for enumerating  $\mathsf{nodes}(A)$  consists in listing, for each  $A \in S$ , all the symbols B for which A appears in  $\mathsf{rhs}(B)$ . The query algorithm, given  $A \in S$ , recursively enumerates  $\mathsf{nodes}(B)$  for each such symbol. Next, for each node  $\nu \in \mathsf{nodes}(B)$  and for each position i where A occurs in  $\mathsf{rhs}(B)$ , the algorithm adds the ith child  $\nu_i$  of  $\nu$  to  $\mathsf{nodes}(A)$ . Since  $\mathsf{depth}(\nu_i) = \mathsf{depth}(\nu) + 1$ , the amortized cost of retrieving  $\nu_i \in \mathsf{nodes}(A)$  is  $\mathcal{O}(\mathsf{depth}(\nu_i))$ .

**Lemma 6.9.** Suppose that LCE queries in a text T[1..n] and in its reverse  $\overline{T}$  can be answered in  $\mathcal{O}(\log n)$  time. Given a multiset  $\mathcal{P}$  of  $n^{\mathcal{O}(1)}$  triples (x, y, w), where x and y are fragments of T and w is an integer, we can in  $\mathcal{O}(|\mathcal{P}|\log^2 n)$  time construct a data structure that, given a pair (x',y') of fragments of T answers the following queries regarding the multiset  $\mathcal{W}(x',y') := \{w : (x,y,w) \in \mathcal{P}, x' \text{ is a suffix of } x, \text{ and } y' \text{ is a prefix of } y\}$ : enumerate  $\mathcal{W}(x',y')$  in  $\mathcal{O}(\log^2 n + |\mathcal{W}(x',y')|)$  time, compute  $\min \mathcal{W}(x',y')$  in  $\mathcal{O}(\log^2 n)$  time, and compute  $\sum \mathcal{W}(x',y')$  in  $\mathcal{O}(\log^2 n)$  time.

Proof. Let  $\mathcal{X} = \{x : (x, y, w) \in \mathcal{P}\}$  and  $\mathcal{Y} = \{y : (x, y, w) \in \mathcal{P}\}$ . At construction time, the algorithm orders  $\mathcal{Y}$  lexicographically and  $\mathcal{X}$  according to the lexicographic order of the reversed strings  $\bar{x}$  for  $x \in \mathcal{X}$ . As a result, each  $y \in \mathcal{Y}$  and each  $x \in \mathcal{X}$  is associated to its rank  $\mathsf{rank}_{\mathcal{Y}}(y)$  and  $\mathsf{rank}_{\mathcal{X}}(x)$ , respectively. Using LCE queries in T and in  $\bar{T}$ , this step can be implemented in  $\mathcal{O}(|\mathcal{P}|\log^2 n)$  time. Next, a range searching data structure of Chazelle [19] is constructed with a point  $(\mathsf{rank}_{\mathcal{X}}(x), \mathsf{rank}_{\mathcal{Y}}(y))$  of weight w associated to each  $(x, y, w) \in \mathcal{P}$ . This data structure costs  $\mathcal{O}(|\mathcal{P}|\log |\mathcal{P}|)$  time to build, and it answers range reporting queries in  $\mathcal{O}(\log |\mathcal{P}| + k)$  time (where k is the number of reported points) and semigroup range searching queries in  $\mathcal{O}(\log^2 |\mathcal{P}|)$  time. In particular, if the semigroup operation is + or min, we obtain (weighted) range counting and range minimum queries.

Given a query (x', y'), the algorithm first finds all  $x \in \mathcal{X}$  with suffix x' and all  $y \in \mathcal{Y}$  with prefix y'. Such fragments appear consecutively in  $\mathcal{X}$  and  $\mathcal{Y}$ , respectively (due to the way these multisets are ordered), and the underlying ranges  $\mathcal{X}[\ell_{\mathcal{X}} \dots r_{\mathcal{X}}]$  and  $\mathcal{Y}[\ell_{\mathcal{Y}} \dots r_{\mathcal{Y}}]$  can be found in  $\mathcal{O}(\log^2 n)$  using binary search (with LCE queries in  $\overline{T}$  and T applied in the comparisons). The multiset  $\mathcal{W}(x', y')$  is precisely the multiset of weights of points in the range  $[\ell_{\mathcal{X}} \dots r_{\mathcal{X}}] \times [\ell_{\mathcal{Y}} \dots r_{\mathcal{Y}}]$  in the data structure of [19]. Hence, the algorithm makes a range query to this data structure with this range. Depending on whether the task is to enumerate  $\mathcal{W}(x', y')$ , compute  $\min \mathcal{W}(x', y')$ , or compute  $\sum \mathcal{W}(x', y')$ , this is a range reporting query, a range minimum query, or a range counting query, respectively. In the latter two cases, the answer is forwarded to the output, while in the first case, the algorithm transforms the multiset of (weighted) points to a multiset of weights. The overall query time is  $\mathcal{O}(\log^2 n)$  plus  $\mathcal{O}(|\mathcal{W}(x', y')|)$  in case of enumeration queries.

Next, we describe indexes for reporting all the occurrences and finding the leftmost occurrence.

**Theorem 6.10.** Given an LZ77-like parsing of a string T[1..n] into f phrases, we can in  $\mathcal{O}(f \log^4 n)$  time construct a data structure that, for any pattern P represented by its arbitrary occurrence in T, reports all the occurrences of P in T in  $\mathcal{O}(\log^3 n + |\operatorname{Occ}(P, T)| \log n)$  time.

Proof. Our data structure consists of a recompression RLSLP  $\mathcal{G}$  generating T (constructed using Theorem 6.2), and a component for LCE queries in T and in its reverse  $\overline{T}$  (built using Theorem 6.3). Moreover, the data structure of Lemma 6.8 is constructed on top of  $\mathcal{G}$ , and the data structure of Lemma 6.9 is built with  $\mathcal{P} = \{(\exp(B), \exp(C), A) : A \in \mathcal{N} \text{ and } A \to BC\} \cup$  $\{(\exp(B), \exp(B)^{k-1}, A) : A \in \mathcal{N} \text{ and } A \to B^k\}$ . Here, A on the third coordinate is technically represented by an integer identifier of A. Moreover,  $\exp(B)$  on the first coordinate and  $\exp(C)$  or  $\exp(B)^{k-1}$  on the second coordinate are represented as fragments of T. Such fragments can be retrieved based on  $\nu = \operatorname{first}(A)$ : if  $\exp(\nu) = T[\ell \dots r)$ , then  $T[\ell \dots \ell + |\exp(B)|)$  matches  $\exp(B)$ , and  $T[\ell + |\exp(B)| \dots r)$  matches  $\exp(C)$  or  $\exp(B)^{k-1}$ . Since  $|\mathcal{N}| = \mathcal{O}(f \log^2 n)$ , the overall construction time is  $\mathcal{O}(f \log^4 n)$ .

If the pattern P consists of a single letter  $A \in \Sigma$ , then the query algorithm simply retrieves  $\mathsf{nodes}(A)$  and reports  $\exp(\nu)$  as an occurrence of  $\nu$  for each  $\nu \in \mathsf{nodes}(A)$ . Since the parse tree  $\mathcal{T}$  is of height  $\mathcal{O}(\log n)$ , this takes  $\mathcal{O}(|\operatorname{Occ}(P,T)|\log n)$  time.

In the following, we assume that  $|P| \ge 2$ . The algorithm first uses Lemma 6.5 to construct a set  $\operatorname{anch}(P)$  of  $\mathcal{O}(\log n)$  potential anchors. For each potential anchor  $a \in \operatorname{anch}(P)$ , the occurrences with anchor a are reported independently. Due to  $|P| \ge 2$ , we may assume that anch $(P) \subseteq [1..|P|)$ , so  $P_L = P[1..a]$  and  $P_R = P[a + 1..|P|]$  are both non-empty. Moreover, both  $P_L$  and  $P_R$  are represented as fragments of T. The algorithm makes a reporting query with  $(x',y') = (P_L, P_R)$  to the data structure of Lemma 6.9. Due to the characterization of Observation 6.4, this results in a set of non-terminals  $\mathcal{A} \subseteq \mathcal{N}$  such that  $\operatorname{Occ}(P, a, \nu) \neq \emptyset$  if and only if  $s(\nu) \in \mathcal{A}$ . Next, for every  $A \in \mathcal{A}$ , the algorithm uses Lemma 6.8 to enumerate  $\operatorname{nodes}(A)$ , and for each  $\nu \in \operatorname{nodes}(A)$ , it appends  $\operatorname{Occ}(P, a, \nu)$  to the constructed set  $\operatorname{Occ}(P, T)$ . Since  $\operatorname{Occ}(P, a, \nu) \neq \emptyset$  and since no occurrence is reported multiple times, the overall running time is  $\mathcal{O}(\log^3 n)$  (dominated by  $\mathcal{O}(\log n)$  queries to the data structure of Lemma 6.9) plus  $\mathcal{O}(|\operatorname{Occ}(P,T)|\log n)$  (dominated by enumerating nodes(A) for each  $A \in \mathcal{A}$ ).

**Theorem 6.11.** Given an LZ77-like parsing of a string T[1..n] into f phrases, we can in  $\mathcal{O}(f \log^4 n)$  time construct a data structure that, for any pattern P represented by its arbitrary occurrence in T, returns the leftmost occurrence of P in T in  $\mathcal{O}(\log^3 n)$  time.

*Proof.* The index is the same as in the proof of Theorem 6.10, except for the weights w in triples  $(x, y, w) \in \mathcal{P}$ : for each  $A \in \mathcal{N}$  with  $A \to BC$  or  $A \to B^k$ , the corresponding weight is set to  $\ell + |\exp(B)|$ , where  $T[\ell ... r) = \exp(\operatorname{first}(A))$ . Given that Lemma 6.8 provides  $\operatorname{first}(A)$  for each  $A \in \mathcal{S}$ , the set  $\mathcal{P}$  can still be constructed in  $\mathcal{O}(f \log^2 n)$  time, and the whole index can be constructed in  $\mathcal{O}(f \log^4 n)$  time.

If the pattern P consists of a single letter  $A \in \Sigma$ , then the query algorithm simply returns  $\exp(\operatorname{first}(A))$ ; this costs  $\mathcal{O}(1)$  time. In the following, we assume that  $|P| \geq 2$ . The algorithm first uses Lemma 6.5 to construct a set  $\operatorname{anch}(P)$  of  $\mathcal{O}(\log n)$  potential anchors. For each potential anchor  $a \in \operatorname{anch}(P)$ , the algorithm makes a minimum query with  $(x', y') = (P[1 \dots a], P[a + 1 \dots |P|))$  to the data structure of Lemma 6.9. By Observation 6.4, for every  $A \in \mathcal{N}$  and  $\nu \in \operatorname{nodes}(A)$ , we have that  $\operatorname{Occ}(P, a, \nu) \neq \emptyset$  if and only if x' is a suffix of x and y' is a prefix of y for the triple  $(x, y, w) \in \mathcal{P}$  corresponding to A. Moreover, w - a is then the starting position of the leftmost occurrence of P with anchor a and hook  $\nu \in \operatorname{nodes}(A)$ . Consequently, if the query to the data structure of Lemma 6.9 yields a value p, then the leftmost occurrence of P with anchor a. If  $p = \infty$ , then there are no occurrences with anchor a.) Therefore, the algorithm reports the minimum among the values p - a obtained for various potential anchors  $a \in \operatorname{anch}(P)$  as the starting position of the leftmost occurrence of P in T. The total query time is  $\mathcal{O}(\log^3 n)$ , dominated by  $|\operatorname{anch}(P)| = \mathcal{O}(\log n)$  queries to the data structure of Lemma 6.9.

Similarly, as in Theorem 6.3, since a recompression RLSLP generating the reversed string T can be obtained by reversing  $\mathsf{rhs}(A)$  for each non-terminal A, the above construction yields also an index for finding the rightmost occurrences.

**Theorem 6.12.** Given an LZ77-like parsing of a string T[1..n] into f phrases, we can in  $\mathcal{O}(f \log^4 n)$  time construct a data structure that, for any pattern P represented by its arbitrary occurrence in T, returns the rightmost occurrence of P in T in  $\mathcal{O}(\log^3 n)$  time.

Counting the occurrences is more challenging, because the size  $|\operatorname{Occ}(P, a, \nu)|$  may vary depending on |P|: if  $s(\nu) = A$ ,  $A \to B^k$ ,  $P[1 \dots a]$  is a suffix of  $\exp(B)$ , and  $P[a + 1 \dots |P|)$  is a prefix of  $\exp(B)^{k-1}$ , then  $|\operatorname{Occ}(P, a, \nu)| = k - \left\lceil \frac{|P|-a}{|\exp(B)|} \right\rceil$  according to Observation 6.4. In other words, we have one occurrence for each exponent  $k' \in [1 \dots k)$  such that  $P[a + 1 \dots |P|)$  is a prefix of  $\exp(B)^{k'}$ . A simple solution would be to add to  $\mathcal{P}$  a triple  $(\exp(B), \exp(B)^{k'}, \operatorname{count}(A))$  for each  $A \to B^k$  and each  $k' \in [1 \dots k)$ . However, this may result in  $|\mathcal{P}| = \Theta(n)$  even if  $f = \mathcal{O}(1)$ . Hence, we apply a more sophisticated solution based on [20, Section 7]. The main idea is to classify the fragments of T (and thus also the occurrences of P in T) into *regular* and *special*.

**Definition 6.13.** Consider an RLSLP  $\mathcal{G}$  representing T. We call a fragment x of T regular if |x| = 1 or x overlaps  $\exp(\nu')$  for at most three children  $\nu'$  of  $\operatorname{hook}(x)$ , and special otherwise.

We design separate data structures for counting the regular and special occurrences of P in T. For this, we partition Occ(P,T) into  $Occ^{r}(P,T)$  and  $Occ^{s}(P,T)$ , which contain the starting positions of regular and special occurrences of P in T, respectively. Similarly, we partition  $Occ(P, a, \nu)$  for any potential anchor a and any node  $\nu$  in  $\mathcal{T}$  into  $Occ^r(P, a, \nu)$  and  $Occ^s(P, a, \nu)$ .

Counting Regular Occurrences. Regular occurrences are counted using an index similar to that in Theorems 6.10 and 6.11, based on the following variant of Observation 6.4.

**Observation 6.14.** Let  $\nu$  be node of  $\mathcal{T}$  with  $s(\nu) = A$ , and let  $P = P_L \cdot P_R$  be a non-empty pattern. Then  $\operatorname{Occ}^r(P,\nu,|P_L|) \neq \emptyset$  only if one of the following holds:

- 1.  $A \in \Sigma$ ,  $P_L = \varepsilon$ , and  $P_R = A$ . Then  $|\operatorname{Occ}^r(P, \nu, |P_L|)| = 1$ . 2.  $A \to BC$ ,  $P_L \neq \varepsilon$  is a suffix of  $\exp(B)$ , and  $P_R \neq \varepsilon$  is a prefix of  $\exp(C)$ . Then  $|\operatorname{Occ}^{r}(P,\nu,|P_{L}|)| = 1.$
- 3.  $A \to B^k$ ,  $P_L \neq \varepsilon$  is a suffix of  $\exp(B)$ , and  $P_R \neq \varepsilon$  is a prefix of  $\exp(B)^2$ . Then  $|\operatorname{Occ}^{r}(P,\nu,|P_{L}|)| = k-1$  if  $P_{R}$  is a prefix of  $\exp(B)$  and  $|\operatorname{Occ}^{r}(P,\nu,|P_{L}|)| = k-2$ otherwise.

**Proposition 6.15.** Given an  $\mathcal{O}(\log n)$ -round recompression RLSLP  $\mathcal{G}$  generating a string T[1..n], we can in  $\mathcal{O}(|\mathcal{S}|\log^2 n)$  time construct a data structure that, for any pattern P represented by its arbitrary occurrence in T, returns the number  $|\operatorname{Occ}^r(P,T)|$  of regular occurrences of P in T in  $\mathcal{O}(\log^3 n)$  time.

*Proof.* Recall that  $\mathcal{G}$  allows answering LCE queries in T and its reverse  $\overline{T}$  in  $\mathcal{O}(\log n)$  time [37]. Our index applies Lemma 6.8 on top of  $\mathcal{G}$ , and builds the data structure of Lemma 6.9 with

$$\mathcal{P} = \{ (\exp(B), \exp(C), \mathsf{count}(A)) : A \in \mathcal{N} \text{ and } A \to BC \}$$
$$\cup \{ (\exp(B), \exp(B), \mathsf{count}(A)) : A \in \mathcal{N} \text{ and } A \to B^k \}$$
$$\cup \{ (\exp(B), \exp(B)^2, (k-2)\mathsf{count}(A)) : A \in \mathcal{N} \text{ and } A \to B^k \}$$

As in the proof of Theorem 6.2, the fragments on the first two coordinates can be retrieved based on exp(first(A)) for each  $A \in \mathcal{N}$ . The overall construction time is  $\mathcal{O}(|\mathcal{S}| + |\mathcal{N}|\log^2 n) =$  $\mathcal{O}(|\mathcal{S}|\log^2 n).$ 

If the pattern P consists of a single letter  $A \in \Sigma$ , then the query algorithm simply returns count(A), which takes  $\mathcal{O}(1)$  time. In the following, we assume that  $|P| \geq 2$ . The algorithm first uses Lemma 6.5 to construct a set  $\operatorname{anch}(P)$  of  $\mathcal{O}(\log n)$  potential anchors. For each potential anchor  $a \in \operatorname{anch}(P)$ , the algorithm makes a counting query with  $(x', y') = (P[1 \dots a], P[a +$  $1 \dots |P|$ ) to the data structure of Lemma 6.9. Due to the characterization of Observation 6.14, this results in the total number of regular occurrences of P with anchor a. Finally, the algorithm sums up the results obtained for all potential anchors  $a \in \operatorname{anch}(P)$ . The overall running time is  $\mathcal{O}(\log^3 n)$  (dominated by  $|\operatorname{anch}(P)| = \mathcal{O}(\log n)$  queries to the data structure of Lemma 6.9).  $\Box$ 

Counting Special Occurrences. First, we need to take a detour and prove two properties of recompression RLSLPs. For this, let us extend the expansion function exp so that  $\exp(X) =$  $\exp(X[0])\cdots\exp(X[|X|-1])$  for  $X \in \mathcal{S}^*$ .

**Lemma 6.16.** If  $\exp(A) = \exp(A')$  for  $A, A \in S$  in a recompression RLSLP  $\mathcal{G}$ , then A = A'.

Proof. Recall the strings  $T_1, \ldots, T_h$  in the definition of  $\mathcal{G}$ . We shall inductively prove the following claim: if  $\exp(x) = \exp(x')$  for two fragments of  $T_j$ , then x = x'. The claim holds trivially for j = 1 due to  $x = \exp(x) = \exp(x') = x'$ . Thus, consider j > 1 and suppose that the claim holds for j - 1. Let  $\bar{x}$  and  $\bar{x}'$  be the fragments of  $T_{j-1}$  corresponding to x and x' in  $T_j$ , respectively. Due to  $\exp(\bar{x}) = \exp(x) = \exp(x') = \exp(\bar{x}')$ , the inductive assumption yields  $\bar{x} = \bar{x}'$ . Notice that block boundaries between two symbols of  $T_{j-1}$  are placed solely based on the values of these symbols (a boundary is *not* placed between matching symbols in run-length encoding and between a left symbol and a right symbol in alphabet partitioning). Hence, block boundaries within  $\bar{x}$  and  $\bar{x}'$  are placed in the same way. Moreover, block boundaries are placed at the endpoints of  $\bar{x}$  and  $\bar{x}'$  (since they are collapsed to x and x', respectively). Thus, the partition of  $T_{j-1}$  into blocks partitions  $\bar{x}$  and  $\bar{x}'$  into full blocks in the same way. As matching blocks are replaced by the same non-terminal, we conclude that x = x'. This completes the inductive proof.

Without loss of generality suppose that the recompression algorithm creates A before A', and suppose that A appears for the first time in  $T_j$  (so that neither A nor A' appears in  $T_{j'}$  for j' < j). Observe that, for each node  $\nu \in \mathsf{nodes}(A) \cup \mathsf{nodes}(A')$ , the fragment  $\exp(\nu)$  is collapsed to a fragment of  $T_j$  (which gets collapsed to A or A' at some iteration  $j' \ge j$ ). As A appears in  $T_j$ , one of these fragments of  $T_j$  consists of a single symbol A, and by the claim above, this means that all the fragments consist of a single symbol A. We conclude that A = A' because blocks of size 1 are never collapsed to new non-terminals

# **Lemma 6.17.** If $A \to B^k$ in a recompression RLSLP $\mathcal{G}$ , then $\operatorname{per}(\exp(B)^2) = |\exp(B)|$ .

*Proof.* Recall the strings  $T_1, \ldots, T_h$  in the definition of  $\mathcal{G}$ . We shall inductively prove the following claim: if  $per(x) = \frac{1}{2}|x|$  for a fragment x of  $T_j$ , then  $per(exp(x)) = \frac{1}{2}|exp(x)|$ . The claim holds trivially for j = 1 due to  $x = \exp(x)$ . Thus, consider j > 1 and suppose that the claim holds for j-1. Let  $\bar{x}$  be the fragment of  $T_{j-1}$  corresponding to x in  $T_j$ . Note that  $\frac{1}{2}|\bar{x}|$  is a period of  $\bar{x}$ . By periodicity lemma, we therefore have  $per(\bar{x}) = \frac{1}{2k}|\bar{x}|$  for some integer  $k \ge 1$ . Consequently,  $\bar{x}$  can be decomposed into 2k matching fragments  $\bar{y}_1, \ldots, \bar{y}_{2k}$ . Now, consider the block boundaries that the recompression algorithm places within  $T_{j-1}$ . By definition of  $\bar{x}$ , there are block boundaries at the endpoints of  $\bar{x}$ , that is, before  $\bar{y}_1$  and after  $\bar{y}_{2k}$ . Moreover, since  $\frac{1}{2}|x|$ is a period of x, there is a block boundary in the midpoint of  $\bar{x}$ , that is, between  $\bar{y}_k$  and  $\bar{y}_{k+1}$ . However, since the block boundaries between two symbols of  $T_{j-1}$  are placed solely based on the values of these two symbols, we conclude that block boundaries are placed between  $\bar{y}_i$  and  $\bar{y}_{i+1}$  for each  $i \in [1..2k)$ . Thus, the partition of  $T_{i-1}$  into blocks partitions each fragment  $\bar{y}_i$ (for  $i \in [1..2k]$ ) into full blocks the same way. As matching blocks are replaced by the same non-terminal, we conclude that x can be partitioned into 2k matching fragments  $y_1, \ldots, y_{2k}$ , that is,  $\frac{1}{2k}|x|$  is a period of x. Hence, k = 1, and thus  $per(\bar{x}) = \frac{1}{2}|\bar{x}|$ . The inductive assumption yields  $\operatorname{per}(\exp(x)) = \operatorname{per}(\exp(\bar{x})) = \frac{1}{2}|\exp(\bar{x})| = \frac{1}{2}|\exp(x)|$ , which completes the inductive proof.

Suppose that A appears for the first time in  $T_j$  (so that it does not appear in  $T_{j'}$  for j' < j). Let us fix any occurrence of A in  $T_j$ . The corresponding fragment of  $T_{j-1}$  matches  $B^k$ . Applying the claim to its prefix matching  $B^2$ , we conclude that  $per(exp(B)^2) = |exp(B)|$ .

We are now ready to characterize special occurrences just like regular occurrences were characterized in Observation 6.14.

**Lemma 6.18.** Let P be a pattern,  $\nu$  be a node of  $\mathcal{T}$ , and a be an integer. Then  $\operatorname{Occ}^{s}(P,\nu,a) \neq \emptyset$ if and only if  $A = s(\nu)$  is of the form  $A \to B^{k}$ ,  $a \in [1 \dots |P|)$ ,  $\max\left(a, \frac{|P|-a}{k-1}\right) \leq \operatorname{per}(P) < \frac{|P|-a}{2}$ , and  $\exp(B) = P[a+1 \dots a+\operatorname{per}(P)]$ . In this case  $|\operatorname{Occ}^{s}(P,a,\nu)| = k - \lceil \frac{|P|-a}{\operatorname{per}(P)} \rceil$ . Proof. First, consider a special occurrence x of P with anchor a and hook  $\nu$ . Since x overlaps  $\exp(\nu')$  for at least 4 children  $\nu'$  of  $\nu$ , the symbol  $A = s(\nu)$  must be of the form  $A \to B^k$  for  $k \ge 4$ . Observation 6.4 implies that  $P_L := P[1 \dots a]$  is a non-empty suffix of  $\exp(B)$  and  $P_R := P[a + 1 \dots |P|)$  is a non-empty prefix of  $\exp(B)^{k-1}$ . In particular,  $a \in [1 \dots |P|)$  and  $|\exp(B)|$  is a period of P. Moreover, since x overlaps  $\exp(\nu')$  for at least 4 children  $\nu'$  of  $\nu$ , the string  $\exp(B)^2$  must be a proper prefix of  $P_R$ , and hence a substring of P. Due to Lemma 6.17, we have  $\operatorname{per}(\exp(B)^2) = |\exp(B)|$ , and thus  $\operatorname{per}(P) = |\exp(B)|$ . Consequently,  $\exp(B) = P_R[1 \dots \operatorname{per}(P)] = P[a + 1 \dots a + \operatorname{per}(P)]$ . Finally,  $\operatorname{per}(P) \ge a$  since  $P_L$  is a suffix of  $\exp(B)$ ,  $\operatorname{per}(P) \ge \frac{|P|-a}{k-1}$  since  $P_R$  is a prefix of  $\exp(B)^{k-1}$ , and  $\operatorname{per}(P) < \frac{|P|-a}{2}$  since  $\exp(B)^2$  is a proper prefix of  $\exp(B)^{k-1}$ .

It remains to prove the converse implication: if P,  $\nu$ , and a satisfy the conditions in the lemma statement, then  $|\operatorname{Occ}^s(P,\nu,a)| = k - \left\lceil \frac{|P|-a}{\operatorname{per}(P)} \right\rceil > 0$ . Again, let  $P_L := P[1..a]$  and  $P_R := P[a + 1..|P|)$ ; these fragments are non-empty due to  $a \in [1..|P|)$ . As  $\exp(B) = P_R[1..\operatorname{per}(P)]$ , we conclude that  $P_L$  is a suffix and  $P_R$  is a prefix of a sufficiently large power  $\exp(B)^{k'}$ . Due to  $|\exp(B)| \ge \max(|P_L|, \frac{|P_R|}{k-1})$ , in fact,  $P_L$  is a suffix of  $\exp(B)$  and  $P_R$  is a prefix of  $\exp(B)^{k-1}$ . Consequently, Observation 6.4 yields  $\operatorname{Occ}(P,\nu,|P_L|) = \{\ell + i|\exp(B)| - |P_L| : i \in [1..k - \lceil \frac{|P_R|}{|\exp(B)|}\rceil]\}$ , where  $\exp(\nu) = T[\ell..r)$ . Since  $|\exp(B)| < \frac{1}{2}|P_R|$ , the occurrence of P at position  $\ell + i|\exp(B)| - |P_L|$  is special as it overlaps the fragments  $\exp(\nu')$  for at least four children of  $\nu$ : these are  $T[\ell+j|\exp(B)|..\ell + (j+1)|\exp(B))$  for  $j \in \{i-1,i,i+1,i+2\}$ . Thus,  $|\operatorname{Occ}^s(P,\nu,a)| = |\operatorname{Occ}(P,\nu,a)| = k - \lceil \frac{|P_R|}{|\exp(B)|}\rceil = k - \lceil \frac{|P|-a}{\operatorname{per}(P)}\rceil > 0$ .

Next, we extend Lemma 6.8 so that generalized count(B) queries can be answered for  $B \in S$ .

**Lemma 6.19.** Given an RLSLP  $\mathcal{G}$ , we can in  $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$  time construct a data structure that, given a symbol B and a positive integer m, computes the following value in  $\mathcal{O}(\log |\mathcal{S}|)$  time:

$$\operatorname{count}(B,m) := \sum_{A \in \mathcal{S} \, : \, \operatorname{rhs}(A) = B^k \, \text{ for } k > m} (k-m) \cdot \operatorname{count}(A).$$

*Proof.* For each symbol  $B \in S$ , let us define a set K(B) containing 1 and all the exponents k such that  $\mathsf{rhs}(A) = B^k$  for some  $A \in S$ . The construction algorithm builds sets K(B) represented as sorted arrays. Each element  $m \in K(B)$  is augmented with two values

$$\sum_{A \in \mathcal{S} : \mathsf{rhs}(A) = B^k \text{ for } k > m} k \cdot \mathsf{count}(A) \quad \text{and} \quad \sum_{A \in \mathcal{S} : \mathsf{rhs}(A) = B^k \text{ for } k > m} \mathsf{count}(A),$$

computed using a right-to-left scan of K(B), after the counts count(A) are obtained using Lemma 6.8. The overall construction time is  $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$ , dominated by sorting K(B).

The query algorithm, given a symbol B and an integer  $m \ge 1$ , finds the predecessor  $m' \in K(B)$  of m (using binary search) and computes count(B,m) based on the values stored with m':

$$\operatorname{count}(B,m) = \sum_{A \in \mathcal{S} \, : \, \operatorname{rhs}(A) = B^k \, \text{ for } k > m'} k \cdot \operatorname{count}(A) \quad - \quad m \cdot \sum_{A \in \mathcal{S} \, : \, \operatorname{rhs}(A) = B^k \, \text{ for } k > m'} \operatorname{count}(A).$$

This equality holds because, by definition of m' as the predecessor of m in K(B), if  $\mathsf{rhs}(A) = B^k$ , then k > m is equivalent to k > m'. The query time is  $\mathcal{O}(\log |\mathcal{S}|)$  dominated by finding the predecessor.

Finally, we describe our index for counting special occurrences.

**Proposition 6.20.** Given an  $\mathcal{O}(\log n)$ -round recompression RLSLP  $\mathcal{G}$  generating a string T[1..n], we can in  $\mathcal{O}(|\mathcal{S}|\log^2 n)$  time construct a data structure that, for any pattern P represented by its arbitrary occurrence in T, returns the number  $|\operatorname{Occ}^s(P,T)|$  of special occurrences of P in T in  $\mathcal{O}(\log^3 n)$  time.

*Proof.* Recall that  $\mathcal{G}$  allows answering LCE queries in T in  $\mathcal{O}(\log n)$  time [37]. The construction algorithm uses these queries, combined with Lemma 6.8, which yields first(A) for each  $A \in \mathcal{S}$ , to sort the symbols  $A \in \mathcal{S}$  according to the lexicographic order of their expansions  $\exp(A)$ . Additionally, the index contains a component for 2-Period queries (Theorem 6.7) and a component for computing the values  $\operatorname{count}(B, m)$  (Lemma 6.19). The overall construction time is  $\mathcal{O}(|\mathcal{S}|\log^2 n)$ , dominated by sorting  $\mathcal{S}$ , which involves  $\mathcal{O}(|\mathcal{S}|\log |\mathcal{S}|)$  comparisons in  $\mathcal{O}(\log n)$  time each.

The query algorithm first makes a 2-period query on the pattern P. If  $\operatorname{per}(P) > \frac{1}{2}|P|$ , then the algorithm returns 0, which is correct due to the condition  $\operatorname{per}(P) < \frac{|P|-a}{2} < \frac{1}{2}|P|$  in Lemma 6.18. Otherwise, the algorithm uses Lemma 6.5 to construct a set  $\operatorname{anch}(P)$  of  $\mathcal{O}(\log n)$ potential anchors. For each potential anchor  $a \in \operatorname{anch}(P) \cap [1 \dots |P|)$ , the algorithm checks whether  $a \leq \operatorname{per}(P) < \frac{|P|-a}{2}$ . If this test fails, then P has no special occurrences according to Lemma 6.18. If the test succeeds, then the algorithm finds a symbol  $B \in S$  such that  $\exp(B) = P[a + 1 \dots a + \operatorname{per}(P)]$ . For this, the algorithm performs a binary search on the list of symbols (sorted by expansions) with an LCE query applied to implement each comparison. If there is no such symbol B, then P has no special occurrences with anchor a according to Lemma 6.18. If a symbol B exists, it is unique by Lemma 6.16. In this case, the algorithm adds  $\operatorname{count}(B, \lceil \frac{|P|-a}{per(P)} \rceil)$  (obtained using Lemma 6.19) to the count of special occurrences of P. This term represents the number of special occurrences with anchor a, because each node  $\nu$  with  $\operatorname{rhs}(s(\nu)) = B^k$  has a non-empty set  $\operatorname{Occ}^s(P,\nu,a)$  only if  $\frac{|P|-a}{k-1} \leq \operatorname{per}(P)$ , which is equivalent to  $k - 1 \geq \frac{|P|-a}{\operatorname{per}(P)}$  and to  $k > \lceil \frac{|P|-a}{\operatorname{per}(P)} \rceil$ . Moreover, the size of  $\operatorname{Occ}^s(P,\nu,a)$  is then equal to  $k - \lceil \frac{|P|-a}{\operatorname{per}(P)} \rceil$  by Lemma 6.18. Consequently, the algorithm correctly counts special occurrences of P in T. The query time is  $\mathcal{O}(\log^3 n)$  dominated by  $\mathcal{O}(\log n)$  LCE queries made for each potential anchor.

Combining Theorem 6.2 (for transforming an LZ77-like parsing into a recompression RLSLP) with Propositions 6.15 and 6.20 (for counting regular and special occurrences, respectively), we obtain our counting index.

**Theorem 6.21.** Given an LZ77-like parsing of a string T[1..n] into f phrases, we can in  $\mathcal{O}(f \log^4 n)$  time construct a data structure that, for any pattern P represented by its arbitrary occurrence in T, returns the number of occurrences of P in T in  $\mathcal{O}(\log^3 n)$  time.

Acknowledgment. The authors would like to thank Barna Saha for helpful discussions.

# References

- Donald Adjeroh, Tim Bell, and Amar Mukherjee. The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching. Springer, Boston, MA, USA, 2008. doi:10.1007/978-0-387-78909-5.
- [2] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In SODA, pages 819–828, 2000.
- [3] Mai Alzamel, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Waleń, and Wiktor Zuba. Quasi-

linear-time algorithm for longest common circular factor. In *CPM*, pages 25:1–25:14, 2019. doi:10.4230/LIPIcs.CPM.2019.25.

- [4] Amihood Amir, Costas S. Iliopoulos, and Jakub Radoszewski. Two strings at Hamming distance 1 cannot be both quasiperiodic. Inf. Process. Lett., 128:54–57, 2017. doi:10.1016/j.ipl.2017.08.005.
- [5] Diego Arroyuelo, Gonzalo Navarro, and Kunihiko Sadakane. Stronger Lempel-Ziv based compressed text indexing. *Algorithmica*, 62(1-2):54–101, 2012. doi:10.1007/s00453-010-9443-8.
- [6] Hideo Bannai, Travis Gagie, and Tomohiro I. Refining the r-index. Theor. Comput. Sci., 812:96-108, 2020. doi:10.1016/j.tcs.2019.08.005.
- Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. SIAM J. Comput., 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
- [8] Djamal Belazzougui, Fabio Cunial, Travis Gagie, Nicola Prezza, and Mathieu Raffinot. Composite repetition-aware data structures. In *CPM*, pages 26–39, 2015. doi:10.1007/978-3-319-19929-0\_3.
- [9] Djamal Belazzougui, Fabio Cunial, Travis Gagie, Nicola Prezza, and Mathieu Raffinot. Flexible indexing of repetitive collections. In *CiE*, pages 162–174, 2017. doi:10.1007/978-3-319-58741-7\_17.
- [10] Djamal Belazzougui, Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Queries on LZ-bounded encodings. In DCC, pages 83–92, 2015. doi:10.1109/DCC.2015.69.
- [11] Philip Bille, Mikko Berggren Ettienne, Inge Li Gørtz, and Hjalte Wedel Vildhøj. Timespace trade-offs for Lempel-Ziv compressed indexing. *Theor. Comput. Sci.*, 713:66–77, 2018. doi:10.1016/j.tcs.2017.12.021.
- [12] Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. SIAM J. Comput., 44(3):513–539, 2015. doi:10.1137/130936889.
- [13] Anselm Blumer, Janet A. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. J. ACM, 34(3):578– 595, 1987. doi:10.1145/28869.28873.
- [14] Dany Breslauer and Zvi Galil. Finding all periods and initial palindromes of a string in parallel. Algorithmica, 14(4):355–366, 1995. doi:10.1007/BF01294132.
- [15] Karl Bringmann and Tobias Friedrich. Exact and efficient generation of geometric random variates and random graphs. In *ICALP*, pages 267–278, 2013. doi:10.1007/978-3-642-39206-1\_23.
- [16] Karl Bringmann and Konstantinos Panagiotou. Efficient sampling methods for discrete distributions. Algorithmica, 79(2):484–508, 2017. doi:10.1007/s00453-016-0205-0.
- [17] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.
- [18] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *Trans. Inf. Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.
- [19] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. SIAM J. Comput., 17(3):427–462, 1988. doi:10.1137/0217026.
- [20] Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes, 2019. arXiv:1811.12779.

- [21] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans. Commun.*, 32(4):396–402, 1984. doi:10.1109/TCOM.1984.1096090.
- [22] Gordon V. Cormack and R. Nigel Horspool. Data compression using dynamic Markov modelling. Comput. J., 30(6):541-550, 1987. doi:10.1093/comjnl/30.6.541.
- [23] Thomas M. Cover and Joy A. Thomas. Elements of Information Theory 2nd Edition. Wiley, 2006.
- [24] Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. Proc. Am. Math. Soc., 16(1):109–114, 1965. doi:10.2307/2034009.
- [25] Travis Gagie. Large alphabets and incompressibility. Inf. Process. Lett., 99(6):246–251, 2006.
- [26] Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A faster grammar-based self-index. In LATA, pages 240–251, 2012. doi:10.1007/978-3-642-28332-1\_21.
- [27] Travis Gagie, Giovanni Manzini, Gonzalo Navarro, and Jens Stoye. 25 Years of the Burrows-Wheeler Transform (Dagstuhl Seminar 19241). *Dagstuhl Reports*, 9(6):55–68, 2019. doi:10.4230/DagRep.9.6.55.
- [28] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully-functional suffix trees and optimal text searching in BWT-runs bounded space, 2018. arXiv:1809.02792.
- [29] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. On the approximation ratio of Lempel-Ziv parsing. In LATIN, pages 490–503, 2018. doi:10.1007/978-3-319-77404-6\_36.
- [30] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. J. ACM, 67(1):1–54, apr 2020. doi:10.1145/3375890.
- [31] Jean-loup Gailly and Mark Adler. gzip Homepage. www.gzip.org. Accessed: 2019-10-19.
- [32] John Kenneth Gallant. *String compression algorithms*. PhD thesis, Princeton University, 1982.
- [33] Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings, 2015. arXiv:1511.02612.
- [34] Genomics England. The 100000 Genomes Project. www.genomicsengland.co.uk/about-genomics-england/the-100000-genomes-project. Accessed: 2019-04-13.
- [35] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In SODA, pages 841–850, 2003.
- [36] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. SIAM J. Comput., 13(2):338–355, 1984. doi:10.1137/0213024.
- [37] Tomohiro I. Longest common extensions with recompression. In CPM, pages 18:1–18:15, 2017. doi:10.4230/LIPIcs.CPM.2017.18.
- [38] Artur Jeż. Faster fully compressed pattern matching by recompression. ACM Trans. Algorithms, 11(3):20:1–20:43, 2015. doi:10.1145/2631920.
- [39] Artur Jeż. A really simple approximation of smallest grammar. *Theor. Comput. Sci.*, 616:141–150, 2016. doi:10.1016/j.tcs.2015.12.032.
- [40] Artur Jeż. Recompression: A simple and powerful technique for word equations. J. ACM, 63(1):4:1-4:51, 2016. doi:10.1145/2743014.
- [41] Juha Kärkkäinen, Dominik Kempa, and Marcin Piątkowski. Tighter bounds for the sum of irreducible LCP values. *Theor. Comput. Sci.*, 656:265–278, 2016. doi:10.1016/j.tcs.2015.12.009.
- [42] Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Lazy Lempel-Ziv factorization algorithms. ACM J. Exp. Algor., 21(1):2.4:1–2.4:19, 2016. doi:10.1145/2699876.

- [43] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *CPM*, pages 181–192, 2001. doi:10.1007/3-540-48194-X\_17.
- [44] Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In SODA, pages 1344–1357, 2019. doi:10.1137/1.9781611975482.82.
- [45] Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: Sublinear-time BWT construction and optimal LCE data structure. In STOC, pages 756–767, 2019. doi:10.1145/3313276.3316368.
- [46] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In STOC, pages 827–840, 2018. doi:10.1145/3188745.3188814.
- [47] Takuya Kida, Tetsuya Matsumoto, Yusuke Shibata, Masayuki Takeda, Ayumi Shinohara, and Setsuo Arikawa. Collage system: A unifying framework for compressed pattern matching. *Theor. Comput. Sci.*, 298(1):253–272, 2003. doi:10.1016/S0304-3975(02)00426-7.
- [48] Tomasz Kociumaka. Efficient Data Structures for Internal Queries in Texts. PhD thesis, University of Warsaw, 2018.
- [49] Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive measure of repetitiveness, 2019. arXiv:1910.02151.
- [50] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Efficient data structures for the factor periodicity problem. In SPIRE, pages 284–294, 2012. doi:10.1007/978-3-642-34109-0\_30.
- [51] Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In SODA, pages 532–551, 2015. doi:10.1137/1.9781611973730.36.
- [52] Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theor. Comput. Sci.*, 483:115–133, 2013.
- [53] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memoryefficient alignment of short DNA sequences to the human genome. *Genome Biol.*, 10(3):R25, 2009.
- [54] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinform.*, 25(14):1754–1760, 2009. doi:10.1093/bioinformatics/btp324.
- [55] Ruiqiang Li, Chang Yu, Yingrui Li, Tak Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinform.*, 25(15):1966–1967, 2009. doi:10.1093/bioinformatics/btp336.
- [56] Matt Mahoney. Large Text Compression Benchmark. http://mattmahoney.net/dc/text.html. Accessed: 2020-09-07.
- [57] Matt Mahoney. Adaptive weighing of context models for lossless data compression. Technical Report CS-2005-16, Florida Institute of Technology, Melbourne, Florida, 2005.
- [58] Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. Genomescale algorithm design: Biological sequence analysis in the era of high-throughput sequencing. Cambridge University Press, Cambridge, UK, 2015. doi:10.1017/cbo9781139940023.
- [59] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. SIAM J. Comput., 22(5):935–948, 1993. doi:10.1137/0222058.
- [60] Sabrina Mantaci, Antonio Restivo, Giuseppe Romana, Giovanna Rosone, and Marinella Sciortino. String attractors and combinatorics on words. In *ICTCS*, pages 57–71, 2019.
- [61] Giovanni Manzini. An analysis of the Burrows-Wheeler transform. J. ACM, 48(3):407–430, 2001. doi:10.1145/382780.382782.
- [62] Gonzalo Navarro. Compact data structures: A practical approach. Cambridge University Press, Cambridge, UK, 2016. doi:10.1017/cbo9781316588284.
- [63] Gonzalo Navarro. Indexing highly repetitive string collections, 2020. arXiv:2004.02781.

- [64] Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. Theor. Comput. Sci., 762:41–50, 2019. doi:10.1016/j.tcs.2018.09.007.
- [65] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic index and LZ factorization in compressed space. *Discret. Appl. Math.*, 274:116–129, 2020. doi:10.1016/j.dam.2019.01.014.
- [66] Enno Ohlebusch. Bioinformatics algorithms: Sequence analysis, genome rearrangements, and phylogenetic reconstruction. Oldenbusch Verlag, Ulm, Germany, 2013.
- [67] Enno Ohlebusch, Timo Beller, and Mohamed Ibrahim Abouelhoda. Computing the Burrows-Wheeler transform of a string and its reverse in parallel. J. Discrete Alg., 25:21–33, 2014. doi:10.1016/j.jda.2013.06.002.
- [68] Tatsuya Ohno, Kensuke Sakai, Yoshimasa Takabatake, Tomohiro I, and Hiroshi Sakamoto. A faster implementation of online RLBWT and its application to LZ77 parsing. J. Discrete Alg., 52-53:18–28, 2018. doi:10.1016/j.jda.2018.11.002.
- [69] Igor Pavlov. 7-zip Homepage. https://www.7-zip.org/. Accessed: 2019-10-19.
- [70] Wojciech Plandowski and Wojciech Rytter. Application of Lempel-Ziv encodings to the solution of words equations. In *ICALP*, pages 731–742, 1998. doi:10.1007/BFb0055097.
- [71] Alberto Policriti and Nicola Prezza. From LZ77 to the run-length encoded Burrows-Wheeler transform, and back. In CPM, pages 17:1–17:10, 2017. doi:10.4230/LIPIcs.CPM.2017.17.
- [72] Alberto Policriti and Nicola Prezza. LZ77 computation based on the run-length encoded BWT. Algorithmica, 80(7):1986–2011, 2018. doi:10.1007/s00453-017-0327-z.
- [73] Nicola Prezza. Can Lempel-Ziv and Burrows-Wheeler compression be asymptotically compared? https://nms.kcl.ac.uk/iwoca/problems/Prezza2016\_updated2019.pdf, 2016. IWOCA 2016 Open Problems.
- [74] Nicola Prezza. Optimal rank and select queries on dictionary-compressed text. In CPM, pages 4:1-4:12, 2019. doi:10.4230/LIPIcs.CPM.2019.4.
- [75] Molly Przeworski, Richard R. Hudson, and Anna Di Rienzo. Adjusting the focus on human variation. Trends Genet., 16(7):296–302, 2000. doi:10.1016/S0168-9525(00)02030-8.
- [76] Wojciech Rytter. Application of Lempel–Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1–3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.
- [77] Julian Seward. bzip2 Homepage. www.sourceware.org/bzip2. Accessed: 2019-10-19.
- [78] Sandip Sinha and Omri Weinstein. Local decodability of the Burrows-Wheeler transform. In STOC, pages 744–755, 2019. doi:10.1145/3313276.3316317.
- [79] Jouni Sirén, Niko Välimäki, Veli Mäkinen, and Gonzalo Navarro. Run-length compressed indexes are superior for highly repetitive sequence collections. In SPIRE, pages 164–175, 2008. doi:10.1007/978-3-540-89097-3\_17.
- [80] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. J. Comput. Syst. Sci., 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.
- [81] James A. Storer and Thomas G. Szymanski. The macro model for data compression. In STOC, pages 30–39, 1978. doi:10.1145/800133.804329.
- [82] Axel Thue. Über unendliche zeichenreihen. Norsk. Vid. Selsk. Skr.I, Mat.-Nat.Kl. Nr.7, pages 1–22, 1906.
- [83] Peter Weiner. Linear pattern matching algorithms. In SWAT/FOCS, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.
- [84] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. Trans. Inf. Theory, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.
- [85] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. Trans. Inf. Theory, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.