Resolution of the Burrows-Wheeler Transform Conjecture

Dominik Kempa
University of California, Berkeley, USA
kempa@berkeley.edu

Tomasz Kociumaka
Bar-Ilan University, Ramat Gan, Israel
kociumaka@mimuw.edu.pl

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 \( r = O(z \log^2 n) \) 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 \( O(z \log n \log \log n) \) 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 \( O(z \log n) \)-time algorithm converting the LZ77 parsing into the run-length compressed BWT. To achieve this, we develop a new number of 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.

Keywords—data compression, compressed indexing, Burrows–Wheeler Transform, Lempel–Ziv compression

I. 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 [66]), statistical compression (this includes, for example, context mixing [43], prediction by partial matching (PPM) [16], and dynamic Markov coding [17]), and Burrows–Wheeler transform (BWT) [13]. As seen in the Large Text Compression Benchmark [42], 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 [54] and gzip [26] 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 \( k \)th order empirical entropy \( H_k(T) \) [18]. 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 [38, Lemma 2.6]. Therefore, \( |T^k| H_k(T^k) \approx t \cdot |T| H_k(T) \) for any \( t > 1 \), i.e., entropy is not sensitive to long repetitions, and hence statistical compressors are outperformed by LZ77, when \( k \) is large, i.e., when the goal is to capture long repetitions [20, 25, 34, 38, 62].

The above analysis raises the question about the nature of compressibility of the Burrows–Wheeler transform. The compression of BWT-based compressors, such as bzip2 [60], 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 [47] proved that \( r \) can be upper-bounded in terms of the \( k \)th order empirical entropy of the input string. On the other hand, already in 2008, Sirén et al. [62] observed that BWT achieves excellent compression (superior to statistical methods) on highly repetitive collections and provided probabilistic analysis exhibiting cases when \( r \) is 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 [65] and suffix arrays [45], 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 [1] produced 75 terabytes of DNA [49]. These databases are, however, highly compressible: Github averages 20 versions per project [49], and two human genomes are 99.9% similar [59]. This area has witnessed a remarkable surge of interest in recent years [5], [9], [10], [11], [15], [21], [23], [32], [50], [58], [61]. BWT-based indexes, such as r-index [25], are among the most powerful [23], and their space usage is up to $O(polylog n)$ factors away from the value $r$. For a comprehensive overview of this field, we refer the reader to a survey by Navarro [49].

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 [39], BWA [40], and Soap2 [41]. Modern textbooks spend dozens of pages describing applications of BWT [2], [44], [48], [51]. The richness of these applications has even spawned a dedicated seminar [22]. 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 [24], [34] to produce output whose size is always within a polylog $n$ factor from $z$, the output size of the LZ77 algorithm (e.g., grammar compression [14], collage systems [35], or macro schemes [63]), or larger by a polynomial factor ($n^\varepsilon$ for some $\varepsilon > 0$) in the worst case (e.g., LZ78 [67], compressed word graphs (CDAWGs) [12]).

BWT is known to never compress much better than LZ77, i.e., $z = O(r \log n)$ [24]. The opposite relation (that $r = O(z \log n)$) was generally conjectured to be false. For example, after presenting how to support suffix array and suffix tree queries in $O(r \log n)$ space, Gagie et al. [23] speculate that “(...) it seems unlikely that one can provide suffix array or tree functionality within space related to $a$, $z$, or $r$, since these measures are not related to the structure of the suffix array: this is likely to be a specific advantage of measure $r$.”

1The choice for LZ77 as a representative in this class follows from the fact that most of the other methods are NP-hard to optimize [14], [27], while LZ77 admits a simple linear-time compression algorithm (see, e.g., [30]).

**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 [57] and Gagie et al. [23], [24]. This result alone has multiple implications for indexing and compression:

1. It is possible to support suffix array and suffix tree functionality in $O(z \log n)$ space [25].
2. It was shown in [32] that many string processing tasks (including BWT and LZ77 construction) can be solved in $O(n/\log_2 n + r \log n)$ time (where $\sigma$ is the alphabet size), i.e., if the text is sufficiently compressible by BWT (formally, when $n/r = \Omega(\log n)$), these tasks can be solved in optimal time (which is unlikely to be possible for general texts [33]). Our result loosens this assumption to $n/z = \Omega(\log n)$.
3. Until now, methods based on the Burrows–Wheeler transform were thought to be neither statistical nor dictionary (LZ-like) compression algorithms [62], [23]. Our result challenges the notion that the BWT is its own compression type: In view of our bound, BWT is much closer to LZ compressors than was previously thought.

Our slightly stronger bound $r = O(\delta \log^2 n)$, where $\delta \leq z$ is symmetric (insensitive to string reversal) repetitiveness measure recently introduced in [37], further shows that:

4. The number $r$ of BWT runs in the reverse of the text satisfies $\bar{r} = O(\log^2 n)$, which is the first non-trivial bound in terms of $r$. This result is of practical importance due to many algorithms whose runtime depends on $\bar{r}$ [6], [7], [8], [52], [53], [55], [56].

After proving $r = O(z \log^2 n)$ and $r = O(\delta \log^2 n)$, by a tighter analysis, we obtain $r = O(z \log z \max(1, \log n \frac{n}{\log z}))$ and $r = O(\delta \log \delta \max(1, \frac{n}{\log r}))$, and we prove 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 $O(n \log \delta)$ on the sum of irreducible LCP values. This improves upon the previously known bound $O(n \log r)$ [29].

We then describe an $O(z \log^3 n)$-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 [55], [53]. 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 technique introduced in [33]. We also describe a variant of wavelet trees [28], designed to work for sequences of long strings. Finally (\checkmark), 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 II introduces the basic notation. We present our upper and lower bounds in Sections III and IV. Finally, in Section V, we outline our algorithm converting LZ77 to run-length compressed BWT.
II. Preliminaries

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

An integer $p \in [1..|S|]$ is a period of a string $S$ if $S[i] = S[i+p]$ for $i \in [1..|S| - p]$. The classic periodicity lemma [19] states that if a string $S$ has periods $p, q$ such that $p + q - \gcd(p, q) \leq |S|$, then $\gcd(p, q)$ is also its period.

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

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

Throughout we consider a string (text) $T[1..n]$ of $n \geq 1$ 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 [45] of $T$ is an array $SA[1..n]$ containing a permutation of the integers $[1..n]$ such that $T[SA[1]..n] < T[SA[2]..n] < \cdots < T[SA[n]..n]$, where $<$ denotes the lexicographical order. The closely related Burrows–Wheeler transform [13] $\text{BWT}[1..n]$ of $T$ is defined by $\text{BWT}[i] = T[SA[i]-1]$ if $SA[i] > 1$ and $\text{BWT}[i] = T[n]$ otherwise. The $\text{BWT}$ is invertible: given $\text{BWT}[1..n]$, the text $T$ can be restored in $O(n)$ time. For any string $S = c_1^l c_2^r \cdots c_h^\ell$, where $c_i \in \Sigma$ and $\ell_i > 0$ for $i \in [1..h]$, and $c_i \neq c_{i+1}$ for $i \in [1..h-1]$, we define the run-length encoding of $S$ as a sequence $RL(S) = (\lambda_1, \lambda_2, \ldots, \lambda_h)$, where $\lambda_i = \ell_1 + \cdots + \ell_i$ for $i \in [1..h]$. Throughout, we let $r = |RL(\text{BWT})|$ denote the number of runs in the $\text{BWT}$, e.g., for the string $T = bbabaababababababa$ in Fig. 1, we have $\text{BWT} = ababaabababababaabababa$, and hence $r = 8$.

By $\text{lcp}(S_1, S_2)$ we denote the length of the longest common prefix of strings $S_1$ and $S_2$. We let $\text{lcp}(j_1, j_2) = \text{lcp}(T[j_1..n], T[j_2..n])$, where $j_1, j_2 \in [1..n]$. The $\text{LCP}$ array (see [45], [31]), $\text{LCP}[1..n]$, is defined as $\text{LCP}[i] = \text{lcp}(SA[i], SA[i-1])$ for $i \in [2..n]$ and $\text{LCP}[1] = 0$. We say that the value $\text{LCP}[i]$ is reducible if $\text{BWT}[i] = \text{BWT}[i-1]$ and irreducible otherwise (including $i = 1$). Note that there are $r$ reducible LCP values.

Theorem II.2 (Kärkkäinen et al. [29]). The sum of all irreducible LCP values is at most $n \log r$.

The LZ77 factorization (or the LZ77 parsing) of a string $T$ is then just a greedy, left-to-right parsing of $T$ into longest previous factors. More precisely, if the $j$th LZ factor (called the $j$th phrase) in the parsing is to start at position $i$, then we output $(p_i, \ell_i)$ to represent the $j$th phrase, and then the $(j + 1)$th phrase starts at position $i + \max(\ell_i, 1)$. We denote the number of phrases in the LZ77 parsing by $z$.

The text $bbabaababababababa$ of Fig. 1 has LZ77 factorization $b \cdot b \cdot a \cdot b \cdot a \cdot b \cdot a \cdot b \cdot a \cdot b \cdot a \cdot b \cdot a$ with $z = 8$ phrases, so its LZ77 representation is $(b, 0), (1, 1), (a, 0), (2, 2), (3, 3), (7, 6), (10, 5), (\$, 0)$.

The following relation between $z$ and $r$ is known.

Theorem II.3 (Gagie et al. [24]). Every string of length $n$ satisfies $z = O(r \log n)$.

III. Upper Bounds

A. Basic Upper Bound

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

Lemma III.1. For every $\ell \in [1..n]$, the number of irreducible LCP values in $[\ell \ldots 2\ell)$ is $O(z \log n)$.

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, $T^\infty[1..n] = T[1..n]$. Note that $T^\infty[SA[1]..] \prec \cdots \prec T^\infty[SA[n]..]$ and $\text{BWT}[i] = T^\infty[SA[i]-1]$ for $i \in [1..n]$. Denote $S_m = \{S \in T^\infty : S$ is a substring of $T^\infty\}$ for $m \geq 1$. Observe that $|S_m| \leq mz$ 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
with $i' \in [b\ldots e]$ share a common prefix $S[k + 1 \ldots 3\ell]$ of length $3\ell - k \geq 2\ell$. Consequently, $i = b$ or $i = e + 1$.

**Theorem III.2.** All length-$n$ strings satisfy $r = O(z \log^2 n)$.

**Proof:** Recall that $r$ is the total number of irreducible LCP values. Thus, the claim follows by applying Lemma III.1 for $\ell_i = 2^i$ with $i \in [0 \ldots \log n]$, and observing that the number of LCP values 0 is $\sigma \leq z$.

**B. Tighter Upper Bound**

To obtain a tighter bound, we refine the ideas from Section III-A, starting with a counterpart of Lemma III.1.

**Lemma III.3.** For every $\ell \in [1 \ldots n]$, the number of irreducible LCP values in $[\ell \cdot 2\ell]$ is $O(z \log z)$.

**Proof:** The proof follows closely that of Lemma III.1. However, with each irreducible value $\text{LCP}[i] \in [\ell \cdot 2\ell]$, we associate cost $\lceil \frac{\ell}{2} \rceil$ 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 $[\ell \cdot 2\ell]$ does not exceed $\frac{z}{2}$ times the total cost, which is bounded by $|S_{3\ell}| \cdot 2 \cdot (3 + \log z) \leq 6\ell z \log n$.  

Recall the trie $\mathcal{T}$ of all reversed length-$\ell$ strings in $S_\ell$. 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 $v \neq \text{root}(\mathcal{T})$ to its parent in $\mathcal{T}$ is called light if $v$ has a sibling $v'$ satisfying size($v'$) $\geq$ size($v$) (see Fig. 2). In the proof of Lemma III.1, we observed that the characters $S[k]$ of $S \in S_{3\ell}$ that can be charged correspond to light edges on the root-to-leaf path in $\mathcal{T}$ spelling $S[1 \ldots \ell]$: whenever $S[k]$ with $k \in [1 \ldots \ell]$ is charged, the edge connecting $v_{S[k-1]}$ to its parent $v_{S[k]}$ is light. We then noted that there are at most $\log |S_\ell| \leq \log n$ 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 $S_{3\ell}$ as in Lemma III.1, but only for units $k \in [1 \ldots \lceil \frac{n}{2\ell} \rceil]$. This implies that only characters $S[k]$ of $S \in S_{3\ell}$ with $k \leq \lceil \frac{n}{2\ell} \rceil$ 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 $\lceil \frac{n}{2\ell} \rceil$ and its child.

Consider a light edge from a node $v$ to its parent $u$ at depth at least $\lceil \frac{n}{2\ell} \rceil$. Let $v'$ be a sibling of $v$ satisfying size($v'$) $\geq$ size($v$), and let $S_u, S_{v'}$ be the labels of the paths from the root to $v$ and $v'$. These labels differ on the last position only, so, by Fact II.1, they cannot be both periodic. Let $\widetilde{v} \in \{v, v'\}$ be such that $S_{\widetilde{v}}$ is not periodic, and let $\widetilde{m} = \text{size}(\widetilde{v})$.

Consider the set $S$ of length-$\ell$ strings corresponding to the leaves in the subtree rooted at $\widetilde{v}$ (i.e., labels of the root-to-leaf paths passing through $\widetilde{v}$). Define $\overline{S} := \{P : P \in S\}$ and note that $\overline{S} \subseteq S_\ell$ because $\mathcal{T}$ is the trie of reversed strings from $S_\ell$. Let $e_1 < \cdots < e_{\overline{m}}$ denote the ending positions of the leftmost occurrences in $T^\infty[1 \ldots]$ of strings in $\overline{S}$. By definition, we have an occurrence of $S_{\overline{m}}$ ending in $T^\infty$ at

![Figure 2. The trie $\mathcal{T}$ of reversed length-4 substrings of $T^\infty$ for $T = \text{bbabaabababababa}$ of Fig. 1. Light edges are thin and dotted.](image-url)
every position $e_i$, $i \in [1..\bar{m}]$. Now, $\text{per}(S_i) > \frac{1}{2}|S_i| \geq \frac{1}{2}\ell$ implies that $e_{i+1} - e_i > \frac{1}{2}\ell$ for every $i \in [1..\bar{m}-1]$ (since otherwise the two close occurrences of $S_i$ would yield $\text{per}(S_i) = \text{per}(S_i) \leq \frac{1}{4}\ell$). Consequently, at least $\frac{1}{2}|S| = \frac{1}{2}\bar{m}$ 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 $z \geq \frac{1}{2}\bar{m}$, and therefore $\text{size}(v) \leq \text{size}(\bar{v}) = \bar{m} \leq 4z$.

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

**Theorem III.4.** Every string of length $n$ satisfies $r = O(z \log z \max(1, \log \frac{n}{z \log z}))$.

**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 III.1, except that we observe that the number of light edges on each root-to-leaf path in $T$ is bounded by $\ell$. Thus, the number of irreducible LCP values in $[\ell..2\ell)$ is $O(\ell^2)$. 

2) $\log z < \ell \leq \frac{n}{2}$. We use the bound of Lemma III.3.

3) $\frac{n}{2} < \ell$. We repeat the proof of Lemma III.3, except that we observe that $|S_{\ell}| \leq n$. Thus, the number of irreducible LCP values in $[\ell..2\ell)$ is $O(n \log z \ell)$. 

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

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

$$= O\left( \sigma + \log z + \log z \log z \max(0, \log \frac{n}{z \log z}) \right) + z \log z$$

$$= O\left( \log z \log z \max(1, \log \frac{n}{z \log z}) \right).$$

**C. Upper Bound in Terms of $\delta$**

Let $\delta = \max_{n=1}^{\infty} \frac{1}{m} |S_m|$ denote the substring complexity of $T$ [37]. Note that letting $\delta = \sup_{n=1}^{\infty} \frac{1}{m} |S_m|$ is equivalent, because $|S_m| \leq n$ holds for $m \geq 1$, which implies $\frac{1}{m} |S_m| \leq 1 \leq |S|$, for $m \geq n$. We start by noting that $\delta \leq z$ because $|S_m| \leq mz$ holds for every $m \geq 1$, as observed in the proof of Lemma III.1. Furthermore, $|S_m| \leq m\delta$ holds by definition of $\delta$, so we can replace $z$ with $\delta$ in the proof of Lemma III.1.

To adapt the proof Lemma III.3, we need the following fact that generalizes the observation that $T^{\infty}[1..]$ contains at most $2\ell\ell$ positions covered by the leftmost occurrence of a substring from $S_i$. This is easy to see since the LZ77 parsing yields a set of positions (phrase boundaries) in $T$. The substring complexity $\delta$ does not provide such information.

As shown below, this property nevertheless generalizes to $\delta$. Its proof is a straightforward modification of the argument used in [37, Lemma 10]. For completeness, below we write down the proof, with technical details tailored to our notation (e.g., $S_i$ defined in terms of $T^{\infty}$ rather than $T$).

**Lemma III.5 (based on [37]).** For any positive integer $\ell$, the total number of positions in $T^{\infty}[1..]$ covered by the leftmost occurrences of strings from $S_i$ 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 $S_i$, 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 $i \in C'$. Observe first 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 $S \in S_i$. 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 $|C'| = |S| \leq |S_{2\ell}|$. Since $|S_{2\ell}| = 2\delta \ell$ holds by definition of $\delta$, we obtain $|C| = |C'| + \ell \leq |S_{2\ell}| + \ell \leq (2\delta + 1) \ell \leq 3\delta \ell$.

**Lemma III.6.** For every $\ell \in [1..n]$, the number of irreducible LCP values in $[\ell..2\ell)$ is $O(\delta \log \delta)$.

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

Let $\bar{m} \geq \text{size}(v)$ be defined as in the proof of Lemma III.3. Recall that we have identified at least $\frac{n}{\bar{m}}$ strings in $S_i$ whose leftmost occurrences in $T^{\infty}[1..]$ are disjoint. By Lemma III.5, there are at most $3\delta$ such substrings. Thus, $\text{size}(v) \leq \bar{m} \leq 12\delta$.

By replacing the thresholds $\log z$ and $\frac{n}{\bar{m}}$ with $\log \delta$ and $\frac{1}{\delta}$, respectively, in the proof of Theorem III.4, we immediately obtain a bound in terms of $\delta$.

**Theorem III.7.** Every string of length $n$ satisfies $r = O(\delta \log \delta \max(1, \log \frac{n}{\delta \log \delta}))$.

Note that the trivial upper bound $r = O(n)$ is tighter if $\delta \log \delta > n$. In Section IV, 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 $O(1)$ and $\Omega(n)$.

By combining the Theorem III.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 $n$) were known before.

**Corollary III.8.** If $r$ and $\bar{r}$ denote the number of runs in the BWT of a length-$n$ text and its reverse, respectively, then $\bar{r} = O(r \log r \max(1, \frac{n}{\log n}))$. 

1006
Since the value of $\delta$ is the same for the text and its reverse, we obtain $\bar{r} = O(\delta \log \delta \max(1, \frac{n}{n \log \delta}))$. Combining [34, Theorem 3.9] and [37, Lemma 2] gives $\delta \leq r$. Consequently, we obtain $\bar{r} = O(r \log r \max(1, \frac{n}{n \log r}))$. 

Our technique also lets us strengthen the bound of Theorem II.2 on the sum of irreducible LCP values.

**Theorem III.9 (♠).** The sum of all irreducible LCP values is $O(n \log \delta)$.

Due to $\delta \leq r$, the presented upper bound is always (asymptotically) at least as strong as the bound of Theorem II.2; furthermore, it can be strictly stronger since $\log \delta = o(\log r)$ is possible when $\delta = \Theta(n)$. In Section IV, we construct strings proving tightness of the new bound.

### IV. Lower Bounds

In this section, we present examples showing asymptotic tightness of the upper bounds in Section III-C. We give two constructions, corresponding to the bound of Theorem III.7 and the trivial bound $r = O(n)$, respectively.

For $\ell \geq 1$, let $\text{bin}_\ell(x) \in \{0, 1\}^\ell$ be the binary representation of $x \in [0, 2^\ell]$.

**Lemma IV.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(\begin{array}{c} K-1 \\binom{2^\ell-1}{0} \\binom{2^{2\ell} \cdot \text{bin}_\ell(i)}{i=0} \end{array}\right) \cdot \$, satisfy $n = \Theta(2^{K+\ell})$, $\delta = \Theta(2^\ell)$, and $r = \Omega(2^\ell K)$.

**Lemma IV.2 (♠).** For all integers $\ell \geq 2$ and $\Delta \in \Omega(2^\ell) \cap O(2^\ell \ell)$, the length $n$, the substring complexity $\delta$, and the number of runs $r$ in the BWT of a string $T_{\ell,\Delta} \in \{0, 1, 2\}^+$, defined with

$$T_{\ell,\Delta} = \left(\begin{array}{c} 2^\ell \\binom{2^\ell \cdot \text{bin}_\ell(i)}{i=0} \end{array}\right) \cdot \$ _1 \$ _2 \cdots \$ _\Delta,$ satisfy $n = \Theta(2^\ell \ell)$, $\delta = \Theta(\Delta)$, and $r = \Omega(2^\ell \ell)$.

Combining Lemmas IV.1 and IV.2, we obtain the following

**Theorem IV.3 (♠).** For all integers $N \geq 1$ and $\Delta \in [1..N]$, there is 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 $r = \Theta(\min(n, \delta \log \delta \max(1, \frac{n}{n \log \delta})))$.

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

**Theorem IV.4 (♠).** For all integers $N \geq 1$ and $\Delta \in [1..N]$, there is a string $T$ whose length $n$, substring complexity $\delta$, and sum $\Sigma \$ of irreducible LCP values satisfy $n = \Theta(N)$, $\delta = \Theta(\Delta)$, and $r \Sigma = \Theta(n \log \delta)$.

V. CONVERTING LZ77 TO RUN LENGTH BWT

In this section, we outline an algorithm that, given the LZ77 parsing of a text $T \in \Sigma^n$, computes its run-length compressed BWT in $O(n \log n)$ time. We start with an overview that explains the key concepts. Then, we present two new data structures utilized in our algorithm: the compressed string synchronizing set (Section V-A) and the compressed wavelet tree (Section V-B). The conversion algorithm is then presented in Section V-C.

For any substring $Y$ of $T^\infty$, we define $\text{ipos}(Y) = \min \{i \in [1..n] : T^\infty[i..i+|Y|] = Y\}$. If $Y$ is a substring of $T^\infty$, we define $\text{ipos}(Y)$ by replacing $T^\infty$ in the definition with $T^\infty$. We say that $Y$ 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 V.1 (BWT modulo $\ell$).** Let $T \in \Sigma^n$, $\ell \geq 1$, and $T_i = T^\infty[\text{SA}[i]..\text{SA}[i+\ell])$ for $i \in [1..n]$. We define the string $\text{BWT}_T \in (\Sigma \cup \mathbb{N})^n$, called the BWT modulo $\ell$ (of $T$), as follows. For $i \in [1..n]$,

$$\text{BWT}_T[i] = \begin{cases} \text{ipos}(Y) & \text{if } Y \text{ is left-maximal,} \\ \text{BWT}_i & \text{otherwise.} \end{cases}$$

The algorithm runs in $k = \lfloor \log n \rfloor$ rounds. For $q \in [0..k)$, the input to the $q$th round is $\text{RL}(\text{BWT}_{T_q})$, where $\ell = 2^q$, and the output is $\text{RL}(\text{BWT}_{T_{q+1}})$. At the end of the algorithm, we have $\text{RL}(\text{BWT}_{T_{k'}}) = \text{RL}(\text{BWT})$ because $X \in S_{2^k}$ is never left-maximal for $2^k \geq n$.

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 $2^q$. $\text{BWT}_{T_q}[b..e] \in \Sigma^+$ implies $\text{BWT}_{T_{q+1}}[b..e] \in \Sigma^+$ (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, $\text{BWT}_{T_q}$ identifies the (leftmost occurrences of) substrings to be inspected in the $q$th round with the aim of replacing their corresponding runs in $\text{BWT}_{T_q}$ with previously unknown BWT symbols (as defined in $\text{BWT}_{T_{q+1}}$).

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

**Lemma V.2.** For any $\ell \geq 1$, it holds $|\text{RL}(\text{BWT}_{T_q})| < 2r$.

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

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 $\text{RL}(\text{BWT}_{T_q})$), or $c_{i+1} \in \mathbb{N}$, which is impossible.
since then $\text{BWT}(\lambda_1 \ldots \lambda_{i+1})$ is non-uniform. Thus, there are at most $r$ runs of symbols from $\Sigma$ in $\text{BWT}_e$.

A. Compressed String Synchronizing Sets

Our algorithm builds on the notion of string synchronizing sets, recently introduced in [33]. Synchronizing sets are one of the most powerful techniques for sampling suffixes. As demonstrated in [36], 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], [33]. 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 V.3 ($\tau$-synchronizing set [33]). Let $T$ be a string of length $n$, and let $\tau \in [1, \ldots \lfloor \frac{n}{2} \rfloor]$. 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 $i \in S$ holds 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 $\text{per}(T[i..i + 3\tau - 2]) \leq \frac{1}{2}\tau$ (for $i \in [1, n - 3\tau + 2]$).

In most applications, we want to minimize $|S|$. Observe that the Thue–Morse sequence $T_{TM}$ [64] 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 $T_{TM}$ satisfies $|S| = \Theta\left(\frac{n}{2}\right)$ unless $n < 3\tau$. Therefore, in general, we cannot hope to achieve an upper bound improving in the worst case upon the following one.

Theorem V.4 ([33]). For any string $T$ of length $n$ and parameter $\tau \in [1, \lfloor \frac{n}{2} \rfloor]$, there exists a $\tau$-synchronizing set $S$ of size $|S| = O\left(\frac{n}{2}\right)$. Moreover, such $S$ can be (deterministically) constructed from $T$ in $O(n)$ time.

Storing $S$ for compressible strings presents the following challenge: As shown in [46], a length-$n$ prefix of $T_{TM}$ satisfies $z = O(\log n)$ and yet, as described above, every synchronizing set of $T_{TM}$ satisfies $|S| = \Omega\left(\frac{n}{2}\right)$. Thus, although $|S|$ can be smaller than $\frac{n}{2}$, $z \ll n$ does not imply $|S| \ll n$, preventing us from keeping plain $S$ when $\tau = o\left(\frac{n}{2}\right)$.

We thus exploit a different property of compressible strings: their substrings $Y$ satisfy $\text{Ipos}(Y) \subseteq \bigcup_{j=1}^{z} (e_j - \lfloor Y \rfloor + e_j)$, 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 $\bigcup_{j=1}^{z} S \cap (e_j - 2\tau \ldots e_j)$. To check if $i \in S$, we then locate $i' = \text{Ipos}(T[i..i + 2\tau])$ and check if $i' \in \bigcup_{j=1}^{z} S \cap (e_j - 2\tau \ldots e_j)$. This motivates the following (more general) definition.

Definition V.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 $j \in [1, \ldots z]$, let $e_j$ denote the last position of the $j$th phase in the LZ77 parsing of $T$. For $k \in \mathbb{N}_{\geq 2}$, we define the compressed representation of $S$ as

$$\text{comp}_k(S) := \bigcup_{j=1}^{z} S \cap (e_j - k\tau \ldots e_j + k\tau).$$

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 $T$.

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

Theorem V.6. Let $T$ be a string of length $n$ and let $\tau \in [1, \lfloor \frac{n}{2} \rfloor]$. Assume that $\text{per}(T[i..i + \tau]) > \frac{1}{3}\tau$ holds for all $i \in [1, n - \tau + 1]$. Then, for every $k \in \mathbb{N}_{\geq 2}$, there exists a $\tau$-synchronizing set $S$ of $T$ with $\text{comp}_k(S) \leq 12k\tau$.

Proof: Let $h : S \rightarrow [0, 1]$ 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 $\text{id} : [1, n - \tau + 1] \rightarrow [0, 1]$ with $\text{id}(i) = h(T[i..i + \tau])$. Observe that (almost surely) $\text{id}$ is an identifier function, that is, $\text{id}(i) = \text{id}(j)$ holds if and only if $T[i..i + \tau] = T[j..j + \tau]$. In [33, Lemma 8.2], it is proved that then

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

is a $\tau$-synchronizing set of $T$. Moreover, $E[|S|] = O\left(\frac{n}{2}\right)$. To see this, observe that, for $j, j' \in [i..i + \tau]$, $\text{id}(j) = \text{id}(j')$ implies $|j' - j| > \frac{1}{3}\tau$ (otherwise, assuming $j < j'$, we have $\text{per}(T[j..j + \tau]) \leq \frac{1}{4}\tau$, which contradicts $\text{per}(T[j..j + \tau]) > \frac{1}{3}\tau$). Thus, $T[i..i + 2\tau - 1]$ contains at least $\frac{2}{3}$ distinct length-$\tau$ substrings. Since each has equal chance of having the smallest id, we have $P[i \in S] \leq \frac{2}{3} \leq \frac{3}{4}$, and consequently, by linearity of expectation, $E[|S|] = \frac{6n}{4}$. More generally, $E[|S \cap (i..i + \ell)|] \leq \frac{3\ell}{4}$, and therefore

$$E\left[|\text{comp}_k(S)|\right] = E\left[\bigcup_{j=1}^{z} S \cap (e_j - k\tau \ldots e_j + k\tau)\right] \leq \sum_{j=1}^{z} E\left[|S \cap (e_j - k\tau \ldots e_j + k\tau)|\right] \leq 12k\tau.$$ 

In particular, $|\text{comp}_k(S)| \leq 12k\tau$ 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 $S$. Indeed, since $|S| = \Theta(\tau n)$ holds in the worst case, we cannot hope to achieve $O(z \text{ polylog } n)$ time this way. Next, we prove that assigning unique names to all elements of $S$ is in fact not necessary.

Lemma V.7. Let $T$ be a string of length $n$ and let $\tau \in [1, \lfloor \frac{n}{2} \rfloor]$. Assume that $\text{per}(T[i..i + \tau]) > \frac{1}{3}\tau$ holds for all $i \in [1, n - \tau + 1]$. For $i \in [1, n - \tau + 1]$, let $\text{id}(i) :=$
h(T[i..i + \tau]), where h : S_\tau \to [0, 1] assigns independent and uniformly random values.

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

Proof: Recall from the proof of Theorem V.6 that T[i..i + 2\tau + 1] contains at least \frac{d}{\tau} distinct length-\tau substrings. Since the values of h are independent and uniformly distributed, we have
\Pr[\min \{\text{id}(j) : j \in [i..i + \tau] \} > \frac{1}{\kappa}] = (1 - \frac{1}{\kappa})^d \leq \exp(-\frac{d}{\kappa}).

The probability above is trivially 0 if \kappa = 1, so we can bound it by \kappa^{-c} for \kappa = \max (1, \frac{1}{3n^c})

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 V.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, 1/\kappa]

(mapping strings in S_{\text{sample}} to real values in [0, 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}^{n} \{ e_i - \tau < 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

\text{E}[|P'_{\text{sample}}|] = \frac{1}{\kappa} \cdot |P_{\text{close}}| \leq \frac{3n \ln n}{\kappa} \cdot \tau = O(\ell \log n).

To finish the construction, it suffices to pick a random function h_{\text{sample}} : S_{\text{sample}} \to [0, 1/\kappa] to obtain id(i) := h_{\text{sample}}(T[i..i + \tau]) (letting id(i) = 1 if T[i..i + \tau] \notin S_{\text{sample}}). Then, by Lemma V.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 V.6. Moreover, we can also detect failures (that \min \{\text{id}(j) : j \in [i..i + \tau] \} = 1 for some i), so the algorithm is Las-Vegas randomized.

Theorem V.8 (\tri). Let T be a string of length n and let \tau \in [1..\frac{\ell}{2}]. Assume that \text{per}(T[i..i + \tau]) > \frac{\tau}{\ell} 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 O(\ell \log^2 n) time a compressed representation \text{comp}_k(S) of a \tau-synchronizing set S of T satisfying \text{comp}_k(S) \leq 24\ell k^2.

2) The General Case: Periodic fragments are handled similarly as in [33]. This yields the following two results, which constitute the main outcome of this section.

Theorem V.9 (\tri). Let T be a string of length n and let \tau \in [1..\frac{\ell}{2}]. For any k \in \mathbb{N}_{\geq 2}, there exists a \tau-synchronizing set S of T satisfying \text{comp}_k(S) \leq 36k\ell.

Theorem V.10 (\tri). Let T be a string of length n and let \tau \in [1..\frac{\ell}{2}]. 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 O(\ell \log^2 n) time a compressed representation \text{comp}_k(S) of a \tau-synchronizing set S of T satisfying |\text{comp}_k(S)| \leq 72k\ell.

B. Compressed Wavelet Trees

Along with string synchronizing sets, wavelet trees [28], 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..m] over the alphabet \Sigma^* 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 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 T whose edges are labelled by Y[1], Y[2], \ldots, 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^\ell, we denote by v_X we denote the node of T labelled X. We let

V(T) := \bigcup_{v_Y \subseteq v_X} \{ v_Y : X \in \Sigma^\ell \}

denote the node set of T.

With each node v_X \in V(T) we associate an increasing sequence I_X[i..i + h] of primary indices such that

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

Based on I_X, we define \text{B}_X \subseteq \Sigma^* such that, for i \in [1..m],

\text{B}_X[i] = W[I_X[i]..|X|] + 1,

if |X| < \ell and \text{B}_X = \varepsilon if |X| = \ell. In other words, \text{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 \text{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 \text{B}_X. The values of primary indices I_X are retrieved using additional data structures, based on the following observation.
Lemma V.11 (28). Let $X \in \Sigma^d$, where $d \in [0..\ell)$. For every $c \in \Sigma$ and $j \in [1..|X_j|]$, we have $I_X[j] = I_X[i]$, where $B_X[i]$ is the $i$th occurrence of $c$ in $B_X$.

We define the compressed wavelet tree $T_c$ of $W$ as the wavelet tree of $W$ in which all strings $B_X$ have been run-length compressed and, with the exception of $\{v_x \cup \langle v_{W[i]} \rangle \}_{i=1}^m$, all nodes $v_x$ satisfying $|RL(B_X)| \leq 1$, 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 tree of strings $W[1]..W[m]$.

We store edge labels of $T_c$ 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 $T_c$ and each element of $RL(B_X)$ can be encoded in $O(1)$ space. Since $|RL(B_X)| \leq m$ holds for every internal node $v_{y} \in V(T_c)$, and unless $|V(T_c)| = 1$, each leaf $v_{z}$ in $T_c$ can be injectively mapped to an element of $RL(B_Y)$ for the parent $v_{z'}$ of $v_{z}$, the space to store $T_c$ is dominated by the run-length compressed strings $B_X$, i.e., $T_c$ needs $O(1 + \sum_{v_x \in V(T_c)} |RL(B_X)|)$ space.

Theorem V.12. Let $W$ be a non-empty sequence of equal-length strings and let $T_c$ be its compressed wavelet tree. Then, $\sum_{v_x \in V(T_c)} |RL(B_X)| = O(1 + |RL(W)|)$ space.

Proof: Let $m = |W|$, $k = |RL(W)| \leq m$, and $k' = \{[W[i]: i \in [1..m]]\} \leq k$. Due to $|V(T_c)| \leq 2k' = O(k)$, we can focus on nodes $v_x \in V(T_c)$ such that $|RL(B_X)| \geq 2$.

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

$$\sum_{v_x \in V(T_c), |RL(B_X)| \geq 2} |RL(B_X)| \leq 4k \log k' = O(k \log k).$$

Consider $X \in \Sigma^d$ with $|RL(B_X)| \geq 2$; note that $d < \ell$.

Let $RL(B_X) = \{(c_1, \lambda_1), (c_2, \lambda_2), \ldots, (c_h, \lambda_h)\}$. Observe that if we let $p_0 = I_X[\lambda_1]$ and $p_1 = I_X[\lambda_1 + 1]$ for some $i \in [1..h)$, then $W[p_0][d + 1] = c_i \neq c_{i+1} = W[p_1][d + 1]$. Moreover, $B_X[\lambda_1] \neq B_X[\lambda_1 + 1]$ 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 $W[p_i]$, where $t \in \{0, 1\}$ is chosen depending on the sizes of subtrees of $T_c$ rooted at the children of $v_x$, so that the subtree containing $v_{W[p_i]}$ has at most as many leaves as the subtree containing $v_{W[p_i - 1]}$.

Now, consider a run $W[b..b'] = Y^\ell$ in $RL(W)$. 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 $v_{y_i}$ to the root $v_c$ of $T_c$ 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 $W[1..m]$ 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 $O(1 + |RL(W)|)$ space. Namely, it suffices to store the length $\ell$ and the sequence $RL([\text{pos}(W[i])]_{i=1}^{m})$. If $W$ is a sequence of substrings of $T^\infty$, we can similarly encode it using $RL([\text{pos}(W[i])]_{i=1}^{m})$.

The key operation that we want to support on $T_c$, given a pointer to $v_x \in V(T_c)$ and an integer $q \in [1..|I_X|]$, is to compute the value $I_X[q]$. The following theorem shows that given the compact encoding of $W$ and the LZ77 parsing of $T$, the compressed wavelet tree of $W$ supporting these primary index queries can be constructed efficiently.

Theorem V.13 (1). Given the LZ77 parsing of $T[1..n]$ and a sequence $W[1..m]$ of $m \leq \sigma$ same-length substrings of $\Sigma^d$ (resp. $\overline{T^\infty}$), as represented by $RL([\text{pos}(W[i])]_{i=1}^{m})$ (resp. $RL([\text{pos}(W[i])]_{i=1}^{m})$), the compressed wavelet tree of $W$, supporting primary index queries in $O(\log^2 n)$ time, can be constructed in $O((z + |RL(W)|) \log^2 n)$ time.

C. The Algorithm

We are now ready to show how to construct the sequences $RL(BWT_t)$, where $\ell = 2^q$ for $q \in [0.. \log n]$. For small $\ell$, constructing $RL(BWT_t)$ reduces to sorting and computing frequencies of length-$\Theta(\ell)$ substrings of $T^\infty$.

Proposition V.14 (1). Given the LZ77 parsing of $T[1..n]$, the sequence $RL(BWT_t)$ can be constructed in $O(z \log^4 n)$ time.

Let $q \geq 4$. We show how to compute $RL(BWT_{2^q})$, given the LZ77 parsing of $T$ and $RL(BWT_t)$. The main idea of the algorithm is, as follows.

Let $S$ be a $\tau$-synchronizing set of $T$, where $\tau = [\underline{\lambda}]$. As noted earlier, $BWT_t[j] \in \Sigma$ implies $BWT_{2^q}[j] \in \Sigma$. Let $BWT_t[y..y'] \in \mathbb{N}^+$ be a run in $BWT_t$. By definition of $BWT_t$, the suffixes of $T^\infty$ starting at positions $i \in \text{SA}[y..y']$ share a common prefix of length $\ell \geq 3\tau$. Thus, assuming that $S \cap [i..i+\tau]$ is not empty for all $i \in \text{SA}[y..y']$ (the periodic case is handled separately), by the consistency of $S$, all text positions $i \in \text{SA}[y..y']$ share a common offset $\Delta$ with $i + \Delta = \min(S \cap [i..i+\tau])$. This lets us deduce the order of length-$2\ell$ prefixes $T[i..i+2\ell]$ 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 $S \in \Sigma$, we extract, using a wavelet tree, those preceded by $T[i..i+\Delta]$ (a prefix common to $T^\infty[i..i]$) for $i \in \text{SA}[y..y']$. Importantly, the synchronizing positions $s$ sharing $T[s-\ell..s+2\ell]$ can be processed together; hence, by Theorem V.9, it suffices to use $O(z)$ distinct substrings. We formalize these ideas as follows. Let

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

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

- \( T^\infty[s'_i \ldots s'_i + 7\tau) \} < T^\infty[s'_{i+1} \ldots s'_{i+1} + 7\tau) \), or
- \( T^\infty[s'_{i+1} \ldots s'_{i+1} + 7\tau) \} = T^\infty[s'_{i} \ldots s'_{i} + 7\tau) \) and
  \( T^\infty[s'_i \ldots s'_i + \tau) \} < T^\infty[s'_i \ldots s'_i + 7\tau) \).

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

\[
\tilde{W}[i] = T^\infty[s'_i \ldots s'_i + 7\tau),
\]

\[
W[i] = T^\infty[s'_i \ldots s'_i + 7\tau),
\]

\[
W'[i] = T^\infty[s'_i \ldots s'_i + 7\tau).
\]

Recall that we can compactly represent \(\tilde{W}\) in \(O(1 + |RL(\tilde{W})|)\) space using \(RL((\text{pos}(\tilde{W}[j]))_{j \in [1..|S|]})\). The sequences \(W\) and \(W'\) can be represented analogously, except that we use \(RL((\text{pos}(W[j]))_{j \in [1..|S|]})\) for \(W\).

Lemma V.15. The sequences \(\tilde{W}, W, \text{ and } W'\) defined above satisfy \(|RL(W)|, |RL(W')| \leq |RL(\tilde{W})| \leq |\text{comp}_\tau(S)|\).

Proof: For the first inequality, note that \(\tilde{W}[i] = \tilde{W}[i+1]\) implies \(W[i] = W[i+1] \) and \(W'[i] = W'[i+1]\).

Let \(RL(\tilde{W}) = (R_1, A_1, \ldots, R_h, A_h)\). Observe that \(i \neq j\) implies \(R_i \neq R_j\). For \(i \in [1..h]\), let \(T^\infty[p..p + 7\tau)\) be the occurrence of \(R_i\) in \(T^\infty\) that minimizes \(p \in [1..n]\). Then, there exists \(j \in [1..s]\) such that \(e_j - 8\tau < p - \tau \leq e_j\). By the consistency of \(S\), we conclude that \(|S\cap(e_j - 7\tau .. e_j + \tau)\| \leq |\text{comp}_\tau(S)|\). The claim follows, since this map is injective.

Importantly, the compact representations of \(\tilde{W}, W, \text{ and } W'\) can be computed efficiently.

Lemma V.16 (\(\Delta\)). Given \(|\text{comp}_\tau(S)|\) and the LZ77 parsing \(T\), the compact representations of \(\tilde{W}, W, \text{ and } W'\) can be constructed in \(O(\log^2 n + |\text{comp}_\tau(S)| \log^3 n)\) time.

Next, we recall the notion of distinguishing prefixes, originally introduced in [33], that allows mapping each suffix \(T^\infty[i..n]\) to the corresponding node of the wavelet tree of \(W\).

Definition V.17 (Distinguishing prefix). For any position \(i \in [1..\max(S \cup \{0\})]\), let \(i_{\text{suc}} = \min\{j \in S : j \geq i\}\). The distinguishing prefix of \(T[i..n]\) is \(D_i = T[i..i_{\text{suc}} + 2\tau)\).

Let \(D = \{D_j : j \in [1..\max(S \cup \{0\})]\}\). Note that if \(Y\) starts with \(D\), then for every occurrence \(T^\infty[i..n+|Y|] = Y\) with \(i \in [1..n]\), the distinguishing prefix \(D_i\) is defined and satisfies \(D_i = D^\tau\). Thus, for any such \(Y\), we define \(D_Y = D\). We denote \(D_Y = D_Y[1..|D_Y| - 2\tau]\).

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

Lemma V.18. Let \(Y = 1..n\) be a string starting with an element of \(D\). 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 \(T^\infty[\text{SA}[i]..n] = Y\) for \(i \in [y..y]\).

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

1. \(B_X[b..b']\) and \(BWT[y..y]\) are equal as multisets.
2. \(|RL(B_X[b..b']|) \leq 3 |RL(BWT[y..y])|).

Proof: 1. Due to \(T[n] = \emptyset\), 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 \(s \in S\) 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 \((s'_i)_{i \in [1..|S|]}\) containing all such \(s\) as a two-step search.

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

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

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

Let \(BWT[y..y] = c^\delta \in \mathbb{N}^+\) be a run in \(BWT\), and let \(Y = T^\infty[c..c + \ell]\) be the left-maximal \(c \leq \ell \leq n\). Since \(Y\) is left-maximal, \(c \leq \ell \leq n\). Thus, by \(3\tau \leq \ell\) and \(R = \emptyset\), we obtain \([c..c + \tau) \cap S \neq \emptyset\). Moreover, if \(Y = XX'\) is such that \(|X'| = \Delta\), where \(c + \Delta = \min(S \cap [c..c + \tau])\), then \(D_Y = X\). Since we also have \(|X'| \leq 2\ell \leq 7\tau\) (due to \(\delta \geq 2\)), Lemma V.18 holds for \(Y\). Let \([b..b']\) be the range of \(B_X\) corresponding to \(Y\) through Lemma V.18. Then, \(|RL(B_X[b..b'])| > 1\). Moreover

- For every \(j \in [0..\delta]\) such that \(BWT_{2\ell}[y + j] \in \Sigma\), the following equality holds: \(BWT_{2\ell}[y + j] = B_X[b + j]\).

To see this, apply Lemma V.18 to all strings \(Y \)\(\Rightarrow\)\( T^\infty[\text{SA}[j]..\text{SA}[j] + 2\ell] : j \in [y..y']\) ordered lexicographically. Since \(D_X = D_Y\), the corresponding ranges \([b..b']\) form a left-to-right partition of \([b..b']\). Thus, if \(Y\) is not left-maximal, its BWT block is \(BWT[y..y'] = B_X[b..b']\).
On the other hand, if \( \ell' = \text{BWT}_2[y + j] \in \mathbb{N} \) holds for some \( j \in [0, \delta) \), then the string \( \text{BWT}_2[b, b'] \) corresponding (through Lemma V.18) to \( Y = T^\infty[\ell', \ell' + 2\ell] \in \mathcal{Y} \) is not unary, i.e., there exists an index \( \hat{b} \in [b, b'] \) satisfying \( \text{BWT}_2[\hat{b}] \neq \text{BWT}_2[b + 1] \), and \( \text{lcp}(W'[\text{BWT}_2[\hat{b}]]), W'[\text{BWT}_2[b + 1]]) \geq 2\ell - |X| \). The converse is also true: if \( \hat{b} \in [b, b'] \) satisfies the two conditions, then \( \text{BWT}_2[y + \hat{b} - b] \in \mathbb{N} \). Consequently, the set of left-maximal strings in \( \mathcal{Y} \) is

\[
\{ X \cdot W'[\text{BWT}_2[\hat{b}]](1..2\ell - |X|) : \hat{b} \in [b, b'), \text{BWT}_2[\hat{b}] \neq \text{BWT}_2[b + 1], \text{ and } \text{lcp}(W'[\text{BWT}_2[\hat{b}]], W'[\text{BWT}_2[b + 1]]) \geq 2\ell - |X| \}.
\]

Moreover, letting \( \tilde{Y} = X \cdot W'[\text{BWT}_2[\hat{b}]](1..2\ell - |X|) \) for any \( \hat{b} \) satisfying the above conditions, the range \( [y..y'] = \{ j \in [1..n] : T^\infty[\text{SA}[j]..\text{SA}[j + 2\ell] = \tilde{Y} \} \) satisfies \( [y..y'] = y - b + [b, b'] \), where \( \text{BWT}_2[b, b'] \) corresponds to \( Y \) through Lemma V.18.

The algorithm processing a run \( \text{BWT}[y..y'] = \epsilon^\ell \in \mathbb{N}^+ \) is thus as follows. Letting \( Y = T^\infty[c..c + \ell] \) and \( X = D'_X \), we first compute \( |X| \) and the pointer to \( v_X \). 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 \( \text{BWT}_2[y..y'] \) is achieved by a series of forward and backward searches (at most one per run of \( \text{BWT}_2[b, b'] \)).

The length \( |X| \) is computed using \( \text{comp}_R(S). \) The pointers to \( v_Y \) are precomputed since \( |\text{RL}(\text{BWT}_2)| > 1 \) implies \( v_Y \in V(T_x) \). 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 V.19 (\( \star \)).** Let \( T \) be a string of length \( n \), and let \( \ell = 2^q \) be such that \( q \in [4..\log n] \). If \( R = \emptyset \), then, given RL(BWT_{\ell}), and the LZ77 parsing of \( T \), the sequence RL(BWT_{2\ell}) can be constructed in \( \mathcal{O}(r(z + \log^2 n)) \) time.

2) The General Case: Periodic fragments are handled similarly as in the BWT construction in [33], resulting in the following algorithm.

**Proposition V.20 (\( \star \)).** Let \( T \) be a string of length \( n \), and let \( \ell = 2^q \) be such that \( q \in [4..\log n] \). Then, given RL(BWT_{\ell}) and the LZ77 parsing of \( T \), the sequence RL(BWT_{2\ell}) can be constructed in \( \mathcal{O}(r(z + \log^2 n)) \) time.

By Proposition V.14 we can compute \( \text{BWT}_r \) for \( \ell = 2^q \) and \( q < 4 \) in \( \mathcal{O}(z \log^4 n) \) time. For \( q \geq 4 \), we use Proposition V.20. Thus, by the upper bound \( r = \mathcal{O}(z \log^2 n) \) from Theorem III.2, we obtain the main result of this section.

**Theorem V.21.** 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 \( \mathcal{O}(r(z + \log^2 n)) = \mathcal{O}(z \log^2 n) \) time.

**Acknowledgments:** The authors would like to thank Barna Saha for helpful discussions. D. Kempa is supported by NSF grants no. 1652303 and 1934846, and an Alfred P. Sloan Fellowship grant. T. Kociumaka is 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).

**References**


