DOI:10.1145/3531445

To view the accompanying Technical Perspective, visit doi.acm.org/10.1145/3531443

Resolution of the Burrows-Wheeler Transform Conjecture

By Dominik Kempa and Tomasz Kociumaka

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 the *r*-index[7]), central in modern bioinformatics. The compressibility of BWT is quantified by the number *r* of equal-letter runs in the output. Despite the practical significance of BWT, no nontrivial upper bound on *r* is known. By contrast, the sizes of nearly all other known compression methods have been shown to be either always within a polylog *n* factor (where n is the length of the text) from *z*, the size of Lempel–Ziv (LZ77) parsing of the text, or much larger in the worst case (by an n^ε 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; in particular: (1) it proves that many results related to BWT automatically apply to methods based on LZ77, for example, 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; and (3) it implies the first nontrivial relation between the number of runs in the BWT of the text and of 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 several new data structures and techniques of independent interest. In particular, we define compressed string synchronizing sets (generalizing the recently introduced powerful technique of string synchronizing sets[11]) and show how to efficiently construct them. Next, we propose a new variant of wavelet trees for sequences of long strings, establish a nontrivial bound on their size, and describe efficient construction algorithms. Finally, we develop new indexes that can be constructed directly from the LZ77 parsing and efficiently support pattern matching queries on text substrings.

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 the following: Lempel–Ziv compression (where the nominal and most commonly used is the LZ77 variant[25]), statistical compression (this includes, e.g., context mixing, prediction by partial matching (PPM), and dynamic Markov coding), and Burrows-Wheeler transform (BWT).[4]

One of the features that best differentiates these algorithms is whether they better remove the redundancy caused by skewed symbol frequencies or by repeated fragments. The idea in LZ77 (which underlies, e.g., `7-zip` and `gzip` 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* $H_k(T)$. For any sufficiently long text T, symbol frequencies (taking length-k contexts into account) in any power of T (the concatenation of several copies of T) do not change significantly (see Kreft and Navarro,[14] Lemma 2.6). Therefore, $|T^t|\, H_k(T^t) \approx t \cdot |T| H_k(T)$ holds for any $t \geq 1$, which means that entropy is not sensitive to long repetitions, and hence it is not able to capture the same type of redundancy as the LZ77 compression.[7,12,14,24]

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`, is quantified by the number r of equal-letter runs in the BWT. The clear picture described above no longer applies to the measure r. On the one hand, Manzini[16] 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.[24] 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 (the size of the LZ77 parsing) 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

The original version of this paper was presented at FOCS 2020 and is available at https://arxiv.org/abs/1910.10631

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) concerning the uncompressed version. Although classical (uncompressed) indexes, such as suffix trees and suffix arrays, have been successful in many applications, they are not suitable for storing and searching big highly repetitive collections, which are virtually impossible to search without preprocessing. A recent survey[17] provides several real-life examples of such datasets. In particular, Github stores more than 20 terabytes of data, with an average of over 20 versions per project, whereas the 100000 Human Genome Project produced over 70 terabytes of DNA, which is highly compressible due to 99.9% similarity between individual human genomes. Motivated by such applications, compressed indexing witnessed a remarkable surge of interest in recent years. BWT-based indexes, such as the r-index,[7] are among the most powerful, 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.[18]

In addition to text indexing, the Burrows-Wheeler transform has many applications in computational biology. In particular, BWT is the main component of the popular genome read aligners such as `Bowtie`, `BWA`, and `Soap2`, which paved its way to general bioinformatics textbooks.[21] Specialized literature on algorithmic analysis of biological sequences[15, 19] devotes dozens of pages to BWT applications.

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?

Except for BWT, essentially every other known compression method has been proven 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, collage systems, and macro schemes)[a] or larger by a polynomial factor (n^{ε} for some $\varepsilon > 0$) in the worst case (e.g., LZ78, compressed word graphs (CDAWGs)). We refer the reader to Navarro[17] for a survey of repetitiveness measures. Notably, BWT is known to never compress much better than LZ77, that is, $z = \mathcal{O}(r \log n)$,[6] and the opposite relation $r = \mathcal{O}(z \text{ polylog } n)$ was often conjectured to be false.[5]

1.1. Our contribution

We prove that $r = \mathcal{O}(z \log^2 n)$ holds for all strings, resolving the BWT conjecture in the more surprising way and answering an open question of Gagie et al.[5,6] This result alone has multiple implications for indexing and compression:

1. It is possible to support suffix array and suffix tree functionality in $\mathcal{O}(z \text{ polylog } n)$ space.[7]
2. It was shown by Kempa[10] that many string processing tasks (such as BWT and LZ77 construction) can be solved in $\mathcal{O}(n/\log_{\sigma} n + r \text{ polylog } n)$ time (where σ is the alphabet size). Thus, if the text is sufficiently compressible by BWT (formally, $n/r = \Omega$ (polylog n)), these tasks can be solved in optimal time (which is unlikely to be possible for general texts[11]). Our result loosens this assumption to $n/z = \Omega$ (polylog n).
3. Until now, methods based on the Burrows-Wheeler transform were thought to be neither statistical nor dictionary (LZ-like) compression algorithms.[5, 24] Our result challenges the notion that the BWT forms a separate compression type: Because of our bound, BWT is much closer to LZ compressors than anticipated.

Our slightly stronger bound $r = \mathcal{O}(\delta \log^2 n)$, where $\delta \le z$ is a symmetric (insensitive to string reversal) repetitiveness measure recently studied by Kociumaka et al.,[13] further shows that:

4. The number $\overline{r}$ of BWT runs in the reverse of the text satisfies $\overline{r} = \mathcal{O}(r \log^2 n)$, which is the first nontrivial bound in terms of r. This result is of practical importance due to many algorithms whose efficiency depends on $\overline{r}$.[2, 20, 22, 23]

After proving $r = \mathcal{O}(z \log^2 n)$, we refine our approach to obtain $r = \mathcal{O}(z \log z \max(1, \log \frac{n}{z \log z}))$ and subsequently $r = \mathcal{O}(\delta \log \delta \max(1, \log \frac{n}{\delta \log \delta}))$. We then show that the latter bound, combined with the trivial one $r \le n$, is asymptotically tight for the full spectrum of values of n and δ. As a side result, we obtain a tight upper bound $\mathcal{O}(n \log \delta)$ on the sum of irreducible LCP values, improving upon the previously known bound $\mathcal{O}(n \log r)$.[9]

Next, we develop an $\mathcal{O}(z \log^8 n)$-time algorithm converting the LZ77 parsing into the 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.[20,22] 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 technique by Kempa and Kociumaka.[11] We then describe a new variant of wavelet trees,[8] designed to work for sequences of long strings. In the full version of this paper, we also propose new indexes that can be built directly from the LZ77 parsing and support fast pattern matching queries on text substrings.

2. PRELIMINARIES

A *string* is a finite sequence of characters from a given *alphabet*. The length of a string S is denoted by $|S|$ and, for $i \in [1 \mathinner{.\,.} |S|]$,[b] the ith character of S is denoted by $S[i]$. A string U is a *substring* of S if $U = S[i]\, S[i+1] \ldots S[j-1]$ for some $1 \le i \le j \le |S| + 1$. We then say that U *occurs* in S at position i. The *occurrence* of U at position i in S is denoted by $S[i \mathinner{.\,.} j)$ or $S[i \mathinner{.\,.} j-1]$. Such an occurrence is a *fragment* of S and can

a The choice for LZ77 as a representative in this class follows from the fact that most other methods are NP-hard to optimize, whereas LZ77 admits a simple linear-time compression.

b For integers $i, j \in \mathbb{Z}$, we denote $[i \mathinner{.\,.} j] = \{k \in \mathbb{Z} : i \le k \le j\}$, $[i \mathinner{.\,.} j) = \{k \in \mathbb{Z} : i \le k < j\}$, and $(i \mathinner{.\,.} j] = \{k \in \mathbb{Z} : i < k < j\}$.

be represented by (a pointer to) S and the two positions i, j. Two fragments (perhaps of different strings) *match* if they are occurrences of the same substring. A fragment $S[i\,..\,j]$ is a *prefix* if $i = 1$ and a *suffix* if $j = |S|$.

We use $\overline{S}$ to denote the *reverse* of S, that is, $S[|S|] \dots S[2]$ $S[1]$. We denote the *concatenation* of two strings U and V, that is, $U[1]U[2] \dots U[|U|]V[1]V[2] \dots V[|V|]$, by UV or $U \cdot V$. Furthermore, $S^k := \bigodot_{i=1}^{k} S$ denotes the concatenation of $k \geq 0$ copies of S; note that $S^0 = \varepsilon$ is the *empty string*.

An integer $p \in [1\,..\,|S|]$ is a *period* of a string S if $S[i] = S[i + p]$ holds for every $i \in [1\,..\,|S| - p]$. The shortest period of S is denoted as $\mathrm{per}(S)$. A string S is called *periodic* if $\mathrm{per}(S) \leq \frac{1}{2}|S|$. The following fact is a consequence of the classic *periodicity lemma* (which we do not use directly).

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

Throughout the paper, we consider a string (called the *text*) T of length $n \geq 1$ over an ordered alphabet Σ of size σ. We assume that $T[n] = \$$, where \$ is the smallest symbol in Σ, and that \$ does not occur anywhere else in T. We use $\preceq$ to denote the order on Σ, extended to the *lexicographic* order on Σ^* (the set of strings over Σ) so that $U, V \in \Sigma^*$ satisfy $U \preceq V$ if and only if either U is a prefix of V, or $U[1\,..\,i) = V[1\,..\,i)$ and $U[i] \prec V[i]$ holds for some $i \in [1\,..\,\min(|U|, |V|)]$.

The *suffix array* $\mathrm{SA}[1\,..\,n]$ of T is a permutation of $[1\,..\,n]$ such that $T[\mathrm{SA}[1]\,..\,n] \prec T[\mathrm{SA}[2]\,..\,n] \prec \dots \prec T[\mathrm{SA}[n]\,..\,n]$. The closely related *Burrows-Wheeler transform* $\mathrm{BWT}[1\,..\,n]$ of T is defined by $\mathrm{BWT}[i] = T[\mathrm{SA}[i] - 1]$ if $\mathrm{SA}[i] > 1$ and $\mathrm{BWT}[i] = T[n]$ otherwise. In the context of BWT, it is convenient to consider an *infinite string* T^∞ defined so that $T^\infty[i] = T[1 + (i - 1) \bmod n]$ for $i \in \mathbb{Z}$; in particular, $T^\infty[1\,..\,n] = T[1\,..\,n]$. We then have $\mathrm{BWT}[i] = T^\infty[\mathrm{SA}[i] - 1]$ for $i \in [1\,..\,n]$. Moreover, the assumption $T[n] = \$$ implies $T^\infty[\mathrm{SA}[1]\,..\,] \prec \dots \prec T^\infty[\mathrm{SA}[n]\,..\,]$.

For any string $S = c_1^{\ell_1} c_2^{\ell_2} \cdots c_h^{\ell_h}$, where $c_i \in \Sigma$ and $\ell_i \in \mathbb{Z}_{>0}$ 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 $\mathrm{RL}(S) = ((c_1, \lambda_1), \dots, (c_h, \lambda_h))$, where $\lambda_i = \ell_1 + \dots + \ell_i$ for $i \in [1\,..\,h]$. Throughout, we let $r = |\mathrm{RL}(\mathrm{BWT})|$ denote the number of runs in the BWT of T. For example, for the text $T =$ bbabaababababaababa\$ in Table 1, we have $\mathrm{BWT} = \mathtt{a}^1\mathtt{b}^6\mathtt{a}^1\mathtt{b}^2\mathtt{a}^6\mathtt{b}^1\mathtt{a}^2\$^1$, and hence $r = 8$.

Table 1. The lexicographically sorted suffixes of a string $T =$ bbabaababababaababa\$ along with the BWT, SA, and LCP tables.

i	SA[i]	LCP[i]	BWT[i]	T[SA[i] . . n]
1	20	**<u>0</u>**	a	\$
2	19	**<u>0</u>**	b	a\$
3	14	1	b	aababa\$
4	5	6	b	aabababababa\$
5	17	1	b	aba\$
6	12	3	b	abaababa\$
7	3	8	b	abaababababaababa\$
8	15	**<u>3</u>**	a	ababa\$
9	10	**<u>5</u>**	b	ababaababa\$
10	8	5	b	abababaababa\$
11	6	**<u>7</u>**	a	ababababaababa\$
12	18	0	a	ba\$
13	13	2	a	baababa\$
14	4	7	a	baababababaababa\$
15	16	2	a	baba\$
16	11	4	a	babaababa\$
17	2	**<u>9</u>**	b	babaababababaababa\$
18	9	**<u>4</u>**	a	bababaababa\$
19	7	6	a	babababaababa\$
20	1	**<u>1</u>**	\$	bbabaababababaababa\$

The irreducible LCP values are bold and underlined.

By $\mathrm{lcp}(U, V)$ we denote the length of the longest common prefix of strings U and V. For $j_1, j_2 \in [1\,..\,n]$, we let $\mathrm{LCE}(j_1, j_2) = \mathrm{lcp}(T[j_1\,..\,], T[j_2\,..\,])$. The *LCP array* $\mathrm{LCP}[1\,..\,n]$ is defined so that $\mathrm{LCP}[i] = \mathrm{LCE}(\mathrm{SA}[i], \mathrm{SA}[i - 1])$ for $i \in [2\,..\,n]$ and $\mathrm{LCP}[1] = 0$. We say that the value $\mathrm{LCP}[i]$ is *reducible* if $\mathrm{BWT}[i] = \mathrm{BWT}[i - 1]$ and *irreducible* otherwise (including when $i = 1$). There are exactly r irreducible LCP values.

We say that a nonempty fragment $T[i\,..\,i + \ell)$ is a *previous factor* if it has an earlier occurrence in T, that is, $\mathrm{LCE}(i, i') \geq \ell$ holds for some $i' \in [1\,..\,i)$. The *LZ77 parsing* of T is a factorization $T = F_1 \cdots F_f$ into nonempty *phrases* such that the jth phrase F_j is the longest previous factor starting at position $1 + |F_1 \cdots F_{j-1}|$; if no previous factor starts there, then F_j consists of a single character. In the underlying *LZ77 representation,* every phrase $F_j = T[i\,..\,i + \ell)$ that is a previous factor is encoded as (i', ℓ), where $i' \in [1\,..\,i)$ satisfies $\mathrm{LCE}(i, i') \geq \ell$ (and is chosen arbitrarily in case of multiple possibilities); if $F_j = T[i]$ is not a previous factor, then it is encoded as $(T[i], 0)$. We denote the number of phrases in the LZ77 parsing by z. For example, the text bbabaababababaababa\$ of Table 1 has LZ77 factorization b · b · a · ba · aba · bababa · ababa · \$ with $z = 8$ phrases, and its LZ77 representation is (b, 0), (1, 1), (a, 0), (2, 2), (3, 3), (7, 6), (10, 5), (\$, 0).

The *trie* of a finite set $\mathcal{S} \subseteq \Sigma^*$ is an edge-labeled rooted tree with a node v_X for every string X that is a prefix of a string $S \in \mathcal{S}$. The trie is rooted at v_ε and the parent of each node v_X with $X \neq \varepsilon$ is $v_{X[1\,..\,|X|)}$. In this case, the edge from $v_{X[1\,..\,|X|)}$ to v_X is *labeled* with $X[|X|]$. See Figure 1 for the trie of {\$aba, aaba, abaa, abab, abb\$, b\$ab, baab, baba, babb, bb\$a}.

3. COMBINATORIAL 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. Denote $\mathcal{S}_m = \{S \in \Sigma^m : S \text{ is a substring of } T^\infty\}$ for $m \geq 1$. Observe that $|\mathcal{S}_m| \leq mz$ because every length-m substring of T^∞ 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 $\mathrm{LCP}[i] \in [\ell\,..\,2\ell)$, we associate a cost of ℓ units, which are charged to individual characters of strings in $\mathcal{S}_{3\ell}$. We then show that each of the strings in $\mathcal{S}_{3\ell}$ 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

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

Figure 1. The trie $\mathcal{T}$ of the reversed length-4 substrings of T^∞ for $T=$ `bbabaababababaababa$` of Table 1. Light edges are thin and dotted.

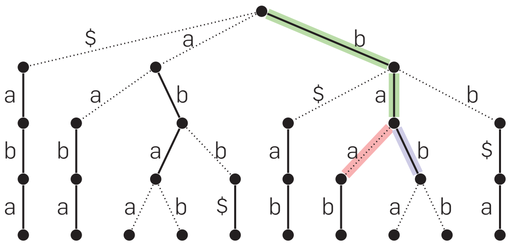
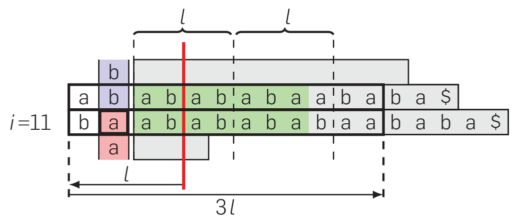


Figure 2. Proof of Lemma 3.1 on T from Table 1 for $\ell = 4$, $i = 11$, and $k = 2$. Strings T^∞ $[j_t - k\,..\,j_t - k + 3\ell)$ are highlighted. The subtree of $\mathcal{T}$ rooted in ν_{baa} is smaller than in ν_{bab} (see Figure 1), and hence we charge the second symbol of T^∞ $[j_1 - k\,..\,j_1 - k + 3\ell)$, that is, $t = 1$.


l
l
b
a b a b a b a b a a b a b a $
i=11
b a a b a b a b a b a a b a b a $
a
l
3l

To devise the announced assignment of cost to characters of strings in $\mathcal{S}_{3\ell}$, consider the trie $\mathcal{T}$ of all reversed strings in $\mathcal{S}_\ell$ (see Figure 1 for example).

Let $\mathrm{LCP}[i] \in [\ell\,..\,2\ell)$ be an irreducible LCP value; note that $i > 1$ due to $\mathrm{LCP}[i] \ge \ell > 0$. Let $j_0 = \mathrm{SA}[i-1]$ and $j_1 = \mathrm{SA}[i]$ so that $\mathrm{LCP}[i] = \mathrm{LCE}(j_0, j_1)$. Because $\mathrm{LCP}[i]$ is irreducible, we have $T^\infty[j_0 - 1] = \mathrm{BWT}[i-1] \ne \mathrm{BWT}[i] = T^\infty[j_1 - 1]$. For $k \in [1\,..\,\ell]$, 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 $T^\infty[j_t - k\,..\,j_t - k + 3\ell) \in \mathcal{S}_{3\ell}$, where $t \in \{0,1\}$ is such that the subtree of $\mathcal{T}$ rooted at $\nu_{\overline{T^\infty[j_t-1\,..\,j_t-k+\ell)}}$ contains fewer leaves than the subtree rooted at $\nu_{\overline{T^\infty[j_{1-t}-1\,..\,j_{1-t}-k+\ell)}}$ (we choose $t = 0$ in case of ties); see Figure 2 for an illustration.

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

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

Theorem 3.2. *All length-n strings satisfy $r = \mathcal{O}(z \log^2 n)$.*

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 $i \in [0\,..\,\lfloor \log n \rfloor]$, and observing that the number of LCP values 0 is exactly $\sigma \le z$. □

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 $\mathrm{LCP}[i] \in [\ell\,..\,2\ell)$, we associate cost $\lceil \frac{1}{2}\ell \rceil$ instead of ℓ. We then show that each of the strings in $\mathcal{S}_{3\ell}$ 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 $[\ell\,..\,2\ell)$ 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 ν of $\mathcal{T}$, by size(ν), we denote the number of leaves in the subtree of $\mathcal{T}$ rooted in ν. An edge connecting $\nu \ne \mathrm{root}(\mathcal{T})$ to its parent in $\mathcal{T}$ is called *light* if ν has a sibling ν' satisfying size(ν') ≥ size(ν) (see Figure 1). 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 $\nu_{\overline{S[1\,..\,\ell]}}$: Whenever $S[k]$, with $k \in [1\,..\,\ell]$, is charged, the edge connecting $\nu_{\overline{S[k\,..\,\ell]}}$ to its parent $\nu_{\overline{S[k+1\,..\,\ell]}}$ is light. We then noted that there are at most $\log|\mathcal{S}_\ell| \le \log n$ light edges on each root-to-leaf path in $\mathcal{T}$. Here, we charge the characters of strings in $\mathcal{S}_{3\ell}$ in the same way as in Lemma 3.1, but only for units $k \in [1\,..\,\lceil \frac{1}{2}\ell \rceil]$. This implies that only characters $S[k]$ of $S \in \mathcal{S}_{3\ell}$ with $k \le \lceil \frac{1}{2}\ell \rceil$ 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 $\lfloor \frac{1}{2}\ell \rfloor$ and its child.

Consider a light edge from a node ν to its parent u at depth at least $\lfloor \frac{1}{2}\ell \rfloor$. Let ν' be a sibling of ν satisfying size(ν') ≥ size(ν), and let S_ν, $S_{\nu'}$ be the labels of the paths from the root to ν and ν', respectively. These labels differ on the last position only so, by Fact 2.1, they cannot be both periodic. Let $\tilde{\nu} \in \{\nu, \nu'\}$ be such that $S_{\tilde{\nu}}$ is not periodic, and let $\tilde{m} = \mathrm{size}(\tilde{\nu})$.

Consider the set $\mathcal{S}$ of length-ℓ strings corresponding to the leaves in the subtree of $\mathcal{T}$ rooted at $\tilde{\nu}$ (i.e., the labels of the root-to-leaf paths passing through $\tilde{\nu}$). Define $\overline{\mathcal{S}} := \{\overline{P} : P \in \mathcal{S}\}$ and note that $\overline{\mathcal{S}} \subseteq \mathcal{S}_\ell$, because $\mathcal{T}$ is the trie of *reversed* strings from $\mathcal{S}_\ell$. Let $e_1 < \cdots < e_{\tilde{m}}$ denote the ending positions of the leftmost occurrences in $T^\infty[1\,.\,.)$ of strings in $\overline{\mathcal{S}}$. By definition, we have an occurrence of $\overline{S_{\tilde{\nu}}}$ ending in T^∞ at every position e_i with $i \in [1\,..\,\tilde{m}]$. Now, $\mathrm{per}(S_{\tilde{\nu}}) > \frac{1}{2}|S_{\tilde{\nu}}| \ge \frac{1}{4}\ell$ 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 $\overline{S_{\tilde{\nu}}}$ would yield $\mathrm{per}(\overline{S_{\tilde{\nu}}}) = \mathrm{per}(S_{\tilde{\nu}}) \le \frac{1}{4}\ell$). Consequently, at least $\frac{1}{4}|\mathcal{S}| = \frac{1}{4}\tilde{m}$ length-ℓ substrings of $T^\infty[1\,.\,.)$ have disjoint leftmost occurrences. Because 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 $\text{size}(v) \leq \text{size}(\tilde{v}) = \tilde{m} \leq 4z$.

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 $\text{size}(v) \leq 4z$. The number of the remaining light edges on the path is at most $\log(\text{size}(v)) \leq 2 + \log z$ by the standard bound applied to the subtree of $\mathcal{T}$ rooted at v. □

THEOREM 3.4. *Every string T 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 $[\ell \mathinner{..} 2\ell)$, we consider three cases:
$\boldsymbol{\ell \leq \log z}$. 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 ℓ. Thus, the number of irreducible LCP values in $[\ell \mathinner{..} 2\ell)$ is $\mathcal{O}(z\,\ell)$.
$\boldsymbol{\log z < \ell \leq \frac{n}{z}}$. We use the bound $\mathcal{O}(z \log z)$ of Lemma 3.3.
$\boldsymbol{\frac{n}{z} < \ell}$. We repeat the proof of Lemma 3.3, except that we observe that $|S_{3\ell}| \leq n$. Thus, the number of irreducible LCP values in $[\ell \mathinner{..} 2\ell)$ is $\mathcal{O}(\frac{n \log z}{\ell})$.
The above upper bounds, applied for every $\ell = 2^i$ with $i \in [0 \mathinner{..} \lfloor \log n \rfloor]$, yield

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

3.3. Tight bounds in terms of δ

Let $\delta = \max_{m=1}^{n} \frac{1}{m} |\mathcal{S}_m|$ denote the (cyclic) *substring complexity* of T.[13] 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 $\delta \leq z$ because $|\mathcal{S}_m| \leq mz$ 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 δ, so δ 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 $\mathcal{S}_\ell$ may have disjoint leftmost occurrences in $T^\infty[1 \mathinner{..})$. This observation is easy because the LZ77 parsing naturally yields a set of z positions (phrase boundaries) in T. The substring complexity δ does not provide such structure, but as the lemma here implies, we can replace z by 3δ in the aforementioned observation. The proof of Lemma 3.5 is a straightforward modification of the argument used in Kociumaka et al.[13] (Lemma 6). For completeness, we write down the full reasoning, with technical details tailored to our notation (e.g., $\mathcal{S}_\ell$ defined in terms of T^∞ rather than T).

LEMMA 3.5 (based on Kociumaka et al.,[13] LEMMA 6). *For any integer $\ell \geq 1$, the total number of positions in $T^\infty[1 \mathinner{..})$ covered by the leftmost occurrences of strings from $\mathcal{S}_\ell$ is at most $3\delta\ell$.*

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

We have thus shown that $|C'| = |\mathcal{S}| \leq |\mathcal{S}_{2\ell}|$. Because $|\mathcal{S}_{2\ell}| \leq 2\delta\ell$ holds by definition of δ, we obtain $|C| < |C'| + \ell \leq |\mathcal{S}_{2\ell}| + \ell \leq (2\delta+1)\ell \leq 3\delta\ell$. □

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

PROOF. Compared to the proof of Lemma 3.3, we use the bound $|\mathcal{S}_{3\ell}| \leq 3\ell\delta$ instead of $|\mathcal{S}_{3\ell}| \leq 3\ell z$. The only other modification required 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 $\text{size}(v) = \mathcal{O}(\delta)$.

Let $\tilde{m} \geq \text{size}(v)$ be defined as in the proof of Lemma 3.3. Recall that there are at least $\frac{\tilde{m}}{4}$ strings in $\mathcal{S}_\ell$ with disjoint leftmost occurrences in $T^\infty[1 \mathinner{..})$. By Lemma 3.5, there are at most 3δ such substrings. Thus, $\text{size}(v) \leq \tilde{m} \leq 12\delta$. □

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 δ.

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

The following construction, analyzed in the full version of this paper, provides tight examples with substring complexity δ covering the whole spectrum between $\mathcal{O}(1)$ and $\Omega(\frac{n}{\log n})$. For $\ell \geq 1$ and $x \in [0 \mathinner{..} 2^\ell)$, we set $\text{bin}_\ell(x) \in \{0, 1\}^\ell$ to be the binary representation of x.

LEMMA 3.8. *For all integers $\ell \geq 2$ and $K \geq 1$, the length n, the substring complexity δ, and the number of runs r in the BWT of a string $T_{\ell,K} \in \{\$, 0, 1, 2\}^+$, defined with*

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

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

If $\delta = \Omega(\frac{n}{\log n})$, on the other hand, then a trivial upper bound $r = \mathcal{O}(n)$ is tighter than that of Theorem 3.7. In this case, the following construction, also analyzed in the full version of this paper, provides tight examples.

LEMMA 3.9. *For all integers $\ell \geq 2$ and $\Delta \in \Omega(2^\ell) \cap \mathcal{O}(2^\ell \ell)$, the length n, the substring complexity δ, and the number of runs r in the BWT of a string $T'_{\ell,\Delta} \in \{\$_1, \ldots, \$_\Delta, 0, 1, 2\}^+$, defined with*

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

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

Overall, combining Lemmas 3.8 and 3.9, we obtain the following tight lower bound for δ ranging from $\mathcal{O}(1)$ to $\Omega(n)$.

Theorem 3.10. *For every $N \geq 1$ and $\Delta \in [1..N]$, there exists a string T whose length n, substring complexity δ, and the number of runs r in the BWT satisfy $n = \Theta(N)$, $\delta = \Theta(\Delta)$, and $r = \Theta(\min(n, \delta \log \delta \max(1, \log \frac{n}{\delta \log \delta})))$.*

3.4. Further combinatorial bounds

By combining Theorem 3.7 with known properties of the substring complexity δ, 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 3.11. *If r and $\overline{r}$ denote the number of runs in the BWT of a length-n text and its reverse, respectively, then $\overline{r} = \mathcal{O}(r \log r \max(1, \log \frac{n}{r \log r}))$.*

Proof. Because the value of δ is the same for the text and its reverse, Theorem 3.7 yields $\overline{r} = \mathcal{O}(\delta \log \delta \max(1, \log \frac{n}{\delta \log \delta}))$. Combining the results of Kempa and Prezza[12] (Theorem 3.9) and Kociumaka et al.[13] (Lemma 2) gives $\delta \leq r$. Consequently, we obtain $\overline{r} = \mathcal{O}(r \log r \max(1, \log \frac{n}{r \log r}))$. □

In the full version of this paper, we also characterize the sum of all irreducible LCP values:

Theorem 3.12. *For every string of length n, the sum r_Σ of all irreducible LCP values satisfies $r_\Sigma = \mathcal{O}(n \log \delta)$.*

Due to $\delta \leq r$, the presented upper bound is always (asymptotically) at least as strong as the previous bound $r_\Sigma \leq n \log r$ by Kärkkäinen et al.[9] Furthermore, it can be strictly stronger because $\log \delta = o(\log r)$ is possible when $\delta = \log^{o(1)} n$. In the full version of this paper, we show that the strings of Lemmas 3.8 and 3.9 yield a matching lower bound:

Theorem 3.13. *For every $N \geq 1$ and $\Delta \in [1..N]$, there exists a string T whose length n, substring complexity δ, and sum r_Σ of irreducible LCP values satisfy $n = \Theta(N)$, $\delta = \Theta(\Delta)$, and $r_\Sigma = \Theta(n \log \delta)$.*

4. FROM LZ77 TO RUN-LENGTH BWT

In this section, we outline our algorithm that, given the LZ77 parsing of a text $T \in \Sigma^n$, computes its run-length compressed BWT in $\mathcal{O}(z \text{ polylog } n)$ 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 4.1) and the compressed wavelet tree (Section 4.2). The sketch of the final algorithm is then presented in Section 4.3.

For any substring Y of T^∞, we define

$$\text{lpos}(Y) = \min\{i \in [1..n] : T^\infty[i..i+|Y|) = Y\}.$$

We say that a substring Y of T^∞ is *left-maximal* if there exist distinct symbols $a, b \in \Sigma$ such that the strings aY and bY are also substrings of T^∞. The following definition, assuming $\Sigma \cap \mathbb{N} = \emptyset$, plays a key role in our construction.

Definition 4.1 (BWT modulo ℓ). *Let $T \in \Sigma^n$, $\ell \geq 1$ be an integer, and $Y_i = T^\infty[\text{SA}[i]..\text{SA}[i]+\ell)$ for $i \in [1..n]$. We define the string $\text{BWT}_\ell \in (\Sigma \cup \mathbb{N})^n$, called the* BWT modulo ℓ *(of T), as follows. For $i \in [1..n]$,*

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

The algorithm runs in $k = \lceil \log n \rceil$ rounds. For $q \in [0..k)$, the input to the qth round is $\text{RL}(\text{BWT}_{2^q})$ and the output is $\text{RL}(\text{BWT}_{2^{q+1}})$. At the end of the algorithm, we have $\text{RL}(\text{BWT}_{2^k}) = \text{RL}(\text{BWT})$, because $X \in \mathcal{S}_{2^k}$ 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. Notice that $\text{BWT}_\ell[b..e] \in \Sigma^*$ implies $\text{BWT}_{\ell+1}[b..e] \in \Sigma^*$ (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, BWT_ℓ identifies the (leftmost occurrences of) substrings to be inspected in the qth round with the aim of replacing their corresponding runs in BWT_ℓ with previously unknown BWT symbols (as defined in $\text{BWT}_{2\ell}$).

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

Lemma 4.2. *For any $\ell \geq 1$, it holds $|\text{RL}(\text{BWT}_\ell)| < 2r$.*

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

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 $\text{RL}(\text{BWT}_\ell)$), or $c_{i+1} \in \mathbb{N}$, which is impossible because $\text{BWT}(\lambda_i..\lambda_{i+1}]$ would then be nonuniform. Thus, there are at most r runs of symbols from Σ in BWT_ℓ. □

4.1. Compressed string synchronizing sets

Our algorithm builds on the notion of *string synchronizing sets,* recently introduced by Kempa and Kociumaka.[11] Synchronizing sets are one of the most powerful techniques for sampling suffixes. In the uncompressed setting, they are the key in obtaining time-optimal solutions to multiple problems, such as a state-of-the-art BWT construction algorithm for texts over small alphabets.[11] 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 4.3 (τ-synchronizing set[11]). *Let $T \in \Sigma^n$ be a string and let $\tau \in \left[1..\left\lfloor \frac{n}{2} \right\rfloor\right]$ be a parameter. A set $\mathsf{S} \subseteq [1..n-2\tau+1]$ is called a* τ-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 $i \in \mathsf{S}$ holds if and only if $j \in \mathsf{S}$ (for $i, j \in [1..n-2\tau+1]$),*

2. $\mathsf{S} \cap [i\,..\,i+\tau) = \emptyset$ *if and only if* $\operatorname{per}(T[i\,..\,i+3\tau-2]) \le \frac{1}{3}\tau$ *(for* $i \in [1\,..\,n-3\tau+2]$).

In most applications, we want to minimize $|\mathsf{S}|$. However, the density condition imposes a lower bound $|\mathsf{S}| = \Omega(\frac{n}{\tau})$ for strings of length $n \ge 3\tau - 1$ that do not contain substrings of length $3\tau - 1$ that are periodic with period at most $\frac{1}{3}\tau$. Therefore, we cannot hope to achieve an upper bound improving in the worst case upon the following one.

THEOREM 4.4 *(Kempa and Kociumaka[11]). For any string T of length n and parameter* $\tau \in \left[1\,..\left\lfloor \frac{n}{2} \right\rfloor\right]$, *there exists a* τ*-synchronizing set* S *of size* $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$. *Moreover, such* S *can be (deterministically) constructed in* $\mathcal{O}(n)$ *time.*

Storing S for compressible strings presents the following challenge: Although $|\mathsf{S}| = o(\frac{n}{\tau})$ *is* sometimes possible, it *is not* implied by $z \ll n$. For example, as noted by Bille et al.,[3] Thue–Morse strings satisfy $z = \mathcal{O}(\log n)$ yet they contain no periodic substring X with $\operatorname{per}(X) < \frac{1}{2}|X|$, and thus their synchronizing sets satisfy $|\mathsf{S}| = \Omega(\frac{n}{\tau})$. This prevents us from keeping the plain representation of S when $\tau = o(\frac{n}{z})$.

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

DEFINITION 4.5 (COMPRESSED τ-SYNCHRONIZING SET). *Let* S *be a* τ*-synchronizing set of string* $T[1\,..\,n]$ *for some* $\tau \in \left[1\,..\left\lfloor \frac{n}{2} \right\rfloor\right]$, *and, for every* $j \in [1\,..\,z]$, *let* e_j *denote the last position of the jth phrase in the LZ77 parsing of T. For* $k \in \mathbb{Z}_{\ge 2}$, *we define the* compressed representation *of* S *as*

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

In the full version of this paper, 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.

THEOREM 4.6. *There exists a Las-Vegas randomized algorithm that, given the LZ77 parsing of a string* $T \in \Sigma^n$, *a parameter* $\tau \in \left[1\,..\left\lfloor \frac{n}{2} \right\rfloor\right]$, *and a constant* $k \in \mathbb{Z}_{\ge 2}$, *constructs in* $\mathcal{O}(z \log^5 n)$ *time a compressed representation* $\operatorname{comp}_k(\mathsf{S})$ *of a* τ*-synchronizing set* S *of T satisfying* $|\operatorname{comp}_k(\mathsf{S})| \le 72kz$.

4.2. Compressed wavelet trees

Along with string synchronizing sets, wavelet trees,[8] 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 because all strings are substrings of the text, which is stored in the LZ77 representation. In this section, we describe this novel variant of wavelet trees, dubbed here *compressed wavelet trees*. In particular, we prove an 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 Σ be an alphabet of size $\sigma \ge 1$. Consider a string $W[1\,..\,m]$ over the alphabet Σ^ℓ so that W is a sequence of $m \ge 0$ strings of length $\ell \ge 0$ over the alphabet Σ. The wavelet tree of W is the trie $\mathcal{T}$ of Σ with each node v_X associated to a string $B_X \in \Sigma^*$ defined here based on W. 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

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

Based on I_X, we define $B_X \in \Sigma^*$ so 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 4.7 (Grossi et al.[8]). *Let* $X \in \Sigma^d$, *where* $d \in [0\,..\,\ell)$. *For every* $c \in \Sigma$ *and* $j \in [1\,..\,|I_{Xc}|]$, *we have* $I_{Xc}[j] = I_X[i]$, *where* $B_X[i]$ *is the jth 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, except $\{v_\varepsilon\} \cup \{v_{W[i]}\}_{i=1}^{m}$, all nodes v_X satisfying $|\mathrm{RL}(B_X)| \le 1$ have been removed (the unary paths are collapsed into single edges with their labels concatenated). The shape and edge labels of the resulting tree are identical to the *compacted trie* of $\{W[1], \ldots, W[m]\}$.

We store the edge labels of $\mathcal{T}_c$ as pointers to substrings in W. We assume that each edge of $\mathcal{T}_c$ and each element of $\mathrm{RL}(B_X)$ can be encoded in $\mathcal{O}(1)$ space. Because $|\mathrm{RL}(B_Y)| \ge 1$ 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 $\mathrm{RL}(B_{Z'})$ for the parent $v_{Z'}$ of v_Z, the tree $\mathcal{T}_c$ uses $\mathcal{O}(1 + \sum_{v_X \in V(\mathcal{T}_c)} |\mathrm{RL}(B_X)|)$ space.

THEOREM 4.8. *If* $\mathcal{T}_c$ *is the compressed wavelet tree of a sequence W of length-*ℓ *strings, then* $\sum_{v_X \in V(\mathcal{T}_c)} |\mathrm{RL}(B_X)| = \mathcal{O}(1 + |\mathrm{RL}(W)| \log |\mathrm{RL}(W)|)$.

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

The proof resembles that of Lemma 3.1. With each $X \in \Sigma^*$ such that $|\mathrm{RL}(B_X)| \ge 2$, we associate $|\mathrm{RL}(B_X)| - 1$ units of cost and charge them to individual elements of W. We then show that each run in $\mathrm{RL}(W)$ 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 $|\mathrm{RL}(B_X)| \geq 2$; note that $d < \ell$. Let $\mathrm{RL}(B_X) = ((c_1, \lambda_1), \ldots, (c_h, \lambda_h))$. Observe that if we let $p_0 = I_X[\lambda_i]$ and $p_1 = I_X[\lambda_i + 1]$ for some $i \in [1 \,..\, h)$, 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 ith 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 $\mathrm{RL}(W)$. 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_ε 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. □

The key operation that we want to support on $\mathcal{T}_c$ is to compute the value $I_X[q]$ given $q \in [1 \,..\, |I_X|]$ and a pointer to $v_X \in V(\mathcal{T}_c)$. Let $W[1 \,..\, m]$ be a sequence of substrings of T^∞ of common length ℓ. Notice that if we have access to T, then the sequence W can be encoded in $\mathcal{O}(1 + |\mathrm{RL}(W)|)$ space. Namely, it suffices to store the length ℓ and the sequence $\mathrm{RL}((\mathrm{lpos}(W[i]))_{i \in [1\,..\,m]})$. In the full version of this paper, we show that, given such encoding of W and the LZ77 parsing of T, the compressed wavelet tree of W supporting fast primary index queries can be constructed efficiently.

Theorem 4.9. *Given the LZ77 representation of $T[1 \,..\, n]$ and a sequence $W[1 \,..\, m]$ of $m \leq n$ same-length substrings of T^∞, represented as $\mathrm{RL}((\mathrm{lpos}(W[i]))_{i \in [1\,..\,m]})$, the compressed wavelet tree of W, supporting primary index queries in time $\mathcal{O}(\log^4 n)$, can be constructed in time $\mathcal{O}((z + |\mathrm{RL}(W)|) \log^2 n)$.*

4.3. The algorithm

We are now ready to sketch the procedure constructing the sequences $\mathrm{RL}(\mathrm{BWT}_{2^q})$, where $q \in [0 \,..\, \lceil \log n \rceil]$.

Let $q \in [4 \,..\, \lceil \log n \rceil]$ (values $q \in [0 \,..\, 3]$ are handled separately) and let $\ell = 2^q$. We show how to compute $\mathrm{RL}(\mathrm{BWT}_{2\ell})$ given $\mathrm{RL}(\mathrm{BWT}_\ell)$ and the LZ77 parsing of T. The main idea of the algorithm is as follows.

Let S be a τ-synchronizing set of T, where $\tau = \lfloor \frac{\ell}{3} \rfloor$. As noted earlier, $\mathrm{BWT}_\ell[j] \in \Sigma$ implies $\mathrm{BWT}_{2\ell}[j] \in \Sigma$. Let $\mathrm{BWT}_\ell[y \,..\, y'] \in \mathbb{N}^+$ be a run in BWT_ℓ. By definition of BWT_ℓ, the suffixes of T^∞ starting at positions $i \in \mathrm{SA}[y \,..\, y']$ share a common prefix of length $\ell \geq 3\tau$. Thus, assuming that $\mathsf{S} \cap [i \,..\, i + \tau) = \varnothing$ holds for all $i \in \mathrm{SA}[y \,..\, y']$ (the periodic case is handled separately), by the consistency of S, all text positions $i \in \mathrm{SA}[y \,..\, y']$ share a common offset Δ with $i + \Delta = \min(\mathsf{S} \cap [i \,..\, i + \tau))$. This lets us deduce the order of length-2ℓ 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 $T[s \,..\, s + 2\ell)$ across $s \in \mathsf{S}$, we extract, using a wavelet tree, those preceded by $T[i \,..\, i + \Delta)$ (a prefix common to $T^\infty[i \,..)$ for $i \in \mathrm{SA}[y \,..\, y']$). Importantly, the synchronizing positions s sharing $T[s - \tau \,..\, s + 2\ell)$ can be processed together; hence, by Theorem 4.6, applied for $k = 7$ so that $2\ell \leq k\tau$, it suffices to use $\mathcal{O}(z)$ distinct substrings.

In the full version of the paper, we provide the details of the above construction and obtain the following result.

Theorem 4.10. *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 time $\mathcal{O}((r + z) \log^6 n) = \mathcal{O}(z \log^8 n)$.*

Acknowledgments

This work was performed when the first author was a postdoctoral scholar at the University of California, Berkeley (supported by NSF grants no. 1652303 and 1934846, and an Alfred P. Sloan Fellowship grant), whereas the second author was a postdoctoral scholar at Bar-Ilan University, Israel (supported by ISF grants no. 1278/16 and 1926/19, a BSF grant no. 2018364, and an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme, agreement no. 683064). **C**

References

1. Amir, A., Iliopoulos, C.S., Radoszewski, J. Two strings at Hamming distance 1 cannot be both quasiperiodic. *Inf. Process. Lett. 128*, (2017), 54–57.
2. Belazzougui, D., Cunial, F., Gagie, T., Prezza, N., Raffinot, M. Composite repetition-aware data structures. In *CPM* (2015), Springer, Cham, Switzerland, 26–39.
3. Bille, P., Gagie, T., Gørtz, I.L., Prezza, N. A separation between RLSLPs and LZ77. *J. Discrete Algorithms 50*, (2018), 36–39.
4. Burrows, M., Wheeler, D.J. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation. Palo Alto, CA, 1994.
5. Gagie, T., Navarro, G., Prezza, N. Fully-functional suffix trees and optimal text searching in BWT-runs bounded space. arXiv 1809.02792 (2018).
6. Gagie, T., Navarro, G., Prezza, N. On the approximation ratio of Lempel–Ziv parsing. In *LATIN* (2018), Springer, Cham, Switzerland, 490–503.
7. Gagie, T., Navarro, G., Prezza, N. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM 67*, 1 (2020), 1–54.
8. Grossi, R., Gupta, A., Vitter, J.S. High-order entropy-compressed text indexes. In *SODA* (2003), ACM/SIAM, Philadelphia, PA, USA, 841–850.
9. Kärkkäinen, J., Kempa, D., Piątkowski, M. Tighter bounds for the sum of irreducible LCP values. *Theor. Comput. Sci. 656*, (2016), 265–278.
10. Kempa, D. Optimal construction of compressed indexes for highly repetitive texts. In *SODA* (2019), SIAM, Philadelphia, PA, USA, 1344–1357.
11. Kempa, D., Kociumaka, T. String synchronizing sets: Sublinear-time BWT construction and optimal LCE data structure. In *STOC* (2019), ACM, New York, NY, USA, 756–767.
12. Kempa, D., Prezza, N. At the roots of dictionary compression: String attractors. In *STOC* (2018), ACM, New York, NY, USA, 827–840.
13. Kociumaka, T., Navarro, G., Prezza, N. Towards a definitive measure of repetitiveness. In *LATIN* (2020), Springer, Cham, Switzerland, 207–219.
14. Kreft, S., Navarro, G. On compressing and indexing repetitive sequences. *Theor. Comput. Sci. 483* (2013), 115–133.
15. Mäkinen, V., Belazzougui, D., Cunial, F., Tomescu, A.I. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing.* Cambridge University Press, Cambridge, UK, 2015.
16. Manzini, G. An analysis of the Burrows-Wheeler transform. *J. ACM 48*, 3 (2001), 407–430.
17. Navarro, G. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Comput. Surv. 54*, 2 (2021), 1–31.
18. Navarro, G. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Comput. Surv. 54*, 2 (2021), 1–32.
19. Ohlebusch, E. *Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction.* Oldenbusch Verlag, Bremen, Germany, 2013.
20. Ohno, T., Sakai, K., Takabatake, Y., Tomohiro, I, Sakamoto, H. A faster implementation of online RLBWT and its application to LZ77 parsing. *J. Discrete Algorithms*, (2018), 52-53:18–28.
21. Pevsner, J. *Bioinformatics and Functional Genomics*, 3rd edn. Wiley-Blackwell, Chichester, UK, 2015.
22. Policriti, A., Prezza, N. From LZ77 to the run-length encoded Burrows-Wheeler transform, and back. In *CPM* (2017), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 17:1–17:10.
23. Policriti, A., Prezza, N. LZ77 computation based on the run-length encoded BWT. *Algorithmica 80*, 7 (2018), 1986–2011.
24. Sirén, J., Välimäki, N., Mäkinen, V., Navarro, G. Run-length compressed indexes are superior for highly repetitive sequence collections. In *SPIRE* (2008), Springer, Berlin, Heidelberg, Germany, 164–175.
25. Ziv, J., Lempel, A. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory 23*, 3 (1977), 337–343.

Dominik Kempa ([kempa]@cs.jhu.edu), Johns Hopkins University, Baltimore, MD, USA.

Tomasz Kociumaka ([kociumaka]@berkeley.edu), University of California, Berkeley, CA, USA.