Resolution of the Burrows-Wheeler Transform Conjecture

Dominik Kempa\textsuperscript{1} and Tomasz Kociumaka\textsuperscript{*}\textsuperscript{,2}

\textsuperscript{1}Department of Computer Science, University of Warwick, United Kingdom
dominik.kempa@warwick.ac.uk

\textsuperscript{2}Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
kociumaka@mimuw.edu.pl

Abstract

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 some of the most popular lossless compression methods as well as of compressed indexes, central in modern bioinformatics. The compression ratio of BWT-based compressors, such as \texttt{bzip2}, is quantified by the number $r$ of maximal equal-letter runs in the BWT. This is also (up to polylog $n$ factors, where $n$ is the length of the text) the space used by the state-of-the-art BWT-based indexes, such as the recent $r$-index [Gagie et al., SODA 2018]. The output size of virtually every known compression method is known to be either within a polylog $n$ factor from $z$, the size of Lempel–Ziv (LZ77) parsing of the text, or significantly larger (by a $n^\varepsilon$ factor for $\varepsilon > 0$). The value of $r$ has resisted, however, all attempts and until now, no non-trivial upper bounds on $r$ were known.

In this paper, we show that every text satisfies $r = O(z \log^2 n)$. This result has a number of immediate implications: (1) it proves that a large body of work related to BWT automatically applies to the so-far disjoint field of Lempel–Ziv indexing and compression, e.g., it is possible to obtain full functionality of the suffix tree and the suffix array in $O(z \text{polylog } n)$ space; (2) it lets us relate the number of runs in the BWT of the text and its reverse; (3) it shows that many fundamental text processing tasks can be solved in the optimal time assuming that the text is compressible by a sufficiently large polylog $n$ factor using LZ77.

1 Introduction

Lossless text compression aims to exploit the redundancy in the data to represent it in a small space. Despite the abundance of compression programs, nearly every existing tool clearly falls into one of the very few general frameworks. As seen in the Large Text Compression Benchmark [24], the three methods underlying most implementations are Lempel–Ziv (LZ) compression [33, 34], Burrows–Wheeler Transform (BWT) [6], and Context Mixing (CM) [25]. Despite the good compression ratio, the CM method is usually orders of magnitude slower than the other two. Thus, the preferred methods in practice are either based on LZ (or more precisely, on LZ77 [33]) or BWT, underlying the popular \texttt{gzip} [13], \texttt{7-zip} [28], and \texttt{bzip2} [31] programs, for example. Outside of data compression, both LZ77 and BWT are common algorithmic tools, in particular, in \textit{compressed indexing} which aims to store a string in compressed form simultaneously supporting various queries (such as random access, pattern matching, or even suffix array queries) on the uncompressed data. This area has witnessed a surge of interest in recent years [1, 2, 3, 4, 8, 9, 10, 12, 19, 27, 30].

\textsuperscript{*}Supported by ISF grants no. 824/17 and 1278/16 and by an ERC grant MPM under the EU’s Horizon 2020 Research and Innovation Programme (grant no. 683064).
The central role of LZ77 and BWT is also motivated theoretically. With the exception of BWT, essentially every other known compression method has been proven \([11, 21]\) to produce output whose size is within a polylog\(n\) factor from \(z\), the output size of LZ77 algorithm (e.g., grammar compression \([7]\), collage systems \([22]\), or macro schemes \([32]\)), or larger by polynomial \((n^\epsilon\text{ for some }\epsilon > 0)\) factor (e.g., LZ78 \([34]\), compressed acyclic word graphs (CDAWGs) \([5]\)).

Given the central position of LZ77 and BWT both in data compression and compressed computation, one of the major open problems that emerged asks:

**Which of these two fundamental paradigms yields stronger compression?**

Recent efforts managed to solve the problem partially, by proving the relation \(z = \mathcal{O}(r \log n)\) \([11]\).

A similar bound in the opposite direction was generally conjectured to be false. After describing how to support suffix array and suffix tree queries in \([10]\), Gagie et al. speculate that "(...) it seems unlikely that one can provide suffix array or tree functionality within space related to \(g, z,\) or \(\gamma,\) since these measures are not related to the structure of the suffix array: this is likely to be a specific advantage of measure \(r\)."

**Our Contribution.** In this paper, we prove that \(r = \mathcal{O}(z \log^2 n)\) holds for all strings, resolving (in the way more surprising than anticipated) an open problem posted by Prezza \([29]\) and Gagie et al. \([10, 11]\). Our result has a number of implications for indexing and data compression:

1. It is possible to support suffix array and suffix tree functionality in \(\mathcal{O}(z \log n)\) space.
2. It implies the first non-trivial relation between the number of BWT runs \(r\) in the string and its reverse (denoted \(\bar{r}\)): \(r = \mathcal{O}(r \log^2 n)\). This result is achieved by a slight modification of our original proof to actually achieve \(r = \mathcal{O}(\delta \log^2 n)\), where \(\delta \leq z\) is a symmetric (insensitive to string reversal) repetitiveness measure recently introduced in \([23]\).
3. It was shown in \([19]\) that a large collection of the fundamental string processing tasks (including BWT and LZ77 construction) can be solved in \(\mathcal{O}(n/\log \sigma n + r \log n)\) time (where \(\sigma\) is the alphabet size). In other words, if the text is sufficiently compressible (formally, when \(n/r = \Omega(\log n)\)) by BWT, these tasks can be solved in optimal time (which, as shown in \([20]\), is unlikely to be possible for general texts). Our result implies that all these tasks can be solved optimally even when \(n/z = \Omega(\log n)\).

## 2 Preliminaries

We consider throughout a string (text) \(T[1...n]\) of \(n \geq 1\) symbols from an alphabet \(\Sigma\) of size \(\sigma\). We assume \(T[n] = \$\), where \(\$ \not\in \Sigma\) is lexicographically smaller than any symbol in \(\Sigma\). For \(i, j \in [1..n]\), we write \(T[i..j]\) to denote a substring of \(T\). If \(i > j\), we assume \(T[i..j]\) to be the string of length 0. By \(\bar{T}\) we denote the reverse of a string \(S\).

The *suffix array* \([26, 15]\) 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* \([6]\) \(BWT[1...n]\) of \(T\) is defined by \(BWT[i] = T[SA[i] - 1]\) if \(SA[i] > 1\) and \(BWT[i] = T[n]\) otherwise. By \(r\) we denote the number of runs, i.e., maximal same-character blocks, in BWT. We can efficiently represent this transform as the list of pairs \((\lambda_i, c_i)^r_{i=1}\), where \(\lambda_i > 0\) is the starting position of the \(i\)th run and \(c_i \in \Sigma\).

Let \(\text{LCE}(j_1, j_2)\) denote the length of the longest common prefix of the suffixes \(T[j_1..n]\) and \(T[j_2..n]\). The *LCP array* \([26, 18]\), \(\text{LCP}[1..n]\), is defined as \(\text{LCP}[i] = \text{LCE}(SA[i], SA[i - 1])\) for \(i = 2, \ldots, n\) (including \(BWT\) and LZ77 construction) can be solved in \(\mathcal{O}(n/\log \sigma n + r \log n)\) time.
irreducible otherwise (in particular, it is irreducible if \( i = 1 \)). Note that there are \( r \) irreducible LCP values. The significance of reducibility is summarized in the following theorem.

**Theorem 2.1** (Kärkkäinen et al. [16]). The sum of all irreducible LCP values is at most \( n \log r \).

The LZ77 factorization [33] uses the notion of the longest previous factors (LPF). The LPF at position \( i \) (denoted LPF\([i]\)) in \( T \) is a pair \((p_i, \ell_i)\) such that, \( p_i < i \) and \( \ell_i = \text{LCE}(p_i, i) > 0 \) is maximized. In other words, \( T[i..i+\ell_i-1] \) is the longest prefix of \( T[i..n] \) which also occurs at some position \( p_i < i \) in \( T \). If \( T[i] \) is the leftmost occurrence of a symbol in \( T \), then such a pair does not exist. In this case, we define \( p_i = T[i] \) and \( \ell_i = 0 \). Note that there may be more than one possibility for \( p_i \), and we do not care which one is used.

The LZ77 factorization (or 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 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 \( i + \ell_i \), unless \( \ell_i = 0 \), in which case the next phrase starts at position \( i + 1 \). For the example string \( T = \text{zzzzzipzip} \), the LZ77 factorization produces:

\[
(z, 0), (1, 4), (i, 0), (p, 0), (5, 3).
\]

We denote the number of phrases in the LZ77 parsing of \( T \) by \( z \). The following relation between \( z \) and \( r \) is known.

**Theorem 2.2** (Gagie et al. [11]). Every string of length \( n \) satisfies \( z = \mathcal{O}(r \log n) \).

## 3 Upper Bound

### 3.1 Basic Upper Bound

To illustrate the main idea of our proof technique, we first show the upper bound in its simplest form \( r = \mathcal{O}(z \log^2 n) \).

**Lemma 3.1.** For any \( \ell \in [1..n] \), the sum of irreducible LCP values smaller than \( \ell \) is \( \mathcal{O}(z \ell \log n) \).

**Proof.** Let \( T^\infty \) be an infinite string defined so that \( T^\infty[i] = T[((i - 1) \mod n) + 1] \) for \( i \in \mathbb{Z} \); in particular, \( T^\infty[1..n] = T[1..n] \). Note that the (infinite) suffixes of \( T^\infty \) satisfy \( T^\infty[SA[1]..] \prec \cdots \prec T^\infty[SA[n]..] \) and that \( \text{BWT}[i] = T^\infty[SA[i]-1] \) for \( i \in [1..n] \).

Denote \( S_m = \{ S \in \Sigma^m : S \text{ is a substring of } T^\infty \} \), where \( m \geq 1 \). 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 substrings overlapping two copies of \( T \), since \( T[n] = $ \) always forms a length-1 phrase in the parsing of \( T \).

The idea of the proof is as follows. With each irreducible LCP value \( k \in (0..\ell) \), we associate cost \( k \) which is charged to the characters of strings in \( S_{2\ell} \). We then show that each of the strings in \( S_{2\ell} \) is charged at most \( 2\log n \) times. The claim follows, since the sum of irreducible LCP values smaller than \( \ell \) equals the total cost, which is bounded by

\[
2 |S_{2\ell}| \log n \leq 4\ell z \log n.
\]

To devise the announced assignment of cost to the symbols of strings in \( S_{2\ell} \), consider the trie \( T \) of all reversed substrings of \( S_\ell \). Let \( \text{LCP}[i] \in (0..\ell) \) be an irreducible LCP value and note that \( i > 1 \) due to \( \text{LCP}[i] > 0 \). Let \( j_0 = SA[i-1] \) and \( j_1 = SA[i] \) so that \( k := \text{LCP}[i] = \text{LCE}(j_0, j_1) \). Since \( \text{LCP}[i] \) is irreducible, we have \( T^\infty[j_0 - 1] = \text{BWT}[i-1] \neq \text{BWT}[i] = T^\infty[j_1 - 1] \). For
$u \in [0..k)$, the $(u+1)$th unit of the cost $k$ associated with LCP[$u$] is charged to the character at position $\ell - u + 1$ (corresponding to symbol $T^\infty[j_h]$) of the substring $T^\infty[j_h - \ell + u..j_h + u + \ell) \in S_{2\ell}$, where $h \in \{0,1\}$ is such that the size (number of leaves) of the subtree of $T$ rooted in $v_{T^\infty[j_h - \ell + u..j_h + u)}$ is smaller than that rooted in $v_{T^\infty[j_1-h-1..j_1-h+u)}$ (in case of ties, we choose $h = 0$).

Observe that at most $\log n$ positions of each $S \in S_{2\ell}$ can be charged during the above procedure, since whenever a symbol $S[j]$, $j \in [3..\ell + 1]$, of $S$ is charged, the subtree of $T$ rooted at $v_{S[j]}$ is at least twice as large as the subtree rooted at $v_{S[j-1]}$, and this can happen for at most $\log |S| \leq \log n$ positions $j$.

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

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

**Proof.** By Lemma 3.1, for any $\ell \in [1..n]$, the number of runs in the BWT corresponding to irreducible LCP values in the range $[\frac{1}{\ell} \ell..\ell]$ is $O(z \log n)$. Thus, the claim follows by applying Lemma 3.1 for $\ell_i = 2^i$ with $1 \leq i \leq \lfloor \log n \rfloor$. (The number of all LCP values 0 is bounded by $\sigma \leq z$, so the number of irreducible LCP values 0 is also at most $z$.)

### 3.2 Upper Bound in Terms of $\delta$

Let $\delta = \max_{m=1}^n \frac{1}{m}|S_m|$ [23]. As observed in the proof of Lemma 3.1, $|S_m| \leq mz$ holds for every $m \geq 1$. By definition, $\delta \geq \frac{1}{m}|S_m|$, and hence we can replace $z$ with $\delta$ obtaining $|S_m| \leq m\delta$. Thus, we derive the following result.

**Theorem 3.3.** Every string of length $n$ satisfies $r = O(\delta \log^2 n)$.

**Corollary 3.4.** If $r$ and $\bar{r}$ denote the number of runs in BWT of a text and its reverse, respectively, then $\bar{r} = O(r \log^2 n)$.

**Proof.** Since the value of $\delta$ is the same for the text and its reverse, we obtain $r, \bar{r} = O(\delta \log^2 n)$. Combining [21, Theorem 3.9] and [23, Lemma 2] gives $\delta \leq r$. Consequently, we obtain the claim $\bar{r} = O(\delta \log^2 n) = O(r \log^2 n)$.

### 4 Concluding Remarks

This paper presents our result in the most basic variant. In an extended version, we will slightly improve Theorem 3.3: we will show that $r = O(\delta \log \delta \max(1, \log \frac{m}{n \log^2 \delta}))$ holds for all strings and provide a matching lower bound.

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

[1] Diego Arroyuelo, Gonzalo Navarro, and Kunihiko Sadakane. Stronger Lempel-Ziv based compressed text indexing. *Algorithmica*, 62(1-2):54–101, 2012. doi:10.1007/s00453-010-9443-8.

[2] Djamel Belazzougui, Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Queries on LZ-bounded encodings. In *Data Compression Conference, DCC 2015*, pages 83–92. IEEE, 2015. doi:10.1109/DCC.2015.69.

[3] Philip Bille, Mikko Berggren Ettienne, Inge Li Gørtz, and Hjalte Wedel Vildhøj. Time-space trade-offs for Lempel-Ziv compressed indexing. *Theoretical Computer Science*, 713:66–77, 2018. doi:10.1016/j.tcs.2017.12.021.

[4] Philip Bille, Gad M. Landau, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM Journal on Computing*, 44(3):513–539, 2015. doi:10.1137/130936889.

[5] Anselm Blumer, J. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, 1987. doi:10.1145/28869.28873.

[6] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994. URL: http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf.

[7] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.

[8] Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes, 2019. arXiv:1811.12779.

[9] Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A faster grammar-based self-index. In *6th International Conference on Language and Automata Theory and Applications (LATA 2012)*, volume 7183 of *LNCS*, pages 240–251. Springer, 2012. doi:10.1007/978-3-642-28332-1_21.

[10] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully-functional suffix trees and optimal text searching in BWT-runs bounded space, 2018. arXiv:1809.02792.

[11] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. On the approximation ratio of Lempel-Ziv parsing. In *13th Latin American Theoretical Informatics Symposium, LATIN 2018*, volume 10807 of *LNCS*, pages 490–503. Springer, 2018. doi:10.1007/978-3-319-77404-6_36.

[12] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in BWT-runs bounded space. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1459–1477. SIAM, 2018. doi:10.1137/1.9781611975031.96.

[13] J. Gailly and Mark Adler. gzip Homepage. www.gzip.org/. Accessed: 2019-10-19.

[14] John Kenneth Gallant. *String compression algorithms*. PhD thesis, Princeton University, 1982. URL: https://search.proquest.com/docview/303254400?accountid=14483.

[15] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. New indices for text: Pat trees and Pat arrays. In William B. Frakes and Ricardo A. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 66–82. Prentice–Hall, 1992.

[16] Juha Kärkkäinen, Dominik Kempa, and Marcin Piątkowski. Tighter bounds for the sum of irreducible LCP values. *Theoretical Computer Science*, 656:265–278, 2016. doi:10.1016/j.tcs.2015.12.009.
[17] Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Lazy Lempel-Ziv factorization algorithms. *ACM Journal of Experimental Algorithmics*, 21(1):2.4:1–2.4:19, 2016. doi:10.1145/2699876.

[18] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *12th Annual Symposium on Combinatorial Pattern Matching, CPM 2001*, volume 2089 of LNCS, pages 181–192. Springer, 2001. doi:10.1007/3-540-48194-X_17.

[19] Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1344–1357. SIAM, 2019. doi:10.1137/1.9781611975482.82.

[20] Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: Sublinear-time BWT construction and optimal LCE data structure. In *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.

[21] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In *50th Annual ACM Symposium on Theory of Computing, STOC 2018*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

[22] Takuya Kida, Tetsuya Matsumoto, Yusuke Shibata, Masayuki Takeda, Ayumi Shinohara, and Setsuo Arikawa. Collage system: A unifying framework for compressed pattern matching. *Theoretical Computer Science*, 298(1):253–272, 2003. doi:10.1016/S0304-3975(02)00426-7.

[23] Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive measure of repetitiveness, 2019. arXiv:1910.02151.

[24] Matt Mahoney. Large Text Compression Benchmark. mattmahoney.net/dc/text.html.

[25] Matt Mahoney. Adaptive weighing of context models for lossless data compression. Technical Report CS-2005-16, Florida Institute of Technology, Melbourne, Florida, 2005. URL: https://cs.fit.edu/~mmahoney/compression/cs200516.pdf.

[26] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993. doi:10.1137/0222058.

[27] Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theoretical Computer Science*, 762:41–50, 2019. doi:10.1016/j.tcs.2018.09.007.

[28] Igor Pavlov. 7-zip Homepage. https://www.7-zip.org/. Accessed: 2019-10-19.

[29] Nicola Prezza. Can Lempel-Ziv and Burrows-Wheeler compression be asymptotically compared? https://nms.kcl.ac.uk/iwoca/problems/Prezza2016_updated2019.pdf, 2016. IWOCA 2016 Open Problems.

[30] Nicola Prezza. Optimal rank and select queries on dictionary-compressed text. In *30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*, volume 128 of LIPIcs, pages 4:1–4:12. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CPM.2019.4.

[31] Julian Seward. bzip2 Homepage. www.sourceforge.org/bzip2/. Accessed: 2019-10-19.

[32] James A. Storer and Thomas G. Szymanski. The macro model for data compression. In *10th Annual ACM Symposium on Theory of Computing, STOC 1978*, pages 30–39. ACM, 1978. doi:10.1145/800133.804329.

[33] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

[34] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.

6