Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 387 (2007) 220–235
Theoretical Computer Science
www.elsevier.com/locate/tcs

A simpler analysis of Burrows–Wheeler-based compression☆

Haim Kaplan, Shir Landau, Elad Verbin*

School of Computer Science, Tel Aviv University, Tel Aviv, Israel

Abstract

In this paper, we present a new technique for worst-case analysis of compression algorithms which are based on the Burrows–Wheeler Transform. We mainly deal with the algorithm proposed by Burrows and Wheeler in their first paper on the subject [M. Burrows, D.J. Wheeler, A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994], called BW0. This algorithm consists of the following three essential steps: (1) Obtain the Burrows–Wheeler Transform of the text, (2) Convert the transform into a sequence of integers using the move-to-front algorithm, (3) Encode the integers using Arithmetic code or any order-0 encoding (possibly with run-length encoding).

We achieve a strong upper bound on the worst-case compression ratio of this algorithm. This bound is significantly better than bounds known to date and is obtained via simple analytical techniques. Specifically, we show that for any input string s, and $\mu > 1$, the length of the compressed string is bounded by $\mu \cdot |s| H_k(s) + \log(\zeta(\mu)) \cdot |s| + \mu g_k + O(\log n)$ where H_k is the kth order empirical entropy, g_k is a constant depending only on k and on the size of the alphabet, and $\zeta(\mu) = \frac{1}{1^\mu} + \frac{1}{2^\mu} + \cdots$ is the standard zeta function. As part of the analysis, we prove a result on the compressibility of integer sequences, which is of independent interest.

Finally, we apply our techniques to prove a worst-case bound on the compression ratio of a compression algorithm based on the Burrows–Wheeler Transform followed by distance coding, for which worst-case guarantees have never been given. We prove that the length of the compressed string is bounded by $1.7286 \cdot |s| H_k(s) + g_k + O(\log n)$. This bound is *better* than the bound we give for BW0.

Keywords: Text compression; Burrows–Wheeler Transform; Distance coding; Worst-case analysis

1. Introduction

In 1994, Burrows and Wheeler [6] introduced the Burrows–Wheeler Transform (BWT), and two new lossless text-compression algorithms that are based on this transform. Following [21], we refer to these algorithms as BW0 and BW0_{RL}. A well-known implementation of these algorithms is bzip2 [25]. This program typically shrinks an English text to about 20% of its original size while gzip only shrinks it to about 26% of the original size (see Table 1 and also [1] for detailed results). In this paper, we refine and tighten the analysis of BW0. For this purpose we introduce new techniques and statistical measures. We believe that these techniques may be useful in the analysis of other compression algorithms, and in predicting the performance of these algorithms in practice.

☆ This work was partially supported by Israel Science Foundation (ISF) grant no. 975/06.
* Corresponding author.
E-mail addresses: haimk@post.tau.ac.il (H. Kaplan), landaush@post.tau.ac.il (S. Landau), eladv@post.tau.ac.il (E. Verbin).

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.07.020

Table 1
Results (in bits) of running various compressors on the non-binary files from the Canterbury Corpus [1]

File name	Size	Gzip	Bzip2	BoostRleAc [11]
alice29.txt	1216 712	433 448	345 568	352 720
asyoulik.txt	1001 432	390 552	316 552	320 936
cp.html	196 824	63 720	61 056	62 256
fields.c	89 200	24 976	24 312	26 400
grammar.lsp	29 768	9 856	10 264	10 784
lcet10.txt	3414 032	1156 496	861 184	872 240
plrabn12.txt	3854 888	1556 408	1164 360	1161 816
xargs.1	33 816	13 984	14 096	14 384

The gzip results are taken from [1]. The column marked BoostRleAc [11] gives results, taken from [11], of the implementation of BoostRleAc. (BoostRleAc is the compression algorithm produced by boosting a Run-Length Encoder followed by Arithmetic Coding). We chose to compare to BoostRleAc because most often it gets the best compression ratio among all booster-based algorithms considered in [11].

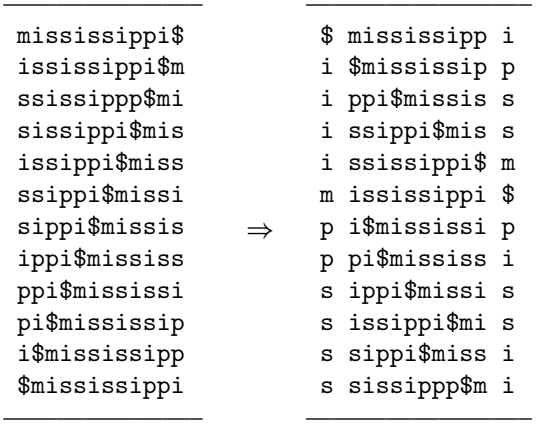

Fig. 1. The Burrows–Wheeler Transform for the string $s = mississippi$. The matrix on the right has the rows sorted in lexicographic order. The string $\hat{s}$ is the last column of the matrix, i.e. *ipssmpissii*, and we need to store the index of the symbol '$', i.e. 6, to be able to compute the original string.

The algorithm BW0 compresses the input text s in three steps.

(1) Compute the Burrows–Wheeler Transform, $\hat{s}$, of s. We elaborate on this stage shortly.

(2) Transform $\hat{s}$ to a string of integers $\dot{s} = \text{MTF}(\hat{s})$ by using the move-to-front algorithm. This algorithm maintains the symbols of the alphabet in a list and encodes the next character by its index in the list (see Section 2).

(3) Encode the string $\dot{s}$ of integers by using an order-0 encoder, to obtain the final bit stream $\text{BW0}(s) = \text{ORDER0}(\dot{s})$. An order-0 encoder assigns a unique bit string to each integer independently of its context, such that we can decode the concatenation of these bit strings. Common order-0 encoders are Huffman code or Arithmetic code.

The algorithm BW0_{RL} performs an additional run-length encoding (RLE) procedure between steps 2 and 3. See [6, 21] for more details on BW0 and BW0_{RL}, including the definition of run-length encoding which we omit here.

Next we define the Burrows–Wheeler Transform (BWT). Let n be the length of s. We obtain $\hat{s}$ as follows. Add a unique end-of-string symbol '$' to s. Place all the cyclic shifts of the string $s\$$ in the rows of an $(n+1) \times (n+1)$ conceptual matrix. One may notice that each row and each column in this matrix is a permutation of $s\$$. Sort the rows of this matrix in lexicographic order ('$' is considered smaller than all other symbols). The permutation of $s\$$ found in the last column of this sorted matrix, with the symbol '$' omitted, is the Burrows–Wheeler Transform, $\hat{s}$. See an example in Fig. 1. Although it may not be obvious at first glance, BWT is an invertible transformation, given that the location of '$' prior to its omission is known to the inverting procedure. In fact, efficient methods exist for computing and inverting $\hat{s}$ in linear time (see for example [22]).

The BWT is effective for compression since in $\hat{s}$ characters with the same context[1] appear consecutively. This is beneficial since if a reasonably small context tends to predict a character in the input text s, then the string $\hat{s}$ will show local similarity — that is, symbols will tend to recur at close vicinity.

Therefore, if s is say a text in English, we would expect $\hat{s}$ to be a string with symbols recurring at close vicinity. As a result $\dot{s} = \text{MTF}(\hat{s})$ is an integer string which we expect to contain many small numbers. (Note that by "integer string" we mean a string over an integer alphabet). Furthermore, the frequencies of the integers in $\dot{s}$ are skewed, and so an order-0 encoding of $\dot{s}$ is likely to be short. This, of course, is an intuitive explanation as to why BW0 "should" work on *typical* inputs. As we discuss next, our work is in a worst-case setting, which means that we give upper bounds that hold for *any* input. These upper bounds are relative to statistics which measure how "well-behaved" our input string is. An interesting question which we try to address is which statistics actually capture the compressibility of the input text.

Introductory definitions. Let s be the string which we compress, and let Σ denote the alphabet (set of symbols in S). Let $n = |s|$ be the length of s, and $h = |\Sigma|$. Let n_σ be the number of occurrences of the symbol σ in s. Let Σ^k denote the set of strings of length k over Σ. For a compression algorithm, A we denote by $\text{A}(s)$ the output of A on a string s. The *zeroth order empirical entropy* of the string s is defined as

$$H_0(s) = \sum_{i=0}^{h-1} \frac{n_i}{n} \log \frac{n}{n_i}.$$

(All logarithms in the paper are to the base 2. We define $0 \log 0 = 0$). For any word $w \in \Sigma^k$, let w_s denote the string consisting of the characters preceding all occurrences of w in s. The value

$$H_k(s) = \frac{1}{n} \sum_{w \in \Sigma^k} |w_s| \, H_0(w_s)$$

is called the kth *order empirical entropy* of the string s.[2]

We also use the *zeta function*, $\zeta(\mu) = \frac{1}{1^\mu} + \frac{1}{2^\mu} + \ldots$ and the *truncated zeta function* $\zeta_h(\mu) = \frac{1}{1^\mu} + \cdots + \frac{1}{h^\mu}$. We denote by $[h]$ the integers $\{0, \ldots, h-1\}$.

History and motivation. Define the *compression ratio* of a compression algorithm to be the average number of bits it produces per character in s. It is well-known that the zeroth order empirical entropy of a string s, $H_0(s)$, is a lower bound on the compression ratio of any order-0 compressor [19,7]. Similarly, the kth order empirical entropy of the string s, $H_k(s)$ gives a lower bound on the compression ratio of any encoder that is allowed to use only the context of length k preceding character x in order to encode it. For this reason, the compression ratio of compression algorithms is traditionally compared to $H_k(s)$, for various values of k. Another widely used statistic is $H_k^*(s)$, called the *modified* kth order empirical entropy of s. This statistic is slightly larger than H_k, yet it still provides a lower bound on the bits-per-character ratio of any encoder that is based on a context of k characters. We do not define H_k^* here, as we present bounds only in terms of H_k. See [21] for more details on H_k^*.

In 1999, Manzini [21] gave the first worst-case upper bounds on the compression ratio of several BWT-based algorithms. In particular, Manzini bounded the total bit-length of the compressed text $\text{BW0}(s)$ by the expression

$$8 \cdot n H_k(s) + (0.08 + C_{\text{ORDER0}}) \cdot n + \log n + g'_k. \tag{1}$$

for any $k \geq 0$. Here C_{ORDER0} is a small constant, defined in Section 2, which depends on the parameters of the Order-0 compressor which we are using, and $g'_k = h^k(2h \log h + 9)$ is a constant that depends only on k and h. Manzini also proved an upper bound of $5 \cdot n H_k^*(s) + g''_k$ on the bit-length of $\text{BW0}_{RL}(s)$, where g''_k is a different constant that depends only on k and h.

[1] The *context of length k* of a character is the string of length k following it.

[2] Actually, the traditional definition of $H_k(s)$ uses w_s which is the string consisting of the characters *following* all occurrences of w. We use a slightly non-standard definition in order to achieve compatibility with the definition of the BWT. Defining H_k in the traditional manner does not significantly effect the compression ratio (see [13] for a discussion on this).

In 2004, Ferragina, Giancarlo, Manzini and Sciortino [13] introduced a BWT-based compression booster. They show a compression algorithm such that the bit-length of its output is bounded by

$$1 \cdot nH_k(s) + C_{\text{ORDER0}}n + \log n + g_k'''. \tag{2}$$

(This algorithm follows from a general compression boosting technique. For details see [13]). As mentioned above, this result is optimal. This upper bound is optimal (up to the $C_{\text{ORDER0}}n + \log n + g_k'''$ term), in the sense that there is no algorithm that compresses every string s to a string of length $0.99 \cdot nH_k(s) + C_{\text{ORDER0}}n + \log n + g_k''$ bits. The upper bounds for this algorithm, and its variants based on the same techniques, are theoretically strictly superior to those in [21] and to those that we present here. However, implementations of the algorithm of [13] by the authors and another implementation by Ferragina, Giancarlo and Manzini [11],[3] give the results summarized in Table 1. These empirical results surprisingly imply that while the algorithm of [13] is optimal with respect to nH_k in a worst-case setting, its compression ratio in practice is comparable with that of algorithms with weaker worst-case guarantees. This seems to indicate that achieving good bounds with respect to H_k does not necessarily guarantee good compression results in practice. This was the starting point of our research. We looked for tight bounds on the length of the compressed text, possibly in terms of statistics of the text that might be more appropriate than H_k.

Our results. In this paper, we tighten the analysis of BW0 and give a tradeoff result that shows that for any constant $\mu > 1$ and for any k, the length of the compressed text is upper bounded by the expression

$$\mu \cdot nH_k(s) + (\log \zeta(\mu) + C_{\text{ORDER0}}) \cdot n + \mu g_k + O(\log n). \tag{3}$$

Here $g_k = (h^k + k) \cdot h \log h$. In particular, for $\mu = 1.5$ we obtain the bound $1.5 \cdot nH_k(s) + (1.5 + C_{\text{ORDER0}}) \cdot n + \log n + 1.5g_k$. For $\mu = 4.45$, we get the bound $4.45 \cdot nH_k(s) + (0.08 + C_{\text{ORDER0}}) \cdot n + 4.45g_k + O(\log n)$, thus surpassing Manzini's upper bound (1). Our proof is considerably simpler than Manzini's proof of (1).

The technique which we use to obtain this bound is even more interesting than the bound itself. We define a new natural statistic of a text s, which we call the *local entropy* of s, and denote it by $\text{LE}(s)$. This statistic was implicitly considered by Bentley et al. [4], and by Manzini [21]. Using two observations on the behavior of LE, we bypass some of the technical hurdles in the analysis of [21].

Our analysis actually proves a considerably stronger result: We define $\widehat{\text{LE}} = \text{LE}(\hat{s})$. That is the statistic $\widehat{\text{LE}}(s)$ is obtained by first applying the Burrows–Wheeler Transform to s and then computing the statistic LE of the result. We show that the size of the compressed text is bounded by

$$\mu \cdot \text{LE}(\hat{s}) + (\log \zeta(\mu) + C_{\text{ORDER0}}) \cdot n + O(\log n). \tag{4}$$

Empirically, this seems to give estimations which are quite close to the actual compression. See Table 2.

We note that the statistic $\widehat{\text{LE}}$ might seem strange, since its definition refers to the BWT itself. We believe that this statistic does have a theoretical value, because analyzing the compression ratio of BWT-based algorithms with respect to this statistic might highlight potential weaknesses of existing compression algorithms and thereby mark the way to invent better compression algorithms.

Here is an overview of the rest of the paper.

(1) We prove a result on compressibility of integer sequences in Section 3. This result is of independent interest.
(2) We define the statistic $\widehat{\text{LE}}$ in Section 2 and show its relation to H_k in Section 4.
(3) We use the last two contributions to give a simple proof of the bound (3). This can be found at the end of Section 4.
(4) We give a tighter upper bound for BW0 for the case that we are working over an alphabet of size 2. This can be found in Section 5.
(5) We outline a further application of our techniques to prove a worst-case bound on the compression of a different BWT-based compressor, which runs BWT, then the so-called distance coder (see [5,2]), and finally an order-0 encoder. The upper bounds proved are strictly superior to those proved for BW0. This can be found in Section 6. In Section 7 we prove a lower bound that shows that our approach cannot give better results for this compression algorithm.

[3] This is a technical report. The conference version of this paper is [12].

Table 2
Results (in bits) of computing various statistics on the non-binary files from the Canterbury Corpus [1]

File name	Size	$H_0(\hat{s})$	$LE(\hat{s})$	(4)	(3)	(1)
alice29.txt	1216712	386367	144247	396813	766940	2328219
asyoulik.txt	1001432	357203	140928	367874	683171	2141646
cp.html	196824	67010	26358	69857.6	105033.2	295714
fields.c	89200	24763	8855	25713	43379	119210
grammar.lsp	29768	9767	3807	10234	16054	45134
lcet10.txt	3414032	805841	357527	1021440	1967240	5867291
plrabn12.txt	3854888	1337475	528855	1391310	2464440	8198976
xargs.1	33816	13417	5571	13858	22317	64673

$H_0(\hat{s})$ gives the result of the algorithm BW0 assuming an optimal order-0 compressor. The final three columns show the bounds given by the Eqs. (4), (3), (1). The small difference between the column showing $H_0(\hat{s})$ and the column marked (4), shows that our bound (4) is quite tight in practice. It should be noted that in order to get the bound of (4) we needed to minimize the expression in (4) over μ. To get the bound of (3) and (1) we calculated their value for all k and picked the best one. We note that the reason for the figures measured in bits is that the theoretical bounds in the literature are customarily measured in bits.

Related work. A lot of work has been devoted recently to develop compressed text indices. A *compressed text index* of s is a compressed representation of s that allows fast pattern matching queries. Furthermore, it also allows to decompress efficiently part of, or the entire string s. The size of the representation is typically much smaller than that of the original text. A compressed text index is therefore simultaneously both a compression algorithm and an indexing data structure. Early progress on compressed text indices was made by Ferragina and Manzini in [14]. A recent result by Grossi, Gupta and Vitter [17] presents a compressed text index whose size is within additive lower-order terms of the order-k entropy of the input text. This result uses data structures for indexable dictionaries by R. Raman, V. Raman, and Rao [24]. For more on compressed text indexing, see [18,14,15].

We leave open the question of how our techniques can be applied to the subject of compressed text indexing.

2. Preliminaries

Our analysis does not use the definitions of H_k and BWT directly. Instead, it uses the following observation of Manzini [21], that $H_k(s)$ is equal to a linear combination of H_0 of parts of $\hat{s}$.

Proposition 1 (*[21]*). *Let s be a string of length n, and $\hat{s} = BWT(s)$. There is a partition $\hat{s} = \hat{s}_1 \dots \hat{s}_t$, with $t \leq h^k + k$, such that:*

$$nH_k(s) = \sum_{i=1}^{t} \left|\hat{s}_i\right| H_0(\hat{s}_i). \tag{5}$$

For completeness, we prove this proposition.

Proof. Consider the lexicographically-sorted matrix of the cyclic shifts of s. This matrix was used as part of the definition of the BWT. Partition the rows of this matrix into t blocks, each consisting of a set of consecutive rows. For each word $w \in \Sigma^k$ that appears as a substring of s there is a block that contains all rows that begin with w. So far we have accounted for all but k of the rows — those in which one of the first k characters is the end-of-string symbol '$'. Each of these rows is in a separate special block. In total, there are $t \leq h^k + k$ blocks.

Now, let $\hat{s}_i$ be the substring of $\hat{s}$ that consists of the last column of the ith block. The reason that (5) holds is that each string $\hat{s}_i$ is a permutation of a string w_s (recall that w_s is the string consisting of the characters preceding all occurrences of w in s). The only exceptions to this are the strings that arise from the k "special" blocks. These have entropy 0, so they do not effect either side of (5). Since permuting the characters of a string s' does not effect $H_0(s')$, correctness of Eq. (5) follows. □

Now we define the move-to-front (MTF) transformation, which was introduced in [4]. MTF encodes the character $s[i] = \sigma$ with an integer equal to the number of distinct symbols encountered since the previous occurrence of σ in s. More precisely, the encoding maintains a list of the symbols ordered by recency of occurrence. When the next symbol arrives, the encoder outputs its current rank and moves it to the front of the list. Therefore, a string over the alphabet

Σ is transformed to a string over $[h]$ (note that the length of the string does not change). To completely determine the encoding we must specify the status of the recency list at the beginning of the procedure. We denote by MTF_π the algorithm in which the initial status of the recency list is given by the permutation π of Σ.

MTF has the property that if the input string has high local similarity, that is if symbols tend to recur at close vicinity, then the output string will consist mainly of small integers. We define the *local entropy* of a string s as follows:

$$\text{LE}_\pi(s) = \sum_{i=1}^{n} \log(\text{MTF}_\pi(s)[i] + 1).$$

That is, LE is the sum of the logarithms of the move-to-front values plus 1 and so it depends on the initial permutation of MTF's recency list. For example, for a string "aabb" and initial list where 'b' is before 'a', $\text{LE}_\pi(s) = 2$ because the MTF values of the second a and the second b are 0, and the MTF values of the first a and the first b are 1. We also define $\text{LE}_W(s) = max_\pi \text{LE}_\pi(s)$. This is the "worst-case" local entropy.[4] Analogously, MTF_W is MTF with an initial recency list that maximizes $\text{LE}_\pi(s)$. We write LE instead of LE_W or LE_π when the initial permutation of the recency list is not significant. (Note that the difference between $\text{LE}_{\pi_1}(s)$ and $\text{LE}_{\pi_2}(s)$ is always $O(h \log h)$). Similarly, we write MTF instead of MTF_W or MTF_π when the initial permutation of the recency list is not significant. We define $\widehat{\text{LE}}_\pi(s) = \text{LE}_\pi(\hat{s})$. The statistic LE was used implicitly in [4,21].

Note that $\text{LE}_\pi(s)$ is the number of bits one needs to write the sequence of integers $\text{MTF}_\pi(s)$ in binary. Optimistically, this is the size we would like to compress the text to. Of course, one cannot decode the integers in $\text{MTF}_\pi(s)$ from the concatenation of their binary representations as these representations are of variable lengths.

The statistics $H_0(s)$ and $H_k(s)$ are normalized in the sense that they represent lower bounds on the *bits-per-character* rate attainable for compressing s, which we call the *compression ratio*. However, for our purposes it is more convenient to work with un-normalized statistics. Thus we define our new statistic LE to be un-normalized. We define the statistics nH_0 and nH_k to be the un-normalized counterparts of the original statistics, i.e. $(nH_0)(s) = n \cdot H_0(s)$ and $(nH_k)(s) = n \cdot H_k(s)$.

Let $f : \Sigma^* \to \mathbb{R}^+$ be an (un-normalized) statistic on strings, for example f can be nH_k or LE.

Definition 2. A compression algorithm A is called (μ, C)-f-competitive if for every string s it holds that $|A(s)| \leq \mu f(s) + Cn + o(n)$, where $o(n)$ denotes a function $g(n)$ such that $\lim_{n\to\infty} \frac{g(n)}{n} = 0$.

Throughout the paper, we refer to an algorithm ORDER0. By this we mean any order-0 algorithm, which is assumed to be a $(1, C_{\text{ORDER0}})$-nH_0-competitive algorithm. For example, $C_{\text{HUFFMAN}} = 1$ and $C_{\text{ARITHMETIC}} \approx 10^{-2}$ for a specific time-efficient implementation of Arithmetic code [26,23]. Furthermore, one can implement arithmetic coding without any optimizations. This gives a compression algorithm for which the bit-length of the compressed text is bounded by $nH_0(s) + O(\log n)$. This algorithm is $(1, 0)$-nH_0-competitive, and thus we can use $C_{\text{ORDER0}} = 0$ in our equations. This implementation of arithmetic coding is theoretically interesting, but is not time-efficient in practice.

We often use the following inequality, derived from Jensen's inequality:

Lemma 3. *For any $k \geq 1$, $x_1, \ldots, x_k > 0$ and $y_1, \ldots, y_k > 0$ it holds that:*

$$\sum_{i=1}^{k} y_i \log x_i \leq \left(\sum_{i=1}^{k} y_i\right) \cdot \log\left(\frac{\sum_{i=1}^{k} x_i y_i}{\sum_{i=1}^{k} y_i}\right). \tag{6}$$

In particular, this inequality implies that if one wishes to maximize the sum of logarithms of k elements under the constraint that the sum of these elements is S, then one needs to pick all the elements to be equal to S/k. (This statement is equivalent to the arithmetic–geometric means inequality).

[4] LE_W is defined to make the presentation more elegant later on, but one could use $\text{LE}_\pi(s)$ for some fixed permutation π, and the analysis would be very similar.

3. Optimal results on compression with respect to SL

In this section, we look at a string s of length n over the alphabet $[h]$. We define the *sum of logarithms statistic*: $\mathrm{SL}(s) = \sum_{i=1}^{n} \log(s[i]+1)$. Note that $\mathrm{LE}(s) = \mathrm{SL}(\mathrm{MTF}(s))$. We show that in a strong sense the best SL-competitive compression algorithm is an order-0 compressor. At the end of this section, we show as to how to get from this good LE-competitive and $\widehat{\mathrm{LE}}$-competitive compression algorithms.

This problem is related to the problem of universal encoding of non-negative integers, where one wants to find a prefix-free encoding for integers, $U : \{0,1,2,\ldots\} \to \{0,1\}^*$. Unlike the case for Huffman coding, The code U cannot depend on the encoded text itself, so the code is fixed in advance. Only after fixing such a code, we measure its compression ratio on a text that consists of non-negative integers. The (not completely concise) objective is to compress well texts which tend to have a large proportion of small numbers. This objective can be made more specific depending on the circumstances. Some well-known universal codes are the Elias gamma, delta, and omega codes, [9], the Fibonacci code, and others. For more information on universal codes see the recent survey of Fenwick [10], or the older survey of Lelewer and Hirschberg [20].

In our setting, we are interested in universal codes U such that for every $x \geq 0$, $|U(x)| \leq \mu \log(x+1) + C$. One such particularly nice universal encoding is the Fibonacci encoding [3,16], for which $\mu = \log_\phi 2$ and $C = 1 + \log_\phi \sqrt{5} \simeq 2.6723$, where $\phi = \frac{\sqrt{5}+1}{2}$.

Clearly, a universal encoding scheme with parameters μ and C gives an (μ, C)-SL-competitive compressor. However, in this section we get a better competitive ratio, taking advantage of the fact that our goal is to encode a long sequence from $[h]$, while allowing an $o(n)$ additive term.

An optimal* (μ, C)-SL-*competitive algorithm. We show that the algorithm ORDER0 is $(\mu, \log \zeta(\mu) + C_{\mathrm{ORDER0}})$-SL-competitive *for any* $\mu > 1$. In fact, we prove a somewhat stronger theorem:

Theorem 4. *For any constant* $\mu > 0$*, the algorithm* ORDER0 *is* $(\mu, \log \zeta_h(\mu) + C_{\mathrm{ORDER0}})$-SL-*competitive.*

Proof. Let s be a string of length n over alphabet $[h]$. Clearly, it suffices to prove that for any constant $\mu > 0$

$$nH_0(s) \leq \mu \mathrm{SL}(s) + n \log \zeta_h(\mu). \tag{7}$$

From the definition of H_0 it follows that $nH_0(s) = \sum_{i=0}^{h-1} n_i \log \frac{n}{n_i}$, and from the definition of SL we get that $\mathrm{SL}(s) = \sum_{j=1}^{n} \log(s[j]+1) = \sum_{i=0}^{h-1} n_i \log(i+1)$. So, (7) is equivalent to

$$\sum_{i=0}^{h-1} n_i \log \frac{n}{n_i} \leq \mu \sum_{i=0}^{h-1} n_i \log(i+1) + n \log \zeta_h(\mu). \tag{8}$$

Pushing the μ into the logarithm and moving terms around we get that (8) is equivalent to

$$\sum_{i=0}^{h-1} n_i \log \frac{n}{n_i (i+1)^\mu} \leq n \log \zeta_h(\mu). \tag{9}$$

Defining $p_i = \frac{n_i}{n}$, and dividing the two sides of the inequality by n we get that (9) is equivalent to

$$\sum_{i=0}^{h-1} p_i \log \frac{1}{p_i (i+1)^\mu} \leq \log \zeta_h(\mu).$$

Using Lemma 3 we obtain that

$$\sum_{i=0}^{h-1} p_i \log \frac{1}{p_i (i+1)^\mu} = \sum_{\substack{0 \leq i \leq h-1 \\ p_i \neq 0}} p_i \log \frac{1}{p_i (i+1)^\mu} \leq \log \left(\sum_{\substack{0 \leq i \leq h-1 \\ p_i \neq 0}} p_i \frac{1}{p_i (i+1)^\mu} \right)$$

$$= \log \left(\sum_{\substack{0 \leq i \leq h-1 \\ p_i \neq 0}} \frac{1}{(i+1)^\mu} \right) \leq \log \zeta_h(\mu). \quad \square$$

In particular, we get the following corollary.

Corollary 5. *For any constant $\mu > 1$, the algorithm* ORDER0 *is* $(\mu, \log \zeta(\mu) + C_{\text{ORDER0}})$-SL-*competitive.*

A lower bound for* SL*-competitive compression. Theorem 4 shows that for any $\mu > 0$ there exists a $(\mu, \log \zeta_h(\mu) + C_{\text{ORDER0}})$-SL-competitive algorithm. We now show that for any fixed values of μ and h, there is no algorithm with better competitive ratio. The lower bounds that we get in this section do not include the constant C_{ORDER0}.

Theorem 6. *Let $\mu > 0$ be some constant. For any $C < \log \zeta_h(\mu)$ there is no (μ, C)-SL-competitive algorithm.*

Proof. For any $\epsilon > 0$, we need to show that no algorithm A satisfies

$$|\text{A}(s)| \leq \mu \text{SL}(s) + (\log \zeta_h(\mu) - \epsilon)\, n + o(n)$$

for every string s of length n. Clearly, if such an algorithm A exists then

$$|\text{A}(s)| \leq \mu \text{SL}(s) + \left(\log \zeta_h(\mu) - \epsilon'\right) n$$

for some $0 < \epsilon' < \epsilon$ and for every string s such that its length n is large enough.

So, in order to establish the theorem we show that for any algorithm A, $\epsilon > 0$ and large enough n, there exists a string s of length n such that

$$|\text{A}(s)| > \mu \text{SL}(s) + (\log \zeta_h(\mu) - \epsilon)\, n, \tag{10}$$

We achieve this by giving a family of strings $S(n)$ for each n such that if n is large enough there must be a string in $S(n)$ that satisfies (10). We prove this by a counting argument.

Let $\alpha_i = n \cdot \frac{1}{\zeta_h(\mu)\cdot(i+1)^\mu}$ for $i \in [h]$. Assume for now that α_i is an integer for every $i \in [h]$. We later show as to how to get rid of this assumption. Let $S(n)$ be the set of strings where integer i appears α_i times. Let $L(n) = \sum_{i=0}^{h-1} \log(i+1) \cdot \alpha_i$ and $N(n) = \frac{n!}{\alpha_0! \cdot \ldots \cdot \alpha_{h-1}!}$. Note that for each $s \in S(n)$, $|s| = n$, $\text{SL}(s) = L(n)$, and $|S(n)| = N(n)$.

Using standard information-theoretic arguments, our algorithm A must compress at least one of the strings in $S(n)$ to at least $\log N(n)$ bits. Thus, it suffices to prove that for large enough n,

$$\log N(n) > \mu L(n) + (\log \zeta_h(\mu) - \epsilon)\, n. \tag{11}$$

We now show a lower bound on $\log N(n) - \mu L(n)$ which implies (11). Using Stirling's approximation $n! = (1 + o(1))\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, so for n large enough, $(1/2)\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq (3/2)\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$. We obtain that

$$\begin{aligned}
\log N(n) &\geq \log \frac{(1/2)\sqrt{2\pi n}\,(n/e)^n}{(3/2)^h \prod_{i=0}^{h-1} \sqrt{2\pi\alpha_i}\,(\alpha_i/e)^{\alpha_i}} \\
&= \log \frac{(1/2)\sqrt{2\pi n}}{(3/2)^h \prod_{i=0}^{h-1} \sqrt{2\pi\alpha_i}} + n \log n - \sum_{i=0}^{h-1} \alpha_i \log \alpha_i \\
&\geq -O(1) - h \log(2\pi n) + \sum_{i=0}^{h-1} \alpha_i \log(n/\alpha_i) \\
&\geq -O(\log n) + \sum_{i=0}^{h-1} \alpha_i \log(n/\alpha_i),
\end{aligned} \tag{12}$$

and therefore

$$\begin{aligned}
\log N(n) - \mu L(n) &= \log N(n) - \mu \sum_{i=0}^{h-1} \log(i+1) \cdot \alpha_i \\
&\geq -O(\log n) + \sum_{i=0}^{h-1} \alpha_i \log(n/\alpha_i) - \mu \sum_{i=0}^{h-1} \log(i+1) \cdot \alpha_i \\
&= -O(\log n) + \sum_{i=0}^{h-1} \alpha_i \log \frac{n}{\alpha_i (i+1)^{\mu}} \\
&= -O(\log n) + \sum_{i=0}^{h-1} \alpha_i \log \zeta_h(\mu) \\
&= -O(\log n) + n \log \zeta_h(\mu), \qquad (13)
\end{aligned}$$

which for large enough n gives (11). (The next to last equality follows by substituting $\alpha_i = n \cdot \frac{1}{\zeta_h(\mu)\cdot(i+1)^{\mu}}$ inside the logarithmic term).

Now we address the fact that for every $i \in [h]$, α_i is not necessarily an integer. For $0 \leq i \leq h-1$, define α'_i to be an integer such that $\lfloor \alpha_i \rfloor \leq \alpha'_i \leq \lceil \alpha_i \rceil$, and such that $\sum_{i=0}^{h-1} \alpha'_i = n$. In other words, we define new values that differ by at most 1, such that the sum is conserved. We define the family of sequences $S(n)$ using α'_i rather than α_i. Repeating the calculations above with α'_i rather than α_i, we get

$$\log N(n) - \mu L(n) \geq -O(\log n) + \sum_{i=0}^{h-1} \alpha'_i \log \frac{n}{\alpha'_i (i+1)^{\mu}}. \qquad (14)$$

Since for every i it holds that $\left|\alpha_i - \alpha'_i\right| \leq 1$, we have

$$\begin{aligned}
&\left| \sum_{i=0}^{h-1} \alpha'_i \log \frac{n}{\alpha'_i (i+1)^{\mu}} - \sum_{i=0}^{h-1} \alpha_i \log \frac{n}{\alpha_i (i+1)^{\mu}} \right| \\
&\quad \leq \left| \sum_{i=0}^{h-1} \alpha'_i \log \frac{n}{(i+1)^{\mu}} - \sum_{i=0}^{h-1} \alpha_i \log \frac{n}{(i+1)^{\mu}} \right| + \sum_{i=0}^{h-1} \left| \alpha'_i \log \alpha'_i - \alpha_i \log \alpha_i \right| \\
&\quad \leq O(h \log n).
\end{aligned}$$

Using this together with (14) gives

$$\log N(n) - \mu L(n) \geq -O(h \log n) + n \log \zeta_h(\mu), \qquad (15)$$

which for large enough n gives (11). □

By setting a large enough alphabet in the proof of Theorem 6, we get the following corollaries:

Corollary 7. *For any $\mu > 1$ and C such that $C < \log \zeta(\mu)$, there is no (μ, C)-SL-competitive algorithm.*

Proof. Suppose in contradiction that there exist $\mu > 1, \epsilon > 0$, and a compression algorithm A such that A is $(\mu, \log \zeta(\mu) - \epsilon)$-SL-competitive. Since $\zeta_h(\mu) \xrightarrow{h\to\infty} \zeta(\mu)$, we can choose h to be an integer such that $\log \zeta_h(\mu) > \log \zeta(\mu) - \frac{\epsilon}{2}$. Thus A is $(\mu, \log \zeta_h(\mu) - \frac{\epsilon}{2})$-SL-competitive. This is a contradiction to Theorem 6. □

Similarly,

Corollary 8. *For any $C \in \mathbb{R}$, there is no $(1, C)$-SL-competitive algorithm.*

Analogous Results With Respect To $\widehat{\text{LE}}$. From Theorem 4 we get

Corollary 9. *For any constant $\mu > 0$, the algorithm* BW0 *is $(\mu, \log \zeta_h(\mu) + C_{\text{ORDER0}})$-$\widehat{\text{LE}}$-competitive.*

Proof. Substituting the string $\text{MTF}(\text{BWT}(s))$ into Theorem 4 gives

$$|\text{ORDER0}(\text{MTF}(\text{BWT}(s)))| \leq \mu \text{SL}(\text{MTF}(\text{BWT}(s))) + (\log \zeta_h(\mu) + C_{\text{ORDER0}})n + o(n),$$

which is exactly

$$|\text{BW0}(s)| \leq \mu \widehat{\text{LE}}(s) + (\log \zeta_h(\mu) + C_{\text{ORDER0}})n + o(n),$$

as required. □

And similarly, Corollary 5 implies:

Corollary 10. *For any constant $\mu > 1$, the algorithm* BW0 *is $(\mu, \log \zeta(\mu) + C_{\text{ORDER0}})$-$\widehat{\text{LE}}$-competitive.*

On the other hand, it is not clear whether the result of Theorem 6 can be used to get the following conjecture:

Conjecture 11. *For any $\mu > 0$ and $C < \log \zeta_h(\mu)$, there is no (μ, C)-$\widehat{\text{LE}}$-competitive algorithm.*

This conjecture would follow from Theorem 6 if the transformations MTF_π and BWT, viewed as functions from Σ^n to Σ^n, were invertible. (Recall that the function $\text{BWT}(s)$ is the outcome of running the Burrows–Wheeler Transform on $s\$$ and then deleting the symbol '$' from the result). But, while MTF_π is invertible, BWT is not.[5] This means that potentially, the image of the transformation BWT could be a small fraction of Σ^n that has better compressibility properties with respect to LE.

4. The entropy hierarchy

In this section we show that the statistics nH_k and $\widehat{\text{LE}}$ form a hierarchy, which allows us to percolate upper bounds down and lower bounds up. Specifically, we show that for each k,

$$\widehat{\text{LE}}(s) \leq nH_k(s) + O(1) \tag{16}$$

where the $O(1)$ term depends on k and h (recall that h is the size of the alphabet). The known entropy hierarchy is

$$\cdots \leq nH_k(s) \leq \cdots \leq nH_2(s) \leq nH_1(s) \leq nH_0(s). \tag{17}$$

Which in addition to (16) gives us:

$$\widehat{\text{LE}}(s) \cdots \lessapprox \cdots \leq nH_k(s) \leq \cdots \leq nH_2(s) \leq nH_1(s) \leq nH_0(s). \tag{18}$$

($O(1)$ additive terms are hidden in the last formula).

Thus any (μ, C)-$\widehat{\text{LE}}$-competitive algorithm is also (μ, C)-nH_k-competitive. To establish this hierarchy we need to prove two properties of LE_W: that it is at most $nH_0 + O(1)$, and that it is convex (in a sense which we will define).

***Some Properties of* LE.** Some of the following claims can be found, explicitly or implicitly, in [21,4]. Specifying them here in this form would help to understand the rest of the analysis. We give references where appropriate.

Define $\text{MTF}_{\textit{ignorefirst}}(s)$ to be a string which is identical to $\text{MTF}_\pi(s)$ except that we omit the integers representing the first occurrence of each symbol (so $\text{MTF}_{\textit{ignorefirst}}(s)$ is of length less than n). Note that in this case when we perform the move-to-front transformation the initial status of the MTF recency list is not significant. Similarly, define $\text{LE}_{\textit{ignorefirst}}(s) = \sum_i \log(\text{MTF}_{\textit{ignorefirst}}(s)[i] + 1)$.

The following is a theorem of Bentley et al. [4].

Theorem 12 (*[4]*). $\text{LE}_{\textit{ignorefirst}}(s) \leq nH_0(s)$.

[5] Take for example the string $s' =$ "bac" over the alphabet $\Sigma = \{a, b, c\}$. String s' is not equal to $\text{BWT}(s)$ for any string s. To see this, suppose in contradiction that there is such s. In the table of lexicographically-sorted cyclically-shifted suffixes of s, the leftmost column is "$abc" while the rightmost column is s' with the character '$' inserted somewhere. One can easily check that no matter where the '$' is inserted, some row must have the same symbol in both the first and last columns, which is impossible since each row of the table is a permutation of s.

Proof. We look separately at the contributions of each of the h different symbols to $\mathrm{LE}_{ignorefirst}(s)$. The contribution of σ to $\mathrm{LE}_{ignorefirst}(s)$ is[6]

$$A_\sigma = \sum_{i:s[i]=\sigma} \log(\mathrm{MTF}_{ignorefirst}(s)[i]+1).$$

It is easy to see that

$$\sum\nolimits_{i:s[i]=\sigma} (\mathrm{MTF}_{ignorefirst}(s)[i]+1) \le n.$$

Let n_σ be the number of occurrences of σ in s. Then using Lemma 3 we get

$$A_\sigma \le n_\sigma \log \frac{\sum_{i:s[i]=\sigma} (\mathrm{MTF}_{ignorefirst}(s)[i]+1)}{n_\sigma} \le n_\sigma \log \frac{n}{n_\sigma}.$$

Summing over all σ we obtain that

$$\mathrm{LE}_{ignorefirst}(s) = \sum_\sigma A_\sigma \le \sum_\sigma n_\sigma \log \frac{n}{n_\sigma} = nH_0(s),$$

as needed. □

Manzini [21] gave the following corollary of this theorem.

Lemma 13 (*[21], Lemma 5.4*). $\mathrm{LE}_W(s) \le nH_0(s) + h\log h$.

Proof. $\mathrm{LE}_W(s)$ is equal to $\mathrm{LE}_{ignorefirst}(s)$ plus the contribution of the first occurrence of each symbol. The number of such contributions is at most h, and each such contribution is bounded by $\log h$, and so we get $\mathrm{LE}_W(s) \le \mathrm{LE}_{ignorefirst}(s) + h\log h \le nH_0(s) + h\log h$. □

We now prove that LE_W is a convex statistic. The intuition behind this is that the encoding MTF_π has a locality property in the sense that if you stop it in the middle and start again from this point using a different recency list then you make little profit if any.

Lemma 14 (LE_W *is a Convex Statistic, Implicitly Stated in [21]*). *Let* $s = s_1 \dots s_t$. *Then* $\mathrm{LE}_W(s) \le \sum_i \mathrm{LE}_W(s_i)$.

Proof. From the definition of LE_W we get that $\mathrm{LE}_W(s) = \sum_{j=1}^n \log(\mathrm{MTF}_{\pi_1}(s)[j]+1)$ for a worst-case permutation π_1. Let us look at the recency list π_i that we use when the $\mathrm{LE}_W(s)$ calculation reaches sub-string s_i. Each of the summands of $\sum_i \mathrm{LE}_W(s_i)$ is calculated with a worst-case permutation, which must be at least as bad as π_i, and the lemma follows. □

The hierarchy result.

Theorem 15. *For any* $k \ge 0$ *and any string* s *of length* n,

$$\widehat{\mathrm{LE}_W}(s) \le nH_k(s) + (h^k + k)\cdot h\log h$$

Proof. Let $\hat{s} = BWT(s)$. By Proposition 1 there is a partition $\hat{s} = \hat{s}_1 \dots \hat{s}_t$, such that $t \le h^k + k$ and $nH_k(s) = \sum_{i=1}^t |\hat{s}_i| H_0(\hat{s}_i)$. Observe that using the convexity of LE_W (Lemma 14) we have

$$\widehat{\mathrm{LE}_W}(s) = \mathrm{LE}_W(\hat{s}) \le \sum_{i=1}^t \mathrm{LE}_W(\hat{s}_i).$$

[6] Note that for the sake of convenience, in the following equations we are disregarding the fact that some elements of $\mathrm{MTF}_{ignorefirst}(s)$ are in a shifted position relative to the characters of s that they represent because the representations of the first appearances of symbols are omitted.

Now, from Lemma 13 we have

$$\sum_{i=1}^{t} \mathrm{LE}_W(\hat{s}_i) \leq \sum_{i=1}^{t} \left|\hat{s}_i\right| H_0(\hat{s}_i) + th\log h = nH_k(s) + th\log h,$$

and using $t \leq h^k + k$ the theorem follows. □

Main results. Using Theorem 4 together with Theorem 15 gives the main result of this paper:

Theorem 16. *For any $k \geq 0$ and for any constant $\mu > 1$, the algorithm* BW0 *is $(\mu, \log\zeta(\mu) + C_{\mathrm{ORDER0}})$-$nH_k$-competitive.*

Proof. Corollary 10 gives that for any string s of length n, $|\mathrm{BW0}(s)| \leq \mu\widehat{\mathrm{LE}}(s) + (\log\zeta(\mu) + C_{\mathrm{ORDER0}} + o(1))n$. Using this together with Theorem 15 gives that for any string s, $|\mathrm{BW0}(s)| \leq \mu nH_k(s) + (\log\zeta(\mu) + C_{\mathrm{ORDER0}} + o(1))n$, which gives the theorem. □

Similarly, using Corollary 9 and Theorem 15 we obtain:

Theorem 17. *For any $k \geq 0$ and for any constant $\mu > 0$, the algorithm* BW0 *is $(\mu, \log\zeta_h(\mu) + C_{\mathrm{ORDER0}})$-$nH_k$-competitive on strings from an alphabet of size h.*

A note on bounds. All bounds discussed in the paper are of the form $|\mathrm{A}(s)| \leq \alpha f(s) + \beta n + o(n)$, where A is a compression algorithm, and f is some statistic such as nH_k, LE, etc. One might ask what about other types of bounds. For example, why not try to prove, say, $|\mathrm{A}(s)| \leq \alpha f(s) + \beta n\log f(s) + o(n)$. The reason that we concentrate on bounds of the former kind is that most bounds in the relevant literature are of this form. This fact is not surprising, since these type of bounds seem relatively easy to work with.

5. An upper bound and a conjecture about BW0

We now prove an upper bound on the performance of BW0 in a specific setting. This bound is tighter than the upper bound of Theorem 16.

Theorem 18. BW0 *is $(2, C_{\mathrm{ORDER0}})$-nH_0-competitive for texts over an alphabet of size 2.*

Proof. Let s be a string of length n over the alphabet $\Sigma = \{a, b\}$. Let n_a be the number of times the symbol 'a' appears in s, and let $p_a = \frac{n_a}{n}$. Suppose w.l.o.g. that $p_a \leq \frac{1}{2}$. We consider the following cases. In each case we prove that

$$H_0(\mathrm{MTF}(\hat{s})) \leq 2H_0(s). \tag{19}$$

Case 1: $p_a = 0$. Here (19) is trivial.
Case 2: $0 < p_a \leq \frac{1}{4}$. The number of 'a's in $\hat{s}$ is equal to n_a. Notice that for every 'a' in $\hat{s}$ there can be at most two '1's in $\mathrm{MTF}(\hat{s})$. Therefore the number of '1's in $\mathrm{MTF}(\hat{s})$ is at most $2n_a \leq \frac{n}{2}$. From the monotonicity of the entropy function[7] it follows that

$$H_0(\mathrm{MTF}(\hat{s})) \leq -2p_a\log(2p_a) - (1-2p_a)\log(1-2p_a),$$

while on the other hand,

$$H_0(s) = -p_a\log p_a - (1-p_a)\log(1-p_a),$$

[7] This is the reason that we need 2 cases. The entropy function $H(p) = -p\log p - (1-p)\log(1-p)$ is monotonically increasing only in the range $p \in (0, \frac{1}{2}]$, so we need to treat the case where $p_a \in (\frac{1}{4}, \frac{1}{2}]$ separately.

and therefore,

$$\begin{aligned} H_0(\mathrm{MTF}(\hat{s})) - 2H_0(s) &\le -2p_a \log(2p_a) - (1-2p_a)\log(1-2p_a) \\ &\quad + 2p_a \log p_a + 2(1-p_a)\log(1-p_a) \\ &= -2p_a - (1-2p_a)\log(1-2p_a) + 2(1-p_a)\log(1-p_a). \end{aligned}$$

Calculating derivative with respect to p_a, one can see that this function is monotonically decreasing. Thus, proving that this expression tends to 0 when p_a tends to 0 (from above) is enough. This fact can be easily verified.
Case 3: $\frac{1}{4} \le p_a \le \frac{1}{2}$. In this case $H_0(s) \ge \frac{1}{2}$ so $H_0(\mathrm{MTF}(\hat{s})) \le 1 \le 2H_0(s)$. Therefore (19) also holds in this case.
In either case we get the following:

$$|\mathrm{BW0}(s)| \le nH_0(\mathrm{MTF}(\hat{s})) + C_{\mathrm{ORDER0}}n \le 2nH_0(s) + C_{\mathrm{ORDER0}}n,$$

so the algorithm BW0 is $(2, C_{\mathrm{ORDER0}})$-nH_0-competitive over an alphabet of size 2. □

We believe that this upper bound is true for larger alphabets as well. Specifically, we leave the following conjecture as an open problem.

Conjecture 19. BW0 *is* $(2, C_{\mathrm{ORDER0}})$-nH_k-*competitive.*

Furthermore, we conjecture that this is optimal:

Conjecture 20. *For any* $\mu < 2$, BW0 *is not* $(\mu, 0)$-nH_k-*competitive.*

Here we are in a similar situation to that we had in Conjecture 11. If the transformation BWT was invertible, then Conjecture 20 would be easy to prove. However, BWT is not invertible.

6. A (1.7286, C_{ORDER0})-nH_k-competitive algorithm

In this section we analyze the BWT *with distance coding* compression algorithm, $\mathrm{BW}_{\mathrm{DC}}$. This algorithm was invented but not published by Binder (see [5,2]), and is described in a paper of Deorowicz [8]. The distance-coding procedure, DC, will be described shortly. The algorithm $\mathrm{BW}_{\mathrm{DC}}$ compresses the text by running the Burrows–Wheeler Transform, then the distance-coding procedure, and then an Order-0 compressor. It also adds to the compressed string auxiliary information consisting of the positions of the first and last occurrence of each character. In this section we prove that $\mathrm{BW}_{\mathrm{DC}}$ is (1.7286, C_{ORDER0})-nH_k-competitive.

First we define the DIST transformation: DIST encodes the character $s[i] = \sigma$ with an integer equal to the number of characters encountered since the previous occurrence of the symbol σ. Therefore, DIST is the same as MTF, except that instead of counting the number of distinct symbols between two consecutive occurrences of σ, it counts the number of characters. In DIST we disregard the first occurrence of each symbol.

The transformation DC converts a text (which would be in our case the Burrows–Wheeler Transform of the original text) to a sequence of integers by applying DIST to s and disregarding all zeroes.[8] It follows that DC produces one integer per block of consecutive occurrences of the same character σ. This integer is the distance to the previous block of consecutive occurrences of σ. It is not hard to see that from $\mathrm{DC}(s)$ and the auxiliary information we can recover s. A formal proof of this fact is in Appendix.

As a tool for our analysis, we define a new statistic of texts, LD. The LD statistic is similar to LE, except that it counts all characters between two successive occurrences of a symbol, instead of disregarding repeating symbols. Specifically, $\mathrm{LD}(s) = \sum_i \log(\mathrm{DIST}(s)[i] + 1)$. For example, the LD value of the string "abbbab" is $\log 4 + \log 2 = 3$. From the definition of LD and DC, it is easy to see that

$$\mathrm{SL}(\mathrm{DC}(s)) = \mathrm{LD}(s). \tag{20}$$

Now we wish to prove that $\mathrm{BW}_{\mathrm{DC}}$ is (1.7286, C_{ORDER0})-nH_k-competitive. We repeat the work of Sections 3 and 4 using LD instead of LE and get the desired result. We omit the proofs of the following lemma and theorem and only give an overview, because the proofs are identical or almost identical to the proofs of the original statements.

[8] This is a simplified version of [8]. Our upper bound applies to the original version as well, since the original algorithm just adds a few more optimizations that may produce an even shorter compressed string.

We first prove, along the lines of Corollary 5, that for any constant $\mu > 1$ and any integer string s all of whose elements are at least 1, the algorithm ORDER0 is $(\mu, \log(\zeta(\mu) - 1) + C_{\text{ORDER0}})$-SL-competitive. The term -1 appears here as the summation that used to give the term $\zeta(\mu)$ now starts at $i = 1$ instead of $i = 0$. From this together with (20) we get the following lemma.

Lemma 21. *The algorithm* DC+ORDER0 *is* $(\mu, \log(\zeta(\mu) - 1) + C_{\text{ORDER0}})$-LD-*competitive.*

We now prove analogously to Lemma 13 that $\text{LD}(s) \leq nH_0(s)$. Then we prove along the lines of Lemma 14 that if $s = s_1 \dots s_t$ then $\text{LD}(s) \leq \sum_i \text{LD}(s_i) + (t-1)h \log n$. All of this together gives the following theorem.

Theorem 22. *If* A *is a* (μ, C)-LD-*competitive algorithm, then* BWT+A *is a* (μ, C)-nH_k-*competitive algorithm for any* $k \geq 0$.

From Theorem 22 together with Lemma 21 we get:

Theorem 23. *For any* $k \geq 0$ *and for any constant* $\mu > 1$*, the algorithm* BW_{DC} *is* $(\mu, \log(\zeta(\mu) - 1) + C_{\text{ORDER0}})$-$nH_k$-*competitive for any* $k \geq 0$

Let $\mu_0 \approx 1.7286$ be the real number such that $\zeta(\mu_0) = 2$. Substituting $\mu = \mu_0$ in the statement of Theorem 23 gives:

Corollary 24. *For any* $k \geq 0$*, the algorithm* BW_{DC} *is* $(\mu_0, C_{\text{ORDER0}})$-$nH_k$-*competitive.*

7. A lower bound with respect to LD

We now prove that using the approach of Section 6 one cannot get a $(1, 0)$-nH_k-competitive algorithm. Specifically, we show:

Theorem 25. *For any* $\mu < \mu_0$*, there is no* $(\mu, 0)$-LD-*competitive algorithm. This holds even if the alphabet size is* 2.

This means that a different approach is needed to get a $(\mu, 0)$-nH_k-competitive algorithm for $\mu < 1.7286$.

Proof. Suppose in contradiction that there is a compression algorithm A which works on the alphabet $\Sigma_1 = \{a, b\}$ and is $(\mu, 0)$-LD-competitive where $\mu < \mu_0$. Since

$$\zeta_h(\mu) \xrightarrow{h \to \infty} \zeta(\mu) > 2,$$

we can choose an integer h such that $\zeta_h(\mu) > 2$. We construct a compression algorithm B which works over the alphabet $\Sigma_2 = \{1, 2, \dots, h\}$ and is $(\mu, 0)$-SL-competitive. We then argue that there cannot be a $(\mu, 0)$-SL-competitive algorithm which works over Σ_2, thereby getting a contradiction.

Given a string s_2 of length n_2 over alphabet Σ_2, algorithm B translates it to a string s_1 of length $n_1 \leq hn_2$ over Σ_1. The string s_1 starts with $s_2[0]$ 'a's, followed by $s_2[1]$ 'b's, followed by $s_2[2]$ 'a's, and so on. Then algorithm B uses algorithm A to compress s_1 and returns the result, that is $\text{B}(s_2) = \text{A}(s_1)$. Clearly, one can recover s_2 from $\text{B}(s_2)$ since the transformation from s_2 to s_1 is invertible.

It is not hard to see that $\text{LD}(s_1) \leq \text{SL}(s_2)$ (the inequality here is from the fact that the first and last characters of s_2 have no impact on $\text{LD}(s_1)$). Thus,

$$|\text{B}(s_2)| = |\text{A}(s_1)| \leq \mu \text{LD}(s_1) + o(n_1) \leq \mu \text{SL}(s_2) + o(n_2), \tag{21}$$

where we could say that the $o(n_1)$ term is also $o(n_2)$ because h only depends on μ, and is independent of n_1 and n_2. From (21) follows that B is $(\mu, 0)$-SL-competitive. We now argue that a $(\mu, 0)$-SL-competitive algorithm which works over Σ_2 does not exist.

One can show analogously to Theorem 6 that there is no constant $C < \log(\zeta_h(\mu) - 1)$ such that there exists a (μ, C)-SL-competitive algorithm that works over alphabet $\{1, 2, \dots, h\}$. The term -1 appears here, since the summation that used to give the term $\zeta(\mu)$ now starts at $i = 1$ instead of $i = 0$. Since h was chosen such that $\zeta_h(\mu) > 2$, algorithm B which is $(\mu, 0)$-SL-competitive does not exist, and the theorem follows. □

***A conjectured lower bound with respect to* LD.** Actually, we would have liked to prove the following lower bound which is somewhat stronger than Theorem 25. We leave this as an open problem.

Conjecture 26. *Let $\mu > 1$ be some constant. Then there is no constant $C < \log(\zeta(\mu) - 1)$ such that there exists a (μ, C)-*LD*-competitive algorithm.*

While Theorem 25 holds even for binary alphabet, it might be the case that this conjecture only holds for asymptotically large alphabet, so for any $\mu > \mu_0$ and for any fixed alphabet size h there might be a constant $C_h(\mu) < \log(\zeta(\mu) - 1)$ such that there is a $(\mu, C_h(\mu))$-LD-competitive algorithm. If this is the case, it is interesting whether the algorithm DC+ORDER0 achieves the optimal ratio for each alphabet size.

8. Conclusions and further research

We leave the following idea for further research: In this paper, we prove that the algorithm BW0 is $(\mu, \log \zeta(\mu))$-$\widehat{\text{LE}}$-competitive. On the other hand, Ferragina et al. [13] show an algorithm which is $(1, 0)$-nH_k-competitive. A natural question to ask is whether there is an algorithm that achieves both ratios. Of course, one can just perform both algorithms and use the shorter result. But the question is whether a direct simple algorithm with such performance exists. We are also curious as to whether the insights gained in this work can be used to produce a better BWT-based compression algorithm.

Acknowledgments

We would like to thank Nir Markus for his work on the implementations. We also thank Gadi Landau and Adi Avidor for their helpful discussions and useful references. We thank Giovanni Manzini and Paolo Ferragina for sharing with us a preliminary implementation of their compression booster. The third author would like to thank Roberto Grossi for some insightful discussions. Finally, we would like to thank the referees for their helpful comments.

Appendix. DC is an invertible transformation

We prove that DC is an invertible transformation. In this section we consider a version of DC that is different from the one discussed in Section 6 in that for each character $s[i] = \sigma$ we write the distance to the next occurrence of σ, instead of the distance to the previous occurrence of σ. This is symmetric and it simplifies the presentation.

The key to the algorithm is to know, at each step of the decoding process, the location of the next occurrence in s of each of the symbols. At the beginning of the process we obviously have this information, because this is part of the auxiliary information that we saved. Denote by $next_\sigma$ the location of the next occurrence of character σ in s. Suppose we next produce the ith character $s[i]$ of s. Then there must be a character σ_0 such that $next_{\sigma_0} = i$. Let $\sigma_1 \neq \sigma_0$ be the character such that $next_{\sigma_1}$ is minimum, and let $j = next_{\sigma_1}$. Then clearly, $s[i], \ldots, s[j-1]$ are all equal to σ_0, and the next integer x in DC(s) gives the distance from $j-1$ of the next occurrence of σ_0 after location $j-1$. We set $next_{\sigma_0} = j + x$ and continue.

The decoding algorithm can be implemented using a heap to run in time $O(n \log h)$.

References

[1] The Canterbury Corpus, http://corpus.canterbury.ac.nz.
[2] Jürgen Abel, Web page about distance coding, http://www.data-compression.info/Algorithms/DC/.
[3] A. Apostolico, A.S. Fraenkel, Robust transmission of unbounded strings using Fibonacci representations, IEEE Transactions on Information Theory 33 (2) (1987) 238–245.
[4] J.L. Bentley, D.D. Sleator, R.E. Tarjan, V.K. Wei, A locally adaptive data compression scheme, Communications of the ACM 29 (4) (1986) 320–330.
[5] E. Binder, Distance coder, Usenet group comp.compression, 2000.
[6] M. Burrows, D.J. Wheeler, A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.
[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press, McGraw-Hill, 2001, pp. 385–392 (Chapter 16.3).

[8] S. Deorowicz, Second step algorithms in the Burrows–Wheeler compression algorithm, Software - Practice and Experience 32 (2) (2002) 99–111.
[9] P. Elias, Universal codeword sets and representation of the integers, IEEE Transactions on Information Theory 21 (2) (1975) 194–203.
[10] P. Fenwick, Universal codes, in: K. Sayood (Ed.), Lossless Data Compression Handbook, Academic Press, 2003.
[11] P. Ferragina, R. Giancarlo, G. Manzini, The engineering of a compression boosting library: Theory vs practice in BWT compression, Technical Report TR-INF-2006-06-03-UNIPMN, Universitá degli Studi del Piemonte Orientale, June 2006.
[12] P. Ferragina, R. Giancarlo, G. Manzini, The engineering of a compression boosting library: Theory vs practice in BWT compression, in: Proc. 14th European Symposium on Algorithms, ESA '06, 2006, pp. 756–767.
[13] P. Ferragina, R. Giancarlo, G. Manzini, M. Sciortino, Boosting textual compression in optimal linear time, Journal of the ACM 52 (2005) 688–713.
[14] P. Ferragina, G. Manzini, Indexing compressed text, Journal of the ACM 52 (2005) 552–581.
[15] P. Ferragina, G. Manzini, V. Mākinen, G. Navarro, An alphabet friendly FM-index, in: Proc. 11th Symposium on String Processing and Information Retrieval, SPIRE '04, 2004, pp. 150–160.
[16] A.S. Fraenkel, S.T. Klein, Robust universal complete codes for transmission and compression, Discrete Applied Mathematics 64 (1) (1996) 31–55.
[17] R. Grossi, A. Gupta, J.S. Vitter, High-order entropy-compressed text indexes, in: Proc. 14th ACM-SIAM Symposium on Discrete algorithms, SODA '03, 2003, pp. 841–850.
[18] R. Grossi, A. Gupta, J.S. Vitter, When indexing equals compression: Experiments with compressing suffix arrays and applications, in: Proc. 15th ACM-SIAM Symposium on Discrete Algorithms, SODA '04, 2004, pp. 636–645.
[19] D.A. Huffman, A method for the construction of minimum-redundancy codes, Proceedings of the Institute of Radio Engineers 40 (9) (1952) 1098–1101.
[20] D.A. Lelewer, D.S. Hirschberg, Data compression, ACM Computing Surveys 19 (3) (1987) 261–296.
[21] G. Manzini, An analysis of the Burrows–Wheeler transform, Journal of the ACM 48 (3) (2001) 407–430.
[22] G. Manzini, P. Ferragina, Engineering a lightweight suffix array construction algorithm, Algorithmica 40 (2004) 33–50.
[23] A. Moffat, R.M. Neal, I.H. Witten, Arithmetic coding revisited, ACM Transactions on Information Systems 16 (3) (1998) 256–294.
[24] R. Raman, V. Raman, S.S. Rao, Succinct indexable dictionaries with applications to encoding k-ary trees and multisets, in: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms, SODA '02, 2002, pp. 233–242.
[25] J. Seward, bzip2, a program and library for data compression, http://www.bzip.org/.
[26] I.H. Witten, R.M. Neal, J.G. Cleary, Arithmetic coding for data compression, Communications of the ACM 30 (6) (1987) 520–540.