A Simpler Analysis of Burrows-Wheeler Based Compression

Haim Kaplan

School of Computer Science, Tel Aviv University, Tel Aviv, Israel; email: haimk@post.tau.ac.il

Shir Landau

School of Computer Science, Tel Aviv University, Tel Aviv, Israel; email: landaush@post.tau.ac.il

Elad Verbin

School of Computer Science, Tel Aviv University, Tel Aviv, Israel; email: eladv@post.tau.ac.il

---

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 deal mainly with the algorithm purposed by Burrows and Wheeler in their first paper on the subject [6], 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 $\mu \cdot |s|H_k(s) + \log(\zeta(\mu)) \cdot |s| + g_k$ where H_k is the k-th order empirical entropy, g_k is a constant depending only on k and on the size of the alphabet, and $\zeta(\mu) = \frac{1}{1^\mu} + \frac{1}{2^\mu} + \dots$ 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 $1.7286 \cdot |s|H_k(s) + g_k$. This bound is *better* than the bound we give for BW0.

Preprint submitted to Elsevier Science *2 March 2006*

Key words: 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 [18], we refer to these algorithms as BW0 and $\mathrm{BW0}_{RL}$. A well known implementation of these algorithms is bzip2 [23]. This program typically shrinks an English text to about 20% of its original size while gzip only shrinks 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 these techniques may be useful for the analysis of other compression algorithms, and in predicting the performance of these algorithms in practice.

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. [1]
(2) Transform $\hat{s}$ to a string of integers $\dot{s} = \mathrm{MTF}(\hat{s})$ 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 $\mathrm{BW0}(s) = \mathrm{ORDER0}(\dot{s})$. 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 $\mathrm{BW0}_{RL}$ performs an additional run-length encoding (RLE) procedure between steps 2 and 3. See [6,18] for more details on BW0 and $\mathrm{BW0}_{RL}$, 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 ('\$'

[1] For compatibility with other definitions, we actually need to compute the BWT of s in reversed order, that is from right to left. This does not change our results and does not effect the compression ratio significantly (see [10] for a discussion on this), so we ignore this point from now on.

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 Figure 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 [19]).

```
mississippi$          $ mississipp i
ississippi$m          i $mississip p
ssissippi$mi          i ppi$missis s
sissippi$mis          i ssippi$mis s
issippi$miss          i ssissippi$ m
ssippi$missi    ==>   m ississippi $
sippi$missis          p i$mississi p
ippi$mississ          p pi$mississ i
ppi$mississi          s ippi$missi s
pi$mississip          s issippi$mi s
i$mississipp          s sippi$miss i
$mississippi          s sissippi$m i
```

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 BWT is effective for compression since in $\hat{s}$ characters with the same context[2] 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.

[2] The context of length k of a character is the string of length k preceding it.

Introductory Definitions. Let s be the string which we compress, and let Σ denote the alphabet (set of symbols in S). Let $n = |s|$, 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 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 following 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 k-th order empirical entropy of the string s. In [18] these terms, as well as BWT, are discussed in greater depth.

We also use the *zeta function*, $\zeta(\mu) = \frac{1}{1^\mu} + \frac{1}{2^\mu} + \ldots$, and the *truncated zeta function* $\zeta_h(\mu) = \frac{1}{1^\mu} + \ldots + \frac{1}{h^\mu}$. 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 [15,7]. Similarly, the k-th 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* k-th 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 [18] for more details on H_k^*.

In 1999, Manzini [18] 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 BW0(s) by the expression

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

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 $5 \cdot nH_k^*(s) + g''_k$ on the bit-length of BW0$_{RL}(s)$, where g_k'' is another constant that depends only on k and h. Here, the use of H_k^* allowed Manzini to achieve a result that lacks a term linear in n.

In 2004, Ferragina, Giancarlo, Manzini and Sciortino [10] 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 [10]). As mentioned above this result is optimal, up to the $C_{\text{ORDER0}} n + \log n + g_k'''$ term. The upper bounds of this algorithm and its variants based on the same techniques are theoretically strictly superior to those in [18] and to those that we present here. However, implementations of the algorithm of [10] by the authors and another implementation by Manzini [17], give the results summarized in Table 1. These empirical results surprisingly imply that while the algorithm of [10] 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.

We define a new statistic of a text s, which we call the *local entropy* of s, and denote it by $\text{LE}(s)$. We also 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. These statistics are theoretically oriented and we find their importance to be two-fold. First they may highlight potential weaknesses of existing compression algorithms and thereby mark the way to invent better compression algorithms. Second, they may be useful in understanding current algorithms and providing better worse-case upper bounds for them.

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

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

Here $g_k = 2k \log h + h^k \cdot h \log h$. 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 $4.45 \cdot nH_k(s) + (0.08 + C_{\text{ORDER0}}) \cdot n + \log n + 4.45g_k$, thus surpassing Manzini's upper bound (1). Our proof is considerably simpler than Manzini's proof of (1).

File Name	size	gzip	bzip2	BW0	[17]	[10](HC)	[10](RHC)
alice29.txt	152089	54181	43196	48915	47856	74576	79946
asyoulik.txt	125179	48819	39569	44961	42267	59924	61757
cp.html	24603	7965	7632	8726	8586	16342	16342
fields.c	11150	3122	3039	3435	3658	10235	10028
grammar.lsp	3721	1232	1283	1409	1369	2297	2297
lcet10.txt	426754	144562	107648	127745	116861	166043	177682
plrabn12.txt	481861	194551	145545	168311	154950	172471	183855
xargs.1	4227	1748	1762	1841	1864	2726	2726

Table 1
Results (in bytes) of running various compressors on the non-binary files from the Canterbury Corpus [1]. The gzip results are taken from [1]. The column marked [17] gives results from a preliminary implementation of the booster-based compression algorithms of [10] by Manzini [17]. BW0 is our implementation (in C++) of the BW0 algorithm, using Huffman encoding as the order-0 compressor. [10](HC) and [10](RHC) are our implementations of the compression booster of [10]. Ferragina et al. [10] suggest two methods to implement it: One using the algorithm HC, and one using the algorithm RHC (the reader is referred to [10] for more details).

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 which we call the "Local Entropy" (LE). This statistic was implicitly considered by Bentley et al. [4], and by Manzini [18]. Using two observations on the behavior of LE we bypass some of the technical hurdles in the analysis of [18].

Our analysis actually proves a considerably stronger result: We show that the size of the compressed text is bounded by

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

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

In order to get our upper bounds we prove in Section 3 a result on compression of integer sequences, which may be of independent interest.

Here is an overview of the rest of the paper.

(1) We prove a result on compressibility of integer sequences in Section 3.
(2) We define the statistic $\widehat{\mathrm{LE}}$ 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 in the end of Section 4.

File Name	size	$H_0(\dot{s})$	$LE(\hat{s})$	(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

Table 2
Results (in bits) of computing various statistics on the non-binary files from the Canterbury Corpus [1]. $H_0(\dot{s})$ gives the result of the algorithm BW0 assuming an optimal order-0 compressor. The final three columns show the bounds given by the Equations (4), (3), (1). The small difference between the column showing $H_0(\dot{s})$ 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 the figures are measured in bits is because the theoretical bounds in the literature are customarily measured in bits.

(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.

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 Indexes was made by Ferragina and Manzini in [20]. A recent result by Grossi, Gupta and Vitter [13] 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 Raman, Raman, and Rao [22].

For more on Compressed Text Indexing, see [14,20,11].

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 [18], that $H_k(s)$ is equal to a linear combination of H_0 of parts of $\hat{s}$, the Burrows Wheeler transform of s.

Proposition 1 ([18]) *Let $\tilde{s}$ be the string obtained from $\hat{s}$ by deleting the occurrences in $\hat{s}$ of the first k characters of s. (Note that these characters can appear in arbitrary positions of $\hat{s}$). There is a partition $\tilde{s} = \tilde{s}_1 \ldots \tilde{s}_t$, with $t \leq h^k$, such that:*

$$|s|\, H_k(s) = \sum_{i=1}^{t} |\tilde{s}_i|\, H_0(\tilde{s}_i)\ . \tag{5}$$

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 'a' is before 'b', $\text{LE}_\pi(s) = 2$ 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.[3] Analogously, MTF_W is MTF with an initial recency list that maximizes $\text{LE}_\pi(s)$. We will 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 implicity in [4,18].

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 $f : \Sigma^* \rightarrow \mathbb{R}^+$ 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 $\lim_{n\rightarrow\infty} \frac{g(n)}{n} = 0$.*

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 [24,21]. 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 interesting theoretically, but is not time-efficient in practice.

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

[3] 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 will be the same.

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.

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*: $\textsc{sl}(s) = \sum_{i=1}^{n} \log(s[i] + 1)$. Note that $\textsc{le}(s) = \textsc{sl}(\textsc{mtf}(s))$. We show that in a strong sense the best $\textsc{sl}$-competitive compression algorithm is an order-0 compressor. In the end of this section we show how to get from this good $\textsc{le}$-competitive and $\widehat{\textsc{le}}$-competitive compression algorithms.

The problem we deal with in this section is related to the problem of universal encoding of integers. In the problem of universal encoding of integers [9,4] the goal is to find a prefix-free encoding for integers, $U : \mathbb{Z}^+ \to \{0,1\}^*$, such that for every $x \geq 0$, $|U(x)| \leq \mu \log(x+1) + C$. A particularly nice solution for this is the Fibonacci encoding [3,12], for which $\mu = \log_\phi 2$ and $C = 1 + \log_\phi \sqrt{5} \simeq 2.6723$. An additional solution for this problem was proposed by Elias [9]. This is an optimal solution, in the sense described in [16]. For more information on universal encoding of integers see the (somewhat outdated) survey paper [16].

Clearly a universal encoding scheme with parameters μ and C gives an (μ, C)-$\textsc{sl}$-competitive compressor. However, in this section we get a better competitive ratio, taking advantage of the fact that out 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, using a technique based on Lemma 3, that the algorithm $\textsc{order0}$ is $(\mu, \log\zeta(\mu) + C_{\textsc{order0}})$-$\textsc{sl}$-competitive *for any* $\mu > 1$. In fact, we prove a somewhat stronger theorem:

Theorem 4 *For any constant $\mu > 0$, the algorithm* $\textsc{order0}$ *is $(\mu, \log\zeta_h(\mu) + C_{\textsc{order0}})$-$\textsc{sl}$-competitive.*

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

$$nH_0(s) \leq \mu\textsc{sl}(s) + n \log \zeta_h(\mu) \; . \tag{7}$$

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 $\textsc{sl}$ we get that $\textsc{sl}(s) = \sum_{j=1}^{n} \log(s[j]+1) = \sum_{i=0}^{h-1} n_i \log(i+1)$. 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

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

□

In particular we get the following corollary.

Corollary 5 *For any constant $\mu > 1$, the algorithm* $\textsc{order0}$ *is* $(\mu, \log \zeta(\mu) + C_{\textsc{order0}})$-$\textsc{sl}$-*competitive.*

A lower bound for **SL**-***Competitive compression.*** In Theorem 4 shows that for any $\mu > 0$ there exists a $(\mu, \log \zeta_h(\mu) + C_{\textsc{order0}})$-$\textsc{sl}$-competitive algorithm. We now show that for any fixed values of μ and h there is no algorithm with better competitive ratio. Note that the lower bounds that we get in this section do not include the constant $C_{\textsc{order0}}$.

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

Proof. We show that for any algorithm $\textsc{a}$, $\mu > 0$, $\epsilon > 0$, and any function f such that $\lim_{n \to \infty} f(n) = 0$, there exists a string s such that

$$|\textsc{a}(s)| > \mu \textsc{sl}(s) + |s| \left(\log \zeta_h(\mu) - \epsilon + f(|s|) \right) \ . \tag{10}$$

We achieve that 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 will later show 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 $s \in S(n)$, $|s| = n$, $\mathrm{SL}(s) = L(n)$, 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 n large enough,

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

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

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

and therefore

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

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

Now we address the fact that for every $i \in [h]$, α_i is not necessarily an integer. Define for $i \in \{1, \ldots h-1\}$, $\alpha'_i = \left\lfloor n \cdot \frac{1}{\zeta_h(\mu)\cdot(i+1)^\mu} \right\rfloor$, and push the excess into α'_0, i.e. define $\alpha'_0 = n - \sum_{i=0}^{h-1} \alpha'_i$. Inequality (12) still holds because the rounding makes α_i and α'_i differ by at most $\pm h$, which contributes only an additional $-O(\log n)$ factor to (12). Inequality (13) continues to hold because by rounding in this specific way we have actually decreased the sum of logarithms, so $\text{SL}(s) \leq L(n)$. □

By setting a large enough alphabet in the proof of Thm. 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 Thm. 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 Thm. 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*

and of course,

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 Thm. 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 Thm. 6 if the transformations MTF_π and BWT, viewed as functions from Σ^n to Σ^n, were invertible. (Recall that the function BWT(s) is the outcome of running the Burrows-Wheeler transform on $s\$$ and then deleting the symbol '\$' from the result). But, while $\textsc{mtf}_\pi$ is invertible, $\textsc{bwt}$ is not.[4] This means that potentially, the image of the transformation $\textsc{bwt}$ could be a small fraction of Σ^n that has better compressibility properties with respect to $\textsc{le}$.

4 The Entropy Hierarchy

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

$$\widehat{\textsc{le}}(s) \leq nH_k(s) + O(1) \tag{14}$$

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

$$\ldots \leq nH_k(s) \leq \ldots \leq nH_2(s) \leq nH_1(s) \leq nH_0(s) \ . \tag{15}$$

Which in addition to (14) gives us:

$$\widehat{\textsc{le}}(s) \ldots \lessapprox \ldots \leq nH_k(s) \leq \ldots \leq nH_2(s) \leq nH_1(s) \leq nH_0(s) \ . \tag{16}$$

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

Thus any (μ, C)-$\widehat{\textsc{le}}$-competitive algorithm is also (μ, C)-nH_k-competitive. To establish this hierarchy we need to prove two properties of $\textsc{le}_W$: that it is at most $nH_0 + o(n)$, 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 [18,4]. Specifying them here in this form would help to understand the rest of the analysis. We give references where appropriate.

Define $\textsc{mtf}_{ignorefirst}(s)$ to be a string which is identical to $\textsc{mtf}_\pi(s)$ except that we omit the integers representing the first occurrence of each symbol (so $\textsc{mtf}_{ignorefirst}(s)$ is of length less than n). Note that in this case when we perform the move-to-front transformation the initial status of the $\textsc{mtf}$ recency list

[4] Take for example the string s'="bac" over the alphabet $\Sigma = \{a, b, c\}$. String s' is not equal to $\textsc{bwt}(s)$ 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. It can be easily seen 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 $\hat{s}$ is a permutation of s.

is not significant. Similarly, define $\text{LE}_{ignorefirst}(s) = \sum_i \log(\text{MTF}_{ignorefirst}(s)[i]+1)$.

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

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

Proof. We look separately at the contributions of each of the h different symbols to. The contribution of σ to $\text{LE}_{ignorefirst}(s)$ is [5]

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

It is easy to see that

$$\sum_{i:s[i]=\sigma} (\text{MTF}_{ignorefirst}(s)[i]+1) \leq n \ .$$

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

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

Summing for all σ we obtain that

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

as needed. $\Box$

Manzini [18] gave the following corollary of this Theorem.

Lemma 13 ([18], Lemma 5.4) $\text{LE}_W(s) \leq nH_0(s) + h \log h$.

Proof. $\text{LE}_W(s)$ is equal to $\text{LE}_{ignorefirst}(s)$ 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 $\text{LE}_W(s) \leq \text{LE}_{ignorefirst}(s) + h\log h \leq nH_0(s) + h \log h$. $\Box$

In addition, we need the following lemma about LE_W.

Lemma 14 *For a string s of length n and a string s' obtained by deleting exactly one character from s, we have that $\text{LE}_W(s) \leq \text{LE}_W(s') + 2\log h$.*

[5] Note that for the sake of convenience, in the following equations we are disregarding the fact that some elements of $\text{MTF}_{ignorefirst}(s)$ 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.

Note 1 *For our purposes in this paper it suffices to prove the bound* $\text{LE}_W(s) \leq \text{LE}_W(s') + h \log h$*, which is easier. We give a proof of the stronger bound because we find the proof interesting by itself.*

Proof. Let x be the character we remove from s to get s'. Let π be a worst-case permutation for s, i.e. $\text{LE}_\pi(s) = \text{LE}_W(s)$. It is enough to show that $\text{LE}_\pi(s) \leq \text{LE}_\pi(s') + 2 \log h$ because from this it follows that

$$\text{LE}_W(s) = \text{LE}_\pi(s) \leq \text{LE}_\pi(s') + 2 \log h \leq \text{LE}_W(s') + 2 \log h \ ,$$

as needed.

Step 1: Observe that $\text{MTF}_\pi(s)$ has one additional element compared to $\text{MTF}_\pi(s')$. This is the element that corresponds to x. One element can contribute at most $\log h$ to $\text{LE}_\pi(s) - \text{LE}_\pi(s')$. In step 2, we bound the difference in the contribution of all the other elements to $\text{LE}_\pi(s)$ and $\text{LE}_\pi(s')$.

Step 2: Observe that the only elements that might differ in $\text{MTF}_\pi(s)$ and $\text{MTF}_\pi(s')$ are the elements that correspond to the first occurrence after x of every symbol. The element that corresponds to the next occurrence of x in s may change. However, it can only decrease, so we disregard it.

Denote $s = \ldots x \ldots a_1 \ldots a_2 \ldots a_i \ldots a_m \ldots$ where $a_1 \ldots a_m$ are the first occurrences after x of all symbols except for the symbol x itself. Let y_i be the value in $\text{MTF}_\pi(s)$ that corresponds to a_i, and y'_i be the value in $\text{MTF}_\pi(s')$ that corresponds to a_i. Observe that

(1) $m \leq h - 1$.
(2) In s, all characters $x, a_1, \ldots, a_{i-1}$ appear between a_i and its previous occurrence.
(3) All symbols that occur between a_i and its previous occurrence in s' also occur between a_i and its previous occurrence in s.
(4) All symbols except x that occur between a_i and its previous occurrence in s also occur between a_i and its previous occurrence in s'.
(5) From item 3 and item 4 it follows that $y'_i \leq y_i \leq y'_i + 1$.
(6) From item 2 it follows that $y_i \geq i$.

From these observations it follows that the part of the difference $\text{LE}_\pi(s) - \text{LE}_\pi(s')$ that we have not accounted for in step 1 is upper-bounded by

$$\sum_{i=1}^{m} (\log(y_i + 1) - \log(y'_i + 1)) \leq \sum_{i=1}^{m} \log \frac{i+1}{i} = \log(m+1) \leq \log h \ .$$

Combined with the $\log h$ upper bound from step 1, the statement follows. □

We would now like to prove that LE_W is a convex statistic. The intuition behind this is that the MTF_π encoding 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 15 (LE_W is a convex statistic, implicitly stated in [18]) *Let $s = s_1 \ldots s_t$. Then $\textsc{le}_W(s) \leq \sum_i \textsc{le}_W(s_i)$*

Proof. From the definition of $\textsc{le}_W$ we get that $\textsc{le}_W(s) = \sum_{j=1}^{n} \log(\textsc{mtf}_{\pi_1}(s)[j]+1)$ for a worst-case permutation π_1. Let us look at the recency list π_i that we use when the $\textsc{le}_W(s)$ calculation reaches sub-string s_i. Each of the summands of $\sum_i \textsc{le}_W(s_i)$ 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 16 *For any $k \geq 0$ and any string s,*

$$|s|\, H_k(s) \geq \widehat{\textsc{le}_W}(s) - 2k \log h - h^k \cdot h \log h$$

Proof. Let $\tilde{s}$ be the string obtained from $\hat{s}$ by deleting the occurrences in $\hat{s}$ of the first k characters of s. By Proposition 1 there is a partition of $\tilde{s}$, $\tilde{s} = \tilde{s}_1 \ldots \tilde{s}_t$, such that $t \leq h^k$ and

$$|s|\, H_k(s) = \sum_{i=1}^{t} |\tilde{s}_i|\, H_0(\tilde{s}_i) \ . \tag{17}$$

Observe that using the convexity of $\textsc{le}_W$ (Lemma 15) and using the relation of $\textsc{le}_W$ to nH_0 (Lemma 13) we have

$$\textsc{le}_W(\tilde{s}) \leq \sum_{i=1}^{t} |\tilde{s}_i|\, H_0(\tilde{s}_i) + th \log h \ . \tag{18}$$

Using Lemma 14 we get

$$\textsc{le}_W(\hat{s}) - 2k \log h \leq \textsc{le}_W(\tilde{s}) \ . \tag{19}$$

From (17), (18) and (19) we get

$$\textsc{le}_W(\hat{s}) - 2k \log h - th \log h \leq |s|\, H_k(s) \ ,$$

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

Main Results. Using Theorem 4 together with Theorem 16 gives the main result of our paper:

Theorem 17 *For any $k \geq 0$ and for any constant $\mu > 1$, the algorithm* $\textsc{bw}0$ *is $(\mu, \log \zeta(\mu) + C_{\textsc{order}0})$-$nH_k$-competitive*

Proof. Corollary 10 gives that for any string s, $|\text{BW0}(s)| \le \mu\widehat{\text{LE}}(s)+(\log\zeta(\mu)+ C_{\text{ORDER0}} + o(1))\,|s|$. Using this together with Theorem 16 gives that for any string s, $|\text{BW0}(s)| \le \mu H_k(s) + (\log\zeta(\mu) + C_{\text{ORDER0}} + o(1))\,|s|$, which gives the theorem. □

Similarly, using Corollary 9 gives

Theorem 18 *For any $k \ge 0$ and for any constant $\mu > 0$, the algorithm* BW0 *is $(\mu, \log\zeta_h(\mu) + C_{\text{ORDER0}})$-$nH_k$-competitive on strings from an alphabet of size h.*

5 An Upper Bound and a Conjecture about BW0

Let us prove an upper-bound on the performance of BW0 in a specific setting. This bound is tighter than the upper bound of Theorem 17.

Theorem 19 BW0 *is $(2, C_{\text{ORDER0}})$-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 $p_a \le \frac{1}{2}$. We consider the following cases. In each case we prove that

$$H_0(\text{MTF}(\hat{s})) \le 2H_0(s)\ . \tag{20}$$

Case 1: $p_a = 0$. Here (20) is trivial.
Case 2: $0 < p_a \le \frac{1}{4}$. 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 $\text{MTF}(\hat{s})$. Therefore the number of '1's in $\text{MTF}(\hat{s})$ is at most $2n_a \le \frac{n}{2}$. From the monotonicity of the entropy function[6] it follows that

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

while on the other hand,

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

and therefore,

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

[6] 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.

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 $H_0(\textsc{mtf}(\hat{s}) \le 1 \le 2H_0(s)$. Therefore (20) also holds in this case.

In either case we get the following:

$$|\textsc{bw0}(s)| \le nH_0(\textsc{mtf}(\hat{s})) + C_{\textsc{order0}}n \le 2nH_0(s) + C_{\textsc{order0}}n \ ,$$

so the algorithm BW0 is $(2, C_{\textsc{order0}})$-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 20 BW0 *is* $(2, C_{\textsc{order0}})$-nH_k*-competitive.*

6 A $(1.7286, C_{\mathbf{order0}})$-$nH_k$-competitive Algorithm

In this section we analyze the BWT *with distance coding* compression algorithm, $\textsc{bw}_{\textsc{dc}}$. 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 $\textsc{bw}_{\textsc{dc}}$ 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 $\textsc{bw}_{\textsc{dc}}$ is $(1.7286, C_{\textsc{order0}})$-$nH_k$-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. [7] It follows that DC produces

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

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 $\text{DC}(s)$ and the auxiliary information we can recover s. A formal proof of this fact can be found in Appendix A.

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,

$$\text{LD}(s) = \sum_i \log(\text{DIST}(s)[i] + 1) \ .$$

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

$$\text{SL}(\text{DC}(s)) = \text{LD}(s) \ . \tag{21}$$

Now we wish to prove that BW_{DC} is $(1.7286, C_{\text{ORDER0}})$-$nH_k$-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.

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 (21) 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 $\text{LD}(s) \le nH_0(s)$. Furthermore, as in Lemma 14, if s is of length n and s' is produced by deleting exactly one character from s then $\text{LD}(s) \le \text{LD}(s') + 2\log n$. Now we prove along the lines of Lemma 15 that if $s = s_1 \ldots s_t$ then $\text{LD}(s) \le \sum_i \text{LD}(s_i) + (t-1)h\log n$. 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* $k \ge 0$.

From Thm. 22 together with Lemma 21 we get:

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

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 another approach must be taken 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 $\Sigma_2 = \{1, 2, \ldots, h\}$ and is $(\mu, 0)$-SL-competitive. We then argue that there cannot be a $(\mu, 0)$-SL-competitive algorithm, 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,

$$|\text{B}(s_2)| = |\text{A}(s_1)| \leq \mu \text{LD}(s_1) + o(n_1) \leq \mu \text{SL}(s_2) + o(n_2) \ , \tag{22}$$

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 (22) follows that B is $(\mu, 0)$-SL-competitive. We now argue that a $(\mu, 0)$-SL-competitive algorithm does not exist.

One can show in a similar fashion to Thm. 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 $\{1, 2, \ldots, h\}$. The term -1 appears here as the summation that used to give the term $\zeta(\mu)$ now starts at $i = 1$ instead of $i = 0$. 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 Thm. 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 $(\mu, \log \zeta(\mu))$-$\widehat{\mathrm{LE}}$-competitive. On the other hand, Ferragina et al. [10] 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.

9 Acknowledgments

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

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 and A. S. Fraenkel. Robust transmission of unbounded strings using fibonacci representations. *IEEE Transactions on Information Theory*, 33(2):238–245, 1987.

[4] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.

[5] E. Binder. Distance coder. Usenet group comp.compression and private communications, 2000.

[6] M. Burrows and 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, and C. Stein. *Introduction to Algorithms, Second Edition*, chapter 16.3, pages 385–392. MIT Press and McGraw-Hill, 2001.

[8] S. Deorowicz. Second step algorithms in the burrowswheeler compression algorithm. *Software - Practice and Experience*, 32(2):99–111, 2002.

[9] P. Elias. Universal codeword sets and representation of the integers. *IEEE Trans. on Information Theory*, 21(2):194–203, 1975.

[10] P. Ferragina, R. Giancarlo, G. Manzini, and M. Sciortino. Boosting textual compression in optimal linear time. *Journal of the ACM*, 52:688–713, 2005.

[11] P. Ferragina, G. Manzini, V. Mākinen, and G. Navarro. An alphabet friendly FM-index. In *Proc. 11th Symposium on String Processing and Information Retrieval (SPIRE '04)*, pages 150–160, 2004.

[12] A. S. Fraenkel and S. T. Klein. Robust universal complete codes for transmission and compression. *Discrete Applied Mathematics*, 64(1):31–55, 1996.

[13] R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 841–850, 2003.

[14] R. Grossi, A. Gupta, and J. S. Vitter. When indexing equals compression: experiments with compressing suffix arrays and applications. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 636–645, 2004. Journal version to appear in ACM Transactions on Algorithms (special issue of ACM-SIAM SODA), 2005.

[15] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[16] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Computing Surveys*, 19(3):261–296, 1987.

[17] G. Manzini. Personal communication.

[18] G. Manzini. An analysis of the burrows-wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.

[19] G. Manzini and P. Ferragina. Engineering a lightweight suffix array construction algorithm. *Algorithmica*, 40:33–50, 2004.

[20] G. Manzini and P. Ferragina. Indexing compressed text. *Journal of the ACM*, 52:552–581, 2005.

[21] A. Moffat, R. M. Neal, and I. H. Witten. Arithmetic coding revisited. *ACM Trans. Inf. Syst.*, 16(3):256–294, 1998.

[22] R. Raman, V. Raman, and S. S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 233–242, 2002.

[23] J. Seward. bzip2, a program and library for data compression. http://www.bzip.org/.

[24] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

A 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, of course, 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 of each of the symbols. In the beginning of the process we obviously have this information, because this is part of the auxiliary information that we saved. Now, suppose that $s[i] = a$ is the first character we have not read yet, and $s[j] = b \neq a$ is the second character we have not read yet (it is possible that $j - i > 1$). Then, obviously, for every $i \leq k < j$, $s[k] = a$. Therefore, the first element of s' that we have not read yet corresponds to the distance between $j - 1$ and the first appearance of a in s after location j. We can continue decoding like this until we get to the end of the string. Special care must be taken with the final appearance of each symbol, because it is not coded in s'.

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