A Simpler Analysis of Burrows-Wheeler Based Compression

Haim Kaplan, Shir Landau, and Elad Verbin

School of Computer Science, Tel Aviv University, Tel Aviv, Israel
{haimk, landaush, 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 steps: 1) Compute 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 prove 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 $\zeta(\mu) = \frac{1}{1^\mu} + \frac{1}{2^\mu} + \ldots$ 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.

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 [15], we refer to these algorithms as BW0 and BW0$_{RL}$. A well known implementation of these algorithms is bzip2 [18]. 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 $\dot{s} = \textsc{mtf}(\hat{s})$ 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 $\textsc{bw0}(s) = \textsc{order0}(\dot{s})$. 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 $\textsc{bw0}_{RL}$ performs an additional run-length encoding (RLE) procedure between steps 2 and 3. See [6, 15] for more details on BW0 and $\textsc{bw0}_{RL}$, 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 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 [16]).

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 $\dot{s} = \textsc{mtf}(\hat{s})$ 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.

[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.

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

```
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. $\hat{s}$ 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 get the original string.

Introductory Definitions. minus .1em 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. Let $\Sigma^* = \bigcup_{k\geq 0} \Sigma^k$ denote the set of all (finite) strings 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. In this context, we define $0 \log(n/0) = 0$, for any n). 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 [15] 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 [7, 12]. Similarly, the k-th order empirical entropy of a string s, denoted by $H_k(s)$, gives a lower bound on the compression ratio of any encoder, that to encode a character x, is allowed to use only the k characters preceding x (the context of length k of x). 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 [15] for more details on H_k^*.

In 1999, Manzini [15] 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 $\text{BW0}_{RL}(s)$, where g''_k is another constant that depends only on k and h.

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 [15] and to those that we present here. However, implementations of the algorithm of [10] by the authors and another implementation by Manzini [14], 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 LE(s). This statistic was implicitly considered by Bentley et al. [4], and by Manzini [15]. We also define $\widehat{\text{LE}}(s) = \text{LE}(\hat{s})$. 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 proving 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

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 BW0 shows the results of our implementation (in C++) of the BW0 algorithm, using Huffman encoding as the order-0 compressor. The column marked [14] gives results from a preliminary implementation of the booster-based compression algorithm supplied to us by Manzini. The columns Booster(HC) and Booster(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 interested reader is referred to [10]).

File Name	size	gzip	bzip2	BW0	[14]	Booster(HC)	Booster(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

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

We prove this bound using two basic observations on the statistic LE that we define. Thereby we bypass some of the technical hurdles in the analysis of [15]. Our analysis actually proves a considerably stronger result. We show that the size of the compressed text is bounded by

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

Empirically, this seems to give estimates 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 $\widehat{\text{LE}}$ 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) in Section 4.
4. We give a tighter upper bound for BW0 for the case that we are working over an alphabet of size 2 in Section 4.1.
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 5.

Table 2. Results of computing various statistics on the non-binary files from the Canterbury Corpus [1]. Numbers are in "bits" since the theoretical bounds in the literature are customarily calculated in bits. The column denoted by $H_0(\dot{s})$ 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), and (1). The difference between the column of $H_0(\dot{s})$ 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 Equation (4) over μ. To get the bound of (3) and (1) we needed to calculate the values of the corresponding equations for all k and pick the best one.

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

Due to space limitation, many proofs are omitted. They can found in the full version of the paper.

2 Preliminaries

Our analysis does not use the definitions of H_k and BWT directly. Instead, it uses the following observation of Manzini [15], 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 ([15]). *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 $t \le h^k$, 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).[3]

[3] In order to completely determine the encoding we must specify the status of the recency list at the beginning of the procedure. Here and in the future we usually ignore this fact.

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 to be $\text{LE}(s) = \sum_{i=1}^{n} \log(\text{MTF}(s)[i] + 1)$. That is, LE is the sum of the logarithms of the move-to-front values plus 1. For example, for a string "aabb" and initial list where 'b' is first and 'a' is second, $\text{LE}(s) = 2$ 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 define $\widehat{\text{LE}}(s) = \text{LE}(\hat{s})$.

Note that $\text{LE}(s)$ is the number of bits one needs to write the sequence of integers $\text{MTF}(s)$ in binary. Optimistically, this is the size we would like to compress the text to. Of course, one cannot decode the integers in $\text{MTF}(s)$ 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 called "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 [17, 19]. Furthermore, one can implement arithmetic code so that it is $(1, 0)$-nH_0-competitive, that is the bit-length of the compressed text is bounded by $nH_0(s) + O(\log n)$. 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:

Lemma 3. *For any $k \geq 1$, $x_1, \ldots, x_k > 0$ and $y_1, \ldots, y_k > 0$ it holds that* $\sum_{i=1}^{k} y_i \log x_i \leq \left(\sum_{i=1}^{k} y_i\right) \cdot \log\left(\frac{\sum_{i=1}^{k} x_i y_i}{\sum_{i=1}^{k} y_i}\right)$.

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 of 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 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, 11], for which $\mu = \log_\phi 2 \simeq 1.4404$ and $C = 1 + \log_\phi \sqrt{5} \simeq 2.6723$ (here ϕ is the golden ratio, $\phi = (1+\sqrt{5})/2$). An additional solution for this problem was proposed by Elias [9]. This is an optimal solution, in the sense described in [13]. For more information on universal encoding of integers see the (somewhat outdated) survey paper [13].

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 our goal is to encode a long sequence over $[h]$, 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 $\textsc{order}0$ is $(\mu, \log\zeta(\mu) + C_{\textsc{order}0})$-$\textsc{sl}$-competitive *for any* $\mu > 1$. In fact, we prove a somewhat stronger theorem:

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

Proof. Let s be a string of length n over alphabet $[h]$. Clearly it suffices to prove that for any constant $\mu > 0$, $nH_0(s) \leq \mu\textsc{sl}(s) + n\log\zeta_h(\mu)$.

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 $\textsc{sl}(s) = \sum_{j=1}^{n} \log(s[j]+1) = \sum_{i=0}^{h-1} n_i \log(i+1)$. Therefore, it suffices to prove that $\sum_{i=0}^{h-1} n_i \log\frac{n}{n_i} \leq \mu \sum_{i=0}^{h-1} n_i \log(i+1) + n\log\zeta_h(\mu)$.

Pushing the μ into the logarithm and moving terms around we get that it suffices to prove that $\sum_{i=0}^{h-1} n_i \log\frac{n}{n_i(i+1)^\mu} \leq n\log\zeta_h(\mu)$.

Defining $p_i = \frac{n_i}{n}$ and dividing the two sides of the inequality by n we get that it suffices to prove that $\sum_{i=0}^{h-1} p_i \log\frac{1}{p_i(i+1)^\mu} \leq \log\zeta_h(\mu)$.

Using Lemma 3 we obtain that $\sum_{i=0}^{h-1} p_i \log\frac{1}{p_i(i+1)^\mu} = \sum_{\substack{0\leq i\leq h-1\\ p_i\neq 0}} p_i \log\frac{1}{p_i(i+1)^\mu}$

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

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.*

One also gets the following analogous result with respect to $\widehat{\text{LE}}$:

Corollary 6. *For any constant $\mu > 0$, the algorithm* BW0 *is* $(\mu, \log\zeta_h(\mu) + C_{\text{ORDER0}})$-$\widehat{\text{LE}}$-*competitive*

A Lower Bound for SL-Competitive Compression. Theorem 4 shows that for any $\mu > 0$ there exists an $(\mu, \log\zeta_h(\mu) + C_{\text{ORDER0}})$-SL-competitive algorithm. We now show that for any fixed values of μ and h there is no algorithm with a better competitive ratio. Note that the lower bound that we get does not include the constant C_{ORDER0}.

Theorem 7. *Let $\mu > 0$ be some constant. For any $C < \log\zeta_h(\mu)$ there is no (μ, C)-*SL-*competitive algorithm.*

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,

$$\widehat{\text{LE}}(s) \leq nH_k(s) + O(1) \tag{6}$$

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

$$\ldots \leq nH_k(s) \leq \ldots \leq nH_2(s) \leq nH_1(s) \leq nH_0(s) \ . \tag{7}$$

Which in addition to (6) gives us:

$$\widehat{\text{LE}}(s) \ldots \lessapprox \ldots \leq nH_k(s) \leq \ldots \leq nH_2(s) \leq nH_1(s) \leq nH_0(s) \ . \tag{8}$$

($O(1)$ 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: that it is at most $nH_0 + o(n)$, and that it is convex (in a sense which we will define).

Some of the following claims appear, explicitly or implicitly, in [4, 15]. We specify them in a form that would help to understand the rest of the analysis.

Manzini [15] gave the following corollary of a theorem of Bentley et al. [4].

Lemma 8 ([15], Lemma 5.4). LE$(s) \leq nH_0(s) + h\log h$.

In addition, we need the following lemma about LE.

Lemma 9. *Let s be a string of length n and let s' be a string obtained by deleting exactly one character from s. Then* LE$(s) \leq$ LE$(s') + 2\log h$.

We now prove that LE is a convex statistic. That is, one cannot gain much by stopping MTF in the middle and restarting it with a different recency list.

Lemma 10 (LE is a convex statistic, implicitly stated in [15]). *Let* $s = s_1 \dots s_t$. *Then* $\textsc{le}(s) \le \sum_i \textsc{le}(s_i)$

(For the last lemma, and for the lemmas before it, we define the initial state of the recency list of LE as the worst possible state. If we define it differently, we might incur an additive term of $th \log h$ in the last lemma. This would not significantly change our bounds).

From these Lemmas one can derive the hierarchy result:

Theorem 11. *For any* $k \ge 0$ *and any string* s, $|s|\, H_k(s) \ge \widehat{\textsc{le}}(s) - 2k \log h - h^k \cdot h \log h$.

Using Corollary 6 together with Theorem 11 allows us to derive the main result of this paper:

Theorem 12. *For any* $k \ge 0$ *and for any constant* $\mu > 0$, *the algorithm* BW0 *is* $(\mu, \log \zeta_h(\mu) + C_{\textsc{order0}})$-$nH_k$*-competitive (on strings from an alphabet of size* h*).*

4.1 An Upper Bound and a Conjecture About BW0

In the case where the alphabet size is 2 we were able to prove an improved upper bound for BW0:

Theorem 13. BW0 *is* $(2, C_{\textsc{order0}})$-nH_0*-competitive for texts over an alphabet of size 2.*

We believe that this upper bound is true in the general setting. Specifically, we leave the following conjecture as an open problem.

Conjecture 1. BW0 *is* $(2, C_{\textsc{order0}})$-nH_k*-competitive.*

5 A $(1.7286, C_{\text{order0}})$-$nH_k$-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 a transformation called DIST: 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, but 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.[4] 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 $\text{DC}(s)$ and the auxiliary information we can recover s.

As a tool for our analysis, let us 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 $\text{SL}(\text{DC}(s)) = \text{LD}(s)$.

We can now state the following theorem.

Theorem 14. *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.*

To prove Theorem 14 we follow the footsteps of the proof of Theorem 12 outlined in Sections 3 and 4, where we use DC instead of MTF, and LD instead of LE. The term $\log(\zeta(\mu) - 1)$ appears instead of $\log \zeta(\mu)$ because the summations in the proof of Theorem 4 now start at $i = 1$ instead of at $i = 0$ (this is because we omitted all zeroes, so all characters of the alphabet are in this case at least 1).

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

Corollary 15. *For any $k \geq 0$, the algorithm BW_{DC} is $(\mu_0, C_{\text{ORDER0}})$-$nH_k$-competitive.*

In the full version of this paper we also show that this approach cannot yield better results. Namely, we prove that for any $\mu < \mu_0$, there is no $(\mu, 0)$-LD-competitive algorithm, so we have a matching lower bound for Corollary 15. We show that this lower bound holds even if the alphabet size is 2. We leave as an open problem to determine whether there is a matching lower bound for Theorem 14, or it can be improved.

6 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. [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

[4] 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.

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 for helpful discussions and useful references. We thank Giovanni Manzini for sharing with us a preliminary implementation of the compression booster. The third author would like to thank Roberto Grossi and Adi Avidor for some insightful discussions.

References

[1] The canterbury corpus. http://corpus.canterbury.ac.nz.

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

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

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

[14] G. Manzini. Personal communication.

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

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

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

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

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