Most Burrows-Wheeler Based Compressors are Not Optimal

Haim Kaplan and Elad Verbin

School of Computer Science, Tel Aviv University, Tel Aviv, Israel,
{haimk,eladv}@post.tau.ac.il

Abstract. We present a technique for proving lower bounds on the compression ratio of algorithms which are based on the *Burrows-Wheeler Transform* (BWT). We study three well known BWT-based compressors: the original algorithm suggested by Burrows and Wheeler; BWT with distance coding; and BWT with run-length encoding. For each compressor, we show a Markov source such that for asymptotically-large text generated by the source, the compression ratio divided by the entropy of the source is a constant greater than 1. This constant is $2 - \epsilon$, 1.26, and 1.29, for each of the three compressors respectively. Our technique is robust, and can be used to prove similar claims for most BWT-based compressors (with a few notable exceptions). This stands in contrast to statistical compressors and Lempel-Ziv-style dictionary compressors, which are long known to be optimal, in the sense that for any Markov source, the compression ratio divided by the entropy of the source asymptotically tends to 1.

We experimentally corroborate our theoretical bounds. Furthermore, we compare BWT-based compressors to other compressors and show that for "realistic" Markov sources they indeed perform bad and often worse than other compressors. This is in contrast with the well known fact that on English text, BWT-based compressors are superior to many other types of compressors.

1 Introduction

In 1994, Burrows and Wheeler [5] introduced the Burrows-Wheeler Transform (BWT) together with two new lossless text-compression algorithms that are based on the BWT. Since then many other BWT-based compressors have emerged. Today, BWT-based compressors are well established alongside with dictionary-based compressors (e.g. the Lempel-Ziv family of compressors, including e.g. gzip) and statistical coders (e.g. PPMd). A widely used implementation of a BWT compressor is bzip2 [19].

BWT-based compressors are typically more effective than dictionary-based compressors, but less effective than statistical coders. For example, bzip2 shrinks the King James version of the Bible to about 21% of its original size while gzip shrinks it to about 29% of the original size, and PPMd to 19% of the original size [1]. See [1, 13, 9] for more experimental results.

In tandem with developing efficient implementations of compression algorithms, analytical upper bounds on compression ratios have been proved. The most famous result of this type is the proof that Huffman codes are the most effective prefix-free codes and, in particular, that the average length of a codeword is close to the entropy of the source (see e.g. [12]).

A compression algorithm is called *optimal* if the algorithm, when applied to a string generated by a Markov source, outputs a number of bits per character which is equal to the entropy of the source, plus a term that tends to 0 as the length of the string tends to infinity. (For a more formal definition see Section 2). It is of particular interest to prove that a compression algorithm is optimal. A famous result of this type is that the Lempel-Ziv-Welch (LZW) compression algorithm [22] is optimal [6, 21]. It is traditionally taken for granted that good compression algorithms should be optimal. In this paper we prove that, surprisingly, most BWT-based compressors are not optimal.

Several papers prove analytical upper bounds on the compression ratio of BWT-based compressors. The bounds found in the papers of Manzini [15] and of Kaplan et al. [13], analyze BWT-based algorithms of the type that is in widespread use today: such algorithms run BWT, followed by a move-to-front encoding or some variant of it, and then apply some type of order-0 encoding (see Section 2 for more information). However, no known upper bound establishes optimality of an algorithm of this type. In Section 3 we show that the reason that optimality was not proved is that these algorithms are in fact not optimal: for each algorithm of this type, there exists a Markov source where the expected number of bits per character that the compressor at stake uses is c times the entropy of the source. Here, $c > 1$ is a constant that depends on which BWT-based algorithm we are considering. For example, for the most well-known algorithm BW_{RL}, which is BWT with run-length encoding, we have $c = 1.297$. For the algorithm $BW0$ first proposed by Burrows and Wheeler, we have $c = 2 - \epsilon$ for any $\epsilon > 0$. For BW_{DC}, which is BWT with distance coding, we get $c = 1.265$. Our technique is quite robust, and can be used to prove non-optimality of most BWT-based compressors.

Two notable exceptions to the above rule are the compression booster of Ferragina et al. [10, 9], and the "block-based" BWT-based compressor of Effros et al. [8]. Both of these compressors have been proved to be optimal. However, none of these compressors is in widespread use today, and both algorithms require further study in order to determine if they are effective in practice, or if they can be made effective in practice. In this paper we do not give an in-depth discussion of these algorithms. For the compression booster, a separate study is in progress to find out whether it can indeed achieve much better compression than other BWT-based compressors, see e.g. [10, 9, 13]. Regarding the block-based compressor of Effros et al., this compressor is proved to be optimal, but only when the block-size is asymptotically large. When the block-size is asymptotically large, the compressor behaves in a way that is not typical of a "classical" BWT-based compressor, and it may behave badly in practice. For constant block-size that is independent of the length of the text, the compressor of Effros et al. is similar to more traditional BWT-based compressors, and Effros et al. state that it is not optimal.

In Section 4 we present experimental evidence that solidify the non-optimality of BWT-based compressors. We begin by applying the compressors to the theoretically bad Markov models, and comparing the compression ratio to the entropy of the source. This confirms our analytical results from Section 3. We briefly outline experiments that were omitted from this version due to lack of space, which show that on texts generated from Markov sources with realistic parameters, the compression ratio divided by the entropy of the source is significantly greater than 1. (We created such "realistic" Markov sources by setting the parameters to values that mimic actual English text).

The results presented in this paper raise an interesting question, which we have not yet managed to fully resolve: if Lempel-Ziv compression is optimal, and BWT-based compression is not optimal, then why does gzip give worse compression rates than bzip2? (and in general, why do practical BWT-based compressors beat dictionary-based compressors). The textbook answer is that the proofs of optimality hold for asymptotically large file-sizes, while these compressors, in order to be practical, make some compromises, such as cutting the input into chunks and compressing each separately (in the case of bzip2), or keeping a limited-size dictionary (for gzip). These compromises imply that effectively, in practice, the compressors do not actually operate on large files, and thus the analysis that implies optimality stays largely theoretical. Granted, this is a good answer, but we would like to propose a different answer, which we think should be the subject of further research: We believe that theoretical results that are proved on Markov sources do not have straightforward theoretical implications to English text, since English text was *not* generated by a Markov source. English text has properties that Markov-generated text does not, such as an abundance of long exact repetitions and approximate repetitions. It is an interesting open question to give a theoretical analysis of compression algorithms that takes into account such differences.

2 Preliminaries

Due to lack of space, we omit the proof of all claims, except that of Thm. 4. All proofs can be found in the full version of this paper, which is available online.

Let s be the string which we compress, and let Σ denote the alphabet (set of symbols in s). Let $n = |s|$ be the length of s, and $h = |\Sigma|$. Let n_σ be the number of occurrences of the symbol σ in s. Let Σ^k denote the set of strings of length k over Σ. We denote by $[h]$ the integers $\{0, \ldots, h-1\}$.

Let $s[i]$ be the i^{th} character in s. The *context of length* k of $s[i]$ is the substring of length k that precedes $s[i]$, i.e. $s[i-k..i-1]$. For a compression algorithm A we denote by $|A(s)|$ the size *in bits* of the output of A on a string s. Note that we always measure the length of strings in characters, and the output of a compression algorithm in bits. The *zeroth order empirical entropy* of the string s is defined as $H_0(s) = \sum_{\sigma \in \Sigma} \frac{n_\sigma}{n} \log \frac{n}{n_\sigma}$. (All logarithms in the paper are to the base 2, and 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. See [15] for a more detailed definition of empirical entropy.

Markov Sources. Let D be a distribution over σ. The *order-zero Markov model* M defined by D is a stochastic process that generates a string by repeatedly picking characters independently according to D.

Let $\{D_w\}$ be a family of distributions over σ, one for each string $w \in \Sigma^k$. The family D_w defines an *order-k Markov model* M over the alphabet Σ. The model M is a stochastic process that generates a string s as follows. The first k characters of s are picked arbitrarily and then each character is generated according to D_w where w is the preceding k characters generated by M.

For a Markov model M we denote by M^n the distribution over Σ^n generated by running M for n steps. In particular, for a string $s \in \Sigma^n$, we denote by $M^n(s)$ the probability that M^n produces s. The *entropy* of M is defined as $H(M) = \lim_{n\to\infty} \frac{1}{n} H(M^n)$, where $H(M^n)$ is the entropy of the distribution M^n, i.e. $H(M^n) = -\sum_{s \in \Sigma^n} M^n(s) \log M^n(s)$.

We remind the reader that Markov models are always discussed asymptotically: we fix a Markov model, and then let n tend to infinity. Thus, the model has "finite memory", while the string's length approaches infinity.

Optimality. A compression algorithm A is called *C-Markov-optimal* if for every Markov model M it holds that $E_{s \in_R M^n}[|A(s)|] \leq CnH(M) + o(n)$. Here, $o(n)$ is a function such that $\frac{o(n)}{n}$ approaches 0 as n tends to infinity. This function may depend arbitrarily on the model M (e.g. it may contain a term exponential in the order of the model). Also, $s \in_R M^n$ denotes that s is a random string chosen according to the distribution M^n. A is called $(\geq C)$*-Markov-optimal* if it is not C'-Markov-optimal for any $C' < C$.[1]

An algorithm A is called *C-w.c.-optimal* (*w.c.* stands for *worst-case*) if for all alphabets Σ, for all $k \geq 0$, for all n large enough, and for all $s \in \Sigma^n$, it holds that $|A(s)| \leq CnH_k(s) + o(n)$. The $o(n)$ term may depend arbitrarily on $|\Sigma|$ and k. An algorithm A is called $(\geq C)$*-w.c.-optimal* if it is not C'-w.c.-optimal for any $C' < C$.

The following connection holds between the notions of Markov-optimality and w.c.-optimality.

Proposition 1. *If A is C-w.c.-optimal then it is C-Markov-optimal. If A is $(\geq C)$-Markov-optimal then it is $(\geq C)$-w.c.-optimal.*

Thus, a lower bound on Markov-optimality is stronger than a lower bound on w.c.-optimality.

[1] Note that if an algorithm is $(\geq C)$-Markov-optimal, this only constitutes a lower bound, and does not imply that the algorithm is optimal in any sense. For example, the algorithm that compresses every string to itself is (≥ 2007)-Markov-optimal, but it is not C-Markov-optimal for any C.

A well-known example of an optimality result is the proof that the LZ77 compression algorithm is 1-Markov-optimal [24]. A similar analysis shows [21] that the LZW compression algorithm is 1-w.c.-optimal. Similar results have been shown for other dictionary-based compressors.

The BWT. Let us define the BWT, which is a function $bwt : \Sigma^n \rightarrow \Sigma^n$. For a string $s \in \Sigma^n$, we obtain $bwt(s)$ 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. Note 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 $bwt(s)$. See an example in Figure 1. Although it may not be obvious at first glance, this function is invertible, assuming that the location of '\$' prior to its omission is known to the inverting procedure. In fact, efficient algorithms exist for computing and inverting bwt in linear time (see for example [16]).

```
mississippi$          $ mississipp i
ississippi$m          i $mississip p
ssissippp$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
$mississippi          s sissippp$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 $bwt(s)$ 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 first and last columns of the right-hand-side matrix have been drawn slightly apart from the rest of the matrix in order to highlight them.

Compressors. Let us describe the three compression algorithms based on the BWT that we analyze in this paper: $BW0$, BW_{DC} and BW_{RL}. Many more algorithms exist, but we have chosen these three as representative examples: $BW0$ is the most basic algorithm, and the easiest to analyze. It was the algorithm originally suggested by Burrows and Wheeler in [5]. $BW0$ is conjectured to be 2-w.c.-optimal by [13]. BW_{RL} is the most famous BWT-based algorithm. It is the algorithm that the program bzip2 is based on. It was analyzed by Manzini [15],

who proved that it is 5-w.c.-optimal. BW_{DC} was invented but not published by Binder (see [4, 2]), and is described in a paper of Deorowicz [7]. BW_{DC} is 1.7286-w.c.-optimal [13].
The algorithm $BW0$ compresses the input text s in three steps.

1. Compute $bwt(s)$.
2. Transform $bwt(s)$ to a string of integers $\dot{s} = mtf(bwt(s))$ by using the move to front algorithm, originally introduced in [3]. 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).[2]
3. Encode the string $\dot{s}$ by using Arithmetic Coding, to obtain the final string of bits $BW0(s) = Arith(\dot{s})$. We do not describe Arithmetic Coding here. The interested reader is referred to [23, 17]. All we use is that for any string s, $|s| H_0(s) \leq |Arith(s)| \leq |s| H_0(s) + O(\log |s|)$.[3]

We denote this algorithm by $BW0 = BWT + MTF + Arith$. Here, '+' is the *composition* operator for compression algorithms, and $Arith$ stands for Arithmetic Coding.

Intuitively, $BW0$ is a good compressor, since in $bwt(s)$ characters with the same context appear consecutively. Therefore, if s is, say, a text in English, we expect $bwt(s)$ to be a string with symbols recurring at close vicinity. As a result $\dot{s} = mtf(bwt(s))$ 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). Therefore, 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. Manzini [15] has formalized this intuition to get an upper bound on the compression ratio of $BW0$. This upper bound was improved and simplified in [13].

The algorithm $BW_{DC} = BWT + DC + Arith$ works like $BW0$, except that MTF is replaced by a transformation called *distance coding* (DC). The transformation DC encodes the character $s[i] = \sigma$ with an integer equal to the distance to the previous occurrence of σ in s, but only if this distance is not zero. When the distance is 0 (i.e. when $s[i-1] = \sigma$), we simply skip this character. Therefore, $DC(s)$ is potentially a much shorter string than s. However, $DC(s)$ might be

[2] To completely determine the encoding we must specify the status of the recency list at the beginning of the procedure. For our purposes, it is enough to assume that in the initial recency list the characters are sorted by any arbitrary order, e.g. their ASCII order.

[3] This upper bound follows from the analysis of [23], but it is not proved anywhere in the literature, since it holds only for a costly version of Arithmetic Coding that is not used in practice. The versions of Arithmetic Coding used in practice actually have, say, $|Arith(s)| \leq |s| H_0(s) + 10^{-2} |s| + O(\log |s|)$ and are more efficient in terms of running time.

a string over a larger alphabet, since the distances might be as big as n. (It is not hard to see that the number of different integers that occur in $DC(s)$ is at most $h\sqrt{2n}$).[4] To be able to decode DC, we also need to store some auxiliary information consisting of the positions of the first and last occurrence of each character. Since this additional information takes at most $2h \log n = O(\log n)$ bits we disregard this from now on. It is not hard to see that from $DC(s)$ and the auxiliary information we can recover s (see [13]). We observe that DC produces one integer per run of s. Furthermore, when s is binary, then this integer is equal to the length of the run.

The algorithm $BW_{RL} = BWT + MTF + RLE + Arith$ works like $BW0$, except that an additional *run-length encoding* (RLE) step is performed between the MTF and the Arithmetic Coding. The particular way RLE is implemented varies in different papers. The general idea is to encode each run as a separate character or sequence of characters so that a long run of, say, m 0's, can be represented using, say, $\log m$ characters in a new alphabet. Afterwards, we perform Arithmetic Coding on a string defined over this new alphabet. A standard version of RLE is described by Manzini [15].

3 Lower Bounds

We first state two lemmas that we use for proving our lower bounds. The first lemma roughly states that if a compression algorithm A performs badly on an order-zero Markov model then $BWT + A$ performs badly on the same model. This gives us a way to derive a lower bound on the optimality ratio of $BWT + A$ from a lower bound on the optimality ratio of A on order-0 Markov models.

Lemma 2. *Suppose that $Pr_{s \in_R M^n}[|A(s)| > CnH(M)] > 1 - 1/n^2$, where M is an order-zero Markov model. Then, $E_{s \in_R M^n}[|A(BWT(s))|] \geq CnH(M) - O(1)$. In particular, the compression algorithm $BWT + A$ is $(\geq C)$-Markov-optimal.*

Proof Sketch. The rough intuition behind the proof of this lemma, is that at least a $\frac{1}{n+1}$-fraction of strings are in the image of the function *bwt* (we call these strings *bwt-images*). The condition of the lemma states that all but a $\frac{1}{n^2}$-fraction of strings are hard to compress. Therefore, via a simple counting argument, we see that a vast majority of the bwt-images are hard to compress, and this gives the lemma. Special care must be taken with measuring what "a vast majority of the bwt-images" means, since some strings are bwt-images of only one string, while some may be bwt-images of $n + 1$ strings. This point accounts for most of the technical difficulty of the lemma.

[4] This is a simplified version of the transformation as described by [4, 2, 7]. Originally, two more properties were used to discard some more characters from $DC(s)$ and make the numbers that do appear in $DC(s)$ smaller. However, in our lower bound we are interested only in a binary alphabet, where the additional properties do not help.

The second lemma bounds the number of runs in binary strings generated by an order-zero Markov model. We use it to analyze the behavior of algorithms such as $MTF + Arith$ on order-zero Markov models.

Here, by *substring* we mean a consecutive substring, and a *run* in a string s is a maximal substring consisting of identical characters. Also, denote by M_p the zero-order Markov model that outputs 'a' with probability p, and 'b' with probability $1 - p$.

Lemma 3. *For any $0 < p \le \frac{1}{2}$ and any constant $c \ge 1$, for all large enough n, a string s chosen from M_p^n has, with probability at least $1 - 1/n^2$, both of the following properties:*

1. *The number of runs in s is between $2p(1-p)(n-n^{2/3})$ and $2p(1-p)(n+n^{2/3})$.*
2. *For every $1 \le k \le c$, the number of runs of 'a' of length exactly k is between $(1-p)^2p^k(n-n^{2/3})$ and $(1-p)^2p^k(n+n^{2/3})$, and the number of runs of 'b' of length exactly k is between $p^2(1-p)^k(n-n^{2/3})$ and $p^2(1-p)^k(n+n^{2/3})$.*

3.1 $BW0$ is (≥ 2)-Markov-Optimal

We now prove that the algorithm $BW0 = BWT + MTF + Arith$ is (≥ 2)-Markov-optimal. Let $\epsilon > 0$ be some positive number. It is enough to prove that $BW0$ is ($\ge 2 - \epsilon$)-Markov-optimal. Let $A0 = MTF + Arith$. By Lemma 2, it is enough to show an order-zero Markov model M such that for large enough n,

$$Pr_{s \in_R M^n}[|A0(s)| > (2-\epsilon)nH(M)] > 1 - 1/n^2 \ .$$

Theorem 4. *For every $\epsilon > 0$ there exists p such that for large enough n,*

$$Pr_{s \in_R M_p^n}[|A0(s)| > (2-\epsilon)nH(M_p)] > 1 - 1/n^2 \ .$$

Therefore $BW0$ is (≥ 2)-Markov-optimal.

Proof. Let $s \in \Sigma_2^n$ be a string with r runs. Then the string $MTF(s)$ contains r '1's and $n - r$ '0's, and thus $|A0(s)| \ge r \log \frac{n}{r} + (n-r)\log\frac{n}{n-r} \ge r \log \frac{n}{r}$. We prove that $r \log \frac{n}{r} > (2-\epsilon)nH(M_p)$ for $r = 2p(1-p)(n \pm n^{2/3})$. By Lemmas 2 and 3 this gives the theorem. In fact, we prove that $r \log \frac{n}{r} > (2 - \epsilon/2)nH(M_p)$ for $r = 2p(1-p)n$. The term $\pm n^{2/3}$ is "swallowed" by the slack of $\epsilon/2$ when n is large enough.
To prove that $\frac{r \log \frac{n}{r}}{nH(M_p)} > 2 - \epsilon/2$ first note that

$$\frac{r \log \frac{n}{r}}{nH(M_p)} = \frac{2p(1-p)n \log \frac{1}{2p(1-p)}}{np \log \frac{1}{p} + n(1-p)\log\frac{1}{1-p}} \ .$$

This expression approaches 2 from below when p tends to 0, since

$$\frac{2p(1-p)\log\frac{1}{2p(1-p)}}{p\log\frac{1}{p} + (1-p)\log\frac{1}{1-p}} = \frac{2p(1-p)\log\frac{1}{p} - \Theta(p)}{p\log\frac{1}{p} + \Theta(p)} \xrightarrow{p \to 0} 2 \ .$$

Thus, we can choose p to be a function of ϵ (but not of n), such that this expression is larger than $2 - \epsilon/2$. $\square$

This gives a positive answer to Conjecture 20 in [13]. Also, it was shown there that for binary alphabet, $BW0$ is 2-w.c.-optimal. Therefore, the lower bound we prove here is tight. In [13] it was also conjectured that $BW0$ is 2-w.c.-optimal for alphabets of any size. We have not been able to prove this so far.

3.2 BW_{DC} is ($\geq$ 1.265)-Markov-Optimal, BW_{RL} is ($\geq$ 1.297)-Markov-Optimal.

In the full version of the paper we also prove that BW_{DC} is ($\geq$ 1.265)-Markov-optimal. Due to Lemma 2, in order to prove this it suffices to prove that the algorithm $A_{DC} = DC + Arith$ performs badly on an order-zero Markov model. The Markov model that was used in the last section, which had infinitesimally small probability for producing 'a', will not work here. A_{DC} works quite well for such Markov models. Instead, we use Lemma 3 to get exact statistics (up to lower-order terms) for the character frequencies in $DC(s)$ where s is taken from the Markov model M_p, and we compute the entropy according to these character frequencies. Using a similar technique we prove that BW_{RL} is ($\geq$ 1.297)-Markov-optimal.

4 Experiments

We confirmed experimentally that our lower bounds indeed hold. For this purpose, we have coded the three algorithms $BW0$, BW_{DC}, and BW_{RL}. Unlike bzip2 which cuts the file into chunks of 900K before compressing, our programs compress the whole file in one piece.

Rather than using Arithmetic Coding on the integer sequence, as the final stage of the compression algorithm, our implementations computed the empirical entropy of the integer sequence. For any string this entropy is a lower bound on the size of Arithmetic Encoding that string. This made our experiment cleaner since our focus is on lower bounds. We also did not consider auxiliary data that one needs to save in order to decompress the file, such as the initial status of the mtf recency list, the location of the end-of-string symbol after bwt, etc. We refer to our implementations as "clean" versions of the respective algorithms.

To compare BWT-based compressors against dictionary-based compressors we also implemented a clean version of the LZW algorithm [22]. This implementation follows closely the theoretical version of LZW described in [22]. In particular it does not have an upper bound on the dictionary size. The compressed representation consists of a sequence of pointers (integer values) to the dictionary and a sequence of characters (the list of addition symbols). We compute the entropy of each of these sequences. The sum of both entropies is the "compressed size" of the file according to our clean LZW implementation. Note that the sequence of pointers is a string defined over a potentially very large alphabet. Therefore to arithmetically code this string one would need to store a very large table which is not realistic. We believe however that the entropy of this string does reflect in a clean way the size of the LZW encoding.

We have generated 100 files of size 10 Megabytes from the source M_p for various values of p. For each compressor and each value of p, we compute the *optimality ratio*, which is the ratio between the number of bits per character that the compressor used and the entropy of the source. These results are given in Table 1.

First notice that our results with regards to $BW0$, BW_{DC}, and BW_{RL} follow closely the theoretical results of Section 3. In particular for $p = 0.001$ the optimality ratio of $BW0$ is 1.823. (A calculation, omitted due to lack of space, shows that this ratio is exactly equal to the theoretical prediction). For $p = 0.051$ the optimality ratio of BW_{DC} is 1.266, and for $p = 0.06$ the optimality ratio of BW_{RL} is 1.298. Furthermore as predicted by our analysis the optimality ratio of $BW0$ increases as p goes to zero.

The optimality ratio of our "clean" implementation of LZW increases when p decreases. This effect is related to results of Kosaraju and Manzini [14]. Recall, that the optimality ratio of LZW on M_p tends to 1 as n tends to infinity. So it is surprising, to see that in our experiments the optimality ratio of LZW may get as large as 1.53, and is always at least 1.11. This indicates that the optimality ratio of LZW tends to 1 quite slowly, and is quite far from 1 for the 10MB strings that we compressed. Also, note that for some values of p, the optimality ratio of LZW is worse than that of the BWT-based compressors.

We have also checked how the optimality ratios of the algorithms change when we fix the value of p, and increase the file size. (The experimental data is not included due to lack of space. It can be found in the full version of the paper). The optimality ratio of each of the BWT-based compressors stays exactly the same when the file size grows. The optimality ratio of LZW, on the other hand, tends to 1 as n grows, but it decreases slowly. This can be explained if we recall that the optimality ratio of LZW is $1 + O\left(\frac{1}{\log n}\right)$, see [18]. Note that the constant hidden by the O-notation depends on the parameters of the Markov model, see [24, 14]. In our case, it depends on p). The low-order $O\left(\frac{1}{\log n}\right)$ term decays slowly. For example, for $n = 10^7$, $\frac{1}{\log n} \approx 0.04$. Nonetheless, for any fixed p, there exists a large enough value of n for which LZW beats the BWT-based compressors for strings of length n drawn from M_p. Thus, we can say that LZW *asymptotically beats* BWT-compressors on Markov models.

We did not consider statistical compressors in any of the experiments here. Intuitively, for an order-zero Markov model, such a compressor is simply Arithmetic Coding (or the "clean" version thereof), which has a constant optimality ratio of 1, no matter what n is. We have continued to see that statistical compressors are the best among all compressors we study, when working over Markov models.

Real Compressors. We have also performed a similar experiment using the "real" compressors *bzip2* [19], *gzip* [11], and *ppmdi* [20]. These compressors seem to work like their theoretical counterparts, with some interesting exceptions. We include these experimental results in the full version of the paper.

p	clean BW0	clean BW_{DC}	clean BW_{RL}	clean LZW	σ
0.001	1.823	1.141	1.143	1.535	0.016
0.005	1.772	1.210	1.216	1.490	0.006
0.01	1.737	1.232	1.242	1.347	0.004
0.051	1.579	1.266	1.296	1.211	0.0012
0.06	1.552	1.265	1.298	1.202	0.0011
0.1	1.450	1.250	1.286	1.171	0.0008
0.2	1.253	1.176	1.202	1.137	0.0004
0.3	1.114	1.093	1.103	1.124	0.0002
0.4	1.029	1.027	1.028	1.113	0.00013
0.5	1.000	1.000	1.000	1.113	0.00004

Table 1. The optimality ratio of the clean compressors, measured on texts generated from the Markov source M_p. Each experiment was run on 100 samples, each of size 10MB. The standard deviations are not listed per-entry, but the maximal standard deviation among all those in the row is listed in the last column (the standard deviations tend to be similar for values in the same row).

Experiments on Realistic High-Order Models. We have also performed experiments on high-order Markov models that were generated using realistic parameters extracted from English text. We found that BWT-based compressors behave quite badly on such Markov models. These experimental results are included in the full version of the paper.

5 Conclusion

In this paper we have shown lower bounds on the optimality ratio of BWT-based compressors on Markov sources. This suboptimal performance on Markov sources is corroborated by experiments with Markov sources that imitate an English text.

On the other hand it is known that on English text, BWT-based compressors work extremely well. We believe that BWT-compressors work on English text better than Dictionary-based compressors (and not much worse than statistical compressors) because of non-Markovian elements in English text.

This discrepancy between the performance of BWT-based compressors on Markov sources that resemble an English text and their performance on the text itself is yet to be explored. What kind of regularity is there in English text that compressors exploit?

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] 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.

[4] E. Binder. Distance coder. Usenet group comp.compression, 2000.

[5] M. Burrows and D. J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[6] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & sons, 1991.

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

[8] M. Effros, K. Visweswariah, S. Kulkarni, and S. Verdu. Universal lossless source coding with the Burrows Wheeler transform. *IEEE Transactions on Information Theory*, 48(5):1061–1081, May 2002.

[9] P. Ferragina, R. Giancarlo, and G. Manzini. The engineering of a compression boosting library: Theory vs practice in BWT compression. In *Proc. 14th European Symposium on Algorithms (ESA '06)*, pages 756–767, 2006.

[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] J. Gailly and M. Adler. The gzip compressor. http://www.gzip.org/.

[12] R. Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24(6):668–674, 1978.

[13] H. Kaplan, S. Landau, and E. Verbin. A simpler analysis of Burrows-Wheeler based compression. *To be puiblished in Theoretical Computer Science, special issue on the Burrows-Wheeler Transform and its Applications*, 2007. Preliminary version published in CPM '06.

[14] S. R. Kosaraju and G. Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM J. Comput.*, 29(3):893–911, 1999.

[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 Transactions on Information Systems*, 16(3):256–294, 1998.

[18] S. A. Savari. Redundancy of the Lempel-Ziv-Welch code. In *Proc. Data Compression Conference (DCC)*, pages 191–200, 1997.

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

[20] D. Shkarin and J. Cheney. ppmdi, a statistical compressor. This is Shkarin's compressor PPMII, as modified and incorporated into XMLPPM by Cheney, and then extracted from XMLPPM by J. Adiego.

[21] P. Shor. Lempel-Ziv compression (lecture notes for the course principles of applied mathematics). `www-math.mit.edu/~shor/PAM/lempel_ziv_notes.pdf`.

[22] T. A. Welch. A technique for high-performance data compression. *Computer*, 17:8–19, 1984.

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

[24] A. D. Wyner and J. Ziv. The sliding-window Lempel-Ziv algorithm is asymptotically optimal. *Proc. IEEE*, 82(8):872–877, 1994.