# An Analysis of the Burrows-Wheeler Transform

Giovanni Manzini
Dipartimento di Informatica, Università del Piemonte Orientale, Italy.

The Burrows-Wheeler Transform (also known as Block-Sorting) is at the base of compression algorithms which are the state of the art in lossless data compression. In this paper we analyze two algorithms which use this technique. The first one is the original algorithm described by Burrows and Wheeler, which, despite its simplicity, outperforms the `Gzip` compressor. The second one uses an additional run-length encoding step to improve compression. We prove that the compression ratio of both algorithms can be bounded in terms of the $k$-th order empirical entropy of the input string for any $k \geq 0$. We make no assumptions on the input and we obtain bounds which hold in the worst case, that is, for every possible input string. All previous results for Block-Sorting algorithms were concerned with the average compression ratio and have been established assuming that the input comes from a finite-order Markov source.

Categories and Subject Descriptors: E.4 [**Coding and Information Theory**]: *data compactions and compression*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: *Nonnumerical Analysis and Problems*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Block sorting, Burrows-Wheeler Transform, move-to-front encoding, worst-case analysis of compression

## 1. INTRODUCTION

A recent breakthrough in data compression has been the introduction of the Burrows-Wheeler Transform (BWT from now on) [Burrows and Wheeler 1994]. Loosely speaking, the BWT produces a permutation $\texttt{bwt}(s)$ of the input string $s$ such that from $\texttt{bwt}(s)$ we can retrieve $s$ but at the same time $\texttt{bwt}(s)$ is much easier to compress. The whole idea of a transformation that makes a string easier to compress is completely new, even if, after the appearance of the BWT some researchers recognized that it is related to some well known compression techniques (see for example [Cleary and Teahan 1997; Fenwick 1996b; Larsson 1998]).

The BWT is a very powerful tool and even the simplest algorithms which use it have surprisingly good performances (the reader may look at the very simple and clean BWT-based algorithm described in [Nelson 1996] which outperforms, in terms of compression ratio, the commercial package `Pkzip`). More advanced BWT-based compressors, such as `Bzip` [Seward 1997] and `Szip` [Schindler 1997], are among the best compressors currently available. As can be seen from the results reported in [Arnold and Bell ; Fenwick 1996a] BWT-based compressors achieve a very good compression ratio using relatively small

THIS VERSION CONTAINS THE CORRECTION OF A FEW TYPOS WHICH UNFORTUNATELY APPEARED IN THE JACM PRINTED VERSION. I POINT OUT THAT THE RESPONSIBILITY OF THESE TYPOS IS ENTIRELY MINE AND NOT OF THE ACM STAFF.

A preliminary version of this paper has been presented to the 10th Symposium on Discrete Algorithms (SODA '99).
Author's address: Dipartimento di Informatica, Università del Piemonte Orientale, Corso Borsalino, 54, I-15100, Alessandria, Italy, and IIT-CNR, Pisa, Italy. Email: `manzini@mfn.unipmn.it`.

resources (time and space). Considering that BWT-based compressors are still in their infancy, we believe that in the next future they are likely to become the new standard in lossless data compression.

Another remarkable property of the BWT is that it can be used to build a data structure which is a sort of *compressed suffix array* for the input string $s$ [Ferragina and Manzini 2000; Ferragina and Manzini 2001]. Such data structure consists of a compressed version of $\texttt{bwt}(s)$ plus $o(|s|)$ bits of auxiliary information. Using this data structure we can compute the number of occurrences of an arbitrary pattern $p$ in $s$ in $O(|p|)$ time, and we can compute the position in $s$ of each one of such occurrences in $O(\log^{\epsilon}|s|)$ time.

Despite of its practical relevance, no satisfactory theoretical analysis of the BWT is available. Although it is easy to intuitively understand why the BWT helps compression, none of the currently used BWT-based algorithms have been analyzed theoretically. In other words, these algorithms work very well in practice but no proof has been given that their compression ratio is, say, within a constant factor of the zeroth order entropy of the input.

In [Sadakane 1997; Sadakane 1998] Sadakane has proposed and analyzed three different algorithms based on the BWT. Assuming the input string is generated by a finite-order Markov source, he proved that the average compression ratio of these algorithms approaches the entropy of the source. More recently, Effros [Effros 1999] has considered similar algorithms and has given bounds on the speed at which the average compression ratio approaches the entropy. Although these results provide useful insight on the BWT, they are not completely satisfying for several reasons. First, these results deal with algorithms which are not realistic (and in fact are not used in practice). For example, some of these algorithms require the knowledge of quantities which are usually unknown such as the order of the Markov source or the number of states in the ergodic source. Secondly, none of these analyses deals with run-length encoding which, as we will see, is a technique frequently used in connection with the BWT. Finally, the hypothesis that the input comes from a finite-order Markov source is not always realistic, and results based on this assumption are only valid on the average and not in the worst case.

In this paper, we compare the compression ratio of BWT-based algorithms with the *empirical entropy* of the input string. The empirical entropy is defined in terms of the number of occurrences of each symbol or group of symbols. Therefore, it is defined for any string without requiring any probabilistic assumption and it can be used to establish worst case results. For $k \geq 0$, the $k$-th order empirical entropy $H_k(s)$ provides a lower bound to the compression we can achieve using for each symbol a code which depends on the $k$ symbols preceding it. Since, as we will see, the value $H_k(s)$ is sometimes a too conservative lower bound, in this paper we consider also the *modified* empirical entropy $H_k^*(s)$ which is defined imposing the additional requirement that the coding of a string $s$ takes at least enough bits to write down its length in binary.

In Section 4 we analyze the algorithm in which the output of the BWT is processed by the move-to-front transformation followed by order-0 encoding. We call this algorithm `BW0`. `BW0` is the basic algorithm described in the paper introducing the BWT [Burrows and Wheeler 1994, Sect. 3] and it has been tested in [Fenwick 1996b] (under the name `bs-Order0`). Although `BW0` is one of the simplest BWT-based algorithms, it achieves a better compression than `Gzip`, which is based on `LZ77`. We prove that for any string $s$ and any $k \geq 0$, the output size of `BW0` on input $s$ is bounded by $\approx 8|s|H_k(s) + (2/25)|s|$. This means that, except for a small overhead for each input symbol, `BW0` compression ratio is within a constant factor of the $k$-th order entropy. Note that $k$ *is not* a parameter of the algorithm, that is, the bound holds *simultaneously* for all $k \geq 0$. Although the constant 8 is admittedly too high for our result to have a practical impact, this is the first non-trivial entropic bound for `BW0`. Our result is obtained through an analysis of the move-to-front encoding which we believe is of independent interest. We prove that move-to-front transforms a string which is locally homogeneous into a string which is globally homogeneous. Although this was well known at the qualitative level, ours is the first analytical proof of this fact.

In Section 5 we consider a variant of BWO, called $\texttt{BWO}_{RL}$, which has an additional step consisting in the run-length encoding of the runs of zeroes produced by the move-to-front transformation. As reported in [Fenwick 1996a] all the best BWT-based compressors make use of this technique. We prove that for any $k \geq 0$ there exists a constant $g_k$ such that for any string $s$

$$\texttt{BWO}_{RL}(s) \leq (5+\epsilon)|s|H_k^*(s) + g_k, \tag{1}$$

where $\texttt{BWO}_{RL}(s)$ is the output size of $\texttt{BWO}_{RL}$ on input $s$, $\epsilon \approx 10^{-2}$, and $H_k^*(s)$ is the modified $k$-th order empirical entropy. The significance of (1) is that the use of run-length encoding makes it possible to get rid of the constant overhead per input symbol, and to reduce the size of the multiplicative constant associated to the entropy. Note that the use of the modified empirical entropy $H_k^*$ is essential since (1) does not hold if we consider $H_k$.

To our knowledge, a bound similar to (1) has not been proven for any other compression algorithm. Indeed, for many of the better known algorithms (including some BWT-based compressors) one can prove that a similar bound *cannot* hold. For example, although the output of LZ77 and LZ78 is bounded by $|s|H_k(s) + O(|s|(\log\log|s|)/\log|s|)$, for any $\lambda > 0$ it is possible to find a string $s$ such that the output of these algorithms is greater than $\lambda|s|H_1^*(s)$ (see [Kosaraju and Manzini 1999]; obviously for such strings we have $H_1^*(s) \ll (\log\log|s|)/\log|s|$). The algorithm PPMC [Moffat 1990], which has been the state of the art compressor for several years, predicts the next symbol on the basis of the $l$ previous symbols, where $l$ is a parameter of the algorithm. Thus, there is no hope that its compression ratio can be bounded in terms of the $k$-th order entropy for $k > l$. Two algorithms for which a bound similar to (1) might hold for any $k \geq 0$ are DMC [Cormack and Horspool 1987] and PPM* [Cleary and Teahan 1997]. Both of them predict the next symbol on the basis of a (potentially) unbounded context and they work very well in practice. Unfortunately, these two algorithms have not been analyzed theoretically.

Although asymptotic optimality does not necessarily mean good performance in the real world, where multiplicative constants and lower order terms are often significant, it is reassuring to know that algorithms which work well in practice have nice theoretical properties. Our results somewhat guarantee that BWT-based algorithms remain competitive for very long strings, or strings with very small entropy.

**Notation for compression algorithms.** In the following we will introduce several algorithms: some of them are complete data compression algorithms designed to reduce the size of the input, others are recoding schemes which transform the input without compressing it. For example the Burrows Wheeler transform is a recoding scheme since its output $\texttt{bwt}(s)$ is simply a permutation of the input string $s$. Throughout the paper, recoding schemes will be denoted using lower-case letters only, while complete data compression algorithms will be denoted with an initial upper-case letter. Given a recoding scheme a we write $\texttt{a}(s)$ to denote the output of a on input $s$. Given a compression algorithm A we write $\texttt{A}(s)$ to denote the *size* (i.e. number of bits) of the output produced by A on input $s$.

## 2. THE EMPIRICAL ENTROPY OF A STRING

Let $s$ be a string of length $n$ over the alphabet $\mathcal{A} = \{\alpha_1, \ldots, \alpha_h\}$, and let $n_i$ denote the number of occurrences of the symbol $\alpha_i$ inside $s$. The zeroth order empirical entropy of the string $s$ is defined as

$$H_0(s) = -\sum_{i=1}^{h} \frac{n_i}{n} \log\left(\frac{n_i}{n}\right), \tag{2}$$

where we assume $0 \log 0 = 0$ (all logarithms in this paper are taken to the base 2). The value $|s|H_0(s)$, represents the output size of an ideal compressor which uses $-\log\frac{n_i}{n}$ bits for coding the symbol $\alpha_i$. It is well known that this is the maximum compression we can achieve using a uniquely decodable code in which a fixed codeword is assigned to each alphabet symbol. We can achieve a greater compression if the

codeword we use for each symbol depends on the $k$ symbols preceding it. For any length-$k$ word $w \in \mathcal{A}^k$ let $w_s$ denote the string consisting of the concatenation of the single characters following each occurrence of $w$ inside $s$. For example, if $s = \texttt{abcabcabd}$ we have $\texttt{ab}_s = \texttt{ccd}$. Note that the length of $w_s$ is equal to the number of occurrences of $w$ in $s$, or to that number minus one if $w$ is a suffix of $s$. The value

$$H_k(s) = \frac{1}{|s|} \sum_{w \in \mathcal{A}^k} |w_s| H_0(w_s) \tag{3}$$

is called the $k$-th order empirical entropy of the string $s$. The value $|s|H_k(s)$ represents a lower bound to the compression we can achieve using codes which depend on the $k$ most recently seen symbols. Not surprisingly, for any string $s$ and $k \geq 0$, we have $H_{k+1}(s) \leq H_k(s)$.

*Example* 1. Let $s = \texttt{mississippi}$. For $k = 1$ we have $\texttt{m}_s = \texttt{i}$, $\texttt{i}_s = \texttt{ssp}$, $\texttt{s}_s = \texttt{sisi}$, $\texttt{p}_s = \texttt{pi}$. Since, according to (2), we have $H_0(\texttt{i}) = 0$, $H_0(\texttt{ssp}) = 0.918\ldots$, $H_0(\texttt{sisi}) = 1$, $H_0(\texttt{pi}) = 1$, the first order empirical entropy is

$$H_1(s) = \frac{1}{11} H_0(\texttt{i}) + \frac{3}{11} H_0(\texttt{ssp}) + \frac{4}{11} H_0(\texttt{sisi}) + \frac{2}{11} H_0(\texttt{pi}) = 0.796\ldots.$$

The empirical entropy resembles the entropy defined in the probabilistic setting (for example, when the input comes from a Markov source). However, the empirical entropy is defined for any string and can be used to measure the performance of compression algorithms without any assumption on the input.

The value $H_k(s)$ is defined assuming that the first $k$ symbols of $s$ are coded for free. Therefore, it provides a reasonable lower bound to the compression ratio only when $|s| \gg k$. The following example shows that even when $|s| \gg k$ the empirical entropy may provide a lower bound which is too conservative and that we cannot reasonably hope to achieve.

*Example* 2. Let $s = \texttt{cc}(\texttt{ab})^n$. We have $\texttt{a}_s = \texttt{b}^n$, $\texttt{b}_s = \texttt{a}^{n-1}$, $\texttt{c}_s = \texttt{ca}$. This yields

$$|s|H_1(s) = nH_0(\texttt{b}^n) + (n-1)H_0(\texttt{a}^{n-1}) + 2H_0(\texttt{ca}) = 2.$$

Hence, $|s|H_1(s)$ does not depend on $n$.

The reason for which in the above example $|s|H_1(s)$ fails to provide a reasonable bound, is that for any string consisting of multiple copies of the same symbol, for example $s = \texttt{a}^n$, we have $H_0(s) = 0$. Since the output of any compression algorithm must contain enough information to recover the length of the input, it is natural to consider the following alternative definition of zeroth order empirical entropy. For any string $s$ let

$$H_0^*(s) = \begin{cases} 0 & \text{if } |s| = 0, \\ (1 + \lfloor \log |s| \rfloor)/|s| & \text{if } |s| \neq 0 \text{ and } H_0(s) = 0, \\ H_0(s) & \text{otherwise.} \end{cases} \tag{4}$$

Note that $1 + \lfloor \log |s| \rfloor$ is the number of bits required to express $|s|$ in binary. $H_0^*$ satisfies many of the properties of the empirical entropy, including the monotonicity property

$$H_0^*(\texttt{a}^n\texttt{b}^m) < H_0^*(\texttt{a}^{n-1}\texttt{b}^{m+1}) \qquad \text{for } 0 \leq m < n-1.$$

Having defined $H_0^*$ it is natural to define the $k$-th order modified empirical entropy $H_k^*$ using a formula similar to (3). Unfortunately, if we simply replace $H_0$ with $H_0^*$ in (3) the resulting entropy $H_k^*$ does not satisfy the inequality $H_{k+1}^*(s) \leq H_k^*(s)$ for every string $s$. This is shown by the following example.

*Example* 3. Let $s = (\mathtt{bad})^n(\mathtt{cad})^n$. We have $\mathtt{a}_s = \mathtt{d}^{2n}$, $\mathtt{b}_s = \mathtt{a}^n$, $\mathtt{c}_s = \mathtt{a}^n$, and $\mathtt{d}_s = \mathtt{b}^{n-1}\mathtt{c}^n$. Thus, according to (3) the first order modified entropy would be

$$\frac{1}{|s|}\left[2nH_0^*(\mathtt{d}^{2n}) + nH_0^*(\mathtt{a}^n) + nH_0^*(\mathtt{a}^n) + (2n-1)H_0^*(\mathtt{b}^{n-1}\mathtt{c}^n)\right]. \tag{5}$$

For $k = 2$ we have $\mathtt{ba}_s = \mathtt{d}^n$, $\mathtt{ca}_s = \mathtt{d}^n$, $\mathtt{db}_s = \mathtt{a}^{n-1}$, $\mathtt{dc}_s = \mathtt{a}^n$, and $\mathtt{ad}_s = \mathtt{b}^{n-1}c^n$. Hence, the second order modified entropy would be

$$\frac{1}{|s|}\left[nH_0^*(\mathtt{d}^n) + nH_0^*(\mathtt{d}^n) + (n-1)H_0^*(\mathtt{a}^{n-1}) + nH_0^*(\mathtt{a}^n) + (2n-1)H_0^*(\mathtt{b}^{n-1}\mathtt{c}^n)\right]. \tag{6}$$

Since $2nH_0^*(\mathtt{d}^{2n}) \approx \log 2n$ and $2nH_0^*(\mathtt{d}^n) \approx 2\log n$ we have that (5) is (asymptotically) smaller than (6).

The above example shows that, when $H_0$ is replaced by $H_0^*$, the use of a longer context for the prediction of the next symbol does not always yield an increase in compression. For this reason, we define $H_k^*$ as the maximum compression ratio we can achieve using for each symbol a codeword which depends on a context of size *at most* $k$ (instead of always using a context of size $k$). We use the following notation. For a given string $s$, let $\mathcal{S}_k$ denote the set of all $k$-letter substrings of $s$. Let $\mathcal{Q}$ be a subset of $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k$. We say that $\mathcal{Q}$ *covers* $\mathcal{S}_k$, and we write $\mathcal{Q} \preceq \mathcal{S}_k$, if every string $w \in \mathcal{S}_k$ has a unique suffix in $\mathcal{Q}$.

*Example* 4. Let $s = \mathtt{mississippi}$. We have $\mathcal{S}_2 = \{\mathtt{mi}, \mathtt{is}, \mathtt{ss}, \mathtt{si}, \mathtt{ip}, \mathtt{pp}, \mathtt{pi}\}$. A set which covers $\mathcal{S}_2$ is $\{\mathtt{i}, \mathtt{is}, \mathtt{ss}, \mathtt{p}\}$. For the string $s = (\mathtt{bad})^n(\mathtt{cad})^n$ of Example 3, we have $\mathcal{S}_2 = \{\mathtt{ba}, \mathtt{ca}, \mathtt{db}, \mathtt{dc}, \mathtt{ad}\}$. A set which covers $\mathcal{S}_2$ is $\{\mathtt{a}, \mathtt{db}, \mathtt{dc}, \mathtt{ad}\}$.

Definition 2.1. *For each $\mathcal{Q}$, such that $\mathcal{Q} \preceq \mathcal{S}_k$, let*

$$H_{\mathcal{Q}}^*(s) = \frac{1}{|s|}\sum_{w \in \mathcal{Q}} |w_s| H_0^*(w_s).$$

*We define the $k$-th order* modified empirical entropy *as*

$$H_k^*(s) = \min_{\mathcal{Q} \preceq \mathcal{S}_k} H_{\mathcal{Q}}^*(s). \tag{7}$$

□

The value $H_{\mathcal{Q}}^*(s)$ denotes the compression we achieve using the strings in $\mathcal{Q}$ as contexts for the prediction of the next symbol. The value $H_k^*(s)$ represents therefore the maximum compression we can achieve using contexts of length up to $k$. In the following we use $\mathcal{T}_k$ to denote the set for which the minimum (7) is achieved. Therefore we write

$$H_k^*(s) = \frac{1}{|s|}\sum_{w \in \mathcal{T}_k} |w_s| H_0^*(w_s). \tag{8}$$

*Example* 5. Let $s = (\mathtt{bad})^n(\mathtt{cad})^n$ as in Example 3. The first order modified entropy $H_1^*(s)$ is achieved for $\mathcal{T}_1 = \mathcal{S}_1$ and is given by (5). The second order modified entropy $H_2^*(s)$ is achieved for $\mathcal{T}_2 = \{\mathtt{a}, \mathtt{db}, \mathtt{dc}, \mathtt{ad}\}$ and is given by

$$H_2^*(s) = \frac{1}{|s|}\left[2nH_0^*(\mathtt{d}^{2n}) + (n-1)H_0^*(\mathtt{a}^{n-1}) + nH_0^*(\mathtt{a}^n) + (2n-1)H_0^*(\mathtt{b}^{n-1}\mathtt{c}^n)\right].$$

It is straightforward to verify that $H_{k+1}^*(s) \leq H_k^*(s)$ for every string $s$. Note that, if we replace $H_0^*$ with $H_0$ inside Definition 2.1 we get an alternative definition of $H_k(s)$ (in this case, the minimum (7) is achieved for $\mathcal{T}_k = \mathcal{S}_k$). For this reason we say that $H_k^*$ generalizes $H_k$ when we consider $H_0^*$ instead of

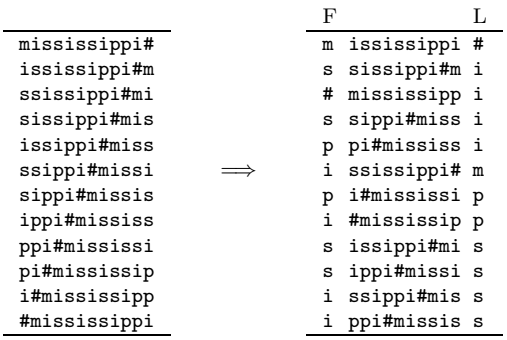


Fig. 1. Example of Burrows-Wheeler transform for the string $s =$ `mississippi`. The matrix on the right has the rows sorted in right-to-left lexicographic order. The string $\texttt{bwt}(s)$ is the first column $F$ with the symbol # removed; in this example $\texttt{bwt}(s) =$ `msspipissii`.

$H_0$. The above discussion shows that $H_k^*$ is a more realistic lower bound to the compression ratio we can achieve using contexts of size $k$ or less. Since $H_k$ has a much simpler definition and $H_k(s) \leq H_k^*(s)$ for any string $s$, in many situations it may be still preferable to work with $H_k$. However, when we are trying to establish tight bounds which hold for every string, the modified empirical entropy is more appropriate.

## 3. THE BURROWS-WHEELER TRANSFORM AND RELATED ALGORITHMS

The Burrows-Wheeler transform [Burrows and Wheeler 1994] consists of a reversible transformation of the input string $s$. The transformed string, that we denote by $\texttt{bwt}(s)$, contains the same characters as $s$ but it is usually easier to compress. The string $\texttt{bwt}(s)$ is obtained as follows (see Fig. 1). First we append to end of $s$ a special character # smaller than any other character. Then we form a (conceptual) matrix $\mathcal{M}$ whose rows are the cyclic shifts of $s\#$ sorted in right-to-left lexicographic order.[1] We set $\texttt{bwt}(s)$ to be the first column of the sorted matrix with the character # removed. Note that $\texttt{bwt}(s)$ can be seen as the result of sorting $s$ using, as a sort key for each character, its context, that is, the set of characters preceding it. The output of the Burrows-Wheeler transform is the string $\texttt{bwt}(s)$ and the index $I$ of the row starting with # in the sorted matrix (for example, in Fig. 1 we have $\texttt{bwt}(s) =$ `msspipissii` and $I = 3$).

Let $F$ (resp. $L$) denote the first (resp. last) column of the sorted matrix $\mathcal{M}$. We write $F_i$ (resp. $L_i$) to denote the $i$-th character of column $F$ (resp. column $L$). The following properties have been proven in [Burrows and Wheeler 1994].

**a.** Every column of $\mathcal{M}$ is a permutation of $s\#$. As a consequence, the last column $L$ can be obtained by lexicographically sorting the characters of $F$.
**b.** For $i = 2, \ldots, |s| + 1$, the character $L_i$ is followed, in the string $s\#$, by the character $F_i$.
**c.** For any character $\alpha$, the $i$-th occurrence of $\alpha$ in $F$ corresponds to the $i$-th occurrence of $\alpha$ in $L$. For example, in Fig. 1 the second `i` of $F$ (which is $F_8$) corresponds to the second `i` of $L$ (which is $L_3$). Indeed both $F_8$ and $L_3$ correspond to the last `i` in `mississippi`.

The above properties are the key to reverse the BWT, that is, to reconstruct $s$ given $\texttt{bwt}(s)$ and $I$. We show how the reconstruction works for the example of Fig. 1. From $\texttt{bwt}(s)$ and $I$ we retrieve $F$, and by sorting $F$, we retrieve $L$. Since # is smaller than any other character, we know that $s\#$ is the first row of the sorted matrix $\mathcal{M}$. Hence, $F_1 =$ `m` is the first character of $s$. By property **c.** we get that $F_1$ (the

[1] In the original formulation given in [Burrows and Wheeler 1994] rows are sorted in left-to-right lexicographic order.

first m in column $F$) corresponds to $L_6$ (the first m in column $L$). By property **b.** we get that the second character of $s$ is $F_6 =$ i. Using again property **c.** we get that $F_6$ (the first i in column $F$) correspond $L_2$ (the first i in column $L$). By property **b.** we get that the third character of $s$ is $F_2 =$ s. The process continues until we reach the character #.

Why is the string $\texttt{bwt}(s)$ important for us? The reason is that $\texttt{bwt}(s)$ has the following remarkable property: for each substring $w$ of $s$, the characters following $w$ in $s$ are grouped together inside $\texttt{bwt}(s)$. This is a consequence of the fact that all rotations ending in $w$ are consecutive in the sorted matrix. Using the notation of the previous section, we have that $\texttt{bwt}(s)$ contains, as a substring, a permutation of the string $w_s$. Recalling the definitions (3) and (8) of $H_k$ and $H_k^*$, we write

$$\texttt{bwt}(s) = \bigcup_{w \in \mathcal{A}^k} \pi_w(w_s) \qquad \left[\text{resp. } \texttt{bwt}(s) = \bigcup_{w \in \mathcal{T}_k} \pi_w(w_s)\right], \tag{9}$$

where $\pi_w(w_s)$ denotes a permutation of the string $w_s$.[2] Permuting a string does not change its zeroth order entropy, that is, $H_0(w_s) = H_0(\pi_w(w_s))$ and the same is true for $H_0^*$. Hence, if we had an ideal algorithm A such that for any partition $s_1 s_2 \cdots s_t$ of $s$

$$\texttt{A}(s) \leq \sum_{i=1}^{t} |s_i| H_0(s_i) \qquad \left[\text{resp. } \texttt{A}(s) \leq \sum_{i=1}^{t} |s_i| H_0^*(s_i)\right] \tag{10}$$

then, by (3) and (8), we would have

$$\texttt{A}(\texttt{bwt}(s)) \leq |s| H_k(s) \qquad \left[\text{resp. } \texttt{A}(\texttt{bwt}(s)) \leq |s| H_k^*(s)\right].$$

In other words, combining A with bwt we would be able to compress any string up to its $k$-th order (modified) empirical entropy *for any* $k \geq 0$. Thus, the Burrows-Wheeler transform can be seen as a tool for reducing the problem of compressing up to the $k$-th order entropy to the problem of compressing *distinct portions* of the input string up to their *zeroth order* entropy. This is a crucial observation for our analysis. Although no algorithm which satisfies (10) is likely to exist, we show that there are algorithms whose output size is "close" to $\sum_{i=1}^{t} |s_i| H_0(s_i)$ or $\sum_{i=1}^{t} |s_i| H_0^*(s_i)$. By using these algorithms to process the output of the BWT we get a compression ratio which is "close" to the $k$-th order entropy.

Most of the compression algorithms based on the BWT, process the string $\texttt{bwt}(s)$ with the move-to-front transformation which is a recoding scheme[3] introduced in [Bentley, Sleator, Tarjan, and Wei 1986; Ryabko 1980]. The move-to-front transformation encodes the symbol $\alpha_i$ with an integer equal to the number of distinct symbols encountered since the previous occurrence of $\alpha_i$. More precisely, the encoder 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 $\mathcal{A} = \{\alpha_1, \ldots, \alpha_h\}$ is transformed to a string over $\{0, \ldots, h-1\}$ (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 $\texttt{mtf}_\pi$ the algorithm in which the initial status of the recency list is given by the permutation $\pi$ defined over $\mathcal{A}$. We denote by mtf the algorithm in which the initial ordering of the recency list is induced by the input string $s$, that is, the $i$-th symbol in the recency list is given by the $i$-th symbol in the input (not counting multiple occurrences of the same symbol).

The reason for which $\texttt{bwt}(s)$ is often processed using move-to-front is the following. We have observed that the BWT collects together the symbols following a given context, that is, $\texttt{bwt}(s) = \cup_w \pi_w(w_s)$

[2] In addition to $\cup_{w \in \mathcal{A}^k} \pi_w(w_s)$ the string $\texttt{bwt}(s)$ also contains the first $k$ symbols of $s$ (which do not belong to any $w_s$). In the following we will ignore the presence of these $k$ symbols in $\texttt{bwt}(s)$.

[3] In its original formulation move-to-front was presented as a compression algorithm.

(see (9)). Every string $w_s$ is likely to contain only a few distinct symbols, but the symbols appearing in $\pi_w(w_s)$ are in general different from those in $\pi_{w'}(w'_s)$. The move-to-front recoding transforms both $\pi_w(w_s)$ and $\pi_{w'}(w'_s)$ into strings containing a large number of small integers. In other words, the move-to-front recoding transforms a string which is locally homogeneous into a string which is globally homogeneous. In Section 4 we will analyze to what extent move-to-front achieves this objective.

Summing up, the string $s' = \texttt{mtf}(\texttt{bwt}(s))$ has exactly the same length of the input string $s$. However, the regularity which is in $s$ is transformed into the presence of many small integers in $s'$. As an example, if $s$ is an English text $s'$ typically contains more than 50% 0's (see Table 1 in [Fenwick 1996b]). The BWT-based algorithms take advantage of this "skewness" of $s'$ to reduce its size (and this is where the actual compression is done). In the paper introducing the BWT, Burrows and Wheeler suggest to compress $s'$ using a zeroth order coder such as Huffman coding [Huffman 1952] or Arithmetic coding [Witten, Neal, and Cleary 1987]. These algorithms encode a symbol which appears $n_i$ times out of $n$ using roughly $-\log(n_i/n)$ bits and therefore achieve a compression ratio close to $H_0(s')$. Burrows and Wheeler also describe an improved algorithm which uses run-length encoding to compress the runs of 0's which typically appear in $s'$. More recent BWT-based algorithms (see for example [Fenwick 1996a; Seward 1997; Wheeler 1995; Wheeler 1997]) do not introduce new techniques for the compression of $s'$. Most of the efforts have been directed to improve the overall compression speed, or to refine the two basic tools (zeroth order encoding[4] and run-length encoding) introduced by Burrows and Wheeler for the compression of $s'$.

In section 4 we analyze the algorithm which compresses $s'$ using a zeroth order coder, and in section 5 we consider the effect of using run-length encoding. To ensure the widest possible applicability of our results, we have tried to make as few assumptions as possible on the algorithms used to compress $s'$.

In the following we denote by `Order0` a generic order-0 coder. We make no assumptions on its inner working, we only assume that its compression ratio is close to the zeroth order entropy of the input string. More precisely, we assume that there exists a constant $\mu$ such that for any string $s$

$$\texttt{Order0}(s) \leq |s|H_0(s) + \mu|s|. \tag{11}$$

It is well known that for static Huffman coding (11) holds with $\mu = 1$. For the dynamic Huffman coding algorithm described in [Vitter 1987] (11) holds with $\mu = 2$. Arithmetic coding routines exist in different flavors (see for example [Howard and Vitter 1992b; Moffat, Neal, and Witten 1995; Witten, Neal, and Cleary 1987]) each one with a different balance between storage requirements, compression, and speed. In [Howard and Vitter 1992a] Howard and Vitter carry out a comprehensive analysis of arithmetic coding which tells us that a simple arithmetic coder, such as the one described in [Witten, Neal, and Cleary 1987], satisfies (11) with $\mu \approx 10^{-2}$.

## 4. ANALYSIS OF THE ALGORITHM BW0

In this section we analyze the algorithm `BW0` = `bwt` + `mtf` + `Order0`, that is, the algorithm in which the output of the BWT is processed by move-to-front followed by a zeroth order coder. The output of this algorithm is therefore $\texttt{BW0}(s) = \texttt{Order0}(\texttt{mtf}(\texttt{bwt}(s)))$. We prove that `BW0` compression ratio can be bounded in terms of the $k$-th order entropy for any $k \geq 0$. The proof is based on the following result on the behavior of the move-to-front transformation.

THEOREM 4.1. *Let $s$ be any string over the alphabet $\{\alpha_1, \ldots, \alpha_h\}$, and $\hat{s} = \texttt{mtf}(s)$. For any partition $s = s_1 \cdots s_t$ we have*

$$|\hat{s}|H_0(\hat{s}) \leq 8\left[\sum_{i=1}^{t} |s_i|H_0(s_i)\right] + \frac{2}{25}|s| + t(2h\log h + 9). \tag{12}$$

[4]Some of the efforts have been directed to avoid the use of arithmetic coding which is covered by quite a few patents.

Let us comment on the above theorem. Note that for any partition $s = s_1 \cdots s_t$ we have $|s|H_0(s) \geq \sum_i |s_i|H_0(s_i)$. If the strings $s_i$'s have similar statistics then $\sum_i |s_i|H_0(s_i)$ will be close to $|s|H_0(s)$, but in general we can have a large gap between the two terms (consider for example the extreme case $s_1 = a^n$, $s_2 = b^n$ for which we have $|s_1 s_2|H_0(s_1 s_2) = 2n$ and $H_0(s_1) = H_0(s_2) = 0$).

Theorem 4.1 establishes that if we use move-to-front the entropy of the resulting string cannot be too far from $\sum_i |s_i|H_0(s_i)$. We do not claim that the bound in (12) is tight (in fact, we believe it is not). However, a bound of the form $|\hat{s}|H_0(\hat{s}) \leq \lambda[\sum_i |s_i|H_0(s_i)] + c$ cannot hold. For the entropy $H_0$ this is obvious (consider again $s_1 = a^n$, $s_2 = b^n$), the following example shows that this is true also for the modified entropy $H_0^*$.

*Example* 6. Let $s = (\mathtt{ab})^k \mathtt{b}^{k^2}$, $s_1 = (\mathtt{ab})^k$, $s_2 = \mathtt{a}^{k^2}$. We have $|s_1|H_0^*(s_1) + |s_2|H_0^*(s_2) = 2k + \lfloor \log k^2 \rfloor + 1$. Let $\hat{s} = \mathtt{mtf}(s) = 01^{2k}0^{k^2-1}$. Since

$$\begin{aligned} |\hat{s}|H_0^*(\hat{s}) &= |\hat{s}|H_0(\hat{s}) \\ &= 2k\left(\log\frac{2k+k^2}{2k}\right) + k^2\left(\log\frac{2k+k^2}{k^2}\right) \\ &\geq k\log k, \end{aligned}$$

we have $\lim_{k\to\infty} |\hat{s}|H_0^*(\hat{s})/(|s_1|H_0^*(s_1) + |s_2|H_0^*(s_2)) = +\infty$.

As an immediate corollary of Theorem 4.1 we get the following result on BWO.

Theorem 4.2. *For any string $s$ over $\mathcal{A} = \{\alpha_1, \ldots, \alpha_h\}$ and $k \geq 0$ we have*

$$\mathtt{BWO}(s) \leq 8|s|H_k(s) + \left(\mu + \frac{2}{25}\right)|s| + h^k(2h\log h + 9), \tag{13}$$

*where $\mu$ is defined in* (11).

Proof. Let $\hat{s} = \mathtt{mtf}(\mathtt{bwt}(s))$. By (11) we have

$$\mathtt{BWO}(s) = \mathtt{Order0}(\hat{s}) \leq |\hat{s}|H_0(\hat{s}) + \mu|s|.$$

By the properties of the BWT we know that there exists $t \leq h^k$ and a partition $s_1 \cdots s_t$ of $\mathtt{bwt}(s)$ such that $|s|H_k(s) = \sum_{i=1}^t |s_i|H_0(s_i)$. The thesis follows by Theorem 4.1. □

Before proving Theorem 4.1 we need to establish several auxiliary results. The following lemma bounds $|s|H_0(s)$ in terms of $\sum_i |s_i|H_0(s_i)$ for a string $s$ over the alphabet $\{0,1\}$ assuming that all strings $s_i$'s contain more 0's than 1's.

Lemma 4.3. *For $i = 1, \ldots, t$, let $s_i$ be a string over the alphabet $\{0,1\}$. Let $m_i$ denote the number of 1's in $s_i$ and. If, for $i = 1, \ldots, t$, $m_i \leq |s_i|/2$, then*

$$|s_1 \cdots s_t|H_0(s_1 \cdots s_t) \leq 3\sum_{i=1}^t |s_i|H_0(s_i) + \frac{1}{40}|s_1 \cdots s_t|.$$

Proof. Let $h(x) = -x\log(x) - (1-x)\log(1-x)$, $s = s_1 \cdots s_t$, and $t = (m_1 + \cdots + m_t)/|s|$. Since $H_0(s_i) = h(m_i/|s_i|)$, our thesis is equivalent to

$$h(t) \leq 3\sum_{i=1}^t \frac{|s_i|}{|s|} h\left(\frac{m_i}{|s_i|}\right) + \frac{1}{40}.$$

Since, for $0 \le x \le 1/2$, we have $h(x) \ge 2x$ we can write

$$\sum_{i=1}^{t} \frac{|s_i|}{|s|}\, h\left(\frac{m_i}{|s_i|}\right) \ge 2\frac{m_1 + \cdots + m_t}{|s|} = 2t$$

and our thesis becomes

$$h(t) \le 6t + \frac{1}{40}.$$

Elementary calculus shows that the function $6t - h(t) + \frac{1}{40}$ has a positive minimum (achieved for $t = 1/65$) and the lemma follows. □

To prove the next lemma we need some additional notation. For $x, y \ge 0$ define

$$G(x,y) = -x \log \frac{x}{x+y} - y \log \frac{y}{x+y}. \tag{14}$$

In addition, let $G(x,0) = G(0,y) = G(0,0) = 0$. The following properties are easily verified

$$G(\lambda x, \lambda y) = \lambda G(x,y), \tag{15}$$

$$0 \le G(x,y) \le x + y, \tag{16}$$

$$G(x, y+z) \le G(x+y, z) + G(x,y). \tag{17}$$

The following lemma bounds $|s_1 s_2| H_0(s_1 s_2)$ when $s_1, s_2$ are strings over the alphabet $\{0,1\}$ which are not homogeneous, that is, $s_1$ contains more 1's than 0's whereas $s_2$ contains more 0's than 1's.

LEMMA 4.4. *Let $s_1, s_2$ be strings over $\{0,1\}$. Let $x_1$ (resp. $y_1$) denote the number of 1's in $s_1$ (resp. $s_2$). Assume $x_1 > |s_1|/2$ and $y_1 \le |s_2|/2$. We have*

$$|s_1 s_2| H_0(s_1 s_2) \le 4.85|s_1| + |s_2| H_0(s_2) + \frac{1}{20}|s_1 s_2|.$$

PROOF. Let $x_0 = |s_1| - x_1$, $y_0 = |s_2| - y_1$ denote the number of 0's in $s_1$ and $s_2$ respectively. Expressing the entropy in terms of the function $G$, our thesis becomes

$$G(x_0 + y_0, x_1 + y_1) \le 4.85(x_1 + x_0) + G(y_0, y_1) + \frac{1}{20}(x_0 + y_0 + x_1 + y_1).$$

Let $r = y_1(x_0/y_0)$. Using (17) with $x = x_0 + y_0$, $y = y_1 + r$, $z = x_1 - r$ we get

$$G(x_0 + y_0, x_1 + y_1) \le G(x_0 + y_0 + y_1 + r, x_1 - r) + G(x_0 + y_0, y_1 + r).$$

By (15) and (16) we have

$$\begin{aligned} G(x_0 + y_0, y_1 + r) &= \left(1 + \frac{x_0}{y_0}\right) G(y_0, y_1) \\ &\le G(y_0, y_1) + x_0 \left(1 + \frac{y_1}{y_0}\right) \\ &= G(y_0, y_1) + x_0 + r. \end{aligned}$$

Hence, to complete the proof it suffices to prove that

$$G(x_0 + y_0 + y_1 + r, x_1 - r) \le 4.85(x_1 - r) + \frac{1}{20}(x_0 + y_0 + x_1 + y_1).$$

Expanding $G$ using (14), dividing by $x_1 - r$, and setting $t = (x_0 + y_0 + y_1 + r)/(x_1 - r)$ the previous inequality becomes

$$(1+t)\log(1+t) - t\log(t) \leq 4.85 + \frac{1+t}{20}.$$

Elementary calculus shows that the maximum of the function $(1+t)\log(1+t) - t\log(t) - (1+t)/20$ is $-\log(2^{1/20} - 1)$ which is less than 4.85 and the lemma follows. □

Lemmas 4.3 and 4.4 both deal with the simple case of strings over the alphabet $\{0, 1\}$. Now we need to face the general case of strings over any finite alphabet. In the following given a string $s$ over the alphabet $\{0, \ldots, h-1\}$ we denote by $s^{01}$ the string in which each nonzero symbol is replaced by 1. For example, $(102300213)^{01} = 101100111$.

LEMMA 4.5. *For any string $s$ let $\hat{s} = \texttt{mtf}(s)$. We have $|\hat{s}^{01}|H_0(\hat{s}^{01}) \leq 2|s|H_0(s)$.*

PROOF. For $i = 1, \ldots, h$, let $n_i$ denote the number of occurrences of the symbol $\alpha_i$ in $s$. Without loss of generality, we can assume that the most frequent symbol of $s$ is $\alpha_1$, that is, $n_1 \geq n_i$ for $i = 2, \ldots, h$. If $n_1 \leq |s|/2$ then

$$|s|H_0(s) = \sum_{i=1}^{h} n_i \log(|s|/n_i) \geq \sum_{i=1}^{h} n_i = |s|,$$

and the lemma follows since $|\hat{s}^{01}|H_0(\hat{s}^{01}) \leq |s|$.

Assume now $n_1 > |s|/2$. Let $r = n_2 + \cdots + n_h$ and $\beta = n_1/|s|$. We have

$$\sum_{i=2}^{h} n_i \log(|s|/n_i) = r\log|s| - \sum_{i=2}^{h} n_i \log n_i \geq r\log(|s|/r), \tag{18}$$

which implies

$$|s|H_0(s) = \sum_{i=1}^{h} n_i \log(|s|/n_i) \geq n_1 \log(|s|/n_1) + r\log(|s|/r) = -n_1\log\beta - r\log(1-\beta). \tag{19}$$

Let $m_0$ denote the number of 0's in $\hat{s}^{01}$. Note that, by definition of `mtf` encoding, the first symbol of $\hat{s}$ is 0; in addition, there is a 0 in $\hat{s}$ for each pair of identical consecutive symbols in $s$. Hence, the symbol $\alpha_1$ alone generates $n_1 - r$ 0's in $\hat{s}$, which implies $m_0 \geq n_1 - r$. Consider now the algorithm which encodes $\hat{s}^{01}$ using $-\log(\beta)$ bits for the symbol 0, and $-\log(1-\beta)$ bits for the symbol 1. Since the codeword lengths satisfy Kraft's inequality we have

$$\begin{aligned}
|\hat{s}^{01}|H_0(\hat{s}^{01}) &\leq m_0 \log\left(\frac{1}{\beta}\right) + (|s| - m_0)\log\left(\frac{1}{1-\beta}\right) \\
&= m_0 \log\left(\frac{1-\beta}{\beta}\right) + |s|\log\left(\frac{1}{1-\beta}\right) \\
&\leq (n_1 - r)\log\left(\frac{1-\beta}{\beta}\right) + (n_1 + r)\log\left(\frac{1}{1-\beta}\right) \\
&= -n_1\log\beta - 2r\log(1-\beta) + r\log\beta \\
&\leq 2[-n_1\log\beta - r\log(1-\beta)].
\end{aligned}$$

By (19) the last term is smaller than $2|s|H_0(s)$ and the lemma follows. □

LEMMA 4.6. *For $i = 0, \ldots, h-1$, let $m_i$ denote the number of occurrences of the symbol $i$ inside $\hat{s} = \texttt{mtf}(s)$. We have*

$$\sum_{i=0}^{h-1} m_i \log(i+1) \leq |s| H_0(s).$$

PROOF. The result can be proven repeating verbatim the proof of Theorem 1 in [Bentley, Sleator, Tarjan, and Wei 1986] with $f(x) = \log(x)$. Note that in [Bentley, Sleator, Tarjan, and Wei 1986] `mtf` ranks are encoded with the symbols $1, \ldots, h$. □

LEMMA 4.7. *For $i = 1, \ldots, t$ let $\hat{s}_i = \texttt{mtf}(s_i)$, and let $\tilde{s} = \hat{s}_1 \cdots \hat{s}_t$. We have*

$$|\tilde{s}| H_0(\tilde{s}) \leq |\tilde{s}^{01}| H_0(\tilde{s}^{01}) + 2 \sum_{i=1}^{t} |s_i| H_0(s_i).$$

PROOF. For $j = 0, \ldots, h-1$, let $m_j$ (resp. $m_j^{(i)}$) denote the number of occurrences of the symbol $j$ in $\tilde{s}$ (resp. $\hat{s}_i$), and let $\beta = m_0/|\tilde{s}|$. Consider the algorithm which encodes $\tilde{s}$ using $-\log(\beta)$ bits for the symbol 0, and $-\log(1-\beta) + 2\log(j+1)$ bits for the symbol $j$, $j = 1, \ldots, h-1$. Since the codeword lengths satisfy Kraft's inequality we have

$$\begin{aligned} |\tilde{s}| H_0(\tilde{s}) \;&\leq\; -m_0 \log(\beta) - (|\tilde{s}| - m_0) \log(1-\beta) + 2 \sum_{j=1}^{h-1} m_j \log(j+1) \\ &=\; |\tilde{s}^{01}| H_0(\tilde{s}^{01}) + 2 \sum_{i=1}^{t} \sum_{j=1}^{h-1} m_j^{(i)} \log(j+1). \end{aligned}$$

The thesis follows by Lemma 4.6. □

Note that Lemma 4.7 bounds the entropy of $\tilde{s} = \hat{s}_1 \cdots \hat{s}_t$ with $\hat{s}_i = \texttt{mtf}(s_i)$. Unfortunately, for the proof of Theorem 4.1 we need to bound the entropy of $\hat{s} = \texttt{mtf}(s) = \texttt{mtf}(s_1 \cdots s_t)$. The difference is a subtle one but cannot be ignored. When we compute $\texttt{mtf}(s_1 \cdots s_t)$ we encode $s_i$ ($i > 1$) using the recency list induced by the processing of $s_1 \cdots s_{i-1}$. This will produce an output different from $\texttt{mtf}(s_i)$ which uses, by definition, the recency list induced by $s_i$ (see the definition of move-to-front encoding in Section 3). Note however, that the difference in the initial status of the recency list influences only the encoding of the first occurrence of any given symbol in $s_i$. Hence, the encoding of $s_i$ in $\hat{s}$ differs from $\texttt{mtf}(s_i)$ in at most $h$ positions. We use this observation to prove the following lemma.

LEMMA 4.8. *For $i = 1, \ldots, t$ let $\hat{s}_i = \texttt{mtf}(s_i)$. Let $s = s_1 \cdots s_t$, $\tilde{s} = \hat{s}_1 \cdots \hat{s}_t$, and $\hat{s} = \texttt{mtf}(s)$. We have*

$$|\hat{s}| H_0(\hat{s}) \leq |\tilde{s}^{01}| H_0(\tilde{s}^{01}) + 2 \sum_{i=1}^{t} |s_i| H_0(s_i) + \frac{|s|}{300} + t(9 + 2h \log h).$$

PROOF. As we have already observed, the strings $\hat{s}$ and $\tilde{s}$ differ in at most $th$ positions. Hence, repeating the proof of Lemma 4.7 we get

$$|\hat{s}| H_0(\hat{s}) \leq |\hat{s}^{01}| H_0(\hat{s}^{01}) + 2 \sum_{i=1}^{t} |s_i| H_0(s_i) + 2th \log h.$$

We complete the proof by showing that

$$|\hat{s}^{01}| H_0(\hat{s}^{01}) - |\tilde{s}^{01}| H_0(\tilde{s}^{01}) - \frac{|s|}{300} \leq 9t. \tag{20}$$

We observe that, except for the first symbol of $\hat{s}$, each 0 in $\hat{s}$ corresponds to repetition of the same symbol in $s$. Hence, we can have a difference between $\hat{s}$ and $\tilde{s}$ only in the positions corresponding to the first symbol of each $s_i$ for $i > 1$. In these positions there is always a 0 in $\tilde{s}$ whereas there is a 0 in $\hat{s}$ only if the first symbol of $s_i$ is equal to the last symbol of $s_{i-1}$. As a result, $\tilde{s}$ contains $r$ more zeros than $\hat{s}$ with $r < t$. Let $n_0$ (resp. $n_1$) denote the number of 0's (resp. 1's) in $\hat{s}^{01}$. Using (14) and (17) we get

$$\begin{aligned} |\hat{s}^{01}|H_0(\hat{s}^{01}) - |\tilde{s}^{01}|H_0(\tilde{s}^{01}) - \frac{|s|}{300} &= G(n_0, n_1) - G(n_0 + r, n_1 - r) - \frac{n_0 + n_1}{300} \\ &\leq G(n_0, r) - \frac{n_0 + r}{300}. \end{aligned}$$

We prove (20) by showing that the last expression is bounded by $9r$. Using (14) and setting $t = n_0/r$ we get

$$\frac{G(n_0, r)}{r} - \frac{n_0 + r}{300r} = (1 + t)\log(1 + t) - t \log t - \frac{1 + t}{300}$$

Elementary calculus shows that the right-hand side is at most $-\log(2^{1/300} - 1)$ which is less than 9 and the lemma follows. □

Proof of Theorem 4.1. For $i = 1, \ldots, t$, let $\hat{s}_i = \texttt{mtf}(s_i)$ and $\tilde{s} = \hat{s}_1 \cdots \hat{s}_t$. By Lemma 4.8 we know that it suffices to prove that

$$|\tilde{s}^{01}|H_0(\tilde{s}^{01}) \leq 6\left[\sum_{i=1}^{t} |s_i|H_0(s_i)\right] + \frac{3}{40}|\hat{s}|. \tag{21}$$

Assume first that in each $\hat{s}_i^{01}$ the number of 0's is at least as large as the number of 1's. By Lemma 4.3 we get

$$|\tilde{s}^{01}|H_0(\tilde{s}^{01}) \leq 3\left[\sum_{i=1}^{t} |\hat{s}_i^{01}|H_0(\hat{s}_i^{01})\right] + \frac{1}{40}|\hat{s}|$$

and (21) follows by Lemma 4.5.

Consider now the general case in which some of the $\hat{s}_i^{01}$ contains more 1's than 0's. We can assume that these are $\hat{s}_1^{01}, \hat{s}_2^{01}, \ldots, \hat{s}_k^{01}$. By Lemmas 4.4 and 4.3 we have

$$\begin{aligned} |\tilde{s}^{01}|H_0(\tilde{s}^{01}) &\leq 4.85|\hat{s}_1^{01} \cdots \hat{s}_k^{01}| + |\hat{s}_{k+1}^{01} \cdots \hat{s}_t^{01}|H_0(\hat{s}_{k+1}^{01} \cdots \hat{s}_t^{01}) + \frac{1}{20}|\tilde{s}^{01}| \\ &\leq 4.85\left[\sum_{i=1}^{k} |s_i|\right] + 6\left[\sum_{i=k+1}^{t} |s_i|H_0(s_i)\right] + \frac{3}{40}|\tilde{s}|. \end{aligned}$$

For $j = 1, \ldots, k$ let $m_j$ denote the number of occurrences of the most frequent symbol in $s_j$. The hypothesis on $\hat{s}_j^{01}$ implies $m_j \leq (3/4)|s_j|$, hence, using (18) we get

$$|s_j|H_0(s_j) \geq m_j \log\left(\frac{|s_j|}{m_j}\right) + (|s_j| - m_j)\log\left(\frac{|s_j|}{|s_j| - m_j}\right) \geq \gamma|s_j|, \tag{22}$$

where $\gamma = -[(3/4)\log(3/4) + (1/4)\log(1/4)]$. Since $(4.85/\gamma) \leq 6$, by (22) we have $4.85|s_j| \leq 6|s_j|H_0(s_j)$ and the thesis follows. □

## 5. ANALYSIS OF THE ALGORITHM BW0$_{RL}$

We have observed in Section 3 that when we process the output of the BWT with move-to-front encoding we get a string which usually contains long sequences of zeroes. In this section we analyze the effects of

compressing these sequences using run-length encoding. As we will see, for the resulting algorithm, that we call $\texttt{BWO}_{RL}$, we are able to prove bounds which are better than the ones given in Section 4 for the algorithm BWO.

We use the following notation. Given a string $s$ over $\{\alpha_1, \ldots, \alpha_h\}$, let $\hat{s} = \texttt{mtf}(s)$. We know that $\hat{s}$ is defined over the alphabet $\{0, 1, \ldots, h-1\}$. Let $\mathbf{0}, \mathbf{1}$ denote two symbols not belonging to any of the above alphabets. For $m \geq 1$ let $B(m)$ denote the number $m+1$ written in binary using $\mathbf{0}, \mathbf{1}$ and discarding the most significant bit. That is

$$B(1) = \mathbf{0}, \quad B(2) = \mathbf{1}, \quad B(3) = \mathbf{00}, \quad B(4) = \mathbf{01}, \quad \ldots.$$

We define $\texttt{rle}(\hat{s})$ (the run-length encoding of $\hat{s}$) as the string obtained from $\hat{s}$ by replacing each (maximal) run $0^m$ with $B(m)$. For example, if $\hat{s} = 110022013000$, then $\texttt{rle}(\hat{s}) = 11\mathbf{1}2\mathbf{2}0\mathbf{3}\mathbf{00}$. Clearly, given $\texttt{rle}(\hat{s})$ we can retrieve $\hat{s}$ since, $\mathbf{0}$ and $\mathbf{1}$ are only used to encode runs of 0's. Note that $|B(m)| = \lfloor \log(m+1) \rfloor \leq m$ which implies $|\texttt{rle}(\hat{s})| \leq |\hat{s}|$.

We define the algorithm $\texttt{BWO}_{RL} = \texttt{bwt} + \texttt{mtf} + \texttt{rle} + \texttt{Order0}$. The output of this algorithm on input $s$ is therefore $\texttt{BWO}_{RL}(s) = \texttt{Order0}(\texttt{rle}(\texttt{mtf}(\texttt{bwt}(s))))$. Although the run-length encoding scheme we have just described has not been used in any actual implementation of a BWT-based algorithm, we claim that our analysis is of general interest. In fact, the only property of our scheme that we use in our proofs is that the sequence $0^m$ is replaced by a binary string of length at most $\log(m+1)$.

The main result of this section is the following theorem which bounds the output size of $\texttt{BWO}_{RL}$ in terms of the modified $k$-th order entropy.

THEOREM 5.1. *For any $k \geq 0$ there exists a constant $g_k$ such that for any string $s$ we have*

$$\texttt{BWO}_{RL}(s) \leq (5 + 3\mu)|s|H_k^*(s) + g_k$$

*where $\mu$ is defined in* (11).

The proof of the above theorem is completely different from the proof of Theorem 4.1 and it is based on the concept of *local $\lambda$-optimality*.

DEFINITION 5.2. *A compression algorithm* A *is* locally $\lambda$-optimal *if for all $t > 0$ there exists a constant $c_t$ such that for each partition $s_1 s_2 \cdots s_t$ of the string $s$ we have*

$$\texttt{A}(s) \leq \lambda \left[ \sum_{i=1}^{t} |s_i| H_0^*(s_i) \right] + c_t.$$

□

From the discussion in Section 3 we know that if A is locally $\lambda$-optimal then, for any $k \geq 0$ the output size of $\texttt{bwt}+\texttt{A}$ is bounded by $\lambda|s|H_k^*(s)+g_k$. Note that many algorithms which work well in practice—Huffman coding, LZ78, PPM, to name a few—are not locally optimal. Suppose for example $t = 2$ and $s = s_1 s_2$. During the processing of $s_1$ these algorithms build internally a model of the input (the dictionary in LZ78, frequency counts in Huffman coding and PPM) which influences the processing of $s_2$. It is possible to find instances in which this model is completely misleading so that the overall output size is much greater than $|s_1|H_0^*(s_1) + |s_2|H_0^*(s_2)$.[5]

The proof of Theorem 5.1 is obtained by showing that $\texttt{mtf} + \texttt{rle} + \texttt{Order0}$ is locally $\lambda$-optimal with $\lambda = 5 + 3\mu$. Note that $\texttt{mtf} + \texttt{Order0}$ *is not* locally optimal, that is, run-length encoding is essential for achieving local optimality. The following lemma, which can be easily proven by induction, provides a sufficient condition for local optimality.

[5]We emphasize that the notion of local $\lambda$-optimality is simply a useful tool for our analysis. We do not claim that locally optimal algorithms are in any sense superior to other compressors.

LEMMA 5.3. *If there exist three constants $\lambda, v, w$ such that for any string $s$ $\mathtt{A}(s) \leq \lambda H_0^*(s) + v$, and for any partition $s = s_1 s_2$ we have*

$$\mathtt{A}(s) \leq \mathtt{A}(s_1) + \mathtt{A}(s_2) + w, \tag{23}$$

*then the algorithm* A *is locally $\lambda$-optimal.* □

Unfortunately, we cannot apply the above lemma directly to $\mathtt{mtf} + \mathtt{rle} + \mathtt{Order0}$. The reason is that we know nothing of the inner working of Order0 and we are only assuming that it satisfies (11). For this reason we introduce the auxiliary compression algorithm Pc (for prefix code) defined as follows. Recall that the output of rle is a string over the alphabet $\{\mathbf{0}, \mathbf{1}, 1, 2, \ldots, h-1\}$. For such strings the algorithm Pc works as follows. The symbols **0** and **1** are coded using two bits (10 for **0**, 11 for **1**). The symbol $i$, $i = 1, 2, \ldots, h-1$, is coded in $1 + 2\lfloor \log(i+1) \rfloor$ bits using a prefix code for the integer $i+1$: $\lfloor \log(i+1) \rfloor$ 0's followed by the binary representation of $i+1$ which takes $1 + \lfloor \log(i+1) \rfloor$ bits, the first one being a 1. It is straightforward to verify that this is a instantaneous code, that is, no codeword is a prefix of any other codeword.

For any ordering $\pi$ of the alphabet $\mathcal{A}$ let $\mathtt{A}_\pi = \mathtt{mtf}_\pi + \mathtt{rle} + \mathtt{Pc}$. For any string $s$, we denote with $\pi(s)$ the ordering which *maximizes* the output size of $\mathtt{A}_\pi$. That is,

$$\mathtt{A}_{\pi(s)}(s) = \max_\pi \mathtt{A}_\pi(s). \tag{24}$$

For any string $s$ we define $\mathtt{mtf}^* = \mathtt{mtf}_{\pi(s)}$ and $\mathtt{A}^* = \mathtt{A}_{\pi(s)} = \mathtt{mtf}^* + \mathtt{rle} + \mathtt{Pc}$. Note that, for every string $s$, $\mathtt{A}^*$ first computes the worst case ordering $\pi(s)$, then uses it as the initial ordering for move-to-front encoding.

The following lemmas establish some auxiliary results on the behavior of the algorithms $\mathtt{mtf}^*$ and $\mathtt{A}^*$.

LEMMA 5.4. *Let $\pi$ denote any ordering of the alphabet $\mathcal{A}$. For $i = 1, \ldots, h-1$, let $m_i$ denote the number of occurrences of the symbol $i$ inside $\mathtt{mtf}_\pi(s)$. We have*

$$\sum_{i=1}^{h-1} m_i \log(i+1) \leq |s| H_0(s) + h \log h.$$

PROOF. The thesis follows by Lemma 4.6 observing that $\mathtt{mtf}(s)$ and $\mathtt{mtf}_\pi(s)$ differ in at most $h$ positions. □

LEMMA 5.5. *Let $\mathtt{A}^* = \mathtt{mtf}^* + \mathtt{rle} + \mathtt{Pc}$. For any string $s$ we have*

$$\mathtt{A}^*(s) \leq 5|s| H_0^*(s) + 2h \log h + 2 \log e - 1.$$

PROOF. Let $s' = \mathtt{mtf}^*(s)$, $s'' = \mathtt{rle}(s')$. For $i = 0, \ldots, h-1$, let $m_i$ denote the number of occurrences of the symbol $i$ in $s'$. By construction, for $i > 0$, $s''$ still contains $m_i$ occurrences of the symbol $i$, whereas the $m_0$ 0's in $s'$ are transformed into sequences of **0**'s and **1**'s. Let $z_0$ denote the number of occurrences of the symbols **0** and **1** in $s''$ (so that $|s''| = |s'| - m_0 + z_0$). Note that, since $|B(i)| = \lfloor \log(i+1) \rfloor \leq i$, we have $z_0 \leq m_0$. By construction we have

$$\begin{aligned} \mathtt{A}^*(s) = \mathtt{Pc}(s'') &= 2z_0 + \sum_{i=1}^{h-1} m_i (1 + 2\log(i+1)) \\ &\leq z_0 + |s''| + 2\left\lceil \sum_{i=1}^{h-1} m_i \log(i+1) \right\rceil, \end{aligned} \tag{25}$$

that, using Lemma 5.4, becomes

$$\mathtt{A}^*(s) \leq z_0 + |s''| + 2|s| H_0(s) + 2h \log h. \tag{26}$$

For $i = 1, \ldots, h$, let $n_i$ denote the number of occurrences of the symbol $\alpha_i$ in $s$. Without loss of generality, we can assume $n_1 \geq n_i$ for $i = 2, \ldots, h$. Let $n = |s| = |s'|$ and $r = n_2 + \cdots + n_h$. To prove the lemma we need to analyze several cases.

*Case 1.*. $r \geq n/3$.
Using (18) we get

$$|s|H_0(s) \geq n_1 \log(n/n_1) + r \log(n/r) \geq \gamma' n$$

where $\gamma' = -[(2/3)\log(2/3) + (1/3)\log(1/3)]$. Since $\gamma' > 2/3$ and $z_0 + |s''| \leq 2n$, from (26) we get

$$\mathtt{A}^*(s) \leq 2n + 2|s|H_0(s) + 2h \log h < 5|s|H_0(s) + 2h \log h.$$

*Case 2.*. $1 < r < n/3$.
This is the most complex case. We start by observing that the $n_1$ occurrences of $\alpha_1$ in $s$ generate at least $n_1 - (r+1)$ 0's in $s' = \mathtt{mtf}^*(s)$. Using our notation, this translates to

$$m_0 \geq n_1 - (r+1), \qquad \text{or, equivalently,} \qquad m_1 + m_2 + \cdots m_{h-1} \leq 2r + 1. \tag{27}$$

The $m_0$ 0's in $s'$ are translated into $z_0$ $\mathbf{0}$'s and $\mathbf{1}$'s in $s'' = \mathtt{rle}(s')$. We now want to bound $z_0$ in terms of $r$ and $n$. Assume $s'$ contains $g$ sequences of 0's of length $l_1, \ldots, l_g$. Let $p = \max_i l_i$. For $j = 1, \ldots, p$ let $q_j$ denote the number of sequences of length $j$ (therefore $\sum_{j=1}^{p} q_j = g$). We have

$$z_0 = \sum_{i=1}^{g} \lfloor \log(1 + l_i) \rfloor \leq \sum_{j=1}^{p} q_j \log(1+j) = g \left[ \sum_{j=1}^{p} \frac{q_j}{g} \log(1+j) \right].$$

Since $\log(1+x)$ is a concave function, by Jensen's inequality we get

$$z_0 \leq g \log\left( \sum_{j=1}^{p} \frac{q_j}{g}(1+j) \right) = g \log\left( \frac{g + m_0}{g} \right).$$

Since each one of the $g$ sequences of 0's in $s'$ is terminated either by a nonzero symbol or by the end of the string, we have $g + m_0 \leq n + 1$ which implies $z_0 \leq g \log((n+1)/g)$. The function $f(x) = x \log((n+1)/x)$ increases for $x \leq (n+1)/e$. It is not difficult to see that $g \leq r + 1$, and our hypothesis implies $r + 1 \leq (n+1)/e$. Using elementary calculus we get

$$\begin{aligned} z_0 \;&\leq\; (r+1) \log\left( \frac{n+1}{r+1} \right), \\ &\leq\; (r+1) \log\left( \frac{n}{r+1} \right) + \log e. \end{aligned} \tag{28}$$

Combining the last inequality with (25), (27) and Lemma 5.4 we get

$$\begin{aligned} \mathtt{A}^*(s) \;&\leq\; 2z_0 + \left[ \sum_{i=1}^{h-1} m_i \right] + 2 \left[ \sum_{i=1}^{h-1} m_i \log(i+1) \right], \\ &\leq\; 2(r+1)\left[ \log(n/(r+1)) + 1 \right] + 2\log e - 1 + 2|s|H_0(s) + 2h \log h. \end{aligned}$$

Hence, to complete the proof we need to show that

$$|s|H_0(s) \geq (2/3)(r+1)\left[ \log(n/(r+1)) + 1 \right]. \tag{29}$$

Using (18) we have

$$|s|H_0(s) \;\geq\; n_1 \log(n/n_1) + r \log(n/r)$$

$$\begin{aligned} &\geq r \log \left(1 + \frac{r}{n_1}\right)^{\frac{n_1}{r}} + r \log(n/r) \\ &\geq r[1 + \log(n/r)]; \end{aligned} \tag{30}$$

where the last inequality holds since $(1 + 1/t)^t \geq 2$ for $t \geq 1$. Being $r \geq 2$, we have $r \geq (2/3)(r + 1)$. Hence

$$|s|H_0(s) \geq (2/3)(r + 1)[\log(n/r) + 1] \geq (2/3)(r + 1)[\log(n/(r + 1)) + 1]$$

as claimed.

*Case 3.*. $r = 1$.
The input string consists of $n - 1$ copies of $\alpha_1$ plus a different symbol. It is easy to see that in the worst case we have $m_1 = 1$ and $m_{h-1} = 2$. From (25) and (28) we get

$$\mathtt{A}^*(s) \leq 4 \log(n/2) + 2 \log e + 4 \log h + 5 = 4 \log n + 2 \log e + 4 \log h + 1$$

The thesis follows since by (30) we have $5|s|H_0(s) \geq 5 \log n + 5$.

*Case 4.*. $r = 0$.
The input string consists of $n$ copies of the same symbol. The thesis follows since $\mathtt{A}^*(s) = 1 + 2 \log h + 2\lfloor \log n \rfloor$ and $|s|H_0^*(s) = 1 + \lfloor \log n \rfloor$.

□

COROLLARY 5.6. *Let $\pi$ denote any ordering of the alphabet $\mathcal{A}$. For any string $s$, we have*

$$|\mathtt{rle}(\mathtt{mtf}_\pi(s))| \leq 3|s|H_0^*(s) + \log e.$$

PROOF. The proof of the corollary is essentially contained in the proof of Lemma 5.5 (the fact that we are now considering an arbitrary ordering $\pi$ instead of the worst case ordering $\pi(s)$ does not alter the substance of the reasoning). Let $s'' = \mathtt{rle}(\mathtt{mtf}_\pi(s))$ and let any other symbol be defined as in the proof of Lemma 5.5. We show how the argument goes for the case when $1 < r < n/3$ which is the most complex one. By (27) and (28), we have

$$\begin{aligned} |s''| &= z_0 + m_1 + m_2 + \cdots + m_{h-1} \\ &\leq (r + 1) \log\left(\frac{n}{r + 1}\right) + \log e + 2r + 1 \\ &\leq 2(r + 1)\,[\log(n/(r + 1)) + 1] + \log e. \end{aligned}$$

The thesis follows since by (29) this is less than $3|s|H_0(s) + \log e$. □

LEMMA 5.7. *The algorithm $\mathtt{A}^* = \mathtt{mtf}^* + \mathtt{rle} + \mathtt{Pc}$ is locally 5-optimal, that is for any string $s$ and any partition $s = s_1 \cdots s_t$ we have*

$$\mathtt{A}^*(s) \leq 5\left[\sum_{i=1}^{t} |s_i|H_0^*(s_i)\right] + c_t. \tag{31}$$

PROOF. By Lemmas 5.5 and 5.3 it suffices to prove that $\mathtt{A}^*$ satisfies (23). Let $s = s_1 s_2$, $\pi = \pi(s)$ (the worst ordering for $s$), and let $\tau$ denote the ordering of the recency list when $\mathtt{mtf}$ has processed the last symbol of $s_1$. Using this notation we have $\mathtt{mtf}^*(s) = \mathtt{mtf}_\pi(s_1) \cup \mathtt{mtf}_\tau(s_2)$. Let $\mathtt{A}_\pi = \mathtt{mtf}_\pi + \mathtt{rle} + \mathtt{Pc}$, $\mathtt{A}_\tau = \mathtt{mtf}_\tau + \mathtt{rle} + \mathtt{Pc}$. If the last symbol of $\mathtt{mtf}_\pi(s_1)$ or first symbol of $\mathtt{mtf}_\tau(s_2)$ are different from 0 we are done since, by (24), we have

$$\mathtt{A}^*(s_1) + \mathtt{A}^*(s_2) \geq \mathtt{A}_\pi(s_1) + \mathtt{A}_\tau(s_2) = \mathtt{A}^*(s).$$

Assume now $\mathtt{mtf}_\pi(s_1) = \hat{s}_1 0^i$, $\mathtt{mtf}_\tau(s_2) = 0^j \hat{s}_2$, where the last symbol of $\hat{s}_1$ and the first symbol of $\hat{s}_2$ are different from 0. We have

$$\begin{aligned}
\mathtt{A}^*(s_1) + \mathtt{A}^*(s_2) &\geq \mathtt{A}_\pi(s_1) + \mathtt{A}_\tau(s_2) \\
&= \mathtt{Pc}(\mathtt{rle}(\hat{s}_1 0^i)) + \mathtt{Pc}(\mathtt{rle}(0^j \hat{s}_2)) \\
&= \mathtt{Pc}(\mathtt{rle}(\hat{s}_1)) + \mathtt{Pc}(B(i)) + \mathtt{Pc}(B(j)) + \mathtt{Pc}(\mathtt{rle}(\hat{s}_2)) \\
&= \mathtt{Pc}(\mathtt{rle}(\hat{s}_1)) + 2(\lfloor \log(i+1) \rfloor + \lfloor \log(j+1) \rfloor) + \mathtt{Pc}(\mathtt{rle}(\hat{s}_2)) \\
&\geq \mathtt{Pc}(\mathtt{rle}(\hat{s}_1)) + \mathtt{Pc}(B(i+j)) + \mathtt{Pc}(\mathtt{rle}(\hat{s}_2)) \\
&= \mathtt{A}^*(s).
\end{aligned}$$

This completes the proof. □

We are now able to prove Theorem 5.1. As we have already pointed out, the proof is obtained by establishing the local optimality of $\mathtt{mtf} + \mathtt{rle} + \mathtt{Order0}$, which is the thesis of the following theorem.

THEOREM 5.8. *The algorithm* $\mathtt{A} = \mathtt{mtf} + \mathtt{rle} + \mathtt{Order0}$ *is locally* $(5 + 3\mu)$*-optimal.*

PROOF. For any string $s$, let $s' = \mathtt{rle}(\mathtt{mtf}(s))$. By (24), and the fact that Pc codewords satisfy Kraft's inequality we get

$$\begin{aligned}
\mathtt{A}^*(s) &= \mathtt{Pc}(\mathtt{rle}(\mathtt{mtf}^*(s))) \\
&\geq \mathtt{Pc}(\mathtt{rle}(\mathtt{mtf}(s))) \\
&\geq |s'| H_0(s').
\end{aligned}$$

Using (11) we can establish the following relationship between $\mathtt{A}(s)$ and $\mathtt{A}^*(s)$

$$\begin{aligned}
\mathtt{A}(s) &= \mathtt{Order0}(s') \\
&\leq |s'| H_0(s') + \mu |s'| \\
&\leq \mathtt{A}^*(s) + \mu |s'|. \qquad (32)
\end{aligned}$$

Let $s = s_1 s_2 \cdots s_t$ denote any partition of $s$. For $i = 1, \ldots, t-1$, let $\pi_i$ denote the ordering of the recency list when mtf has processed the last symbol of $s_i$. Finally, let $\hat{s}_1 = \mathtt{mtf}(s_1)$, and for $i = 2, \ldots, t$ let $\hat{s}_i = \mathtt{mtf}_{\pi_{i-1}}(s_i)$. We have $\mathtt{mtf}(s) = \hat{s}_1 \cup \hat{s}_2 \cup \cdots \cup \hat{s}_t$. Hence, using Lemma 5.6, we get

$$\begin{aligned}
|s'| &= |\mathtt{rle}(\hat{s}_1 \cup \hat{s}_2 \cup \cdots \cup \hat{s}_t)| \\
&\leq |\mathtt{rle}(\hat{s}_1)| + |\mathtt{rle}(\hat{s}_2)| + \cdots + |\mathtt{rle}(\hat{s}_t)| \\
&\leq 3[|s_1| H_0^*(s_1) + \cdots + |s_t| H_0^*(s_t)] + t \log e.
\end{aligned}$$

Combining the last inequality with (32) and (31) we get

$$\begin{aligned}
\mathtt{A}(s) &\leq \mathtt{A}^*(s) + \mu |s'| \\
&\leq 5 \left[ \sum_{i=1}^{t} |s_i| H_0^*(s_i) \right] + c_t + 3\mu \left[ \sum_{i=1}^{t} |s_i| H_0^*(s_i) \right] + t\mu \log e \\
&= (5 + 3\mu) \left[ \sum_{i=1}^{t} |s_i| H_0^*(s_i) \right] + c_t'.
\end{aligned}$$

This completes the proof. □

## 6. CONCLUDING REMARKS

In this paper we have analyzed two algorithms based on the BWT. We have bounded the output size of these algorithms in terms on the $k$-th order empirical entropy of the input string. Our results hold for every string without any probabilistic assumption on the input.

We believe that, with a more careful analysis, it is possible to reduce the size of the constants which appear in our bounds. However, we conjecture that such constants cannot be made equal to one, that is, the BWT-based algorithms are not optimal in the classical sense. In other words, our conjecture is that any worst case analysis of BWT-based algorithms must take into account small overheads which do not degrade the performance in practice even if they do not go to zero asymptotically.

In our analysis we concentrated on the algorithms which process the output of the BWT. Our major efforts have been aimed at bounding their compression ratio in terms of $\sum_{i=1}^{t} |s_i| H_0(s_i)$, where $s_1 \cdots s_t$ is a partition of $\mathtt{bwt}(s)$. We ignored a "second order" effect which could be considered in future works. In fact, we did not take into account that, in the partitions we are interested in, the substrings $s_i$ and $s_{i+1}$ are associated to contexts which in most cases are similar (being consecutive in the lexicographic order). Therefore, $s_i$ and $s_{i+1}$ have in general similar statistics which makes the move-to-front transformation more effective.

## References


Arnold, R. and Bell, T. The Canterbury corpus home page. `http://corpus.canterbury.ac.nz`.

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

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

Cleary, J. G. and Teahan, W. J. 1997. Unbounded length contexts for PPM. *The Computer Journal 40*, 2/3, 67–75.

Cormack, G. V. and Horspool, R. N. S. 1987. Data compression using dynamic Markov modelling. *The Computer Journal 30*, 6, 541–550.

Effros, M. 1999. Universal lossless source coding with the Burrows-Wheeler transform. In *DCC: Data Compression Conference* (1999). IEEE Computer Society TCC.

Fenwick, P. 1996a. Block sorting text compression — final report. Technical Report 130, Dept. of Computer Science, The University of Auckland New Zeland.

Fenwick, P. 1996b. The Burrows-Wheeler transform for block sorting text compression: principles and improvements. *The Computer Journal 39*, 9, 731–740.

Ferragina, P. and Manzini, G. 2000. Opportunistic data structures with applications. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science* (Redondo Beach, CA, 2000). 390–398.

Ferragina, P. and Manzini, G. 2001. An experimental study of an opportunistic index. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms* (Washington, D.C., 2001).

Howard, P. and Vitter, J. 1992a. Analysis of arithmetic coding for data compression. *Information Processing and Management 28*, 6, 749–763.

Howard, P. and Vitter, J. 1992b. Practical implementations of arithmetic coding. In *Image and Text Compression*, J. A. Storer, Ed., 85–112. Kluwer Academic.

Huffman, D. A. 1952. A method for the construction of minimim redundancy codes. *Proc.of the IRE 40*, (Sept.), 1098–1101.

Kosaraju, R. and Manzini, G. 1999. Compression of low entropy strings with Lempel–Ziv algorithms. *SIAM Journal on Computing 29*, 3, 893–911.

Larsson, N. J. 1998. The context trees of block sorting compression. In *Proceedings of the IEEE Data Compression Conference* (Mar.–Apr. 1998). 189–198.

Moffat, A. 1990. Implementing the PPM data compression scheme. *IEEE Transactions on Communications COM-38*, 1917–1921.

Moffat, A., Neal, R., and Witten, I. 1995. Arithmetic coding revisited. In *Data Compression Conference* (1995). IEEE Computer Society TCC, 202–211.

Nelson, M. 1996. Data compression with the Burrows-Wheeler transform. *Dr. Dobb's Journal of Software Tools 21*, 9, 46–50. `http://www.dogma.net/markn/articles/bwt/bwt.htm`.

Ryabko, B. Y. 1980. Data compression by means of a 'book stack'. *Prob.Inf.Transm 16*, 4, 265–269.

Sadakane, K. 1997. Text compression using recency rank with context and relation to context sorting, block sorting and PPM*. In *Proc. Int. Conference on Compression and Complexity of Sequences (SEQUENCES '97)* (1997). IEEE Computer Society TCC, 305–319.

Sadakane, K. 1998. On optimality of variants of the block sorting compression. In *Data Compression Conference* (Snowbird, Utah, 1998). IEEE Computer Society TCC.

Schindler, M. 1997. A fast block-sorting algorithm for lossless data compression. In *Data Compression Conference* (1997). IEEE Computer Society TCC. `http://www.compressconsult.com/szip/`.

Seward, J. 1997. The bzip2 home page. `http://sourceware.cygnus.com/bzip2/index.html`.

Vitter, J. 1987. Design and analysis of dynamic Huffman codes. *Journal of the ACM 34*, 4 (Oct.), 825–845.

Wheeler, D. 1995. An implementation of block coding. Computer Laboratory, Cambridge University, UK, `ftp://ftp.cl.cam.ac.uk/users/djw3/bred.ps`.

Wheeler, D. 1997. Upgrading bred with multiples tables. Computer Laboratory, Cambridge University, UK, `ftp://ftp.cl.cam.ac.uk/users/djw3/bred3.ps`.

Witten, I., Neal, R., and Cleary, J. 1987. Arithmetic coding for data compression. *Communications of the ACM 30*, 6 (June), 520–540.