SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2002; **32**:99–111 (DOI: 10.1002/spe.426)

Second step algorithms in the Burrows–Wheeler compression algorithm

Sebastian Deorowicz*,†

Institute of Computer Science, Silesian University of Technology, Akademicka 16, Gliwice 44-100, Poland

SUMMARY

In this paper we focus our attention on the second step algorithms of the Burrows–Wheeler compression algorithm, which in the original version is the Move To Front transform. We discuss many of its replacements presented so far, and compare compression results obtained using these replacements. Then we propose a new algorithm that yields a better compression ratio than the previous algorithms. Copyright © 2001 John Wiley & Sons, Ltd.

KEY WORDS: data compression; lossless compression; Burrows–Wheeler transform; block-sorting; List Update Problem; Calgary Corpus

INTRODUCTION

In 1994, Burrows and Wheeler developed a new data compression algorithm [1], based on the Burrows–Wheeler transform introduced in the same paper. The high compression efficiency of this algorithm has stimulated a lot of research. Many of its modifications presented so far are described and discussed in [2].

In the first part of this paper we describe the Burrows–Wheeler compression algorithm (BWCA). Later we focus our attention on the second step of this algorithm (Move To Front in the original paper [1]). We discuss recent suggestions of changing the BWCA and propose a new algorithm. Next we compare the efficiency of different second step algorithms. Finally we compare the new compression algorithm with other known methods.

*Correspondence to: Sebastian Deorowicz, Institute of Computer Science, Silesian University of Technology, Akademicka 16, Gliwice 44-100, Poland.
†E-mail: sdeor@star.iinf.polsl.gliwice.pl

Contract/grant sponsor: Silesian University of Technology; contract/grant number: Bk-210/RAu2/2000

Copyright © 2001 John Wiley & Sons, Ltd.

Received 22 May 2001
Revised 26 July 2001
Accepted 3 September 2001

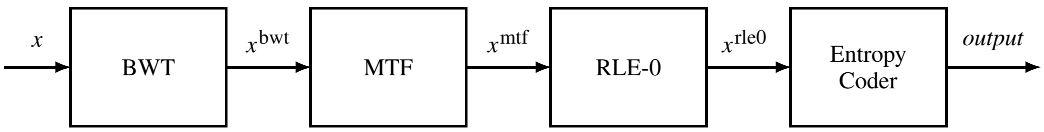

Figure 1. The Burrows–Wheeler compression algorithm.

THE BURROWS–WHEELER COMPRESSION ALGORITHM

In order to describe the BWCA we have to introduce some definitions and notations. Let $x = x_1 x_2 \ldots x_n$ be a *sequence*. The *length* of the sequence—denoted by n—is the number of elements in the sequence x. Every element x_i of the sequence belongs to a finite ordered set $\mathcal{A} = \{a_0, a_1, \ldots, a_{k-1}\}$ which we call an *alphabet*. The *size* of the alphabet, k, is the number of elements belonging to it. We call the elements of the alphabet *symbols* or *characters*. If $x = uvw$ for some possibly empty sequences u, v and w, then u is a *prefix*, v a *component*, and w a *suffix* of x. If a component is non-empty and consists of identical symbols then we call it a *run*.

A finite memory Context Tree source [3] ω is defined by a set $\mathcal{S}$ of contexts s (sequences over the alphabet $\mathcal{A}$) of length not greater than the maximum order D ($|s| \le D$). The set S of contexts should be complete, which means that for every possible sequence x there exists a context $s \in \mathcal{S}$, being a suffix of x. There is exactly one such context, called a proper context. A conditional probability distribution $\{\Theta_s, s \in \mathcal{S}\} = \{\{\theta(a|s), a \in \mathcal{A}\}, s \in \mathcal{S}\}$ is related with each context $s \in \mathcal{S}$.

The algorithm

The scheme of the BWCA is presented in Figure 1. It is composed of four stages. The first step performs the Burrows–Wheeler transform (BWT). The second step implements the Move To Front (MTF) transform or another algorithm converting the output of the BWT (sequence x^{bwt}) to a form that can be better compressed by an entropy coder. The third step is a specialized version of a Run Length Encoding (RLE) called RLE-0 (in some versions of the BWCA this step does not occur). The last step is an entropy coder, which typically is an arithmetic coder. A more precise description of these algorithms can be found in [2].

For completeness, we should note that some early versions of the BWCA used another kind of RLE before the BWT step to speed up the sorting procedure. Today, however, fast ways to compute the BWT are known and it is not necessary to apply this preliminary step. For this reason we will not consider this kind of RLE further in the paper.

An example of the BWT

Now we show an example of the operation of the BWT that will be helpful for further discussion. Let $x = abracadabra\$$. The last character of the sequence x, \$, is called a *sentinel*. It is the last character

Copyright © 2001 John Wiley & Sons, Ltd.

in the alphabet and it appears exactly once in the sequence x. In fact, the sentinel is not part of the data that we want to compress, and is appended to the sequence before the BWT stage.

At the beginning of the BWT, we form a matrix $M(x)$ of cyclic shifts of x:

$$M(x) = \begin{bmatrix} a & b & r & a & c & a & d & a & b & r & a & \$ \\ b & r & a & c & a & d & a & b & r & a & \$ & a \\ r & a & c & a & d & a & b & r & a & \$ & a & b \\ a & c & a & d & a & b & r & a & \$ & a & b & r \\ c & a & d & a & b & r & a & \$ & a & b & r & a \\ a & d & a & b & r & a & \$ & a & b & r & a & c \\ d & a & b & r & a & \$ & a & b & r & a & c & a \\ a & b & r & a & \$ & a & b & r & a & c & a & d \\ b & r & a & \$ & a & b & r & a & c & a & d & a \\ r & a & \$ & a & b & r & a & c & a & d & a & b \\ a & \$ & a & b & r & a & c & a & d & a & b & r \\ \$ & a & b & r & a & c & a & d & a & b & r & a \end{bmatrix}$$

Then we sort the rows of $M(x)$ in lexicographic order, obtaining the matrix $\widetilde{M}(x)$:

$$\widetilde{M}(x) = \begin{bmatrix} \underline{a} & \underline{b} & \underline{r} & \underline{a} & \underline{c} & \underline{a} & \underline{d} & \underline{a} & \underline{b} & \underline{r} & \underline{a} & \underline{\$} \\ a & b & r & a & \$ & a & b & r & a & c & a & \mathbf{d} \\ a & c & a & d & a & b & r & a & \$ & a & b & \mathbf{r} \\ a & d & a & b & r & a & \$ & a & b & r & a & \mathbf{c} \\ a & \$ & a & b & r & a & c & a & d & a & b & \mathbf{r} \\ b & r & a & c & a & d & a & b & r & a & \$ & \mathbf{a} \\ b & r & a & \$ & a & b & r & a & c & a & d & \mathbf{a} \\ c & a & d & a & b & r & a & \$ & a & b & r & \mathbf{a} \\ d & a & b & r & a & \$ & a & b & r & a & c & \mathbf{a} \\ r & a & c & a & d & a & b & r & a & \$ & a & \mathbf{b} \\ r & a & \$ & a & b & r & a & c & a & d & a & \mathbf{b} \\ \$ & a & b & r & a & c & a & d & a & b & r & \mathbf{a} \end{bmatrix}$$

The result of BWT is a sequence x^{bwt} that appears in the last column of the matrix $\widetilde{M}(x)$, plus the index, $R(x)$, of the row of $\widetilde{M}(x)$ that contains the original sequence x. In our example, we get $x^{\text{bwt}} = \$drcraaaabba$ and $R(x) = 1$.

The sentinel is employed because sorting the cyclic shifts is a computationaly intensive problem. For a sequence ended by a sentinel it reduces to the problem of sorting suffixes. The latter problem can be solved by building a suffix tree and searching it in lexicographic order, which can be done very effectively. The other advantage of using the sentinel is better relation of the BWT to Context Tree sources.

BWT relation to Context Tree sources

Balkenhol and Kurtz [4] showed that if we assume that the sequence x is generated by a Context Tree source, then the BWT groups together the similar contexts. This means that the symbols appearing in

Copyright © 2001 John Wiley & Sons, Ltd.

a specified context $s \in \mathcal{S}$ are grouped together in the last column of the consecutive rows of the matrix $\widetilde{M}(x)$. Furthermore, the probability distribution in a component of the sequence x^{bwt}, related to one context s (CT component), does not change. The contexts contained in the set $\mathcal{S}$ appear in $\widetilde{M}(x)$ in the lexicographic order, so the sequence x^{bwt} is a concatenation of the CT components, each of them corresponding to one of the leaves in the Context Tree source model.

PREVIOUS WORKS ON THE SECOND STEP ALGORITHMS

As was mentioned above, the sequence x^{bwt} is a concatenation of components corresponding to separate contexts. Unfortunately, there is no information on where exactly each such CT component starts in the sequence x^{bwt}. (This is the main difference between the BWCA and the PPM algorithms [30].) However, on monitoring the probability distribution in the sequence x^{bwt} we can try to uncover some of this information [5,6]. Surely the sequence x^{bwt} is a permutation of x. To exploit the properties of the sequence x^{bwt} we have to transform its local structure into a global structure in some way. Alternatively, we have to find a way to rapidly adapt to a changing probability distribution in the sequence x^{bwt} without information on where the CT components start.

Several algorithms have been proposed to solve this problem, some of which are based on the observation that the problem is similar to the List Update Problem (LUP) [7]. (A review of many algorithms solving the LUP was recently presented by Bachrach and El-Yaniv [8].)

The formulation of the LUP states that there is a list of items and a sequence of requests. A request can be an insertion, a deletion or an access to an item in the list. The algorithm solving the problem must serve these requests. The cost of serving an access request to an item p on the ith position from the front of the list is i, which is the number of comparisons needed to find p. After processing a request, the algorithm can reorganize the list in order to minimize the total cost of its maintenance. Once an item p is accessed, it may be moved free of charge to any position closer to the front of the list (*free transpositions*). Other transpositions, with elements located closer than p to the end of the list are called *paid* and cost 1. (In the BWCA, we have a list of items L, containing the characters from alphabet $\mathcal{A}$, and a sequence of requests x^{bwt}. The list L contains all the possible items, so we never do any insertions or deletions.) The algorithms solving the LUP should minimize the total cost of maintaining the list (the sum of all costs of serving the items from the sequence of requests).

In the second step of the BWCA we do not want to minimize the total cost of maintaining the list L. However, when we apply the algorithm solving the LUP to the list L and the sequence of requests x^{bwt} we obtain the sequence x^{lup} that contains the numbers corresponding to the costs of accessing the symbols from x^{bwt}. When we sum all the numbers from the sequence x^{lup}, we get the total cost of maintaining the list L (we assume here that the algorithm for the LUP does not perform paid transpositions). The main goal of the second step of the BWCA is not to minimize the total cost, however typically when the total cost is smaller, then the distribution of probabilities of symbol occurrences in the sequence x^{lup} is less uniform. In the last step of the BWCA, we apply the entropy coder to the sequence x^{lup}, and when the distribution of the symbols in this sequence is less uniform then a better compression ratio can be achieved. This observation is of course not precise because the compression ratio depends on the probability distribution of symbols in the sequence x^{lup} in a more complicated way. When we use the RLE-0 algorithm, we in fact encode the sequence x^{rle0}, the probability distribution of which is slightly different. The overall compression ratio also depends on

Copyright © 2001 John Wiley & Sons, Ltd.

the probability estimation method in the last step of the BWCA. However, typically minimizing the total cost of maintaining the list L yields a better compression ratio.

We also note that the additional cost of paid transpositions can be neglected because the symbols in x^{lup} correspond to the cost of finding the current item, and there is no cost related to reorganizing the list L. In particular, we can say that in the BWCA the problem is similar to the modified LUP, in which the paid transpositions cost 0. The sequence x^{bwt} has also a specific structure (it is composed of CT components). In the LUP we do not assume anything about the sequence of requests. Therefore, using the best algorithms specialized for the LUP does not always lead to the best compression results. Several modifications to these algorithms were proposed to exploit the properties of the BWT in a better way.

MTF and its modifications

Burrows and Wheeler [1] suggested using the MTF transform [9] as the second step of the BWCA. The MTF maintains a character list L. When a character x_i^{bwt} appears, the list L is scanned and the position of the character x_i^{bwt} in the list L is assigned to x_i^{mtf}. (As a result we get the sequence x^{mtf} over the alphabet $\mathcal{A} = \{0, 1, \ldots, k-1\}$.) Then the character is moved to the beginning of the list L. This is a very simple strategy, but it gives quite good results is practice.

Burrows and Wheeler suggested that refraining from moving the current character to the very first position may be sometimes useful. Fenwick [10,11] and Schindler [12] explored such a possibility but failed to obtain better compression results. Recently, Balkenhol *et al.* [13] proposed a modification called MTF-1 which improves the compression ratio. Their only modification to the MTF algorithm is that only the symbols from the second position in the list L are moved to the first position. The symbols from higher positions are moved to the second position. Balkenhol and Shtarkov [6] proposed a further modification to the MTF-1—the symbols from the second position are moved to the beginning of the list L only when the previous transformed symbol is at the first position (following the authors we call this version MTF-2).

Time Stamp

One of the best algorithms for the LUP is Time Stamp presented by Albers [14]. The deterministic version of this algorithm—TimeStamp(0) (TS(0)) [15]—scans for a processed character x_i^{bwt} in the list L and outputs its position. Then it moves the character in front of the first item in the list L that has been requested at most once since the last request for the character x_i^{bwt}. If the character x_i^{bwt} has not been requested so far, it is left at the same position.

TS(0) was theoretically analysed by Albers and Mitzenmacher [15], who showed that theoretically the TS(0) is better than the MTF. The results obtained for sequences from Calgary Corpus [16] were worse than those obtained with the MTF, but the authors do not provide explicit details.

Inversion Frequencies

A completely new approach to the problem of transforming the sequence x^{bwt} to a form that can be better compressed by an entropy coder was proposed by Arnavut and Magliveras [17]. They described an algorithm called Inversion Frequencies (IF). The IF algorithm is not the algorithm solving the LUP.

Copyright © 2001 John Wiley & Sons, Ltd.

It forms a sequence x^{if} over an alphabet of integers from the range $[0, n-1]$. For each character a_j from the alphabet, the IF algorithm scans the sequence x^{bwt}. When it finds the first occurrence of the character a_j it outputs its position in the sequence x^{bwt}. For further occurrences of the character a_j the IF outputs an integer which is the number of characters greater than a_j that occurred since the last request for the character a_j. The sequence x^{if}, however, is not sufficient to recover the sequence x^{bwt} correctly. We also need to know the number of occurrences of each character from the alphabet in the sequence x^{bwt}. This disadvantage is especially important for short sequences.

Distance Coding

Recently a new proposition for the second step algorithm—Distance Coding (DC)—was suggested by Binder. This algorithm has not been published yet and we describe it according to references [18,19].

For each character x_i^{bwt}, DC finds its next occurrence in the sequence x^{bwt} (x_p^{bwt}) and outputs the distance to it, i.e. $p - i$. When there are no further occurrences of x_i^{bwt}, DC outputs 0. To establish the sequence x^{bwt} correctly we also have to know the first position of all the alphabet characters. The basic version of DC, described above, is in fact a small modification of Interval Encoding proposed by Elias [20].

To improve the compression ratio, Binder proposed three improvements to the basic algorithm. First, we can notice that in the sequence x^{dc} some of the ending zeroes are redundant. Second, when we are scanning the sequence x^{bwt} for the next occurrence of the current character, we may count only the characters that are unknown at this moment. Third, and most importantly, if the next character is the same as the current character, we do not need to encode anything, and we can simply proceed to the next character.

The Balkenhol–Shtarkov method for coding MTF output

Recently Balkenhol and Shtarkov [6] proposed a new approach to the coding of the sequence x^{bwt}. Their method is based on the observation that the entropy coder codes the sequence over the alphabet $\mathcal{A}^{\mathrm{mtf}}$ (or $\mathcal{A}^{\mathrm{rle0}}$) consisting of integers. For typical sequences x, the probability distribution of the integers in x^{mtf} decreases monotonically for larger integers. Unfortunately this distribution varies in the sequence and—what is worse—for a given integer, we do not know which character it represents. For example, two identical integers greater than 1 that appear at successive positions in x^{mtf} represent different characters.

Balkenhol and Shtarkov suggest dividing the sequence x^{mtf} into two sequences[‡]. The first sequence, $x^{\mathrm{mtf},1}$, is over the alphabet $\mathcal{A}^{\mathrm{mtf},1} = \{0, 1, 2\}$. It is constructed from x^{mtf} by replacing all occurrences of integers greater than 1 with 2. This means that the ternary sequence $x^{\mathrm{mtf},1}$ holds only the information whether the transformed character was at the first (integer 0), the second (integer 1) or at some other position (integer 2) in the list L. The second sequence, $x^{\mathrm{mtf},2}$, is over the alphabet $\mathcal{A}$. It is constructed from the sequence x^{bwt} by removing the characters for which the integers 0 or 1 appear in the sequence $x^{\mathrm{mtf},1}$.

[‡]In fact Balkenhol and Shtarkov use the MTF-2 algorithm instead of the MTF transform and the sequence $x^{\mathrm{mtf}\text{-}2}$.

Copyright © 2001 John Wiley & Sons, Ltd.

Further, the sequence $x^{\mathrm{mtf},1}$ is treated as a Markov chain and is encoded using the standard universal coding scheme. To encode the sequence $x^{\mathrm{mtf},2}$ the alphabet is divided into four groups of changing sizes and contents. The symbols from the sequence $x^{\mathrm{mtf},2}$ are encoded with the probability estimation from one of these groups. The groups are maintained in such a way that they work as sliding windows and contain statistics of symbol occurrences in these windows. Because, due to their high probability, the last symbols have similar probability distributions, such an estimation helps to obtain good compression results.

WEIGHTED FREQUENCY COUNT

We propose a new algorithm, which can be considered as a generalization of the well known Frequency Count (FC). We call this algorithm a Weighted Frequency Count (WFC).

First we formulate the FC algorithm in an alternative way. We assign to each character a_j appearing prior to the ith position in the sequence x^{bwt} a sum

$$W_i(a_j) = \sum_{\substack{1 \le p < i \\ a_j = x_p}} 1$$

and sort the list L according to the decreasing values of counters $W_i(a_j)$.

Next, we note that instead of summing 1's for all characters we can condition the numbers that are summed from their relative position in the sequence x^{bwt}. To this end we introduce the weight function w and reformulate the sum

$$W_i(a_j) = \sum_{\substack{1 \le p < i \\ a_j = x_p}} w(i - p)$$

If two characters have the same value $W_i(.)$, we find their relative order using the values $W_{i-1}(.)$, $W_{i-2}(.)$ and so on, until the counters are different. For completion we define $W_0(a_j) = -j$. The algorithm outputting the position of processed characters in the list L and maintaining the list in the described way is called the Weighted Frequency Count.

Let us notice that if we set $w(t) = 1,\ \text{for } t > 0$, we obtain the FC algorithm and if we set

$$w(t) = \begin{cases} 1, & \text{for } t = 1, \\ 0, & \text{for } t > 1, \end{cases}$$

we obtain the MTF algorithm.

The algorithm Sort By Time (SBT)—recently proposed by Schulz [21]—is also a special case of the WFC algorithm. We get it by setting $w(t) = q^t$, for $t > 0$. Theoretical properties of the SBT algorithm for different values of q have been examined by Schulz. He has shown that by establishing $0 < q \le 0.5$ one obtains the MTF algorithm.

WFC relation to CT sources

As mentioned above, the sequence x^{bwt} is a concatenation of the CT components. Therefore, the character which has occurred at the previous position is more likely to be at the same context than

Copyright © 2001 John Wiley & Sons, Ltd.

the last but one, and so on. Generally, the characters which have appeared at recent positions are more likely described by the same probability distribution as the current character than those from more distant positions. Unfortunately, we do not know where in the CT we are, how long the CT component of the current leaf is, and where in that CT component we are.

There is also one more property to consider: typically similar contexts have only a slightly different probability distribution (this property is one of the bases of the PPM algorithms). It may be useful to explore some of the information on the probability distribution of previous contexts. All in all, the weight function w should decrease. It is not clear, however, how fast it should decrease.

We have explored many functions w. Our results show that for different sequences from Calgary Corpus different functions w give the best results. This is of course what one would expect. Short sequences have typically shorter CT components than longer sequences. Also the sequences x are generated by different sources which may have a different number of contexts.

An efficient implementation

The formulation of the WFC algorithm does not give us a method for fast computing of the values of $W_i(.)$. When we switch to the next character we have to recalculate the values of all counters $W_i(.)$. To this end, we have to rescan the encoded part of the sequence x^{bwt}. This yields the computational complexity $O(n(n + k \log k))$.

The complexity of the WFC algorithm can be improved by sacrificing its precision. One possibility is to quantize the values of w to integer powers of 2. This decreases the number of different values of w to at most $l = \log_2 w(1)/w(t_{\max}) + 1$ (we assume that the values w are decreasing), which is typically small. For such values of w we can obtain the values of $W_i(.)$ from $W_{i-1}(.)$ by updating only the counters for the characters where the values w are changing (for all such t that $w(t) \neq w(t-1)$). Using this approach we obtain the algorithm of the worst-case complexity $O(nlk)$, which is not much greater than $O(nk)$ for algorithms like the MTF. In practice, the characters on the list L move only by a small number of positions. Using a modified insertion sort algorithm the cost of maintaining the list is small (e.g. for the function w_{6q} the average number of swaps of characters on the list L per input character is less than six for almost all files from Calgary Corpus, except for binary files such as `geo` where it is larger).

The disadvantage of using this approach in the compression ratio depends on the weight function w and properties of the sequence. If necessary we can double the number of different values of w by also using the powers of 2 for half exponents.

Comparison of different weight functions

In the first experiment we examined many weight functions w. In this section we describe the compression results obtained for some of these weight functions (see Figure 2). In our experiments we have used the BWT-based compression algorithm described in [2] (D99), where the MTF-1 algorithm was replaced by the WFC algorithm with the examined weight functions w.

For each of the examined weight functions we have looked for the best set of parameters. The obtained results are shown in Table I. We achieved the best overall results for the function w_6. The use of different parameters for different ranges in the function w_6 is motivated by the fact that typically characters in these ranges are from different (but similar) contexts. It is useful to exploit the information

Copyright © 2001 John Wiley & Sons, Ltd.

$$w_1(t) = \begin{cases} 1, & \text{for } t = 1 \\ 0, & \text{for } t > 1 \end{cases}$$

$$w_2(t) = \begin{cases} q^t, & \text{for } 1 \le t \le t_{\max} \\ 0, & \text{for } t > t_{\max} \end{cases}$$

$$w_3(t) = \begin{cases} \dfrac{1}{p * t}, & \text{for } 1 \le t \le t_{\max} \\ 0, & \text{for } t > t_{\max} \end{cases}$$

$$w_4(t) = \begin{cases} 1, & \text{for } t = 1 \\ \dfrac{1}{p * t}, & \text{for } 1 < t \le t_{\max} \\ 0, & \text{for } t > t_{\max} \end{cases}$$

$$w_5(t) = \begin{cases} 1, & \text{for } t = 1 \\ p * t^q, & \text{for } 1 < t \le t_{\max} \\ 0, & \text{for } t > t_{\max} \end{cases}$$

$$w_6(t) = \begin{cases} 1, & \text{for } t = 1 \\ \dfrac{1}{p * t}, & \text{for } 1 < t \le 64 \\ \dfrac{1}{2 * p * t}, & \text{for } 64 < t \le 256 \\ \dfrac{1}{4 * p * t}, & \text{for } 256 < t \le 1024 \\ \dfrac{1}{8 * p * t}, & \text{for } 1024 < t \le t_{\max} \\ 0, & \text{for } t > t_{\max} \end{cases}$$

Figure 2. Examined weight functions.

of the probability distribution in such contexts, but it should not dominate the probability distribution of the current context.

The last column (w_{6q}) contains the results for the function w_6, where all values were replaced by the nearest negative integer powers of 2. As we can see, the disadvantage caused by the quantization can be neglected. However, for other functions w this difference may not be so small. In further experiments we use the best of the examined functions in the quantized form, w_{6q}.

Copyright © 2001 John Wiley & Sons, Ltd.

Table I. Compression results for different weight functions (for all functions we used $t_{\max} = 2048$).

File	Size	Weight functions w_1	w_2 $q = 0.7$	w_3 $p = 4$	w_4 $p = 4$	w_5 $p = 0.5$ $q = -1.25$	w_6 $p = 4$	w_{6q} $p = 4$
bib	111 261	1.915	1.916	1.969	1.916	1.899	**1.896**	**1.896**
book1	768 771	2.344	2.311	2.280	2.283	2.279	**2.273**	2.274
book2	610 856	1.999	1.980	1.999	1.973	1.962	1.959	**1.958**
geo	102 400	4.235	4.229	**4.115**	4.121	4.146	4.150	4.152
news	377 109	2.464	2.461	2.464	2.415	2.410	**2.409**	**2.409**
obj1	21 504	3.766	3.757	3.724	**3.695**	**3.695**	3.697	**3.695**
obj2	246 814	2.439	2.448	2.492	2.432	2.416	**2.413**	2.414
paper1	53 161	2.420	2.422	2.488	2.424	2.405	**2.403**	**2.403**
paper2	82 199	2.382	2.370	2.405	2.364	2.351	**2.347**	**2.347**
pic	513 216	0.761	0.741	**0.703**	0.706	0.716	0.718	0.717
progc	39 611	2.455	2.461	2.521	2.451	**2.431**	**2.431**	**2.431**
progl	71 646	1.684	1.697	1.769	1.682	1.672	**1.670**	**1.670**
progp	49 379	**1.667**	1.690	1.787	1.690	1.673	1.672	1.672
trans	93 695	**1.450**	1.483	1.611	1.466	1.456	**1.450**	1.452
Average	3 141 622	2.284	2.283	2.309	2.258	2.251	2.249	2.249

COMPARISON OF COMPRESSION ALGORITHMS

In the second experiment, we compared the compression results of different second step algorithms (see Table II). The results in columns denoted by MTF, MTF-1, MTF-2, TS(0) and WFC are obtained using the compression method described in reference [2], where the MTF-1 algorithm was replaced by the mentioned algorithms. For the rest of the algorithms different probability estimation is needed and we cannot simply replace only the second step algorithm. The results for the DC algorithm are taken from experiments with its implementation [22]. The results for the BS99 algorithm are obtained from reference [6]. The IF algorithm was implemented as follows. For each number x_i^{if} the Elias code [23] of length $1 + \lfloor 2 \log x_i^{\text{if}} \rfloor$ bits is calculated and encoded using a binary arithmetic coder. Each bit of the Elias code is encoded in a separate context, where the probability estimation is calculated as weighted probability for orders 0 and 2 (similarly to reference [2]). Instead of encoding the number of occurrences of all characters we have modified the IF algorithm as follows. When we process a character a_j and reach the end of the sequence x^{bwt}, we increment the current character to the a_{j+1} and count further from the beginning of the sequence x^{bwt}. Therefore we sometimes can get a number greater than n, but when we use the binary arithmetic coder and encode each bit separately it is not a problem.

The results show that different algorithms are best for different files from Calgary Corpus, but most of the top results are obtained using the WFC algorithm. However, one should remember that improving the probability estimation for the IF, DC or BS99 algorithms may change this result.

Copyright © 2001 John Wiley & Sons, Ltd.

Table II. Comparison of different second step algorithms for the files from Calgary Corpus.

File	Size	MTF	MTF-1	MTF-2	TS(0)	IF	DC	BS99	WFC
bib	111 261	1.915	1.904	1.904	2.012	1.963	1.931	1.91	**1.896**
book1	768 771	2.344	2.317	2.304	2.308	**2.239**	2.241	2.27	2.274
book2	610 856	1.999	1.983	1.976	2.027	1.964	**1.938**	1.96	1.958
geo	102 400	4.235	4.221	4.220	4.186	4.190	4.510	4.16	**4.152**
news	377 109	2.464	2.450	2.449	2.586	2.459	**2.397**	2.42	2.409
obj1	21 504	3.766	3.737	3.740	3.900	3.889	3.969	3.73	**3.695**
obj2	246 814	2.439	2.427	2.429	2.637	2.548	2.451	2.45	**2.414**
paper1	53 161	2.420	2.411	2.411	2.588	2.454	2.407	2.41	**2.403**
paper2	82 199	2.382	2.369	2.364	2.458	2.366	**2.343**	2.36	2.347
pic	513 216	0.761	0.741	0.737	0.732	**0.706**	0.717	0.72	0.717
progc	39 611	2.455	2.446	2.450	2.643	2.500	2.473	2.45	**2.431**
progl	71 646	1.684	1.678	1.681	1.851	1.747	1.692	1.68	**1.670**
progp	49 379	1.667	**1.665**	1.670	1.887	1.745	1.705	1.68	1.672
trans	93 695	1.450	**1.448**	1.452	1.704	1.557	1.473	1.46	1.452
Average	3 141 622	2.284	2.271	2.270	2.394	2.309	2.303	2.26	2.249

Table III. Comparison of compression algorithms for Calgary Corpus.

File	Size	gzip	PPMD+	B97	CTW	VW98	BW94	D99	BS99	D00
bib	111 261	2.509	1.862	1.786	1.79	1.714	2.07	1.904	1.91	1.896
book1	768 771	3.250	2.303	2.184	2.19	2.150	2.49	2.317	2.27	2.274
book2	610 856	2.700	1.963	1.862	1.87	1.820	2.13	1.983	1.96	1.958
geo	102 400	5.345	4.733	4.458	4.46	4.526	4.45	4.221	4.16	4.152
news	377 109	3.063	2.355	2.285	2.29	2.210	2.59	2.450	2.42	2.409
obj1	21 504	3.839	3.728	3.678	3.68	3.607	3.98	3.737	3.73	3.695
obj2	246 814	2.628	2.378	2.283	2.31	2.245	2.64	2.427	2.45	2.414
paper1	53 161	2.791	2.330	2.250	2.25	2.152	2.55	2.411	2.41	2.403
paper2	82 199	2.887	2.315	2.213	2.21	2.136	2.51	2.369	2.36	2.347
pic	513 216	0.817	0.795	0.781	0.79	0.764	0.83	0.741	0.72	0.717
progc	39 611	2.678	2.363	2.291	2.29	2.195	2.58	2.446	2.45	2.431
progl	71 646	1.805	1.677	1.545	1.56	1.482	1.80	1.678	1.68	1.670
progp	49 379	1.812	1.696	1.531	1.60	1.460	1.79	1.665	1.68	1.672
trans	93 695	1.611	1.467	1.325	1.34	1.256	1.57	1.448	1.46	1.452
Average	3 141 622	2.695	2.283	2.177	2.19	2.123	2.43	2.271	2.26	2.249

Copyright © 2001 John Wiley & Sons, Ltd.

In the third experiment we compared our modified algorithm D00 (D99 when the MTF-1 algorithm is replaced by the WFC algorithm) to the best known BWT-based and other algorithms. Table III contains the compression results as output bits per input character (bpc) for the following methods:

- gzip: standard *gzip* program with option −9 (this is an implementation of the well known LZ77 algorithm [24]);
- PPMD+: PPMD+ algorithm implemented by Teahan [25];
- B97: the best version of PPM algorithms proposed by Bunton [26];
- CTW: Context Tree Weighting method proposed by Willems *et al.* [27];
- BW94: the original Burrows–Wheeler algorithm [1];
- VW98: switching method, algorithm presented by Volf and Willems [28,29];
- D99: BWT-based algorithm proposed by Deorowicz [2];
- BS99: BWT-based algorithm proposed by Balkenhol and Shtarkov [6];
- D00: BWT-based algorithm proposed in this paper.

The algorithm D00 yields the best results in the class of BWT-based algorithms, though its advantage over the BS99 algorithm is rather small (only about 0.01 bpc).

CONCLUSIONS

We have described different second step algorithms for the BWCA. We have also presented a new algorithm (WFC) for the second step. The results of our *ad hoc* proposed functions w are promising. We expect that future experiments could provide further improvements in the compression ratio. Also a theoretical analysis of the WFC algorithm should be provided. It would be interesting to find a more precise relation between the presented weight functions w and the CT-source model.

The overall results for the best BWT-based compression algorithms are close to the results of the best compression methods known. We expect that further improvements are possible and we can get even closer to the best algorithms. It is however probable that BWT-based algorithms cannot match the performance of the PPM algorithms, because the knowledge of the locations where the contexts change gives an important advantage to the latter family of algorithms.

ACKNOWLEDGEMENTS

The research was carried out under the Silesian University of Technology Research Project Bk-210/RAu2/2000. The author wishes to thank N. Jesper Larsson for the source code of the suffix tree creation algorithm and John Carpinelli, Alistair Moffat, Radford Neal, Wayne Salamonsen, Lang Stuiver and Ian Witten for the source code of the arithmetic coding algorithm which were both used in the experiments. I also wish to thank Zbigniew J. Czech, Marcin Ciura and Roman Starosolski for reviewing the manuscript and suggesting improvements, Szymon Grabowski for pointing out the DC algorithm and for interesting discussions on the BWT-based compression algorithms, and anonymous referees for helpful comments.

REFERENCES

1. Burrows M, Wheeler DJ. A block-sorting lossless data compression algorithm. *SRC Research Report 124*, Digital Equipment Corporation, 1994. ftp://ftp.digital.com/pub/DEC/SRC/research-reports/SRC-124.ps.zip.

Copyright © 2001 John Wiley & Sons, Ltd.

2. Deorowicz S. Improvements to Burrows–Wheeler compression algorithm. *Software—Practice and Experience* 2000; **30**(13):1465–1483.
3. Weinberger MJ, Rissanen J, Feder M. A universal finite memory source. *IEEE Transactions on Information Theory* 1995; **41**:643–652.
4. Balkenhol B, Kurtz S. Universal data compression based on the Burrows–Wheeler transformation: Theory and practice. *IEEE Transactions on Computers* 2000; **49**(20):1043–1053.
5. Larsson NJ. The context trees of block sorting compression. *Proceedings DCC'98 Data Compression Conference*, Storer JA, Cohn M (eds.). IEEE Computer Society Press: Los Alamitos, CA, 1998; 189–198.
6. Balkenhol B, Shtarkov YM. One attempt of a compression algorithm using the BWT. *Preprint 99-133, SFB343*: Discrete structures in mathematics. Faculty of Mathematics, University of Bielefeld, 1999. http://www.mathematik.uni-bielefeld.de/sfb343/preprints/pr99133.ps.gz.
7. McCabe J. On serial files with relocatable records. *Operations Research* 1965; **13**:609–618.
8. Bachrach R, El-Yaniv R. Online list accessing algorithms and their applications: Recent empirical evidence. *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*. ACM Press: New York, NY, 1997; 53–62.
9. Bentley JL, Sleator DD, Tarjan RE, Wei VK. A locally adaptive data compression scheme. *Communications of ACM* 1986; **29**(4):320–330.
10. Fenwick P. Block sorting text compression—final report. *Technical Report 130*, The University of Auckland, Department of Computer Science, 1996. ftp://ftp.cs.auckland.ac.nz/pub/peter-f/TechRep130.ps.
11. Fenwick P. The Burrows–Wheeler transform for block sorting text compression: Principles and improvments. *The Computer Journal* 1996; **39**(9):731–740.
12. Schindler M. A fast block-sorting algorithm for lossless data compression. *Proceedings DCC'97 Data Compression Conference*, Storer JA, Cohn M (eds.). IEEE Computer Society Press: Los Alamitos, CA, 1997; 469.
13. Balkenhol B, Kurtz S, Shtarkov YM. Modifications of the Burrows and Wheeler data compression algorithm. *Proceedings DCC'99 Data Compression Conference*, Storer JA, Cohn M (eds.). IEEE Computer Society Press: Los Alamitos, CA, 1999; 188–197.
14. Albers S. Improved randomized on-line algorithms for the list update problem. *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM Press: New York, NY, 1995; 412–419.
15. Albers A, Mitzenmacher M. Average case analyses of list update algorithms, with applications to data compression. *Algorithmica* 1998; **21**(3):312–329.
16. Bell TC, Cleary JG, Witten IH. *Text Compression*. Prentice Hall: Englewood Cliffs, NJ, 1990.
17. Arnavut Z, Magliveras SS. Block sorting and compression. *Proceedings DCC'97 Data Compression Conference*, Storer JA, Cohn M (eds.). IEEE Computer Society Press: Los Alamitos, CA, 1997; 181–190.
18. Binder E. Distance Coder. *Usenet group: comp.compression* [2000].
19. Binder E. Private communications, 2001.
20. Elias P. Interval and recency rank source coding: Two on-line adaptive variable length schemes. *IEEE Transactions on Information Theory* 1987; **33**:3–10.
21. Schulz F. Adaptive Suchverfahren (in German). *Doctoral dissertation*, der Technischen Fakultät der Universität des Saarlandes, Saarbrücken, Germany, 1999. http://www-hotz.cs.uni-sb.de/~schulz/publications/Dissertation-Schulz.ps.gz.
22. Binder E. The DC program. ftp://ftp.elf.stuba.sk/pub/pc/pack/dc124.zip [2000].
23. Elias P. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory* 1975; **21**:194–203.
24. Ziv J, Lempel A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 1977; **23**:337–343.
25. Teahan B. The PPMD+ program. http://www.cs.waikato.ac.nz/~wjt/software/ppm.tar.gz [1996].
26. Bunton S. Semantically motivated improvements for PPM variants. *The Computer Journal* 1997; **40**(2–3):76–93.
27. Willems FMJ, Shtarkov YM, Tjalkens TJ. The context tree weighting method: Basic properties. *IEEE Transactions on Information Theory* 1995; **41**:653–664.
28. Volf PAJ, Willems FMJ. Switching between two universal source coding algorithms. *Proceedings DCC'98 Data Compression Conference*, Storer JA, Cohn M (eds.). IEEE Computer Society Press: Los Alamitos, CA, 1998; 491–500.
29. Volf PAJ, Willems FMJ. The switching method: Elaborations. *Proceedings of the 19th Symposium on Information Theory in the Benelux*, Werkgemeenschap voor Informatie- en Communicatietheorie, Enschede, Netherlands, de With PHN, v.d. Schaar-Mitrea M, (eds.), 1998; 13–20. http://ei1.ei.ele.tue.nl/~paul/p_and_w/ben98.ps.zip.
30. Cleary J, Witten I. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications* 1984; **COM-32**:396–402.

Copyright © 2001 John Wiley & Sons, Ltd.