Experimental Evaluation of List Update Algorithms for Data Compression

Reza Dorrigiv[1], Alejandro López-Ortiz[1], and J. Ian Munro[1]

Cheriton School of Computer Science,
University of Waterloo,
Waterloo, Ont., N2L 3G1, Canada,
{rdorrigiv, alopez-o, imunro}@uwaterloo.ca

Technical Report CS-2007-38
Cheriton School of Computer Science
University of Waterloo
October 2007

Abstract. List update algorithms have been widely used as subroutines in compression schemas, specially after the introduction of the Burrows-Wheeler transform. The Burrows-Wheeler transform (BWT), which is the basis of most state-of-the-art general purpose compressors, defines a permutation of the text with increased apparent regularity. Then it applies a compression algorithm on the permuted version of the original text. List update algorithms are a common choice for this second stage of BWT-based compression. As well, list update algorithms have been shown to be a reasonable alternative to simple compression schemes such as Huffman coding [1]. In this paper we perform an experimental comparison of various list update algorithms both as stand alone compression mechanisms and as a second stage of the BWT-based compression. Our experiments show that for straightforward compression competitive list update algorithms such as MTF while simpler, are an inferior choice to Huffman based compression on, in contrast they are a good choice as a second stage in BWT. Then we address the question of which list update algorithm is best for this second stage. We show that MTF outperforms other list update algorithms in practice after BWT. This is consistent with the intuition that BWT increases locality of reference and the predicted result from the locality of reference model of Angelopoulos et al. [2]. Lastly, we show theoretically that due to an often neglected difference in the cost models, good list update algorithms may be far from optimal for BWT compression.

1 Introduction

It has long been observed that list update algorithms can be used for compression. In 1986, Bentley et al. [1] proposed a compression scheme that

uses Move-To-Front as a subroutine. They proved that their compression scheme, based on move-to-front (MTF) is guaranteed to be within twice the compression ratio of the best static Huffman code. Furthermore, they implemented the scheme and showed that their algorithm achieves compression ratios that are comparable or better than Huffman encoding on some text and Pascal files. Observe that Move-To-Front can be replaced with any other on-line list update algorithm, which may or may not improve the compression rate. Albers and Mitzenmacher [3] studied the use of Timestamp and showed theoretical and experimental evidence for its efficiency in data compression. Several on-line list update algorithms were compared according to their efficiency in compression by Bachrach et al. [4]. In general, their results surprisingly show that usually algorithms with bad competitive ratios outperform those that are optimal according to competitive analysis in terms of compression ratio. Therefore there is an inconsistency between theory and practice.

Additionally, Burrows and Wheeler [5] combined their transform with list update algorithms to obtain a compression scheme. The Burrows-Wheeler Transform (BWT) rearranges a string of symbols to one of its permutations. Then they used Move-To-Front algorithm in a similar way to the scheme proposed by Bentley et al. [1]. The resulting scheme is shown to be very effective in theory and practice and many improvements and variants have been proposed for it [5–12]. The well known compression program bzip2 [13] is based on the BWT.

Our study was motivated by recent theoretical results on the impact of locality of reference assumptions for online algorithms [2]. Compression via list update hinges on an implicit assumption that the text (raw or after the BWT transform) exhibits locality of reference which can then be used advantageously by list update algorithms. BWT based compression uses first the BWT transform and then applies a compression step. A common choice is to use a list update algorithm followed by run length encoding or some other basic compressor. In this paper we systematically study different sensible choices for the list update algorithm as well as for the basic compressor.

Our Results In this paper we perform an experimental evaluation of the various list update algorithms for compression. We compare their performance both in stand alone form and as part of BWT based compression. We show that in most cases MTF is the best choice. For the basic compressor a Huffman variant which incorporates run length encoding into the alphabet gives the best compression possible. Additionally as a con-

sequence of this study we observed that list update algorithms optimize for a similar but different objective than a compressor and give an example of an algorithm which is a good choice for list update but not for compression.

2 Preliminaries

2.1 List Update Algorithms

Three standard deterministic on-line algorithms are *Move-To-Front* (MTF), *Transpose* (TR), and *Frequency-Count* (FC). MTF moves the requested item to the front of the list whereas Transpose exchanges the requested item with the item that immediately precedes it. FC maintains a frequency count for each item, updates this count after each access, and updates the list so that it always contains items in non-increasing order of frequency count. Sleator and Tarjan showed that MTF is 2-competitive, while TR and FC do not have constant competitive ratios [14]. Since then, several other deterministic and randomized on-line algorithms have been studied using competitive analysis. We only consider deterministic algorithms because randomized list update algorithms cannot be used in the compression scheme in a straightforward way. Albers introduced the algorithm *Timestamp* (TS) and showed that it is 2-competitive [15]. After accessing an item a, TS inserts a in front of the first item b that appears before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorganize the list.

Schulz [16] introduced an infinite (uncountable) family of list update algorithms called *Sort-By-Rank* (SBR). All algorithms in this family achieve the optimal competitive ratio 2 and they mediate between MTF and TS. Consider a sequence $\sigma = \sigma_1\sigma_2\cdots\sigma_m$ of length m. For an item a and a time $1 \leq t \leq m$, denote by $w_1(a,t)$ and $w_2(a,t)$ the time of the last and the second last access to a in $\sigma_1\sigma_2\cdots\sigma_t$, respectively. If a has not been accessed so far, set $w_1(a,t) = 0$ and if a has been accessed at most once, set $w_2(a,t) = 0$. Then we define $s_1(a,t) = t - w_1(a,t)$ and $s_2(a,t) = t - w_2(a,t)$. Note that after each access, MTF and TS reorganize their lists so that the items are in increasing order of their s_1 and s_2, respectively[1]. For a parameter $0 \leq \alpha \leq 1$, $SBR(\alpha)$ reorganizes its list after the tth access so that items are sorted by their α-rank function

[1] For TS, strictly speaking, this applies only to items that have been accessed at list twice.

defined as $r_\alpha(a,t) = (1-\alpha) \times s_1(a,t) + \alpha \times s_2(a,t)$[2]. More formally, upon a request for an item a in time t, $SBR(\alpha)$ inserts a just after the last item b in front of a with $r_\alpha(b,t) < r_\alpha(a,t)$. Also if there is no such item b or this is the first access to a, $SBR(\alpha)$ inserts a in front of the list. Therefore $SBR(0)$ is equivalent to MTF and $SBR(1)$ is equivalent to TS except modulo the handling of the first accesses, i.e., they were equivalent if TS moves an item that has been accessed only once so far to the front of the list. Intuitively, for values of $0 < \alpha < 1$, $SBR(\alpha)$ is supposed to have behaviour between MTF and TS. We will check this in the case of compression efficiency in Subsection 4.2.

2.2 Compression Schemas

Bentley et al. [1] proposed using list update algorithms as subroutines in compression. The idea of this compression scheme is simple. Both encoder and decoder maintain a list L of all symbols in the file and agree on some on-line list update algorithm $\mathcal{A}$ as well as an initial arrangement for L. The encoder encodes every symbol by its current position in L and then rearranges L according to $\mathcal{A}$. It uses some variable length prefix-free binary code to transmit these integers (positions). Since the decoder knows the initial arrangement of the list and the list update algorithm, it can maintain the same list as the encoder and recover all the symbols. Several variable length prefix-free binary codes can be used in this scheme, e.g., Elias encoding, δ-encoding, ω-encoding, and ω'-encoding. Here we only describe Elias encoding that we use in this paper; see [4], Appendix E for a description of other encodings. Elias encoding of an integer i contains $2\lfloor \log i \rfloor + 1$ bits. It starts with $\lfloor \log i \rfloor$ bits of 0, followed by the binary representation of i.

2.3 Burrows-Wheeler Transform

Burrows and Wheeler [5] introduced the novel idea of a preprocessing phase called the Burrows-Wheeler Transform (BWT) combined with a compression scheme on the resulting text. Informally, the BWT rearranges a string of symbols to one of its permutations in a reversible way so that the resulting string is "more compressible" or has more "locality of reference". A string has high locality of reference if when a symbol occurs in some position of the string, it is more likely to occur in the nearby positions. For a detailed explanation of the BWT transform we refer the reader to [5, 6].

[2] Schulz [16] denoted this by $r_t(a,\alpha)$.

3 Competitiveness of List Update Algorithms for Compression

A list update algorithm $\mathcal{A}$ incurs cost i to access the ith item of the list. However, when we use $\mathcal{A}$ as a subroutine for compression we need $\Theta(\log i)$ bits to represent that the symbol is at the ith position of the list. We show that this difference in the cost model can lead to different competitiveness results for list update algorithms. Other papers that have studied the use of list update algorithms in compression have generally not considered this phenomenon and assumed that competitive list update algorithms are also competitive for compression.

We consider a family of deterministic list update algorithms called *Move-Fraction (MF)*. Upon a request to an item in the ith position, MF(k) moves that item $\lceil i/k \rceil$-1 positions towards the front. Note that MF(1) is equivalent to MTF. This family of algorithms was proposed by Sleator and Tarjan [14], who also showed that MF(k) is $2k$-competitive. Therefore algorithm MF(2) is 4-competitive for list update. We show that under the $\Theta(\log i)$ cost model, MF(2) is not competitive, i.e., does not have constant competitive ratio. Let the cost of compressing for an item in the ith position in the list be $c\lfloor \log i \rfloor + b$ for some constants c and b. For simplicity assume that we have $l = 2^p$ symbols for some integer p. Suppose that symbols are initially ordered as $a_1 a_2 \cdots a_l$ in the list. Now consider the sequence $\sigma_1 = a_l^p$. On the ith request to a_l, MF(2) incurs cost at least $c\lfloor \log \frac{2^p}{2^{i-1}} \rfloor + b = c(p - i + 1) + b$ and moves a_l to a position of index at least $\frac{2^p}{2^i}$. Therefore the cost of MF(2) on σ_1 is at least

$$\sum_{i=1}^{p} (c(p-i+1)+b) = \frac{cp(p+1)}{2} + bp = \frac{c \log l(\log l + 1)}{2} + b \log l = \Theta(\log^2 l).$$

On the other hand, MTF moves a_l to the front of the list and incurs cost $c\lfloor \log l \rfloor + b + (p-1)b = (b+c)\log l$ on σ_1. Thus the cost of OPT on this sequence is at most $(b+c)\log l = \Theta(\log l)$. We can request the item that is now in the ith position of MF(2)'s list p times. Therefore the competitive ratio of MF(2) is at least

$$\frac{c \times \log l(\log l + 1)/2 + b \log l}{(b+c)\log l} = \frac{c(\log l + 1)}{2(b+c)} + \frac{b}{b+c} = \Theta(\log l),$$

which is not a constant. We can prove the same non-competitiveness for MF(k) for $k \geq 3$. The poor performance of these algorithms for compression was empirically observed by Bachrach et al. [4].

4 Experimental Results

We consider two experimental setups. The first one consists of a straightforward compression scheme similar to that of Bentley et al. [1] or Albers et al. [3]. While in practice these compression techniques are unlikely to be of use, the study of their behaviour allows us to understand their differences and comparative advantages.

The second setup consists of the realistic setting of BWT based compression. To be more precise, given a text we compute its BWT and then compare the role of various list update algorithms for compressing the transformed string.

4.1 Experimental Settings

We compare the compression ratios achieved by different list update algorithms on files in the Calgary Corpus [17] and the Canterbury Corpus [18]. These are the standard benchmark files used for comparing data compression schemas. We considered the following list update algorithms: MTF, MTF2, SBR, FC, FC$'$, TS, and TR. MTF, SBR, FC, TS and TR are described in Subsection 2.1. MTF2 is a variant of MTF that on the ith access to an item a, moves a to the front of the list if i is even and does not change a's position if i is odd. We considered two implementations for FC depending on the order of items with the same frequency count. In FC, an item that is less recently used precedes an item that is more recently used and has equal frequency count. FC$'$ adopts the reverse of this ordering. We consider different parameters for SBR since a compressor can, at time of compression, select the parameter α which achieves the most compression and then prepend the compressed file with the choice of α. If not explicitly mentioned otherwise, we use the standard prefix integer encoding of Elias [19] that encodes an integer i using $1 + 2\lfloor \log i \rfloor$ bits. We propose and evaluate some alternative ways for encoding the integers.

4.2 SBR Parameters

Recall that SBR(0) is equivalent to MTF and SBR(1) is equivalent to TS modulo handling the first accesses. Additionally intuitively for $0 < \alpha < 1$, $SBR(\alpha)$ mediates between the behaviour of MTF and TS. We test this intuition in the case of compression. Figure 1 shows the percentage of the size of the file obtained using the compression algorithm that is based on $SBR(\alpha)$ to the original file size for different values of parameter α and

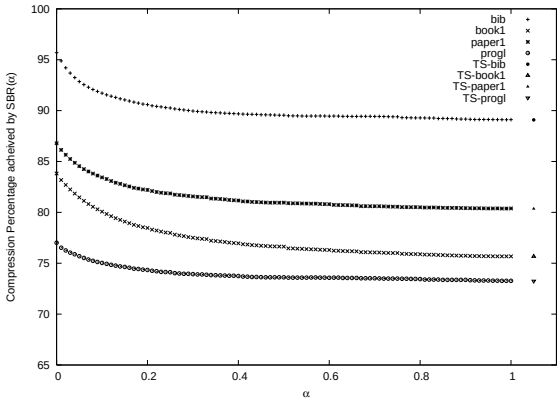

Fig. 1. The percentage of compressed file size achieved by using $SBR(\alpha)$ to the original file size for four different files in the Calgary Corpus in terms of parameter α

four files of the Calgary Corpus. Figure 2 shows the same values after applying BWT.

From these graphs it is clear that as α goes from 0 to 1, the behaviour of SBR(α) goes from MTF to TS. This change in behaviour is faster for small values of α. Although SBR(α) usually achieves best compression for the extremal values of α ($\alpha = 0$ or $\alpha = 1$), there are a few cases in which the optimal value of α is different. For example for file book1 after BWT, the best compression is 32.34 % and it is achieved by SBR(0.32). Therefore the compressor can find the best parameter α and attach it to the compressed file. The decompresser then uses SBR(α) in its decoding process.

4.3 Comparing List Update Algorithms

In order to compare the effect of BWT on the behaviour of compression algorithms, we have computed the result of using different list update algorithms on text files of the Calgary Corpus and the Canterbury Corpus before and after BWT. Table 1 shows the performance of the described algorithms on text files of these corpora and Table 2 shows the corresponding values if we first apply BWT to the files. For each list update algorithm $\mathcal{A}$ and each file f, the table shows the percentage of the size of the compressed file obtained by using $\mathcal{A}$ on f to the original size of f.

From Table 1 we can see that TR and FC usually outperform MTF and TS. This is in contrast with competitive analysis in which MTF

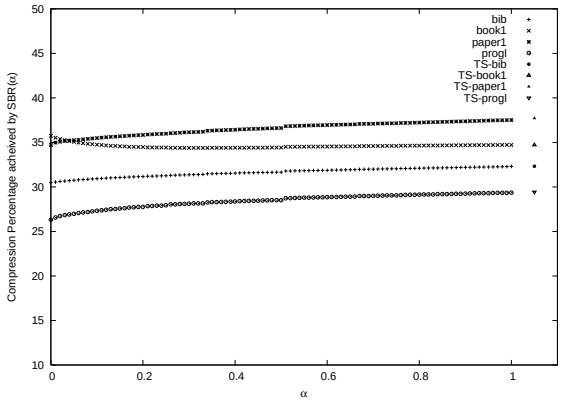

Fig. 2. The percentage of compressed file size achieved by using $SBR(\alpha)$ to the original file size for four different files in the Calgary Corpus after BWT in terms of parameter α

and TS are superior to TS and FC. MTF has the worst performance on all the files and TR is the best algorithm in most cases. MTF2 and FC$'$ always have performance close to their variants, i.e., MTF and FC, respectively. Note that the results for MTF and TS were also reported by Albers and Mitzenmacher [3], who observed that TS outperforms MTF. $SBR(0.5)$ always mediated between the performance of MTF and TS. Thus our experimental results are not consistent with theory. This has been observed by other researches as well [4].

However, for the BWT of the files, the situation is different. Table 2 shows that in this case MTF has the best performance for most of the files. In general, MTF and TS (and thus MTF2 and $SBR(0.5)$) have close performance and always outperform FC and TR. Also the compression ratio that they achieve is much better than the case without BWT. These results are consistent with other experimental results that show the effect of BWT on the performance of compression schemas. This might due to the fact that the BWT is designed to increase the amount of locality in the string, i.e., in the BWT of a string most equal characters occur close to each other. Superiority of MTF to other algorithms is consistent with our recent result that proves MTF outperforms all other on-line list update algorithm on sequences with high locality of reference [2]. Hence, this provides evidence that the locality of reference model proposed accurately reflects reality. We emphasize that our focus here is comparing different list update algorithms and therefore we have not applied any optimization

File	Size (bytes)	MTF	SBR(0.5)	TS	FC	TR	MTF2	FC′
bib	111261	95.69	89.55	89.08	81.42	81.64	94.16	81.42
book1	768771	83.82	76.64	75.67	81.34	69.62	81.27	81.34
book2	610856	84.35	78.36	77.55	75.74	72.44	82.35	75.74
news	377109	88.50	82.68	82.20	88.10	77.87	87.08	87.99
paper1	53161	86.79	80.96	80.35	79.48	74.87	85.19	79.45
paper2	82199	84.47	78.34	77.43	79.27	71.02	82.26	80.45
progc	39611	88.74	84.02	83.62	81.59	77.67	88.16	81.54
progl	71646	77.01	73.62	73.25	82.61	69.02	76.50	82.40
progp	49379	81.09	76.15	75.45	82.41	71.64	80.00	81.68
trans	93695	87.58	84.96	84.59	91.21	83.02	87.36	91.18
alice29.txt	152089	83.69	76.49	75.62	74.74	69.24	81.48	74.85
asyoulik.txt	125179	88.54	81.71	80.67	79.36	73.92	86.44	79.36
cp.html	24603	92.45	91.47	91.90	87.12	85.53	93.01	88.31
fields.c	11150	84.05	80.05	79.81	74.41	73.52	83.73	74.25
grammar.lsp	3721	79.82	75.30	73.45	77.88	68.56	78.63	77.69
lcet10.txt	426754	82.50	76.07	75.23	79.14	69.91	80.42	85.39
plrabn12.txt	481861	86.16	77.67	76.29	88.48	69.98	82.85	88.48

Table 1. Percentage of the size of the compressed files to the size of the original files for different algorithms on text files of the Calgary and Canterbury Corpus without BWT

to the compression scheme, in the presumption that these optimizations would generally affect all schemes equally. It is interesting that such a simple scheme can lead to such good compression ratios.

We also observe that FC and FC′ perform badly compared to other algorithms. One explanation for this is the fact that FC considers the global rather than local environment. For example if an item is accessed a lot at the beginning and then it is not accessed at all, FC will maintain it close to the front of the list.

4.4 Static and Adaptive Huffman Encoding

Table 3 shows the performance ratios achieved by the static Huffman algorithm, as well as an implementation of the Vitter's algorithm on files of the Calgary Corpus before and after BWT. Vitter's algorithm [20, 21] is an adaptive or dynamic Huffman algorithm that adapts to the changes in data. Comparing the ratios before BWT to the ratios of Table 1, we conclude that Vitter's algorithm performs better than plain list update based strategies before BWT. However Vitter's algorithm does not achieve any improvement by applying BWT; the performance remains the same or is worse on BWT of files. Comparing this table to Table 2 we

File	Size (bytes)	MTF	SBR(0.5)	TS	FC	TR	MTF2	FC′
bib	111261	30.49	31.66	32.32	93.42	39.81	31.99	93.33
book1	768771	35.74	34.42	34.71	76.63	36.31	36.04	76.50
book2	610856	31.14	31.03	31.48	80.44	35.31	31.96	80.11
news	377109	36.21	37.75	38.67	85.27	44.90	38.26	85.53
paper1	53161	34.70	36.62	37.70	83.42	47.73	36.87	83.34
paper2	82199	34.86	35.35	36.04	79.00	41.28	36.17	76.46
progc	39611	35.04	37.32	38.54	79.03	51.09	37.54	78.91
progl	71646	26.31	28.52	29.43	81.23	36.18	28.33	79.77
progp	49379	26.00	29.08	30.22	89.11	41.13	28.57	86.08
trans	93695	24.12	27.64	28.71	96.08	41.52	26.76	90.22
alice29.txt	152089	33.15	32.97	33.45	81.34	37.43	33.99	81.37
asyoulik.txt	125179	36.96	36.53	37.08	83.08	41.50	37.79	80.96
cp.html	24603	36.10	38.22	39.37	91.54	49.62	38.58	89.03
fields.c	11150	29.96	33.24	35.03	80.73	51.75	32.84	78.74
grammar.lsp	3721	34.64	38.48	40.85	74.36	52.06	39.26	70.63
lcet10.txt	426754	30.76	30.55	31.01	76.49	34.21	31.51	76.64
plrabn12.txt	481861	36.30	35.23	35.57	78.88	37.13	36.65	78.88

Table 2. Percentage of the size of the compressed files to the size of the original files for different algorithms on text files of the Calgary and Canterbury Corpus after BWT

conclude that after BWT, list update based strategies beat the Vitter's algorithm. Static Huffman has very close performance to the Vitter's algorithm on these files.

4.5 Alternative Techniques for Encoding of Integers

In this subsection we consider other possibilities for the last step of list update based compression schemes, i.e., the prefix-free binary code for integers. These algorithms try to use the following intuition: there is considerable locality of reference in the BWTs of text files and therefore applying a competitive list update algorithm to them leads to a sequence with many small integers. These algorithms assign smaller codes to small integers. In particular, there are many "1"s in the sequence. Therefore almost all these algorithms use a run length on "1"s.

RL(1)+Elias This algorithm combine Elias encoding with a run length encoding for values of 1. More specifically, when the encoded integer is 1, the following Elias-encoded integer shows the number of consecutive 1's starting from that 1. Otherwise, that is the next integer encoded in Elias encoding. Table 4 shows the result of applying this algorithm to text files of the Calgary Corpus after BWT.

File	Size (bytes)	Vitter Before BWT	Vitter After BWT	Static Huffman
bib	111261	65.50	65.51	65.40
book1	768771	57.04	57.04	57.02
book2	610856	60.32	60.32	60.29
news	377109	65.38	65.38	65.34
paper1	53161	62.95	62.96	62.71
paper2	82199	58.08	58.08	57.93
progc	39611	65.73	65.75	65.42
progl	71646	60.15	60.16	60.14
progp	49379	61.44	61.44	61.19
trans	93695	69.76	69.76	66.54

Table 3. Percentage of the size of the compressed files to the size of the original files for Vitter's adaptive Huffman encoding and static Huffman encoding on the text files of the Calgary Corpus before and after BWT

File	Size (bytes)	MTF	SBR(0.5)	TS	FC	TR	MTF2	FC′
bib	111261	27.87	28.92	29.55	93.42	37.06	29.28	93.42
book1	768771	35.78	34.50	34.77	76.78	36.46	36.02	78.68
book2	610856	29.72	29.56	30.00	80.52	33.98	30.48	80.53
news	377109	35.51	36.82	37.71	85.33	43.96	37.37	85.50
paper1	53161	34.60	36.32	37.38	83.36	47.56	36.64	84.96
paper2	82199	34.59	35.01	35.66	79.00	41.02	35.80	78.96
progc	39611	34.83	36.89	38.07	79.15	50.83	37.15	82.32
progl	71646	24.15	26.17	27.07	81.25	33.96	26.07	84.32
progp	49379	23.87	26.68	27.80	89.14	38.92	26.29	91.77
trans	93695	20.92	24.26	25.31	95.58	38.32	23.46	102.71

Table 4. Percentage of the size of the compressed files to the size of the original files for different algorithms on text files of the Calgary Corpus for RL(1)+Elias after BWT

As these results show, this small change improves the compression factor for most list update algorithms. This can be explained by the fact that BWTs of text files have many repetitions. Each such repetition leads to a 1 in the sequence of integers. Therefore we will have many 1's.

1-5-6-17+RL(1): This algorithm uses the following scheme instead of Elias encoding: 1 is encoded by "0", 2 to 9 are encoded by "10000", "10001", ..., "10111", 10 to 17 are encoded by "110000", "110001", ..., "110111", and integers greater than 17 are encoded by their binary representation prepended by "111". Note that there are $l-17$ such numbers, and so we can use a fixed code of length $\lceil \log_2 (l-17) \rceil$ for their binary representations. It also uses run length on "1"s, i.e., when it encodes a "1" the following integer, encoded using the same scheme, denotes the number of consecutive ones started from that "1". Table 5 shows the performance of this algorithm on text files of the Calgary Corpus after BWT.

File	Size (bytes)	MTF	SBR(0.5)	TS	FC	TR	MTF2	FC′
bib	111261	29.54	30.61	31.10	82.72	37.22	30.53	82.25
book1	768771	40.43	39.41	39.50	74.74	40.77	40.41	73.34
book2	610856	33.50	33.49	33.76	77.62	36.98	33.98	77.64
news	377109	38.36	39.68	40.44	82.37	45.62	39.69	82.68
paper1	53161	37.80	39.51	40.33	76.96	48.98	39.20	78.38
paper2	82199	38.10	38.54	39.04	77.75	43.43	38.92	77.72
progc	39611	37.90	39.92	40.94	75.69	51.50	39.53	84.28
progl	71646	26.93	29.02	29.89	80.58	35.73	28.37	83.62
progp	49379	26.73	29.40	30.52	85.70	40.28	28.29	86.80
trans	93695	22.53	26.10	27.01	90.77	38.38	24.03	96.63

Table 5. Percentage of the size of the compressed files to the size of the original files for different algorithms on text files of the Calgary Corpus for 1-5-6-17+RL(1) after BWT

1-2+RL(1) This algorithm uses the following scheme instead of Elias encoding. Encode 1 with a single bit 0, and encode all other numbers with their binary representations prepended by 1. We need $\lceil \log_2 l \rceil$ bits for this binary representation. For most of the cases, this gives a code of length 8 for each integer greater than 1, as $64 \leq l < 128$. Also it uses run length on "1"s.

File	Size (bytes)	MTF	SBR(0.5)	TS	FC	TR	MTF2	FC′
bib	111261	36.36	37.77	38.18	87.44	43.09	37.44	87.44
book1	768771	59.33	57.89	57.92	98.47	59.05	59.25	96.34
book2	610856	47.94	47.89	48.10	97.96	50.78	48.47	97.96
news	377109	51.60	53.52	54.16	97.87	58.28	53.09	97.87
paper1	53161	52.15	54.29	54.93	88.66	62.02	53.56	88.66
paper2	82199	54.25	54.92	55.35	99.97	59.67	55.24	99.97
progc	39611	50.31	53.00	53.93	85.96	61.76	52.06	99.40
progl	71646	36.93	40.08	41.04	99.76	47.21	38.94	99.76
progp	49379	36.20	39.70	40.80	99.72	48.68	37.97	99.72
trans	93695	30.01	34.98	35.70	90.49	45.81	31.78	99.99

Table 6. Percentage of the size of the compressed files to the size of the original files for different algorithms on text files of the Calgary Corpus for 1-2+RL(1) after BWT

Algorithm 2-2-3+RL(1) This algorithm uses the following scheme instead of Elias encoding. Encode 1 and 2 with "00" and "01", respectively. Encode all other numbers with their binary representations prepended by 1. It also uses run length on "1"s.

File	Size (bytes)	MTF	SBR(0.5)	TS	FC	TR	MTF2	FC′
bib	111261	31.74	32.83	33.32	86.54	38.76	32.78	86.54
book1	768771	48.54	47.29	47.51	93.96	48.94	48.67	94.98
book2	610856	39.16	39.05	39.47	97.93	42.77	39.75	97.93
news	377109	44.63	45.94	46.66	97.89	51.72	45.98	97.89
paper1	53161	44.31	46.15	47.03	88.68	55.53	45.97	88.68
paper2	82199	45.67	46.42	47.02	88.92	52.06	46.78	88.92
progc	39611	42.64	44.85	45.73	83.80	55.28	44.27	86.35
progl	71646	31.09	33.16	33.94	86.96	41.10	32.55	86.96
progp	49379	29.87	32.87	33.80	97.04	43.30	31.53	99.70
trans	93695	26.40	29.71	30.64	88.14	41.64	27.90	92.06

Table 7. Percentage of the size of the compressed files to the size of the original files for different algorithms on text files of the Calgary Corpus for Algorithm 2-2-3+RL(1) after BWT

Modified Huffman Inspired by the fact that there are many blocks of "1"s in our integer sequence we treat them as symbols of our alphabet. Thus our alphabet is $\{1, 2, \cdots, l, 11, 111, \cdots, 1^n\}$,where 1^n means n consecutive "1"s. Then we run Huffman encoding on elements of this alphabet with non-zero frequency. The results are shown in Table 8. Note that we should

also encode the dictionary or Huffman tree in this method. We do not consider this in our computations as it becomes negligible for large files, specially if one considers innovative representation of Huffman code.

File	Size (bytes)	MTF	SBR(0.5)	TS	FC	TR	MTF2	FC′
bib	111261	25.95	26.70	27.22	65.62	32.99	27.04	65.62
book1	768771	32.48	31.60	31.83	56.88	33.42	32.65	56.84
book2	610856	27.60	27.51	27.90	59.90	31.49	28.21	59.91
news	377109	33.32	34.12	34.79	64.50	39.51	34.63	64.58
paper1	53161	32.52	33.74	34.63	59.13	42.40	34.10	59.15
paper2	82199	32.06	32.42	32.98	58.59	37.34	33.00	58.54
progc	39611	32.70	34.24	35.12	61.99	44.46	34.47	64.47
progl	71646	23.03	24.44	25.15	60.24	30.84	24.51	61.68
progp	49379	22.73	24.92	25.78	62.29	34.25	24.74	62.35
trans	93695	20.07	22.63	23.47	65.35	33.11	22.15	71.08

Table 8. Percentage of the size of the compressed files to the size of the original files for different algorithms on text files of the Calgary Corpus for the Modified Huffman algorithm after BWT

According to these results, this schema outperforms all other algorithms in our study. Figure 3 compares other compression algorithms to the modified Huffman algorithm.

5 Conclusions

We have considered a variety of list update algorithms in the context of data compression with and without the Burrows-Wheeler transform. We observed that list update algorithms optimize for a similar but different objective than a compressor and give an example of an algorithm which is a good choice for list update but not for compression. Our experiments showed that competitive list update algorithms are not effective as compressors without BWT, while they perform well after BWT. We also considered several schemas for encoding a sequence of integers that is obtained after applying the list update algorithms. In our study, a Huffman variant which incorporates run length encoding into the alphabet led to the best compression.

References

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

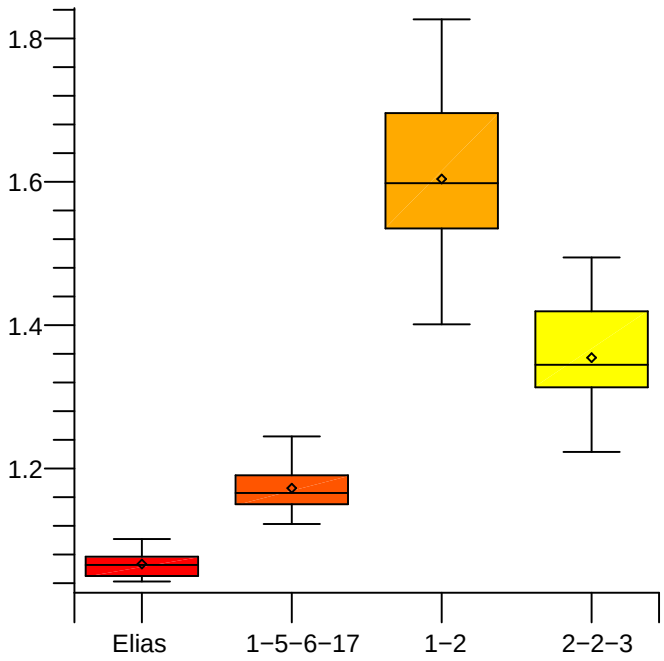

Fig. 3. Comparison of the relative compression ratios between various compression algorithms and modified Huffman. For each file, Modified Huffman equals 1

2. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: List update with locality of reference: Mtf outperforms all other algorithms. Technical Report CS-2006-46, University of Waterloo, School of Computer science (2006)
3. Albers, S., Mitzenmacher, M.: Average case analyses of list update algorithms, with applications to data compression. Algorithmica **21**(3) (1998) 312–329
4. Bachrach, R., El-Yaniv, R., Reinstadtler, M.: On the competitive theory and practice of online list accessing algorithms. Algorithmica **32**(2) (2002) 201–245
5. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. Technical Report 124, DEC SRC (1994)
6. Kaplan, H., Landau, S., Verbin, E.: A simpler analysis of burrows-wheeler based compression. In: Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM '06). Volume 4009 of Lecture Notes in Computer Science. (2006) 282–293
7. Chapin, B.: Switching between two on-line list update algorithms for higher compression of burrows-wheeler transformed data. In: Data Compression Conference. (2000) 183–192
8. Nagy, D.A., Linder, T.: Experimental study of a binary block sorting compression scheme. In: Data Compression Conference. (2003) 439–448

9. Deorowicz, S.: Improvements to burrows-wheeler compression algorithm. Software, Practice, and Experience **30**(13) (2000) 1465–1483
10. Fenwick, P.M.: The Burrows-Wheeler Transform for block sorting text compression: principles and improvements. The Computer Journal **39**(9) (1996) 731–740
11. Balkenhol, B., Kurtz, S.: Universal data compression based on the burrows-wheeler transformation: Theory and practice. IEEE Transactions on Computers **49**(10) (2000) 1043–1053
12. Balkenhol, B., Kurtz, S., Shtarkov, Y.M.: Modifications of the burrows and wheeler data compression algorithm. In: Data Compression Conference. (1999) 188–197
13. Seward, J.: (bzip2, a program and library for data compression. http://www.bzip.org/.)
14. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM **28** (1985) 202–208
15. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM Journal on Computing **27**(3) (1998) 682–693
16. Schulz, F.: Two new families of list update algorithms. In: Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC '98). Volume 1533 of Lecture Notes in Computer Science. (1998) 99–108
17. Witten, I.H., Bell, T.: The Calgary text compression corpus. (Anonymous ftp from ftp.cpsc.ucalgary.ca /pub/text.compression/corpus/text.compression.corpus.tar.Z.)
18. Arnold, R., Bell, T.C.: A corpus for the evaluation of lossless compression algorithms. In: Data Compression Conference. (1997) 201–210
19. Elias, P.: Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory **21**(2) (1975) 194–203
20. Vitter, J.S.: Design and analysis of dynamic Huffman codes. Journal of the ACM **34**(4) (1987) 825–845
21. Vitter: ALGORITHM 673: Dynamic huffman coding. ACMTMS: ACM Transactions on Mathematical Software **15** (1989)