# An Application of Self-Organizing Data Structures to Compression

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

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

**Abstract.** List update algorithms have been widely used as subroutines in compression schemas, most notably as part of Burrows-Wheeler compression. The Burrows-Wheeler transform (BWT), which is the basis of many state-of-the-art general purpose compressors applies a compression algorithm to a permuted version of the original text. List update algorithms are a common choice for this second stage of BWT-based compression. 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 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. [1]. Lastly, we observe that due to an often neglected difference in the cost models, good list update algorithms may be far from optimal for BWT compression and construct an explicit example of this phenomena. This is a fact that had yet to be supported theoretically in the literature.

## 1 Introduction

It has long been observed that list update algorithms can be used in compression. In 1986, Bentley et al. [2] 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. Experimentally their algorithm performs even better achieving compression ratios equal or better than Huffman's. In principle MTF can be replaced with any other online 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 online list update algorithms were compared according to their efficiency in compression by Bachrach et al. [4]. Surprisingly, their results show that some algorithms with bad competitive ratios outperform those that are optimal according to competitive analysis in terms of compression ratio.

A second application of list update is to Burrows and Wheeler compression. The Burrows-Wheeler transform (BWT) rearranges a string of symbols to one of its permutations and in doing so brings the issue of higher order entropy into

play. Then MTF is used to encode this transform in a way similar to the scheme proposed by Bentley et al. [2]. The resulting scheme is shown to be very effective in theory and practice and many improvements and several variants have been proposed [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 [1]. 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. In this paper we systematically study different sensible choices for the list update algorithm as well as for the basic compressor.

*Our Results.* We perform an experimental comparison of the latest list update algorithms for compression, both in stand alone form and as part of BWT based compression. We show that in most cases MTF is the best choice. Additionally we observe 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, a fact that had yet to be reported in the literature.

## 2 Preliminaries

*The List Update Problem.* Consider an unsorted list of $l$ items stored using a linked list. The input is a series of $n$ requests to be served in an online manner. To serve a request to an item $x$, the algorithm should linearly search the list until it finds $x$ at position $i$, for some $i$ between 1 and $l$. The cost of such an access is $i$ units. Immediately after accessing $x$, $x$ can be moved to any position closer to the front of the list at no extra cost. An efficient algorithm should re-arrange the items after each access so as to minimize the overall cost of serving a sequence.

*Standard List Update Algorithms.* Three standard deterministic online algorithms are *move-to-front* (MTF), *transpose* (TR), and *frequency-count* (FC). MTF moves the requested item to the front of the list whereas TR 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 online 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 reorder 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 $t$th access so that items are sorted by their $\alpha$-rank function 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 $r_\alpha(b,t) < r_\alpha(a,t)$. Furthermore, if there is no such item $b$ or this is the first access to $a$, $SBR(\alpha)$ inserts $a$ at the front of the list. Therefore $SBR(0)$ is equivalent to MTF and $SBR(1)$ is equivalent to TS except for 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.

*Compression Schemas.* Bentley et al. [2] proposed using list update algorithms as subroutines in compression. The idea is simple enough: both the encoder and the decoder maintain a list $L$ of all symbols in the file and agree on some online 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 $L$ 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, $\delta$-encoding, and $\omega$-encoding. We refer the reader to [4] for a full description.

*Burrows-Wheeler Transform.* Burrows and Wheeler [5] introduced the idea of a preprocessing phase based on the BWT which is combined with a compression scheme on the transformed 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". The permutation is such that high order entropy is in line with locality of reference. Recall that 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 a nearby position. For a detailed explanation of the BWT we refer the reader to [5, 6].

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

[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 $i$th 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 $i$th position of the list. Other papers that have studied the use of list update algorithms in compression are silent on this issue and apparently simply assumed that competitive list update algorithms are also competitive for compression. We show via an example that this is not necessarily the case, i.e. there exist algorithms which are competitive under one model but not the other. Consider the *move-fraction (MF)* family of deterministic list update algorithms as introduced by Sleator and Tarjan [14]. Upon a request to an item in the $i$th position, MF($k$) moves that item $\lceil i/k \rceil$-1 positions towards the front. MF($k$) is known to be $2k$-competitive [14], therefore algorithm MF(2) is 4-competitive for list update. We show that under the $\Theta(\log i)$ cost model, MF(2) does not have constant competitive ratio. Let the cost of compressing for an item in the $i$th position 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 $i$th 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 $\sigma_1$ is at least $\sum_{i=1}^{p} (c(p - i + 1) + b) = \frac{cp(p+1)}{2} + bp = \Theta(\log^2 l)$. 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 $\sigma_1$. 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 $l$th 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. The same holds for MF($k$) for $k \geq 3$. This fact had been observed empirically by Bachrach et al. [4], who reported on the poor performance of this family for data compression purposes. It remains an open question to determine the competitive ratios of the various list update algorithms under the $c\lfloor \log i \rfloor + b$ cost of access model.

## 4 Experimental Results

We consider two experimental setups. The first one consists of a straightforward compression scheme similar to that of Bentley et al. [2] 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 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 computed the compression ratios achieved by different list update algorithms on files in the Calgary Corpus [17] and the Canterbury Corpus [18]. These are standard benchmarks for data compression. Due to space constraints, we only present the results for the Calgary Corpus; the results for the Canterbury Corpus are similar. We considered the list update algorithms described in Section 2 as well as MTF$'$; this algorithm, on the $i$th 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 frequency-count 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 performed comprehensive experiments on the compression ratios achieved by SBR($\alpha$) for different values of $0 \leq \alpha \leq 1$. These experiments showed that as $\alpha$ goes from 0 to 1, the behaviour of SBR($\alpha$) goes from MTF to TS. Thus we only report the results for SBR(0.5). Due to space constraints these experimental results are not included in this paper. 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. Observe that nonetheless we propose and evaluate other alternative ways for encoding integers.

### 4.2 Comparing List Update Algorithms

We compare the effect of different list update algorithms on text files of the Calgary Corpus before and after BWT. Table 1 shows their performance as stand alone compression algorithms while Table 2 shows their performance as a second stage of BWT compression. 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 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. MTF$'$ 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 MTF$'$ and $SBR(0.5)$) have comparable performance and always outperform FC and TR. The compression ratio they achieve after the BWT is much better than without the BWT, as one would expect given that the BWT increases the amount of locality in the string. The superiority of MTF to other algorithms is consistent with the recent result of Angelopoulos et al. proving that MTF outperforms all other online list update algorithm on sequences with high locality of reference [1]. Hence, this provides evidence that the locality of reference model proposed accurately reflects reality. We emphasize that our focus here is comparing the effect of different list update algorithms

| File | Size (bytes) | MTF | SBR(0.5) | TS | FC | TR | MTF′ | 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 |

**Table 1.** Compression of the Calgary Corpus without BWT

and therefore we have not applied any post-optimizations to the compression scheme, in the presumption that these optimizations are orthogonal and hence would generally benefit all schemes equally.

| File | Size (bytes) | MTF | SBR(0.5) | TS | FC | TR | MTF′ | 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 |

**Table 2.** Compression of the Calgary Corpus after BWT

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 frequently accessed near the beginning and then it is not accessed at all, FC will maintain it close to the front of the list.

### 4.3 Alternative Techniques for Encoding of Integers

We consider other possibilities for the last step of list update based compression schemes, i.e., the prefix-free binary code for integers. As there is considerable locality of reference in the BWTs of text files intuitively a competitive list update

algorithm leads to a sequence with many small integers. These algorithms assign smaller codes to small integers.

*RL(1)+Elias.* This algorithm combines Elias encoding with run length encoding for the value 1, i.e. when the encoded integer is 1, the following Elias-encoded integer shows the number of consecutive 1's starting from that 1. Otherwise, is the next integer encoded in Elias encoding.

| File | Size (bytes) | MTF | SBR(0.5) | TS | FC | TR | MTF′ | 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 3.** Compression of the Calgary Corpus using RL(1)+Elias after BWT

*RL(1)+1-2.* This algorithm encodes 1 with a single bit 0, and encodes 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 | MTF′ | 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 4.** Compression of the Calgary Corpus using RL(1)+1-2 after BWT

*RL(1)+2-2-3:* This algorithm encodes 1 and 2 with "00" and "01", respectively, and encodes 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 | MTF′ | 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 5.** Compression of the Calgary Corpus using Algorithm RL(1)+2-2-3 after BWT

*RL(1)+1-5-6-17:* This algorithm encodes 1 by "0", 2 to 9 by "10000", "10001", ..., "10111", 10 to 17 by "110000", "110001", ..., "110111", and integers greater than 17 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".

| File | Size (bytes) | MTF | SBR(0.5) | TS | FC | TR | MTF′ | 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 6.** Compression of the Calgary Corpus using 1-5-6-17+RL(1) after BWT

Tables 4-7 show the performance of these algorithms on text files of the Calgary Corpus after BWT. According to these results, RL(1)+Elias leads to the

best compression among these algorithms, then RL(1)+5-6-17, then RL(1)+2-2-3, and finally RL(1)+1-2. Comparing Table 3 to Table 2 shows that using RL(1) 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 and RL(1) should be effective. Also according to Tables 4-7, replacing Elias with other proposed integer encodings does not give better compression ratios.

*Modified Huffman.* Inspired by the fact that there are many blocks of "1"s in the 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 Huffman encode the elements of this alphabet. The results are shown in Table 7. Note that we should also encode the Huffman tree. This cost becomes negligible for large files, especially if one considers implicit representations of the portions of the Huffman code corresponding to $1^k$. Indeed the Huffman tree has an impact of in the order of 0.3% uniformly across the different variants for these rather modest file sizes.

| File | Size (bytes) | MTF | SBR(0.5) | TS | FC | TR | MTF′ | FC′ |
|---|---|---|---|---|---|---|---|---|
| bib | 111261 | **26.25** | 27.01 | 27.53 | 65.70 | 33.29 | 27.34 | 65.70 |
| book1 | 768771 | 32.54 | **31.66** | 31.89 | 56.91 | 33.49 | 32.71 | 56.86 |
| book2 | 610856 | 27.70 | **27.61** | 27.99 | 59.93 | 31.58 | 28.30 | 59.94 |
| news | 377109 | **33.44** | 34.25 | 34.91 | 64.55 | 39.64 | 34.75 | 64.63 |
| paper1 | 53161 | **32.96** | 34.17 | 35.06 | 59.46 | 42.78 | 34.51 | 59.48 |
| paper2 | 82199 | **32.39** | 32.75 | 33.30 | 58.72 | 37.65 | 33.33 | 58.67 |
| progc | 39611 | **33.21** | 34.76 | 35.64 | 62.38 | 44.96 | 34.99 | 64.78 |
| progl | 71646 | **23.43** | 24.82 | 25.53 | 60.39 | 31.22 | 24.91 | 61.83 |
| progp | 49379 | **23.22** | 25.40 | 26.26 | 62.51 | 34.74 | 25.24 | 62.56 |
| trans | 93695 | **20.42** | 22.99 | 23.84 | 65.59 | 33.45 | 22.51 | 71.19 |

**Table 7.** Compression of the Calgary Corpus using Modified Huffman after BWT

According to these results, this schema outperforms all other algorithms in our study. Figure 1 reports the mean, median and variance of the comparison of other compression algorithms to the modified Huffman algorithm.

### 4.4 Splay Trees

List update algorithms belong to the area of self-organizing data structures. Another well known self-organizing data structure is the splay tree [22]. The splay tree is a binary search tree which applies a splay operation after each access to an item. This operation reorganizes the tree such that the most recently accessed item is moved to the root of the tree. Splay trees are believed to have good performance on sequences with high locality of reference. The working

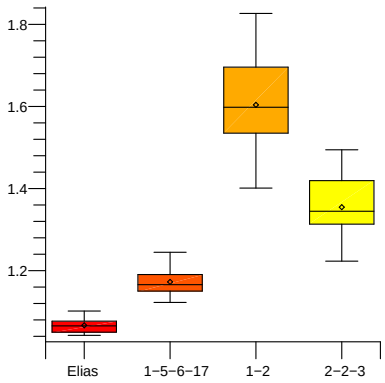


**Fig. 1.** Relative compression ratio versus modified Huffman. For each file, Modified Huffman equals 1

set theorem of [22] shows that splay trees have the working set property. The working property is based on the idea that an operation on a recently accessed item should take less time. Informally, a structure has the working set property if it performs well on sequences with high locality of reference. As stated before there is usually high locality of reference in texts (especially after applying BWT) and thus splay trees are good candidates for text compression. Jones [23] and Grinberg et al. [24] have already studies the application of splay trees to data compression, but they did not consider the BWT.

We studied the effect of using splay trees instead of list update algorithms in our compression schemas. We constructed a splay tree on the characters of the text file. Each character corresponds to a node of the tree and has a binary code that corresponds to the path from the root to its node, i.e., starting from the root, append 0 for each left traversal and 1 for each right traversal. Note that as we proceed with the compression process, the tree changes dynamically and thus the codes for characters are changing as well. Since characters can be in internal nodes, the corresponding codes are not prefix-free. To obtain a prefix-free code, we first add a single 1 to the beginning of each code. Then we consider the number that corresponds to this binary representation and encode these integers using Elias encoding. Note that the code for the root character would be 1.

| File | Size (bytes) | Elias | RL(1)+Elias | Modified Huffman |
|---|---|---|---|---|
| bib | 111261 | 37.76 | 35.14 | 31.43 |
| book1 | 768771 | 44.91 | 44.94 | 40.40 |
| book2 | 610856 | 38.53 | 37.12 | 33.75 |
| news | 377109 | 46.05 | 45.35 | 40.49 |
| paper1 | 53161 | 43.63 | 43.53 | 38.94 |
| paper2 | 82199 | 43.71 | 43.44 | 38.97 |
| progc | 39611 | 44.14 | 43.95 | 39.06 |
| progl | 71646 | 32.14 | 29.98 | 27.43 |
| progp | 49379 | 31.34 | 29.21 | 26.63 |
| trans | 93695 | 28.92 | 25.71 | 23.32 |

**Table 8.** Compression of the Calgary Corpus using splay trees after BWT

We can also apply alternative techniques for encoding integers proposed in Subsection 4.3. We tested the RL(1)+Elias and the modified Huffman techniques. The compression percentages obtained by applying these schemas to the text files of the Calgary Corpus after BWT are shown in Table 8. According to these results, the modified Huffman algorithm is again the best technique for encoding integers. Furthermore, the splay trees lead to less compression compared to the good list update algorithms.

## 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. Furthermore, we experimentally tested the efficacy of splay trees in data compression and observed that they are not as effective as list update algorithms.

## References


1. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: List update with locality of reference. In: Proceedings of the 8th Latin American Symposium on Theoretical Informatics (LATIN '08). (2008) 399–410
2. 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
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: Proc. 17th Annual Symp. on Comb. Pattern Matching (CPM '06). Volume 4009 of LNCS. (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) (June 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 LNCS. (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)
22. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. Journal of the ACM **32**(3) (1985) 652–686
23. Jones, D.W.: Application of splay trees to data compression. Communications of the ACM **31**(8) (1988) 996–1007
24. Grinberg, D., Rajagopalan, S., Venkatesan, R., Wei, V.K.: Splay trees for data compression. In: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms (SODA '95). (1995) 522–530