Context-Based Algorithms for the List-Update Problem under Alternative Cost Models*

Shahin Kamali[a], Susana Ladra[b], Alejandro López-Ortiz[a], and Diego Seco[bc]

[a] *Cheriton School of Computer Science, University of Waterloo, Canada, {s3kamali,alopez-o}@uwaterloo.ca*
[b] *Database Laboratory, University of A Coruña, Spain, sladra@udc.es*
[c] *Department of Computer Science, University of Concepción, Chile, dseco@udec.cl*

Abstract: The List-Update Problem is a well studied online problem with direct applications in data compression. Although the model proposed by Sleator & Tarjan [16] has become the standard in the field for the problem, its applicability in some domains, and in particular for compression purposes, has been questioned. In this paper, we focus on two alternative models for the problem that arguably have more practical significance than the standard model. We provide new algorithms for these models, and show that these algorithms outperform all classical algorithms under the discussed models. This is done via an empirical study of the performance of these algorithms on the reference data set for the list-update problem. The presented algorithms make use of the context-based strategies for compression, which have not been considered before in the context of the list-update problem and lead to improved compression algorithms. In addition, we study the adaptability of these algorithms to different measures of locality of reference and compressibility.

1 Introduction

The list-update problem is a classical online problem that has been extensively studied in the past decades due to its applicability for various data structures and algorithms. The input to the problem is a set of items stored in a list, and a sequence of requests to access said items. This sequence is revealed in an online manner, i.e., upon serving the t-th request, the algorithm does not know any of the requests coming after t. In practice, input sequences have *locality of reference*, i.e., the items that have been recently requested are more likely to be requested again. To serve a request, an algorithm needs to probe all items in the list, starting from the front of the list, up until accessing the requested item. The cost of accessing the i-th item in the list (called *access cost*) is linear in i. A list-update algorithm can rearrange the list so that the items that are more likely to be requested appear closer to the front. Different models for the problem assume different costs for reordering items in the list. In the standard model proposed by Sleator & Tarjan [16], after an access to an item in the i-th position (which has an access cost of i), the algorithm can move the requested item closer to the front using a *free exchange* at no additional cost. It can also swap the positions of two consecutive items in the list with a *paid exchange*, at a unit cost.

*This work was supported in part by Xunta de Galicia (co-funded with FEDER), ref. 2010/17 and CN 2012/211, and by MICINN grant (PGE/FEDER) TIN2009-14560-C03-02 for S.L. and D.S.

Although many algorithms have been proposed and analyzed for the standard cost model, the validity of the assumptions behind this model has been questioned, and consequently other models have been proposed. In particular, Martínez and Roura [12], and Munro [13], independently, proposed an alternative model called the MRM model. This model is arguably more realistic for practical scenarios like maintaining dictionaries. In [7], it is argued that the standard model is not suitable for compression purposes, and an alternative model is suggested. In our opinion, the prevalence of the standard model over all other models has precluded the study of list-update algorithms that do not perform well in the standard model, but may outperform known ones in some actual practical domains.

In this paper, we provide efficient algorithms for the list-update problem under alternative cost models. Based on the ideas in [7], we introduce a compression model for the list-update problem, which provides a suitable framework for applying list-update algorithms for compression purposes. The main contribution of this paper is to present list-update algorithms that outperform all previous algorithms under the MRM and compression models. In contrast with previous list-update algorithms, which avoid reordering items in the list (in order to save on paid exchanges under standard model), our algorithms effectively rearrange the list after each request based on the patterns observed between the previously served items. In providing these algorithms, we have been inspired by the context-based symbol ranking algorithms [9, 11] for data compression, which constitute the base of the outstanding compression technique PPM [5]. To the best of our knowledge, these techniques have not been studied in the context of the list-update problem (probably because of their poor performance under the standard model due to the high cost of list rearrangements).

The efficiency of online algorithms is usually studied using *competitive analysis*, which measures the *competitive ratio* between the cost of the algorithm and that of an offline optimum with unbounded computational power. In [13], it is proved that no online algorithm can have a constant competitive ratio under the MRM model. In Section 1.1 we extend this result to the compression model. We take an experimental approach to evaluate several list-update algorithms, including the ones proposed in the paper, under alternative cost models. Our empirical study shows that context-based algorithms perform well under both MRM and compression models, and confirms that the study of list-update algorithms in different models is worthwhile and may open a door to improvements in practice. We also consider the relation between the list-update problem and locality of reference, which has been addressed in [1, 2]. We push a bit further on this direction, and study the adaptability of several list-update algorithms to different measures of locality of reference, entropy, and compressibility. Our experiments show a high correlation between context-based algorithms and the T-entropy [18, 17].

1.1 Alternative List-Update Models

In this section, we review the MRM model, and also formally define the compression model that was implicitly defined in [7].

MRM Model. As mentioned earlier, the standard model is not realistic in some practical settings. This is particularly the case when the list is represented by a linked list (e.g., in order to maintain a dictionary data structure). To see that, consider a scenario in which an algorithm accesses the item at the end of the list and then reverses the entire list. In practice, reordering items takes time linear on the size of the list, while under the standard model this has quadratic cost. The MRM model assumes that, after an access to the item of the list at position i, all items up to position i can be rearranged at cost proportional to i; while rearranging items at positions after i requires paid exchanges.

Compression Model. List-update algorithms have proven to be efficient solutions for data compression schemes [3], particularly in combination with the Burrows-Wheeler transform [4] where compression ratios compare favourably with Lempel-Ziv based schemes. Consider a text that needs to be encoded, and let each character of the text alphabet be an item in the list, while the whole text forms the input sequence. A list-update algorithm $\mathcal{A}$ can work as a compression algorithm as follows. To compress the text, $\mathcal{A}$ writes the access cost for serving each character in the compressed file. Decompression is similar by following the algorithms' steps (starting from the same initial configuration). Since the compression algorithm only writes the index of the accessed item, under a desirable compression model for list-update, an algorithm only pays for the access costs and not for rearrangements of the list. Note that if we have a randomized algorithm, in order to follow the same steps as the compressor, the random bits used by the algorithm are required to be written in the compressed file. We propose the following definition.

Definition 1 *In the list-update problem under the compression model, the cost of an algorithm for serving a request to the i-th item of the list is equal to i. After serving each request, the whole list can be rearranged free of charge. In case of a randomized algorithm, the cost is increased by the number of random bits used by the algorithm.*

Hence, the compression model is the same as the standard model except that there is no cost for paid exchanges. It is also similar to the MRM model except that it can rearrange the *whole* list, rather than the first i items, free of charge. Consequently, all the algorithms defined for the standard model are well-defined for the MRM and compression models. The above definition of the compression model implies that the position of each item in the list is written in unary format in the compressed file. This can be replaced by a binary code, hence charging $\log i$ for accessing an item in the i-th position. This variant is considered in [7], in which a cost of $c\lfloor \log i \rfloor + b$ is charged, where b and c are constants. In our experiments, we consider both variants of the compression model, and observe that our algorithm performs well in both cases.

If an algorithm has cost $\mathcal{C}$ under the MRM model, it is well-defined and has cost at most $\mathcal{C}$ under the compression model (since the costs of paid exchanges under the MRM model are relaxed under the compression model). In [13], an offline algorithm is introduced for the MRM model, which has a cost of at most $n \log l$ for serving any sequence of size n for a list L of l items, and consequently a cost of at most $n \log l$ under the compression model. Thus, the cost of the best offline algorithm is not more than $n \log l$ for serving any sequence of size n. For any online algorithm, there are

sequences that have total access cost of nl (a cruel adversary asks for the last item of the list in each request). As a result, the competitive ratio of any online algorithm is lower bounded by $l/\log l$.

Proposition 1 *Under the compression model, there is no online algorithm with a constant competitive ratio.*

This implies that online algorithms under the compression model using competitive analysis do not compare well since all ratios are non-constant for large lists. Instead, we apply an experimental approach to compare the algorithms.

1.2 List-Update Algorithms

For serving an item x, a list-update algorithm should linearly probe the list to find x. After this access, the online algorithm can rearrange the list. Different online algorithms are distinguished by their respective policies for reordering the list. Although the standard model allows paid exchanges, almost all previously studied algorithm use free exchanges only. Here we list a few of them. Note that all these algorithms are well-defined and have the same costs under both MRM and compression models.

Move-To-Front (MTF) moves the requested item to the front of the list.

Transpose (TR) exchanges the requested item with its immediate predecessor.

Frequency Count (FC) maintains an access count for each item ensuring that the list is always sorted in non-increasing order of frequency of access.

Timestamp (TS) inserts the requested item x in front of the closest item to the head y that precedes x in the list and was requested at most once since the last request for x. If there is no such item y (or if this is the first access to x), TS does not reorganize the list.

BIT maintains a bit (initialized independently and uniformly at random) for each item. This bit is complemented and, if its value changes to 1, the item is moved to the front of the list. Note that unlike previous algorithms, BIT is randomized. This implies that, when the algorithm is used for more than one agent in a synchronized fashion (for example, for compression), the random source needs to be shared by both agents as well as the initial configuration.

COMB executes BIT with probability 4/5 and TS with probability 1/5. Obviously, this is also a randomized algorithm.

2 Context-based Algorithms for List-Update

In this section we propose new algorithms for the list-update problem under the compression and MRM models. These algorithms are inspired by previous works that use contexts for symbol ranking, which have been generally applied for data compression [9, 11]. We first describe the basic algorithm, named *Context Based (CB)*, which is designed for the compression model and rearranges the whole list after each request. Note that under the MRM model rearranging the whole list is not free. We present a slightly modified version of CB, called *Restricted Context Based*

(RCB) for the MRM model. Our experiments in Section 3 indicate that CB and RCB outperform other list-update algorithms under the compression and MRM models.

Let $L = \{a_1, a_2, \ldots, a_l\}$ be the set of l items in the list. We call any subsequence of size c of members of L a *c-context* of L. The idea behind algorithm CB is to keep track of the frequencies of all possible c-contexts for any value of $c, 1 \leq c \leq C$. Here C is a parameter of the algorithm that is set according to the locality of the input sequence. Given the frequencies of the c-contexts for the first $t-1$ requests of an input sequence, the algorithm predicts the likeliness of each item of the list to be the t-th item, and rearranges the list according to that. For example, let $C = 4$ and assume the last three requests have been to items a, b, and c, in that order. Now if the 4-context *abcb* has been more frequent than *abca* in the previous requests, the algorithm assumes it is more likely to have b in the next request, and arranges items so that b appears before a. In other words, the algorithm assumes that the frequent patterns in the previous requests are more likely to appear in the future (the parameter C bounds the maximum length of such patterns). Note that this assumption is stronger than a simple locality assumption by algorithms like MTF. Our experiments verify that the assumption is valid for real-world sequences.

To keep track of the context frequencies, the algorithm maintains a *context tree*, which is a weighted tree of degree l and height $C + 1$ (See Figure 1). The values of the nodes in level $c + 1$ indicate the frequencies of the c-contexts in previous requests ($1 \leq c \leq C$). Right after serving the t-th request, the value of C of the contexts should be increased by one unit. These are the c-contexts that include the last $c - 1$ items as their prefix ($1 \leq c \leq C$). For example, when $C = 4$, if the last three requests have been respectively to a, b, and c, and the t-th request is to a, then the frequencies of contexts a, ca, bca, and $abca$ should be increased (this requires updating up to C branches of the tree).

The algorithm looks at the last $C-1$ requests and at the context tree to rearrange the items according to their likeliness to appear as the next item. More precisely, for each item x in the list, the algorithm uses the context tree to find the frequency of the C-context if x were to follow as next character in the sequence, i.e. the frequency of the C-context that is formed by concatenating the last $C-1$ requests with x. The list is then rearranged according to the frequencies of these l C-contexts. If the C-contexts formed by two items x and y have the same frequencies, the algorithm compares the frequencies of their $(C-1)$-contexts, and again in case of equal frequencies, compares the frequencies for $(C-2)$-contexts, and so on. In an extreme case, the algorithm might need to look at the frequencies of the 1-contexts for x and y, which are indeed the number of requests to these items (if these are also equal, the relative order of x and y is set in some arbitrary but previously agreed fashion, e.g. alphabetically).

To summarize, the algorithm performs three steps to serve a request to item x. In the first step, it linearly probes the list to find x. In the second step, it updates (increases) the frequencies of the contexts that include x as their last item in the context tree. In the last step, the algorithm rearranges the list by looking at the last $C - 1$ requests, and the (updated) frequencies in the context tree. A straightforward time analysis shows that the time complexity of the algorithm is $O(n\, l \log l)$ for constant values of C. Figure 1 illustrates the details of these steps.

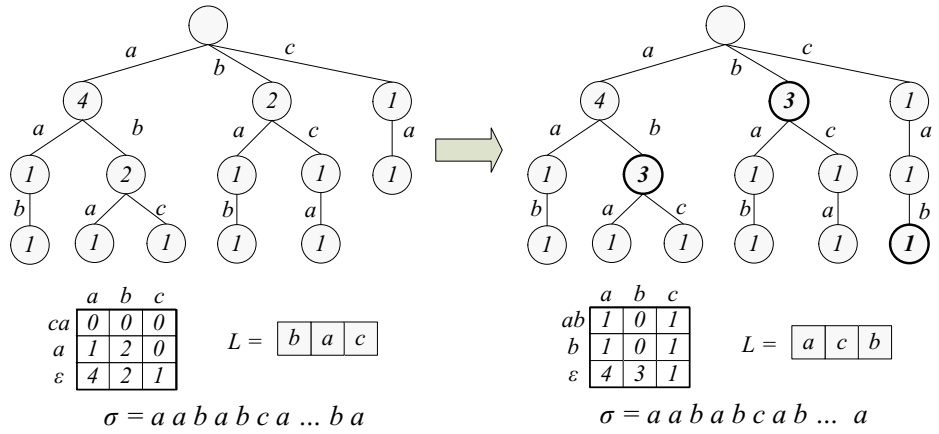


Figure 1: The context tree and list reordering of CB with $C = 3$ for a sequence $\sigma = aababcaba$. (a) After accessing seven items, the algorithm updates the context tree to achieve the tree on the left. To rearrange items, the algorithm looks at the last two requests (i.e., c and a), and checks the frequencies of the 3-contexts caa, cab, and cac. Since they all have the same frequencies (i.e., 0), CB compares the frequencies of the 2-contexts aa, ab, and ac, which respectively have frequencies 1, 2, and 0. Consecutively the algorithm rearranges the list as b, a, c. (b) After serving the eighth item, the context tree is updated (bolded nodes). To rearrange the items, the algorithm looks at the last two requests (i.e., a and b), and forms the 3-contexts aba, abb, and abc. The frequencies of these 3-contexts imply that a and c should appear before than b. To find the relative order of a and c, the algorithm checks the frequencies of ba and bc (which are both 1), and eventually those of 1-contexts a and c (which are consecutively 1 and 4). Consequently, the items are reordered as a, c, b.

We can modify Step 3 of the algorithm to obtain different variations of this context-based approach. Specifically, instead of a complete rearrangement of the list according to the symbol ranking explained above, we can just sort the items of the list that appear before the requested item. We call this variant *Restricted Context Based (RCB)*, and as we will see, it outperforms other list-update algorithms under the MRM model. In particular, since the complete rearrangement of the list might be too costly under the MRM model, this restricted variation of the context-based algorithm has an advantage over CB, under the MRM model.

In the experimental evaluation we also consider two modifications of the described algorithms that do not use the cumulative frequency of the contexts. Instead, they use the timestamp of the last occurrence of the contexts to rearrange the list. These algorithms sort the list by recency of the contexts rather than their frequency. We call these algorithms CB$'$ (when the whole list is rearranged) and RCB$'$ (when the items located before the requested item are rearranged). Other variants of the algorithm can be implemented, e.g., using a time window to consider local context information.

3 Experimental Evaluation

This section presents an experimental evaluation of the proposed context-based algorithms for the list-update problem under different cost models. Our results demonstrate that using the context-based algorithms is an efficient strategy under the compression cost model, as well as the MRM model. We used the reference dataset for

the list-update problem, that is, the Calgary text compression corpus. In Section 3.1 we perform an extended comparison of several methods under different cost models. We analyze the compression ratios obtained by context-based algorithms and their relation with locality of reference and different entropy measures in Section 3.2.

3.1 Total Costs under Different Models

We evaluate the performance of two of the most efficient algorithms (MTF and TS) under different cost models and compare their results with the two context-based algorithms proposed in this paper (CB and RCB). The parameter C of these algorithms is fixed to be 3. Table 1 shows the results for some of the files of the Calgary corpus.

	book1						
	Free	Paid	AC	STD	MRM	CC_linear	CC_log
MTF	9,002,657	0	9,771,428	9,771,428	9,771,428	9,771,428	5,155,343
TS	3,215,445	0	8,326,983	**8,326,983**	8,326,983	8,326,983	4,653,983
CB	907,829	351,564,554	3,216,242	354,780,796	52,941,414	**3,216,242**	**2,542,883**
RCB	2,415,021	19,399,435	7,373,290	26,772,725	**7,373,290**	7,373,290	4,008,491
	news						
	Free	Paid	AC	STD	MRM	CC_linear	CC_log
MTF	6,033,483	0	6,410,592	6,410,592	6,410,592	6,410,592	2,669,833
TS	2,324,979	0	5,788,482	**5,788,482**	5,788,482	5,788,482	2,479,925
CB	967,786	357,520,313	2,271,967	359,792,280	32,919,516	**2,271,967**	**1,384,551**
RCB	2,363,356	24,002,408	5,141,341	29,143,749	**5,141,341**	5,141,341	2,128,293
	progc						
	Free	Paid	AC	STD	MRM	CC_linear	CC_log
MTF	645,069	0	684,680	684,680	684,680	684,680	281,197
TS	262,363	0	632,842	**632,842**	632,842	632,842	264,907
CB	78,196	26,575,729	223,937	26,799,666	3,010,434	**223,937**	**129,079**
RCB	159,749	1,580,853	529,991	2,110,844	**529,991**	529,991	213,139
	trans						
	Free	Paid	AC	STD	MRM	CC_linear	CC_log
MTF	1,531,585	0	1,625,280	1,625,280	1,625,280	1,625,280	656,437
TS	651,001	0	1,548,988	**1,548,988**	1,548,988	1,548,988	634,037
CB	129,422	69,476,148	380,704	69,856,852	7,339,262	**380,704**	**264,085**
RCB	485,861	4,937,879	1,344,826	6,282,705	**1,344,826**	1,344,826	522,627

Table 1: Performance of MTF, TS, CB, and RCB algorithms over several files of the Calgary corpus under different cost models. Legend: Free = free exchanges; Paid = paid exchanges; AC = access cost; STD = cost in the standard model (access cost + paid exchanges); MRM = cost in the MRM model (access cost + some paid exchanges); CC_linear = cost in the compression model (when accessing i costs i); CC_log = cost in the compression model (when accessing i costs $1 + 2\lfloor \log i \rfloor$). We use bold type to highlight the best values for the column of each cost model.

We observe that MTF and TS obtain the best results for the standard model, while the context-based algorithms perform poorly, as they require many paid exchanges for the rearrangement of the list. In contrast, for the alternative cost models, the context-based algorithms achieve interesting results. CB outperforms other methods for the compression model, since it exploits the context information to completely reorder the list at free cost. RCB is the preferred choice for the MRM model, as it uses the cost-free reordering of the list up to the accessed position. One can conclude that the context-based algorithms, which might have been originally discarded for list-update due to their unsatisfactory performance under the standard model, become attractive practical solutions for the problem under alternative models.

3.2 Compression Ratio, Locality of Reference and Entropy

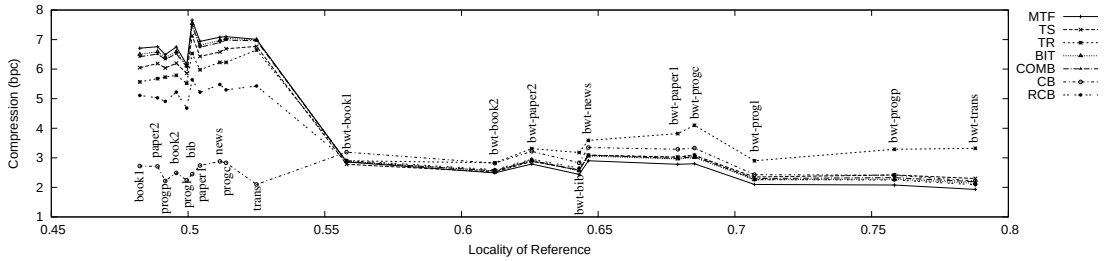
We compare the performance of the proposed context-based list-update algorithms on text files of the Calgary Corpus before and after the *Burrows-Wheeler Transform (BWT)* [4]. The BWT produces a permutation of the sequence with a high number of runs of repeated symbols, and it is commonly used as a first stage of data compression methods. MTF strategy is known to obtain good results after BWT [6, 7], and it is commonly used as a second stage after BWT [3]. Table 2 shows the comparison of our context-based algorithms (CB, CB′, RCB, and RCB′) with MTF. The results reported for the context-based algorithms are obtained when using a maximum context length of $C = 10$ before BWT, and a maximum context length of $C = 3$ after BWT. Since the goal is to compare these algorithms, and not to necessarily obtain the best compression, we simply represent the obtained sequence using the standard prefix integer encoding of Elias [8] that encodes an integer i using $1 + 2\lfloor \log i \rfloor$ bits.

File	Size (bytes)	Before BWT					After BWT				
		MTF	CB	CB′	RCB	RCB′	MTF	CB	CB′	RCB	RCB′
bib	111261	95.69	**29.78**	30.47	70.44	72.16	30.49	34.04	36.03	32.90	32.24
book1	768771	83.82	**34.15**	35.75	63.99	65.97	35.74	38.66	40.22	36.37	36.21
book2	610856	84.35	**29.97**	30.54	65.00	65.39	31.14	34.08	35.87	32.32	32.20
geo	102400	104.91	76.69	80.46	99.43	104.37	50.78	47.87	51.79	**47.13**	48.53
news	377109	88.50	**35.05**	35.72	68.31	69.01	36.21	39.85	43.16	38.25	38.47
obj1	21504	89.99	59.38	57.39	80.40	76.11	**43.75**	46.02	49.04	45.38	44.66
obj2	246814	101.68	36.72	34.81	88.20	79.39	**28.06**	30.29	32.49	29.34	29.25
paper1	53161	86.79	**33.64**	34.21	65.11	66.82	34.70	39.44	41.93	37.68	37.08
paper2	82199	84.47	**33.50**	34.62	62.83	65.35	34.86	38.43	41.06	36.52	36.35
pic	513216	23.21	**19.54**	20.14	21.55	21.78	20.08	19.77	21.07	19.60	19.84
progc	39611	88.74	34.46	**34.34**	66.28	66.28	35.04	40.01	42.20	38.48	37.23
progl	71646	77.01	26.08	**25.71**	58.15	57.58	26.31	29.29	31.36	28.02	27.80
progp	49379	81.09	26.32	**25.90**	61.23	59.90	26.00	29.20	30.91	28.05	27.70
trans	93695	87.58	24.35	24.31	65.63	65.25	**24.12**	26.92	28.76	26.02	25.78

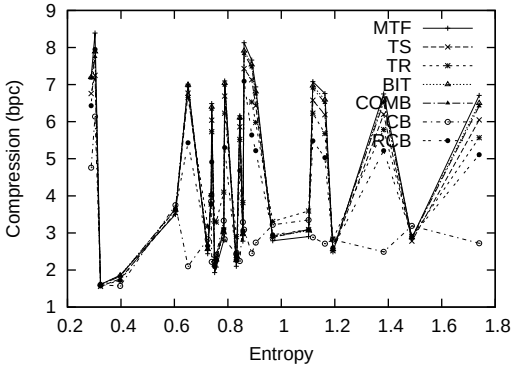
Table 2: Compression percentage of text files of the Calgary Corpus using different list-update algorithms. We use bold type to highlight the best values for each file.

We observe that context-based algorithms obtain the best performance for the original dataset (especially CB algorithm). This is consistent with the results obtained by compression methods that exploit the same idea as PPM [5]. MTF obtains the best results after BWT, proving their combined efficiency for data compression, such in bzip2 [15]. However, CB competes with BWT+MTF and achieves better compression ratios for some files of the dataset. In addition, in pure online scenarios where the sequence of requests is not known in advance and it is not feasible to compute the BWT of the sequence, CB may become the preferred choice.

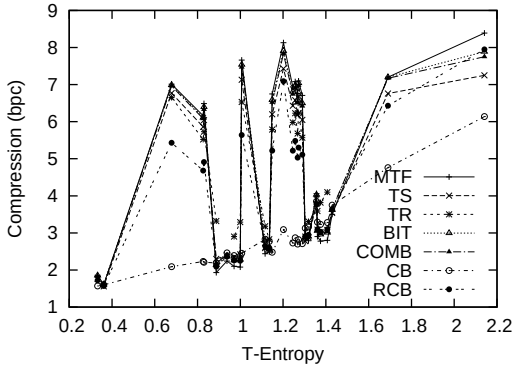
To improve our understanding of these results, we relate the behavior of the list-update algorithms with some properties of the sequence such as its locality of reference or its entropy. All previous works on this topic conclude that MTF is the best strategy when the locality of reference is high [1]. The relation between entropy, compression, and list-update algorithms has been previously studied [14]. We analyze the behavior of the proposed context-based algorithms depending on the k-th order entropy and the *T-entropy* of the sequence [18, 17]. We show that the T-entropy is a good performance characterizer for the use of context-based algorithms in the list-update problem.



(a) Compression vs. locality of reference (as defined by Albers).



(b) Compression vs. k-th order entropy ($k = 4$).



(c) Compression vs. T-entropy.

Figure 2: Compression vs. different entropy measures.

Figure 2a shows the compression ratio (in bits per character) for the ten ASCII files of the Calgary Corpus before and after the BWT transform, and its relation with the locality of reference of the text, as defined by Albers [1]. We observe that most of the list-update algorithms behave differently depending on the locality of reference of the text. For the original text, we obtain higher values than for the text transformed with BWT. After BWT, the best values are obtained by MTF. From the original text, the best values are obtained by CB, whose behavior is not significantly affected by the locality of reference of the input text.

Figures 2b and 2c show different entropy measures of the input sequences (k-th order empirical entropy and T-entropy). As we can see from the figures, there seems to be no relation between the performance of list-update algorithms and k-th order entropy. However, there is relation between T-entropy and the performance of the CB algorithm. This measure, proposed for finite strings, and similar to the LZ production complexity [10], measures the effective number of steps required to build a string from an alphabet based on the decomposition of the string into constituent patterns, determined by a recursive hierarchical pattern copying algorithm. For lack of space, we refer the reader to [18, 17] for details of this measure.

4 Conclusions

The list-update problem has been extensively studied in the past decades, especially under the standard model proposed by Sleator & Tarjan [16]. In the last years al-

ternative cost models have been proposed to address the drawbacks of the standard model in practical settings such as data compression. In this paper, we studied the applicability of context-based algorithms for the list-update problem under three alternative cost models. Specifically, we proposed a Context Based (CB) algorithm that outperforms all existing algorithms under the compression cost model. These results are consistent with the existing results on applying the context information to data compression [5, 9, 11]. More strikingly, we introduced a similar context-based algorithm, called Restricted Context Based (RCB), which outperforms all other algorithms under the MRM model (which is the model realized in practice for list accessing purposes). We also analyzed how the locality of reference and several entropy measures are related to the performance of these algorithms, and observed that the T-entropy is a good characterizer of the behavior of the context-based algorithms.

References

[1] Albers, S., Lauer, S.: On list update with locality of reference. In: Procs. of ICALP. pp. 96–107 (2008)

[2] Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: List update with locality of reference. In: LATIN. pp. 399–410 (2008)

[3] Bentley, J.L., Sleator, D.D., Tarjan, R.E., Wei, V.K.: A locally adaptive data compression scheme. Commun. ACM 29(4), 320–330 (1986)

[4] Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. Tech. Rep. 124, Digital Equipment Corporation (1994)

[5] Cleary, J.G., Witten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Transactions on Communications 32, 396–402 (1984)

[6] Dorrigiv, R., López-Ortiz, A., Munro, J.I.: List update algorithms for data compression. In: Procs. of DCC. p. 512 (2008)

[7] Dorrigiv, R., López-Ortiz, A., Munro, J.I.: An application of self-organizing data structures to compression. In: Procs. of SEA. pp. 137–148 (2009)

[8] Elias, P.: Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory 21(2), 194 – 203 (1975)

[9] Howard, P., Vitter, J.: Design and analysis of fast text compression based on quasi-arithmetic coding. In: Procs. of DCC. pp. 98 –107 (1993)

[10] Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE Transactions on Information Theory 22, 75–81 (1976)

[11] Manzini, G.: Efficient algorithms for on-line symbol ranking compression. In: Procs. of ESA. pp. 694–694 (1999)

[12] Martínez, C., Roura, S.: On the competitiveness of the move-to-front rule. Theoretical Computer Science 242(1-2), 313–325 (2000)

[13] Munro, J.I.: On the competitiveness of linear search. In: Procs. of ESA. pp. 338–345 (2000)

[14] Pandurangan, G., Upfal, E.: Can entropy characterize performance of online algorithms? In: Procs. of SODA. pp. 727–734 (2001)

[15] Seward, J.: bzip2, `http://www.bzip.org`

[16] Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28, 202–208 (1985)

[17] Titchener, M.R.: A measure of information. In: Procs. of DCC. pp. 353–362 (2000)

[18] Titchener, M.: Deterministic computation of complexity, information and entropy. In: Procs. of ISIT. pp. 16–21 (1998)