On the competitive theory and practice of online list accessing algorithms

Ran Gilad-Bachrach * Ran El-Yaniv † Martin Reinstädtler ‡

November, 1999

Abstract

This paper concerns the online list accessing problem. In the first part of the paper we present two new families of list accessing algorithms. The first family is of optimal, 2-competitive, deterministic online algorithms. This family, called the MRI (MOVE-TO-RECENT-ITEM) family, includes as members the well known MOVE-TO-FRONT (MTF) algorithm, and the recent, more "conservative" algorithm TIMESTAMP due to Albers. So far MOVE-TO-FRONT and TIMESTAMP were the only algorithms known to be optimal in terms of their competitive ratio. This new family contains a sequence of algorithms $\{A(i)\}_{i\geq 1}$ where $A(1)$ is equivalent to TIMESTAMP and the limit element $A(\infty)$ is MTF. Further, in this class, for each i, the algorithm $A(i)$ is more conservative than algorithm $A(i+1)$ in the sense that it is more reluctant to move an accessed item to the front, thus giving a gradual transition from the conservative TIMESTAMP to the "reckless" MTF. The second new family , called the PRI (PASS-RECENT-ITEM) family is also infinite and contains TIMESTAMP; We show that most algorithms in this family attain a competitive ratio of 3.

In the second, experimental part of the paper, we report the results of an extensive empirical study of the performances of a large set of online list accessing algorithms (including members of our MRI and PRI families). The algorithms' access cost performances were tested with respect to a number of different request sequences. These include sequences of independent requests generated by probability distributions and sequences generated by Markov sources to examine the influence of locality. It turns out that the degree of locality has a considerable influence on the algorithms' absolute and relative costs, as well as on their rankings. In another experiment, we tested the algorithms' performances as data compressors in two variants of the compression scheme of Bentley *et al.* In both experiments, members of the MRI and PRI families were found to be among the best performing algorithms.

*Institute of Computer Science, The Hebrew University, email: ranb@cs.huji.ac.il

†Department of Computer Science, Technion - Israel Institute of Technology, email: rani@cs.technion.ac.il

‡Max-Planck-Institut für Informatik, Saarbrücken, email: marei@mpi-sb.mpg.de

1 Introduction

In the *list accessing problem* an online algorithm maintains a set of items as an unsorted linear list. A sequence of requests for items on the list is given and the algorithm must serve these requests in the order of their arrival. Upon a request for an item x, the algorithm must access x by searching for x starting from the front of the list. In a standard model of this problem the cost associated with accessing the ith item from the front is i. The algorithm may reorganise the list at any time. The incentive for such reorganization is to decrease the cost of future accesses. Reorganization is done via a sequence of transpositions of consecutive items. After an item x is accessed, x may be moved free of charge to any position closer to the front. Thus the transpositions that bring x forward after an access are called *free exchanges*. Any other transposition costs 1 and is called *paid*.

The above variant of the list accessing problem is called the *static* model. The *dynamic* model deals with a dynamic list where new items may be inserted and old items may be deleted (or accessed). Unless otherwise is specified we usually assume the static model.

The importance of the list accessing problem arises from the fact that online list accessing algorithms are often used by practitioners. Although organizations of dictionaries as linear lists are relatively inefficient, there are various situations in which a linear list is the implementation of choice. For instance, when the dictionary is small (say for organising the list of identifiers maintained by a compiler, or for organising collisions in a hash table), or when there is no space to implement efficient but space consuming data structures, etc. [6, 11]. In addition, any list accessing algorithm can be used as the "heart" of a data compression algorithm [7, 2]. Due to its great relevancy the list accessing problem has been studied since 1965 (see e.g. [15, 19, 6, 21, 17, 3]). Nevertheless, despite these extensive studies, this simple-to-state problem is not yet well understood. In particular, the basic question of which are the best list accessing algorithms and for which inputs, remains unsolved except for two extreme cases, where inputs are independent observations of a probability distribution, or when inputs are generated by an adversary aiming to maximize the multiplicative "regeret" of the algorithm (i.e. its competitive ratio, see below). Each of these models implies a different notion of optimality, and the work done so far in this area identified many interesting optimality and approximate-optimality results. However, with respect to hardest "model" of real-life inputs, the problem remains illusive for the most part.

Competitive list accessing algorithms. Let ALG be any list accessing algorithm. For any sequence of requests σ we denote by $\text{ALG}(\sigma)$ the total cost incurred by ALG to service σ. Following Sleator and Tarjan [21] we measure the performance of an online list accessing algorithm ALG by its *competitive ratio*, defined as follows: we say that ALG attains a competitive ratio c (or that ALG is *c-competitive*) if there exists a constant α such that for all request sequences σ, $\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + \alpha$ where OPT is an optimal *offline* list accessing algorithm. (The constant α is usually called "the additive constant".) If ALG is randomized, it is c competitive against an *oblivious adversary* if for every request sequence σ, $E[\text{ALG}(\sigma)] \leq c \cdot \text{OPT}(\sigma) + \alpha$ where α is a constant independent of σ, and $E[\cdot]$ is the mathematical expectation taken with respect to the random choices made by ALG. The use of the competitive ratio for the analysis of online algorithms is termed competitive analysis.[1]

On known competitive optimal list accessing algorithms and their applications. In their seminal paper, Sleator and Tarjan [21] showed that the well-known algorithm MOVE-TO-FRONT (MTF), a deterministic algorithm that moves each requested item to the front, is 2-competitive. Raghavan and Karp (reported in [12]) proved a lower bound of $2 - 2/(\ell + 1)$ on the competitive ratio of any deterministic algorithm maintaining a list of size ℓ. Irani [12] gave a matching upper bound for MTF showing that MTF is a strictly optimal online algorithm judged from the perspective of competitive

[1] The term "competitive analysis" was coined by Karlin *et al.* [14].

analysis. Recently, Albers [1] discovered another deterministic algorithm called TIMESTAMP and proved that it is 2-competitive. So far MTF and TIMESTAMP have been the only deterministic algorithms known to attain an optimal competitive ratio of 2.[2][3]

The importance and practical usefulness of algorithm MTF has been well known for quite some time (see e.g. Bentley and McGeoch [6]). This algorithm provides a very simple and efficient method of maintaining a linked list with a guaranteed performance. Further, MTF can be used to devise a very simple and successful (text) compression algorithm (see e.g. [7, 9, 10]).

Soon after its discovery, algorithm TIMESTAMP was also shown to play an important role. Continuing Albers' work, Albers, von Stengel and Werchner [3] combined TIMESTAMP and algorithm BIT [17, 13] in a 1/5-4/5 probability mixture and showed that the resulting randomized algorithm (called COMB) is 8/5-competitive against an oblivious adversary. So far this is the best known randomized upper bound that leaves a small gap to the best known lower bound of 3/2 [22]. Another work due to Albers and Mitzenmacher [2] demonstrated the usefulness of TIMESTAMP by considering it as an engine for the data compression scheme of Bentley *et al.* [7, 2]. With respect to the Calgary Corpus, a collection of standard benchmark files for (text) compression [5], they found that the TIMESTAMP-based compression algorithm is in most cases superior to the MTF-based one. They argued that this phenomenon is related to the fact that TIMESTAMP is more "conservative" than MTF and therefore it is better suited to the degree of *locality of reference* exhibited in (the corpus) text files.

These results concerning MTF and TIMESTAMP emphasize compelling arguments supporting the quest for other deterministic (list accessing) algorithms, and in particular, optimal ones.[4] First, different (optimal) deterministic algorithms have different behaviors on different input sequences and by identifying the characteristic behaviors with respect to various inputs one can better match an algorithm to the task in question. For instance, it could be rewarding to be able to match (or adapt online) the choice of a list accessing algorithm to varying degrees of locality of reference exhibited in the input. Also, a set of (optimal) deterministic algorithms can be used to obtain randomized algorithms by considering probability distributions over the set. The discovery of algorithm COMB is an important example for such possibilities. (Note that algorithm BIT, the randomized component of COMB, is in fact a uniform probability mixture over a set of 2^ℓ deterministic algorithms each of which is 2-competitive.)

In the first part of this paper we introduce two new families of deterministic online list accessing algorithms. We analyze one of the families and provide some initial results regarding algorithms of the second family.

The list accessing problem has been studied for Despite the (over) thirty years of list accessing research, with plenty of interesting theoretical results, this topic suffers from a lack of experimental feedback that could lead to better and more realistic theory. In the second part of this paper we report the results of extensive empirical study testing the access cost performance and compression performance of more than 40 list accessing algorithms. These include many well known as well as our new algorithms.

[2] Any deterministic list accessing algorithm is termed here *optimal* if it attains a competitive ratio of 2. An algorithm which attains the upper bound of $2 - 2/(\ell + 1)$ is termed *strictly optimal*.

[3] Another variant of MTF that moves an item to the front every other access can be also shown to be 2-competitive.

[4] Note that this goal of discovering more (or all) optimal algorithms is very often overlooked in competitive analysis once a single, more or less "satisfying," optimal algorithm is found.

1.1 Contents and paper organization

The paper is organized as follows. In Section 2 we introduce the new MRI and PRI families and state our results. In Section 3 we develop some techniques that will be used later in the analyses of the MRI and PRI families. In particular, we present some features of the well-known "list factoring" technique that in this paper will be used only for lower bounding optimal offline costs. In Section 4 we prove the optimality of the algorithms in the MRI family and prove the equivalence between MRI(1) and TIMESTAMP. We also prove a lower bound of $\Omega(\sqrt{\ell})$ on the competitive ratio of algorithm MRI(0). In Section 5 we extend our results to the dynamic list model.

The second part of the paper starts in Section 6 and is concerned with the results of an extensive experimental study of many list accessing algorithms. In Section 6.1 we describe all the algorithms tested. These include representatives of 18 different families and more than 40 algorithms in all. Whenever known we specify bounds on the competitive ratios of the algorithms described. We have performed two experiments. The first attempts to rank the various algorithms in terms of their access cost performance, and the second, in terms of their performance as data compressors. General descriptions of these two experiments are given in Sections 6.2 and 6.3. Sections 7 and 8 treat each of these experiments in detail. In particular, they describe the data sets, the experiments, the conclusions and other relevant and related experiments. Finally, in Section 9 we draw our conclusions.

2 New list accessing algorithms

In this section we present new deterministic list accessing algorithms. In particular, we introduce two classes of algorithms called the MRI and the sc pri families.

2.1 The MRI Family

Let m be a non-negative integer. Consider the following deterministic list accessing algorithm called MOVE-TO-RECENT-ITEM(m) (or MRI(m) for short).

> **Algorithm MRI(m)**: Upon a request for item x, move x forward just after the last item z on the list that is in front of x and that was requested at least $m+1$ times since the last request for x. If there is no such item z or if this is the first request for x, move x to the front.

We call the set of algorithms $\{\text{MRI}(m)\}_{m\geq 0}$ "the MRI Family". In this paper we prove that the MRI Family includes all deterministic list accessing algorithms that are so far known to be optimal (i.e. MTF and TIMESTAMP). First, notice that MTF is the limit element of the MRI Family as m approaches infinity. In fact, for each particular (finite) request sequence σ each of the algorithms MRI(m) with $m \geq |\sigma|$ is equivalent to MTF with respect to σ (i.e. it acts identically to MTF). As noted above MTF is optimal. Let us now describe algorithm TIMESTAMP due to Albers [1][5].

> **Algorithm TIMESTAMP:** Upon a request for an item x, insert x in front of the first (from the front of the list) item y that precedes x on the list and was requested at most once since the last request for x. Do nothing if there is no such item y, or if x has been requested for the first time.

[5]The precise name of TIMESTAMP in [1] is TIMESTAMP(0).

The algorithms in the MRI family and in particular, MRI(1) look similar to algorithm TIMESTAMP. Indeed, all these algorithms maintain and use time "stamps" in order to determine their actions. Note the following two differences between MRI(1) and TIMESTAMP. The first difference is in the manner they handle first requests for items: upon a first request for an item x, TIMESTAMP keeps x in place whereas MRI(1) moves x to the front. The second difference is that MRI(1) moves the requested item x forward to a position just *after* the *last* element in front of x that was requested *at least twice* since the last request for x whereas TIMESTAMP moves x forward just *before* the *first* element in front of x that was requested *at most once* since the last request for x. The first difference is of minor significance. The second difference, although subtle, appears to be (at least at the outset) of major significance. Nevertheless, later we shall prove that MRI(1) and TIMESTAMP are equivalent (modulo the above first difference).

Proposition 1 *Algorithms* MRI(1) *and* TIMESTAMP *are equivalent.*

(The precise definition of this equivalence will be given later, in Section 4.1.)

For each $i \geq 0$, algorithm MRI(i) appears more "reluctant" than algorithm MRI($i+1$) to move an accessed item forward. This property makes the MRI family attractive as it provides a gradual transition from the "conservative" TIMESTAMP to the more "hasty" MTF. An example that demonstrates this feature will be presented. In particular this property of the MRI family may be used in applications to dynamically adapt to varying degrees of locality of reference, by "shifting gears" between algorithms in the MRI family.

Theorem 1 *For any list of size ℓ and for each integer $m \geq 1$,* MRI(m) *is $(2 - 1/\ell)$-competitive in the static model.*[6]

This result significantly expands our knowledge of competitive-optimal deterministic list accessing algorithms. The proof of Theorem 1 (Section 4) uses a standard potential function argument in conjunction with standard list factoring (to bound below the optimal offline cost). These two techniques have been usually used separately. Note that standard list factoring by itself cannot be employed for proving upper bounds for the MRI family (except when $m = 1$). The reason is that for $m > 1$ algorithm MRI(m) does not satisfy the "Pairwise Property" [6]. On the other hand, we could not obtain a pure potential function proof. It is interesting to note that both MTF and TIMESTAMP do satisfy the Pairwise Property and thus allow for a standard and simple list factoring analysis (as well as a potential function analysis).

The "odd" element of the MRI family is the algorithm MRI(0) which is no better than $\Omega(\sqrt{\ell})$-competitive. This is established in Section 4.2. It is interesting to note that in the experimental study, presented later in this paper, members of the MRI family and in particular MRI(0) were found to be among the best list accessing algorithms in terms of their access cost performance (as well as data compressors).

Theorem 1 also holds within the dynamic list accessing model (in which case ℓ is the total number of elements that were inserted to the list). This is established in Section 5.

2.2 The PRI Family

Analogous to the MRI Family and generalizing the algorithm TIMESTAMP of Albers in a straightforward manner, we now define the following family of algorithms. Let m be a non-negative

[6] As pointed out by an anonymous referee it is possible to prove that MRI(m), $m \geq 1$, attain the optimal upper bound of $2 - 2/(\ell + 1)$ using a different proof technique than ours.

integer. Consider the following deterministic list accessing algorithm called PASS-RECENT-ITEM(m) (or PRI(m) for short).

> **Algorithm PRI(m)**: Upon a request for item x, move x forward just in front of the first item z on the list that was requested at most m times since the last request for x. Do nothing if there is no such item z. If this is the first request for x, move x to the front.

We call this infinite set of algorithms $\{\text{PRI}(m)\}_{m\geq 0}$, "the PRI Family". It is readily seen that algorithm PRI(1) is identical to algorithm TIMESTAMP (modulo handling of first requests). Here again, the limit algorithm of this family (as m approaches infinity) is MTF. We state without a proof the following result.[7]

Proposition 2 *For all $m \geq 1$ PRI(m) is 3-competitive.*

The exact competitive ratio of the algorithm in the PRI Family (other than PRI(1)) is an open question.

It is not hard to see that for all $m \neq 1$ ($m \geq 0$) PRI(m) acts differently than MRI(m); Consider the following example.

Example 1: Consider a list of 5 elements initially ordered $\langle 1,2,3,4,5\rangle$ where the element 1 is at the front. Consider the following request sequence

$$\sigma = 5,4,3,2,1,1,3,1,1,3,2,4,5,1,3,1,3,2,5,2,5,1,1,4,5,5,4,2.$$

Consider algorithm MRI(2). After processing σ the list maintained by MRI(2) is $\langle 4,5,2,1,3\rangle$ and the cost it incurs is 87. For comparison consider algorithms PRI(1) = TIMESTAMP, PRI(2), and MTF. The costs incurred by these algorithms for processing σ are, respectively, 95, 93 and 93 and they end up with the following lists, respectively: $\langle 5,4,1,2,3\rangle$, $\langle 2,4,5,1,3\rangle$ and $\langle 2,4,5,1,3\rangle$. Thus, in some sense algorithms MRI(2), PRI(1) and PRI(2) all have different "predictions" for what is likely to be the next request.

Example 2: We now consider three request sequences, each of which distinguishes between the algorithms MRI(m), $m = 1,2,\ldots,8$ in a different way. The following are the three sequences.

$$\begin{aligned}
\sigma_1 &= 5432154423231421111123143245212421423542435142123131444325144421 \\
&\quad 4231233231433143233235242331233242523243251333; \\
\sigma_2 &= 5432121131252135511335242331113545542554442135455432332421423133 \\
&\quad 3 \\
&\quad 1114242331232113552442131132452543351245254212433323142124333242 \\
&\quad 1 \\
&\quad 31252143; \\
\sigma_3 &= 5432131251444325252454543121442335554511135442443321255133331125 \\
&\quad 4 \\
&\quad 4545423543551132321131252523311211423112114235542554455244251133.
\end{aligned}$$

The respective costs of the algorithms for each of these sequences are summarized in Table 1. As can be seen in Table 1, with respect to σ_1 the minimum cost of 314 is obtained by MRI(1) and the cost monotonically increases with m, so that the maximum cost of 356 is obtained by both

[7] The proof of Proposition 2 has a similar structure to the proof of Theorem 1.

	MRI(1)	MRI(2)	MRI(3)	MRI(4)	MRI(5)	MRI(6)	MRI(7)	MRI(8)
σ_1	314 $*$	323	330	340	349	352	356	356
σ_2	449	435	414	407 $*$	411	415	415	418
σ_3	418	399	391	386	382	375	372	368 $*$

Table 1: The costs of MRI(m), $m = 1, 2 \ldots, 8$ with respect to σ_i, $i = 1, 2, 3$. The minimum cost in each row is marked by $*$

MRI(7) and MRI(8). With respect to σ_2, the minimum cost is obtained by MRI(4) and the sequence of costs incurred by MRI(m), $m = 1, 2, 3, 4$ is monotone decreasing; on the other hand, the sequence of costs incurred by MRI(m), $m = 4, \ldots, 8$ is monotone increasing. Lastly, with respect to σ_3 the minimum cost is obtained by MRI(8) and the sequence of costs for MRI(m) is a monotone decreasing with m.

3 Preliminaries and list factoring

We begin by introducing some notation that will be used throughout. Let $\{x_1, x_2, \ldots, x_\ell\}$ be a set of items. These items are the (fixed) set of items on the list, L. Let $\sigma = r_1, r_2, \ldots, r_n$ be any request sequence with $r_i \in L$. For each pair of items x and y ($x \neq y$) denote by σ_{xy} the "projection" of σ over x and y, defined to be σ after deletion of all the requests for items other than x and y. Similarly we define L_{xy} to be the projection of the list L over x and y (i.e. L_{xy} is the two element list holding x and y).

The list factoring technique was first discovered by Bentley and McGeoch [6] and was extended and used in several papers [12, 22, 3]. This technique enables a reduction of the cost analysis of list accessing algorithms to lists of size two. In this paper we use this technique only to bound from below the optimal offline cost. We now present parts of this technique that are essential to our analysis or to the discussion that follows.

Call the variant where we charge $i - 1$ for accessing the ith item on the list the *partial-cost model*. The usual model where we charge i for accessing the ith item will be called the *full-cost model*. For any list accessing algorithm ALG and request sequence σ denote by ALG$^*(\sigma)$ the cost incurred by ALG for serving σ within the partial-cost model.

Suppose ALG is a deterministic list accessing algorithm that does not use paid exchanges. Let $\sigma = r_1, r_2, \ldots, r_n$ be any request sequence. For each item $x \in L$, and integer $1 \leq j \leq n$, we denote by ALG(x, r_j) a cost function measuring the penalty attributed to item x for being in the way while ALG accesses r_j (the jth request). That is

$$\text{ALG}(x, r_j) = \begin{cases} 1 & \text{if } x \text{ is in front of } r_j, \\ 0 & \text{otherwise (including the case } x = r_j). \end{cases} \tag{1}$$

Using the above notation and this cost function we have,

$$\begin{aligned} \text{ALG}^*(\sigma) &= \sum_{1 \leq j \leq n} \sum_{x \in L} \text{ALG}(x, r_j) \\ &= \sum_{x \in L} \sum_{1 \leq j \leq n} \text{ALG}(x, r_j) \end{aligned}$$

$$= \sum_{x\in L}\sum_{y\in L}\sum_{j:\, r_j=y} \textsc{alg}(x, r_j)$$
$$= \sum_{\substack{x,y\in L\\ x\neq y}}\sum_{j:\, r_j=y} \textsc{alg}(x, r_j)$$
$$= \sum_{\substack{\{x,y\}\subseteq L\\ x\neq y}}\sum_{j:\, r_j\in\{x,y\}} [\textsc{alg}(x, r_j) + \textsc{alg}(y, r_j)]\,. \tag{2}$$

For every x and y in L, and request sequence σ we denote the internal summation of the expression in (2) by $\textsc{alg}^*_{xy}(\sigma)$. That is,

$$\textsc{alg}^*_{xy}(\sigma) = \sum_{j:\, r_j\in\{x,y\}} [\textsc{alg}(x, r_j) + \textsc{alg}(y, r_j)]\,.$$

We can now write equation (2) as

$$\textsc{alg}^*(\sigma) = \sum_{\substack{\{x,y\}\subseteq L\\ x\neq y}} \textsc{alg}^*_{xy}(\sigma). \tag{3}$$

Define $\textsc{alg}^*(\sigma_{xy})$ to be the cost that ALG pays for serving the projected request sequence σ_{xy} while operating on the projected list L_{xy}. We say that the algorithm ALG satisfies the *Pairwise Property* if for every pair $\{x, y\} \subseteq L$, $\textsc{alg}^*(\sigma_{xy}) = \textsc{alg}^*_{xy}(\sigma)$.

Which online algorithms satisfy the Pairwise Property? The following is a useful characterization due to Bentley and McGeoch [6]: *An algorithm satisfies the Pairwise Property iff for every request sequence σ, when* ALG *serves σ, the relative order of every two elements x and y in L is the same as their relative order in L_{xy} when* ALG *serves σ_{xy}.*

Equations (2) and (3) do not in general apply to an optimal offline algorithm, OPT ($\textsc{opt}^*$) as in general, OPT cannot avoid paid exchanges.[8] Nevertheless, it is not hard to obtain the following inequality

$$\sum_{\substack{\{x,y\}\subseteq L\\ x\neq y}} \textsc{opt}^*(\sigma_{xy}) \leq \textsc{opt}^*(\sigma) \tag{4}$$

Moreover, it is possible to extend this inequality to the full cost model. This is established in the following lemma.

Lemma 1 *For any request sequence σ,*

$$\sum_{\substack{\{x,y\}\subseteq L\\ x\neq y}} \textsc{opt}(\sigma_{xy}) \leq \textsc{opt}(\sigma)$$

The proof of inequality (4) and Lemma 1 can be found in [8]. (For the reader's convenience we included them in Appendix A. What makes Lemma 1 so useful is the fact that the optimal offline algorithm, restricted to lists of size two, has a very simple structure[9] (in fact, it is not unique and we refer to one particular optimal offline algorithm), which is summarized in the following lemma.

[8]The following example due to Reingold and Westbrook [17] proves this assertion. Consider a list $L = \{x_1, x_2, x_3\}$ of size 3 initially ordered x_1, x_2, x_3 (x_1 at the front). The optimal offline cost to serve the request sequence x_3, x_2, x_3, x_2 is 8. An optimal offline algorithm without paid exchanges pays 9 to serve the same sequence.

[9]The best known list accessing optimal offline algorithm for lists of sizes ℓ larger than 2 [17] does not have a concise description, which is probably essential to make it useful for analyses. We note that this algorithm computes the optimal solution in time exponential in ℓ.

Lemma 2 *Assume L is a list of two elements x and y. Then there exists an optimal offline algorithm* OPT *for L that satisfies the following properties: (i)* OPT *does not use paid exchanges; and (ii) Whenever there is a run of two or more consecutive requests for x (y),* OPT *moves x (y) to the front (if it is not already there) after the first request (of this run) using free exchanges.*[10]

The proof of Lemma 2 is simple and is left to the reader.

This lemma will be essential to our analyses. Using it in conjunction with Lemma 1, one can easily bound the optimal offline cost to serve a request sequence σ by summing up the costs $\text{OPT}(\sigma_{xy})$ for every pair $\{x, y\} \subseteq L$.

Let OPT be any optimal offline list accessing algorithm and let L be a list of ℓ items. Denote by FOPT ("factored" OPT) the collection of all $\binom{\ell}{2}$ 2-element lists L_{xy} with $\{x, y\} \subseteq L$ that are each maintained by an optimal offline algorithm satisfying properties (i) and (ii) of Lemma 2. Let σ be any request sequence. Abusing notation, we shall refer to FOPT both as the *set* of all these 2-element lists and also as the optimal offline *algorithm* that maintains these lists and serves the request sequences σ_{xy} ($\{x, y\} \subseteq L$). Define FOPT_i to be the cost incurred by FOPT to serve the ith request in σ in the full cost model. Formally, if the ith request is for an element x then FOPT_i is defined to be one plus the number of 2-element lists $L_{xy} \in$ FOPT that contain x in the second position. By the definition of the extended cost function (10) it is clear that

$$\sum_{i=1}^{|\sigma|} \text{FOPT}_i = \sum_{\substack{\{x,y\} \subseteq L \\ x \neq y}} \text{OPT}(\sigma_{xy}) \tag{5}$$

Therefore, using Lemma 1 we obtain

Lemma 3

$$\sum_{i=1}^{|\sigma|} \text{FOPT}_i \leq \text{OPT}(\sigma)$$

4 Analysis of the MRI Family

We use the following notation. Suppose L is the list maintained by some algorithm. For each pair of items x and y in L, we say that "$\langle x, y \rangle$ in L" when x is currently in front of y in L. If $\langle x, y \rangle$ in the list maintained by the online algorithm but $\langle y, x \rangle$ in the list maintained by the optimal offline algorithm then we say that $\langle x, y \rangle$ is an *inversion*. With respect to $\text{MRI}(m)$ consider a request for an item x. Define $p(x)$, the *current pivot of x* to be the last element on the list that is in front of x and was requested at least $m + 1$ times since the last request for x. If there is no such element or if this is the first request for x then $p(x)$ is undefined. Thus, upon a request for an element x, $\text{MRI}(m)$ moves x one position after $p(x)$, and if $p(x)$ is undefined, x is moved to the front.

Fix any request sequence σ and set L to be the ℓ-item list maintained by $\text{MRI}(m)$. We compare configurations of L to configurations of FOPT. Consider any configuration of L and a configuration of FOPT. For each (ordered) pair of items $(x, y) \subseteq L$, $x \neq y$, we define the following weight function

[10] In fact, it is not hard to see that an optimal offline algorithm (for lists of size two) that satisfies property (i) *must* satisfy property (ii).

$w(x,y) = w(x,y,L,\text{FOPT})$.

$$w(x,y) = \begin{cases} 0, & \text{if } \langle x,y\rangle \text{ is not an inversion;} \\ 1, & \text{if } \langle x,y\rangle \text{ is an inversion and } y \text{ will pass } x \\ & \quad \text{in } L \text{ if } y \text{ is requested next;} \\ 2, & \text{if } \langle x,y\rangle \text{ is an inversion and } y \text{ will pass } x \\ & \quad \text{in } L \text{ iff } y \text{ is now requested twice in a row;} \end{cases}$$

It is easy to verify that $w(x,y)$ is well defined. Moreover $w(x,y)$ satisfies the following independence property that is essential for the proof of Theorem 1: the weight of two elements y and z cannot be increased (from 1 to 2) after a request for x is made.

Lemma 4 *Let $m > 0$ be given. Let y and z be two elements such that $\langle z,y\rangle$ in L is an inversion with $w(z,y) = 1$. Let x be any element other than y and z. Then $w(z,y)$ remains 1 after a request for x is serviced by* MRI(m).

Proof: The proof is by contradiction. Consider a configuration of L with $\langle z,y\rangle$ in L an inversion with $w(z,y) = 1$. Consider the ith request for an element x, $x \neq y$, $x \neq z$, and assume that after MRI(m) processes this request for x, the weight $w(z,y)$ increases from 1 to 2. By the definition of the function w it must be that due to this request for x, MRI(m) positions x in between y and z so that x becomes the pivot of y. This means that the pivot of x just before the ith request, $p(x)$, is an element located in front of y and after z (or $p(x)$ is z itself). Set $p = p(x)$. Since after the service of the ith request x becomes the pivot of y it must be that after servicing the ith request there were $m+1$ requests for x since the last request for y. In between the times of the two last requests for x there must be $m+1$ requests for p. As $m \geq 1$ the second last request for x occurred after the last request for y which means that p was a pivot of y even before MRI(m) serviced the ith request for x. But this means that $w(z,y) = 2$ before the ith request in contradiction to our assumption. It follows that $w(z,y)$ cannot increase from 1 to 2. Note also that $w(z,y)$ cannot decrease to 0 if z or y are not requested. ∎

We now define the following potential function.

$$\Phi = \Phi(L,\text{FOPT}) = \sum_{\substack{\{x,y\}\subseteq L \\ x\neq y}} w(x,y).$$

It is clear that Φ is bounded below as it is always non-negative. Let $\sigma = r_1, r_2, \ldots, r_n$ be any request sequence. The *amortized cost* a_i for MRI(m) to serve the ith request is defined as $a_i = \text{MRI}(m)_i + \Delta\Phi_i$ where MRI$(m)_i$ is the *actual cost* cost incurred by MRI(m) while serving r_i, and $\Delta\Phi_i = \Phi_i - \Phi_{i-1}$, is the difference of the potential *after* both MRI(m) and FOPT served r_i minus the potential *before* their moves). Similarly, define FOPT$_i$ to be the actual cost incurred by FOPT while serving r_i. We shall use the standard potential function technique that is summarized in the following simple lemma [21, 8]

Lemma 5 *Suppose that (i) Φ is bounded below; and (ii) there is a positive constant c such that for each $i = 1,2,\ldots,n$,* MRI$(m)_i \leq c \cdot$ FOPT$_i$. *Then* MRI(m) *is c-competitive.*

Proof of Theorem 1: Imagine that algorithms MRI(m) and FOPT are working concurrently and processing the request sequence σ. Consider the ith request (r_i) for some item x. At the penalty of

increased additive constant assume that r_i is not the first request for x. (We can pad the request sequence σ with a prefix consisting of ℓ requests, one for each item. The additional cost due to this prefix can be attributed to the additive constant and will not alter the competitive ratio.) Assume that just before this request is revealed, x is located at the kth position in L and that x appears at the second position (i.e. last position) in j of the 2-item lists in FOPT. The former assumption means in particular that the actual cost for MRI(m), is MRI(m)$_i = k$. Similarly, the latter assumption implies that FOPT$_i = j + 1$.

Define $\mathcal{I}$ to be the set of elements z such that $\langle z, x\rangle$ in L is an inversion (i.e. $\langle z, x\rangle$ in L but $\langle x, z\rangle$ in $L_{xz} \in$ FOPT). Define $\mathcal{C}$ to be the set of elements z such that $\langle z, x\rangle$ in L is not an inversion, and define $\mathcal{S}$ to be the set of elements z such that $\langle x, z\rangle$ in L is an inversion. Set $I = |\mathcal{I}|$, $C = |\mathcal{C}|$ and $S = |\mathcal{S}|$.

We are now in a position to bound the amortized cost a_i. First consider the change in potential after MRI(m)'s move due to inversions in L corresponding to elements in $\mathcal{I}$. By the definitions of MRI(m) and of the weight function w, if in L an element $z \in \mathcal{I}$ is passed by x, the weight of the inversion $\langle z, x\rangle$ in L was 1 before the move and this inversion is eliminated by this move. An element z corresponding to an inversion $\langle z, x\rangle$ that is not passed by x, contributed (together with x) a weight of 2 before the move and by the definition of MRI(m) and w, reduces this weight to 1 after MRI(m) moves x. Hence there is a decrease of exactly I in Φ due to elements in $\mathcal{I}$. Due to MRI(m)'s move and before FOPT moves, the weight of each inversion $\langle x, z\rangle$ in L with $z \in \mathcal{S}$ may increase by one giving a total increase of up to S.

Since FOPT does not use any paid exchanges its move cannot affect the above change in potential due to inversions corresponding to elements in $\mathcal{I}$. (Of course, FOPT's move can eliminate inversions due to elements in $\mathcal{S}$; this possibility will be considered next.)

We shall make use of the following claim.

Claim 1 *If, in the list L, x passes an element $z \in \mathcal{C}$ then z was requested at least once since the last request for x.*

To prove this claim suppose that $z \in \mathcal{C}$ was not requested at all since the last request for x. It follows (property (ii) of Lemma 2) that z cannot be in $\mathcal{C}$ since FOPT would have moved x in front of z in L_{xz} before this request for x. Hence, if an element $z \in \mathcal{C}$ is passed by x, then z was requested at least once since the last request for x, which proves the above claim.

Suppose that a new inversion $\langle x, z\rangle$ in L is created (i.e. x does not pass z in L_{xz} but x does pass z in L). By definition of the weight function, the weight of this new inversion is 1 if and only if z will pass x if z is requested next. We now show that this is indeed the case (i.e. that if the subsequent request is for z, then z will pass x).

Consider the configuration of L just after MRI(m) serves the ith request. By our assumption, x has just moved in front of z. Consider any element y that is now positioned between x and z; that is, in the last move x also passed y and now in L, x appears in front of y, which appears in front of z. Suppose that the previous request for x is $r_{i'}$. By the definition of MRI(m) since x passed y, it must be that y was requested at most m times between the last (ith) and second last (i'th) requests for x. Assume that the next request r_{i+1} is for z. Then z will pass x if and only if the following two conditions hold: (i) all items y now appearing between x and z were requested at at most m times between the current (i.e. $(i+1)$st) and last request for z; and (ii) x was requested at most m times since the last request for z. Since, by the above claim, z was requested at least once between the two most recent request for x, and since any such item y was requested at most m times between the two most recent requests for x it must be that these two conditions hold.

Therefore if $r_{i+1} = z$, the element z will pass x and the weight of the new inversion $\langle x, z\rangle \in L$ (if it is created) is exactly 1.

Other possibilities for new inversions are when for some element $z \in \mathcal{C}$, in L_{xz}, x passes z but x does not pass z in L, the weight of a new such inversion must be 1 as well, since the last request was for x so an additional request for x will move x to the front of L. It follows that the increase in Φ due to new inversions created is at most C.

It is easy to see that the following identities hold.

$$\begin{aligned} k - 1 &= C + I \\ j &= S + C. \end{aligned}$$

For example, to justify the first identity notice that the elements that are in front of x in L (just before the ith request) are exactly those that are either in $\mathcal{C}$ or in $\mathcal{I}$ and these sets are disjoint. The second identity follows from the definitions of j, $\mathcal{S}$ and $\mathcal{C}$. From these identities it readily follows that $k = I + j - S + 1$ and that $C \leq j$. Here again $j \leq \ell - 1$.

Putting all this together, the amortized cost a_i for MRI(m) to serve the ith request is bounded above as follows.

$$\begin{aligned} a_i &\leq k - I + S + C \\ &= I + j - S + 1 - I + S + C \\ &= j + C + 1 \\ &\leq 2j + 1 \\ &\leq (2 - 1/\ell) \cdot (j + 1) \leq (2 - 1/\ell) \cdot \text{FOPT}_i. \end{aligned}$$

Using lemmas 5 and 3 we conclude that MRI(m) is $(2 - 1/\ell)$-competitive. ∎

4.1 The equivalence of MRI(1) and TIMESTAMP

Consider algorithm TIMESTAMP (see Section 2.1). Here we prove that except for the difference in handling first requests for items algorithm TIMESTAMP is equivalent to algorithm MRI(1) . Specifically, we shall prove that if we alter the definition of either MRI(1) TIMESTAMP (or both) so that they handle first requests for items in the same manner, then the two algorithms are equivalent in the sense that they process any request sequence in exactly the same manner.

Lemma 6 *Each of the algorithms* TIMESTAMP *and* MRI(1) *maintains the following invariant: upon a request for x, all the items that are requested twice or more since the last request for x are in front of all items that were requested less than twice since the last request for x.*

Using Lemma 6 we learn that upon a request for an item x both MRI(1) and TIMESTAMP move x forward just after all the elements that were requested twice or more since the last request for x (equivalently, just in front of all the elements that were requested less than twice since the last request for x). This means that both algorithms maintain identical lists at all times. Hence we have proved Proposition 1.

Interestingly, other algorithms in the MRI Family (i.e. $m \neq 1$) do not maintain analogous invariants. For example the algorithm MRI(0) *does not* maintain the following invariant: "upon a request for x, all the items that are requested once or more since the last request for x are in front of all items that were not requested since the last request for x".

4.2 The algorithm MRI(0)

Algorithm MRI(0) is significantly more "conservative" than MRI(1) (and the rest of the algorithms in the MRI family). Unfortunately not only is MRI(0) not optimal, it cannot attain a constant competitive ratio.

Lemma 7 *The competitive ratio of* MRI(0) *is not smaller than* $\Omega(\sqrt{\ell})$.

Proof: For simplicity, we start with a lower bound of 3 and then note how to extend it to the higher lower bound as stated in the lemma. Assume that the initial list, of ℓ items is $\langle x_1, x_2, \ldots, x_{\ell-1}, a\rangle$. The initial segment, σ_0, of the nemesis request sequence, σ, is $\sigma_0 = a, x_{\ell-1}, x_{\ell-2}, \ldots, x_1$. Notice that after processing σ_0 MRI(0) returns to the above initial configuration (and now each item was requested once). The rest of σ consists of an arbitrary number of repetitions of the following segment

$$\sigma_1 = (x_{\ell-1}, a, x_{\ell-1}), (x_{\ell-2}, a, x_{\ell-2}), \ldots, (x_1, a, x_1).$$

For each $i = \ell-1, \ell-2, \ldots, 1$, the item x_i will be brought by MRI(0) to the front only after the second request for x_i in the subsequence (x_i, a, x_i) is processed. The element a will always be left at the back of the list. Clearly, after servicing σ_1 the list maintained by MRI(0) returns to the initial configuration. The cost incurred by MRI(0) for the initial segment σ_0 is some constant $C = \ell^2$ and the cost of servicing the segment σ_1 is $(\ell-1)(3\ell-2) = 3\ell^2 - 5\ell + 2$. Consider an offline algorithm OPT$'$ that serves σ as follows. After the first request for a, OPT$'$ brings a to the front and then it keeps the list static forever. That is, each subsequent request for an item costs OPT$'$ the position of the item in the following list

$$\langle a, x_1, x_2, \ldots, x_{\ell-1}\rangle.$$

Thus, the cost incurred by OPT$'$ for the initial segment σ_0 is some constant $D = \ell + \sum_{2\le i\le \ell} i$ and the cost to serve the segment σ_1 is $\ell - 1 + 2\cdot\sum_{2\le i\le\ell} i = \ell^2 + 2\ell - 3$. The costs ratio, online to offline, for the segment σ_1 is thus $(3\ell-2)/(\ell+3)$ which approaches 3 as ℓ grows.

This idea can be carried further. We now describe the following straightforward generalization of the segment σ_1 that will yield a nemesis request sequence that will force a competitive ratio of at least $\Omega(\sqrt{\ell})$. Assume that the initial order of MRI(0)'s list is

$$\langle x_1, x_2, \ldots, x_{\ell-k}, a_1, a_2, \ldots, a_k\rangle,$$

and that of OPT$'$ is

$$\langle a_1, a_2, \ldots, a_k, x_1, x_2, \ldots, x_{\ell-k}\rangle,$$

where $1 \le k \le \ell - 1$. These initial configurations can be obtained after a fixed initial segment requesting all items. The repetition is now on the segment

$$(x_{\ell-k}, a_1, a_2, \ldots, a_k, x_{\ell-k}), \ldots, (x_1, a_1, a_2, \ldots, a_k, x_1).$$

Here again, after completion of this segment MRI(0)'s list returns to the initial configuration and OPT$'$ keeps its list static at the initial configuration. The cost incurred by MRI(0) for this segment is

$$\frac{\ell-k}{2}\left(4\ell + 2\ell k - k^2 - 3k\right).$$

The cost incurred by OPT$'$ for this segment is

$$\frac{k}{2}\left(\ell k + \ell - k^2 - 3k\right) + \ell^2 + \ell - k.$$

By differentiating the costs ratio (online to offline) with respect to k it is possible to see that the maximum ratio is obtained at

$$k^* = \left(2\ell^2(\ell+1) + 3\ell + 1\right)^{1/2} / \ell - 3 - 1/\ell.$$

Substituting k^* for k in the expression for costs ratio results in a function of ℓ that can be bounded below by $\Omega(\sqrt{\ell})$. ∎

The true competitive ratio of MRI(0) remains an open question but we conjecture that it is indeed $\Theta(\sqrt{\ell})$. We note that the competitive ratio of one of the most conservative (yet "reasonable") algorithms called TRANSPOSE is $\ell/3$ [21].

5 Extensions to the dynamic model

Theorem 1 can be extended to the dynamic (standard) list accessing model [21, 8]. In this model a request is either an *insertion* of an item, an *access* of an item, or a *deletion* of an item. An access or deletion of an item positioned ith from the front cost i whereas an insertion of an item to a list currently holding ℓ items costs $\ell + 1$. This insertion cost implicitly requires that a new item is inserted at the back of the list (but of course, the new item can then move, using free exchanges, to any position closer to the front).

Each of the algorithms MRI(m) is naturally extended to handle deletions and insertions. Depending on the exact definition of MRI(m), in particular, how it deals with first requests for items, this extension is done in a straightforward manner. In accordance with the original definition of MRI(m) we require that each new item is inserted at the front of the list.

Theorem 2 *For each $m \geq 1$ MRI(m) is $(2 - 1/\ell)$-competitive in the dynamic model, where ℓ is the number of items inserted.*

Proof: The extension of Theorem 1 to this dynamic model is easily established by proving that for each insertion and deletion the amortized cost for MRI(m) is within a factor $2 - 1/\ell$ of the actual cost incurred by FOPT where ℓ is the total number of insertions (starting from an empty list). Consider a request for an insertion of an item x. Suppose that at this stage there are exactly $\ell' \geq 2$ items on the list.[11] Since the cost of an insertion in the dynamic model is $\ell' + 1$ we can interpret it as if both OPT and MRI(m) first insert x at the back of their lists where for FOPT, which contains $\binom{\ell'}{2}$ 2-item lists before the insertion, we add ℓ' new 2-item lists, L_{xz}, one for each item z that was on the list before this request. In each of these 2-item lists x appears on the back (i.e. second). At this stage there are no inversions corresponding to x but new inversions can be created by FOPT's move that may move x forward, using free transpositions, in some number k of its 2-item lists. Each of the new k inversions is clearly of weight 1 since a subsequent request for x will cause MRI(m) to move x to the front. Further, the maximum number of such new inversions is ℓ', the number of new 2-item lists containing x. Since $\ell' + 1$ is the actual cost (of both MRI(m) and FOPT), the amortized cost for MRI(m) is in the worst case

$$\ell' + 1 + \Delta\Phi = \ell' + 1 + k \leq 2\ell' + 1 = (2 - 1/\ell')(\ell' + 1) \leq (2 - 1/\ell)(\ell' + 1).$$

The analysis of the case of a request for deletion of an item x is even simpler because all old inversions with respect to x are eliminated and no new inversions are created. (Note that in this case we remove from the set FOPT all the 2-item lists containing x.) ∎

[11] The trivial cases where $\ell' = 0, 1$ should be treated separately but do not bear any special difficulty.

6 An experimental study of list accessing algorithms

In this section, we give an introduction to the experimental part of the paper. We specify the set of algorithms tested and shortly outline the experiments we conducted. Detailed descriptions of these experiments will then be given in the corresponding Sections 7 and 8 of the paper.

6.1 Algorithms

We now describe the algorithms we tested. In the description of most algorithms we specify the algorithm's action only with respect to an ACCESS request. Unless otherwise specified, it is implicitly assumed that an INSERT request for an item x places x at the back of the list, and then x is treated as if it was accessed. With respect to each algorithm we also specify bounds on its competitive ratio whenever they are known. In the sequel ℓ denotes the maximum number of items present in the list at any point in time.

6.1.1 Deterministic algorithms

Algorithm MOVE-TO-FRONT (MTF) is one of the most well known and used algorithms. This algorithm attains an optimal competitive ratio of $2 - 2/(\ell + 1)$ [21, 12].

Algorithm MTF: Upon an access for an item x, move x to the front.

Algorithm TRANSPOSE (TRANS) is perhaps the most "conservative" algorithm presented here and is an extreme opposite to MTF. The competitive ratio of TRANS is bounded below by $2\ell/3$ [21].

Algorithm TRANS: Upon an access to an item x, transpose x with the immediately preceding item.

Algorithm FREQUENCY-COUNT (FC) attempts to adapt its list to the empirical distribution of requests observed so far. A lower bound of $(\ell + 1)/2$ on its competitive ratio is known [21].

Algorithm FC: Maintain a frequency counter for each item. Upon inserting an item, initialize its counter to 0. After accessing an item increment its counter by one and then reorganize the list so that items on the list are ordered in non-increasing order of their frequencies.

In our implementation we further require that if two items have the same frequency count then the item requested less recently is positioned in front of the item requested more recently. The variant adopting the reverse of this ordering is denoted by FC$'$.

The following algorithm called MTF2 is a more "relaxed" version of MTF. MTF2 can be shown to be 2-competitive.

Algorithm MTF2: Upon the ith request for an item x, move x to the front if and only if i is even.

Algorithm MOVE-AHEAD(k) (MHD(k)) proposed by Rivest [19] is a simple compromise between the relative extremes of TRANS and MTF.

Algorithm MHD(k): Upon a request for an item x, move x forward k positions.

Thus algorithm MHD(1) is TRANS. We have tested MHD(k) for $k = 2, 4, 8, 16, 32, \ldots, 1024$.

Algorithm MOVE-FRACTION(k) (MF(k)) proposed by Sleator and Tarjan is a slightly more sophisticated compromise between TRANS and MTF. For each k, algorithm MF(k) is $2k$-competitive [21].

Algorithm MF(k): Upon a request for an item x currently at the ith position, move x $\lceil i/k \rceil - 1$ positions towards the front.

Thus, algorithm MF(1) is MTF. The algorithms of the MF(k) family as described above have the weakness that they do not perform any update operation upon an access for one of the first k items in the list. Therefore, we included in the experiments a variant MF$'(k)$ treating the first k items in its list as a "MTF-cache". The algorithm MF$'(k)$ behaves just like MF(k), but upon a request for one of the first k items in the list, the requested item is moved to the front. We have tested the algorithms MF(k) and MF$'(k)$ for $k = 2, 4, 8, 16, \ldots, 1024$.

Algorithm TIMESTAMP (TS) due to Albers is an optimal algorithm attaining a competitive ratio of 2. [1]

Algorithm TS: Upon a request for an item x, insert x in front of the first (from the front of the list) item y that precedes x on the list and was requested at most once since the last request for x. Do nothing if there is no such item y, or if x has been requested for the first time.

The following family of algorithms is the PRI FAMILY (introduced in Section 2.2). As discussed earlier, for each $m \geq 2$ PRI(m) is 3-competitive and MRI(1) is 2-competitive.

Algorithm PRI(m): Upon a request for item x, move x forward just in front of the first item z on the list that was requested at most m times since the last request for x. Do nothing if there is no such item z. If this is the first request for x, leave x in place (move x to the front).

Notice that modulo the handling of first requests, algorithm PRI(1) is identical to algorithm TS. Also, as noted above, the limit element of this family, as m grows, is MTF. We have tested the algorithm PRI(m) with $m = 0, 1, \ldots, 8$ and with the insertion position being the last and the front. In the sequel each member of the PRI family, PRI(m), that inserts a new item at the first (resp. last) position is denoted PRI$'(m)$ (resp. PRI(m)).

The following family of algorithms is the PRI FAMILY introduced in Section 2.1). For each $m \geq 1$ algorithm MRI(m) is 2-competitive and is thus optimal. The competitive ratio of algorithm MRI(0) is bounded below by $\Theta(\sqrt{\ell})$.

Algorithm MRI(m): Upon a request for item x, move x forward just after the last item z on the list that is in front of x and that was requested at least $m+1$ times since the last request for x. If there is no such item z move x to the front. If this is the first request for x, leave x in place (move x to the front).

Recall that MRI(1) is equivalent to algorithm TS (modulo first requests for items). We have tested the algorithms MRI(m) with $m = 0, 1, \ldots, 8$ and with the insertion position being the last and the front. In the sequel each member of the MRI family, MRI(m), that inserts a new item at the first (resp. last) position is denoted MRI$'(m)$ (resp. MRI(m)).

6.1.2 Randomized algorithms

Algorithm SPLIT (SPL) due to Irani [12] was the first randomized algorithm that was shown to attain a competitive ratio lower than the deterministic lower bound of $2 - 2/(\ell + 1)$. It is known that SPL is 31/16-competitive ($= 1.9375$).

Algorithm SPL: For each item x on the list maintain a pointer $p(x)$ pointing to some item on the list. Initially set $p(x) = x$. Upon a request for an item x, with probability $1/2$ move x to the front and with probability $1/2$ insert x just in front of $p(x)$. Then set $p(x)$ to point the first item on the list.

Algorithm BIT due to Reingold and Westbrook [17] attains a competitive ratio of $1.75 - 1.75/\ell$.

Algorithm BIT: For each item on the list maintain a mod-2 counter $b(x)$ initially set to either 0 or 1 randomly, independently and uniformly. Upon a request for an item x first complement $b(x)$. Then if $b(x) = 0$ move x to the front.

The following algorithm called RMTF is very similar to BIT but somewhat surprisingly its worst case performance is inferior. It can be shown that its competitive ratio is not smaller than 2 [24].

Algorithm RMTF: Upon a request for x, with probability $1/2$ move x to the front, and with probability x leave x in place.

The algorithm RMHD is a simple randomized relative of the deterministic MHD family.

Algorithm RMHD: Upon a request for an item x currently at the ith position, randomly choose a position from the set $\{1, \ldots, i\}$ and move x to that position.

The family of algorithms COUNTER(s, S) (CTR(s, S)) due to Reingold, Westbrook and Sleator [18] is a sophisticated generalization of algorithm BIT. Let s be a positive integer and S, a nonempty subset of $\{0, 1, \ldots, s-1\}$.

Algorithm CTR(s, S): For each item x on the list maintain a mod s counter $c(x)$, initially set randomly, independently and uniformly to a number in $\{0, 1, \ldots, s-1\}$. Upon a request for an item x, decrement $c(x)$ by 1 (mod s) and then if $c(x) \in S$ move x to the front.

Thus, CTR$(2, \{1\})$ is BIT. Reingold *et al.* prove that CTR$(7, \{0, 2, 4\})$ is 85/49-competitive (≈ 1.735). From this family we have tested algorithm CTR$(7, \{0, 2, 4\})$.

The family of algorithms RANDOM-RESET(s, D) (RST(s, D)) due to Reingold *et al.* [18] is a variation on the COUNTER algorithms. Let s be a positive integer and D, a probability distribution on the set $S = \{0, 1, \ldots, s-1\}$ such that for $i \in S$,$D(i)$ is the probability of i.

Algorithm RST(s, D): For each item x on the list maintain a counter $c(x)$, initially set randomly a number in $i \in \{0, 1, \ldots, s-1\}$ with probability $D(i)$. Upon a request for an item x, decrement $c(x)$ by 1. If $c(x) = 0$ then move x to the front and then randomly reset $c(x)$ using D.

The best RST(s, D) algorithm, in terms of the competitive ratio, is obtained with $s = 3$ and D such that $D(2) = (\sqrt{3} - 1)/2$ and $D(1) = (3 - \sqrt{3})/2$. The competitive ratio attained in this case is $\sqrt{3} \approx 1.732$. In this family we have only tested this algorithm.

Let $p \in [0, 1]$ The following is family of algorithm called TIMESTAMP(p) (TS(p)) due to Albers [1] that is a kind of randomized combination of algorithm TS and MTF. For each p, TS(p) is proven to be $\max\{2 - p, 1 + p(2 - p)\}$-competitive.

Algorithm TS(p): Upon a request for an item x, with probability p execute (i) move x to the front; and with probability $1 - p$ execute (ii) let y be the first item on the list such that either (a) y was not requested since the last request for x; or (b) y was requested exactly once since the last request for x and that request for y was served by the algorithm using step (ii). Insert x just in front of y. If there is no such y or if this is the first request for x leave x in place.

The optimal choice is easily shown to be $p = (3-\sqrt{5})/2$ in which case the competitive ratio attained is the golden ratio $\phi = (1+\sqrt{5})/2 \approx 1.62$. From this family we have tested this algorithm and denote it by TSP.

In [3], Albers, von Stengel and Wechner present the following algorithm called COMB which is a probability mixture of BIT and TS. Algorithm COMB is shown to be 8/5-competitive. To date this bound is the best known for a list accessing algorithm.

> **Algorithm COMB:** Before serving any request choose algorithm BIT with probability 4/5, and algorithm TS with probability 1/5. Serve the entire request sequence with the chosen algorithm.

Although for any data set the empirical performance of COMB can be determined from the performances of BIT and TS, we have tested COMB separately, as a control for the statistical significance of our tests of randomized algorithms.

6.1.3 A Benchmark algorithm

In order to have a reference result we tested the performance of the following "bad" algorithm, MTL, which acts in a way that seems to be the worst possible.

> **Algorithm MTL:** Upon a request for an item x move x to the back of the list.

Although the model requires that we charge MTL for the paid exchanges it performs we ignored these costs and only counted access costs.

6.2 Experiment 1: Access cost performance

In the access cost experiment, we made the algorithms serve several types of request sequences starting on an empty list. The goal was to rank the access cost performance of the various algorithms within the traditional dynamic model. The algorithms' performance was measured with respect to the full-cost model where an access to the ith item in the list is charged i. To obtain statistically significant results for the randomized algorithms, we averaged their results over 16 executions of each experiment. [12]

We examined:

- Request sequences produced by memoryless sources according to Zipf's law;
- Request sequences produced by Markov sources in order to examine the influence of locality;
- Request sequences extracted from the Calgary Corpus [5] (see Section 8.2).

We used the data type `random_source` of the LEDA library [16] to produce uniformly distributed random numbers. The access cost experiment is discussed in detail in Section 7.

[12] We computed 98% confidence intervals for the compression experiment we conducted. Just in one case two confidence interval did intersect. In all cases the intervals were very narrow such that our qualitive results holds for any mistake in the confidence interval.

6.3 Experiment 2: Compression performance

In the compression experiment, we measured the algorithms' performance as data compressors in the compression scheme of Bentley *et al.* [7]. The algorithms were ranked according to the compression ratio they obtained on the Calgary Corpus [5], a data set designed to test the performance of text compression algorithms. We examined both a byte-level and a word-level variant of this compression scheme. Here again the results for the randomized algorithms were averaged over 16 executions. The compression experiment is discussed in detail in Section 8.

7 Experiment 1: Access cost performance

7.1 Previous experimental studies

In this section, we describe our experiment for testing the access cost performance of online list-accessing algorithms. A few such empirical studies have been conducted in the past.

Bentley and McGeoch [6] tested the performance of MTF, FC, TRANS with respect to request sequences generated from several text files (4 files containing Pascal programs and 6 other English text files). The sequences generated from the text files by parsing the files to "words" with a word defined as an "alphanumeric string delimited by spaces or punctuation marks". It was found that FC is always superior to TRANS and that MTF is often superior to FC. Tenenbaum [23] tested the performance of various algorithms from the MOVE-AHEAD(k) family (MHD(k)) with respect to request sequences distributed by Zipf's law. A few other simulation results testing various properties of particular algorithms with respect to sequences generated via Zipf's distribution are summarized in a survey by Hester and Hirschberg [11].

7.2 Request sequences distributed by Zipf's law

7.2.1 Zipf sequences: description

In the first access cost experiment, we carry on the studies of Tenenbaum [23] and Hester and Hirschberg [11] and test our more extensive set of online list accessing algorithms on request sequences distributed by *Zipf's law*. Given m distinct items, Zipf's law assigns to the ith item the probability

$$p_i := \frac{1}{i \cdot H_m} \text{ for } 1 \leq i \leq m,$$

where $H_m = \sum_{j=1}^{m} \frac{1}{j}$ is the mth Harmonic number and serves to normalize the sum of probabilities to 1. The requested items are selected independently according to this distribution.

7.2.2 Zipf sequences: discussion of results

We performed the access cost experiment for the values $m = 8, 16$ and 32 on request sequences of length 10^6. The detailed results of in terms of cost ratios can be found in Table 4 in Appendix D

The results for the different values of m are similar. We observe that more conservative algorithms seem to be favorized. The most conservative algorithm in both cases is FC, and this algorithms indeed performs best. After a certain time, FC will almost certainly have ordered its list according to the given probability distribution and will not perform any update operations any more. Recall that FC is not optimal and not even competitive.

Interesting observations can be made for the families MRI(m) and PRI(m). First, we observe that MRI(m) always outperforms PRI(m). Secondly, both families exhibited consistent and perfect

monotonicity in terms of their performance as a function of the parameter m. The algorithms MRI(0) and PRI(0) achieved the best performance in their families. We observe a considerable increase in costs between MRI(0) and MRI(1).

The costs of the algorithms of the MHD(k) class increase with k. Again, more careful (or conservative) behavior is rewarded. We observe that the extrem member MTF of this class is among the worst performing algorithms on all examined sequences. On the other hand, TRANS (which is equivalent to MHD(1)) is always one of the best algorithms.

There is no consistent behaviour of the algorithms of the MF(k) family. Indeed, different Zipf sequences (among those presented here, but also others) yielded very different results. The reason is that these algorithms do not update the front of their list, and therefore their performance depends strongly on the first requested items. As for the members of the variant family MF$'$(k), these algorithms perform well, as long as the MTF-cache remains small.

Finally, the randomized algorithm have considerably higher costs than the best deterministic algorithms in all cases, and they mostly appear at the end of the field. These results stand in strong contrast to known analyses of randomized algorithms which say that randomization can yield algorithms with a competitivity below the deterministic lower bound of 2. However we should recall that in competitive analysis, requests are produced by an adversary whose goal is to increase the algorithm's costs with respect to an optimal offline algorithm. This includes that a request may depend on the algorithm's reactions on former requests, and in this case randomization on the algorithm's side may make the adversaries life more difficult. However in this experiment, request sequences are generated independently from each other without memory. Therefore, there is no reason to expect that the competitive results will be applicable here, and in particular that randomization will give any advantage.

7.3 Request sequences produced by Markov sources

7.3.1 Markov sequences: description

We say that a request sequence exhibits "locality of reference", if the number of distinct requested items in a small part of the sequence is considerably smaller than the number of distinct items in the entire sequence, or alternatively, if requests depend strongly on immediately preceding requests. It is clear that the degree of locality in the request sequence must have a considerable influence on the performance of online algorithms, depending on how quickly they adapt on local changes in the request sequence. In the second access cost experiment, we want to examine this influence.

We use a simple probabilistic model to produce request sequences with different degrees of locality. The model is based on *Markov sources*. A Markov source consists of a set of m states and is described by a transition matrix

$$(p_{\mu,\nu})_{\mu,\nu=1,\ldots,m} \text{ where } p_{\mu,\nu} \geq 0 \ \ (1 \leq \mu, \nu \leq m) \quad \text{and} \quad \sum_{\nu=1}^{m} p_{\mu,\nu} = 1 \ \ (1 \leq \mu \leq m).$$

Thus each line of the transition matrix gives a probability distribution, and if the Markov source is in state μ, the probability that is goes to state ν is exactly $p_{\mu,\nu}$. The initial state may again be chosen according to an initial distribution.

We now describe the Markov source used to generate request sequences with different degrees of locality. The source will request the item μ, if it is in state μ. Let $k, n \in \mathbb{N}$. The Markov source has $m = k \cdot n$ states which are partitioned in k disjoint n-element sets S_i

$$S_i := \{(i-1) \cdot n + 1, \ldots, i \cdot n\}.$$

We will call these sets the *local sets*. In order to receive request sequences with locality, the Markov source should have the following property. Once it enters a local set S_i, it is to stay there for a certain time before entering a new local set S_j $(i \neq j)$. The degree of locality is varied by a parameter α $(0 < \alpha < 1)$ denoting the probability that the next request is in the current local set.

It remains to specify the exact transition probabilities. For reasons of simplicity, we use a fixed probability distribution $(q_1, \ldots, q_n)$ where $q_j > 0$ for all $1 \leq j \leq n$. The probability that the jth smallest state in a local set is entered on the next request will be proportional to q_j. Let Q be the $(n \times n)$-transition matrix where all lines are identical to the vector $(q_1, \ldots, q_n)$. We are now ready to define the $(k \cdot n \times k \cdot n)$-transition matrix P of our Markov source:

$$P := P(k, n, \alpha, q_1, \ldots, q_n) := \begin{pmatrix} \alpha Q & \frac{1-\alpha}{k-1}Q & \cdots & \frac{1-\alpha}{k-1}Q \\ \frac{1-\alpha}{k-1}Q & \alpha Q & \cdots & \frac{1-\alpha}{k-1}Q \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1-\alpha}{k-1}Q & \frac{1-\alpha}{k-1}Q & \cdots & \alpha Q \end{pmatrix} \tag{6}$$

It is easy to verify that, by our choice of α and the q_j, all entries in P are non-negative and that the entries along each line of P sum up to 1, i.e. P is a valid transition matrix. Furthermore, we see that the probability for the source to stay in its current local set is exactly α.

The simple probabilistic model just described can easily be generalized by varying the sizes of the local sets and the transition probabilities within and between the local sets. Before specifying the parameters used in our experimental study, we want to show some properties confirming that the above model is reasonable for abstraction of locality.

The theory of Markov chains tells us that if all entries of P are positive, then the sequence $\{P^l\}_{l \geq 1}$ of transition matrixes converges towards a limiting matrix in which all lines are identical to a probability vector $(p_1, \ldots, p_m)$. This vector is called the *stationary distribution* of the Markov source. It is independent of the initial distribution and gives the probabilities for the occurrences of the states in an infinite sequence produced by the Markov source. The following Lemma 8 describes two important properties of our Markov source.

Lemma 8 *Let $k, n \in \mathbb{N}$, $0 < \alpha < 1$, and let $(q_1, \ldots, q_n)$ be a probability distribution with $q_j > 0$ for $1 \leq j \leq n$. Consider the Markov source with $m = k \cdot n$ states and the transition matrix P as defined in equation (6).*

1. *The expected number of subsequent requests in a local set is $1/(1-\alpha)$.*
2. *The stationary distribution $(p_1, \ldots, p_m)$ is given by*

$$p_{(i-1)n+j} = \frac{q_j}{k} \quad (1 \leq i \leq k, 1 \leq j \leq n). \tag{7}$$

The proof of Lemma 8 can be found in Appendix C. The first part of the lemma confirms that the parameter α is an indicator for the degree of locality: the closer α is to 1, the longer the Markov source stays in a local set, thus producing a request sequence with stronger locality. The second part of the lemma reveals an interesting property of our Markov source, namely that the stationary distribution is independent of the parameter α. This means that the frequencies of the requested items do not depend on the degree of locality in our request sequence. Only the number of request before a local set is left again varies with α. Furthermore, we see from the periodicity of the vector $(p_1, \ldots, p_m)$ in equation (7) that our Markov source requests each local set with the same probability.

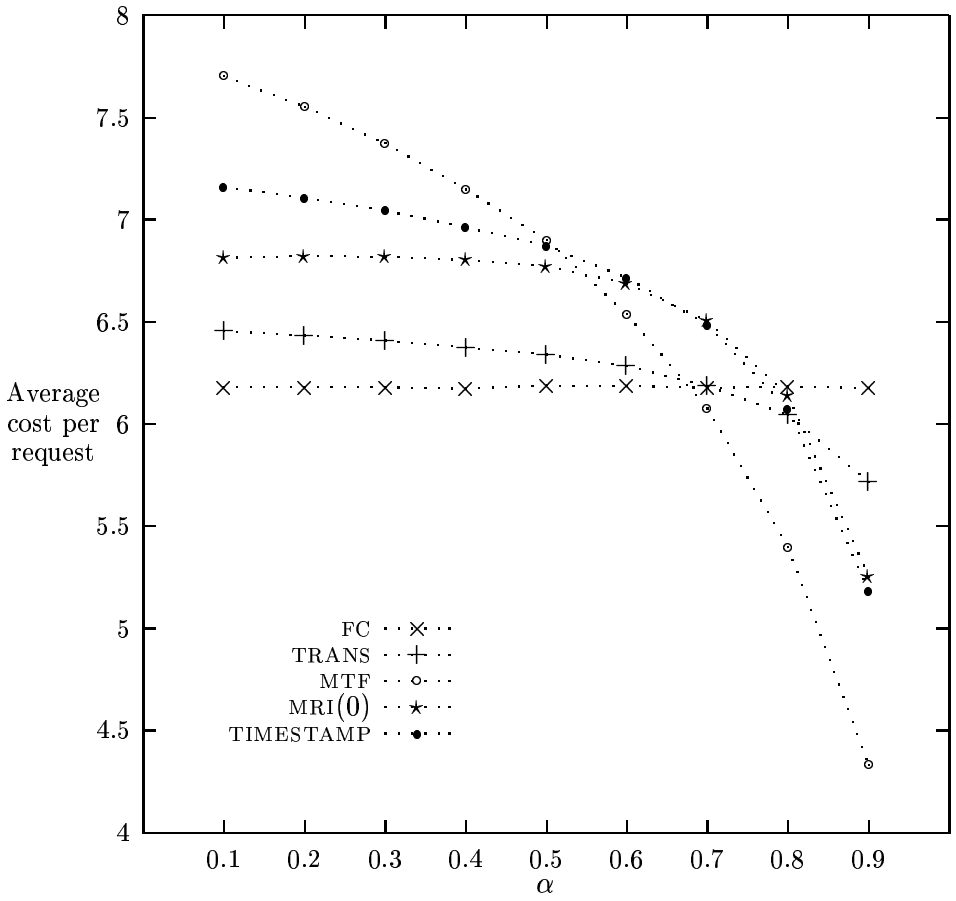

Figure 1: Average access cost per request when serving sequences of different degree of locality of some selected algorithms.

7.3.2 Markov sequences: discussion of results

For the experiment, we set $k = n = 4$ and chose the probabilities q_j for the distribution $(q_1, \ldots, q_n)$ according to Zipf's law. The probability α was varied in order to examine the influence of locality. We produced request sequences of length 10^6 for all values $\alpha = 0.1, 0.2, \ldots, 0.9$. The initial state was chosen at random. Table 5 in Appendix D presents the average access costs of all algorithms for the values $\alpha = 0.1, 0.5$ and 0.9. Figure 1 graphically presents the average access costs of some selected algorithms for all examined values of α.

Let us first turn to Figure 1. We observe that the degree of locality has a considerable influence on the performance of many, but not all algorithms. The algorithm FC has about the same costs for all values of α. After a certain time, its list will be ordered according to the limiting distribution which, as we have seen, is independent of α, and FC serves the entire sequence without remarkable reorganisations of its list. The results for the other algorithms show that the locality in the sequences produced by our Markov source may have a positive influence on the algorithms' performance. Their access costs decrease with increasing α. This effect is particularly strong for MTF, which is the worst performing algorithm for $\alpha = 0.1$, but performs best for $\alpha = 0.9$. Figure 1 shows – and

this is confirmed by the tables in Appendix D – that FC performs best while $\alpha \leq 0.6$. Only when the locality becomes very strong, FC is outperformed by other algorithms reacting faster on local changes.

We now turn to the complete results summarized in the tables of Appendix D. We start with the case $\alpha = 0.1$ and make the following observations. The algorithm FC performs best. The costs of the MHD(k) algorithms increase with k. The same holds for the algorithms of the MRI(m) and PRI(m) family when m increases. Furthermore, MRI(m) outperforms PRI(m) for each m. The MF$'$(k) family performs well as long as k is relatively small. All randomized algorithms perform considerably worse than the best deterministic algorithms.

In the case $\alpha = 0.5$, FC is still the best algorithm, however its lead over all other algorithms has decreased; the costs of the worst algorithm are only 15 % higher. The algorithms of the MHD(k) family still appear in order of k. Still MRI(m) outperforms PRI(m) for each m, but for both families the members for extreme values of m have higher costs than the members for modest values of m like $2 \leq m \leq 5$. Again the MF$'$(k) family performs well for k. All randomized algorithms appear at the end of the list, without exception having higher costs than MTF which is the second worst performing deterministic algorithm.

Let us now consider the interesting case of $\alpha = 0.9$, where we have a strong locality in our request sequence. We observe that the results in this case are extremely different from the results for the other values of α, and in some parts are even turned to the opposite. The algorithm FC is now among the worst algorithms, and its costs are more than 42 % over the costs of the now best performing algorithm MTF. The algorithms of the families MHD(k), MRI(m) and PRI(m) now appear ordered by decreasing k or m, respectively. Furthermore, PRI(m) outperforms MRI(m) for each considered value of m. This time the members of MF$'$(k) family show a relatively poor performance, whereas the randomized algorithms yield relatively good results.

The Markov model we used in this section to examine the influence of locality is certainly arguable. However, it is simple, and we were able to show some properties indicating that it is a reasonable abstraction of locality. In any case, the model was sufficient for our purposes, namely to illustrate that the ranking of list accessing algorithm depends strongly on properties of the served request sequence.

7.4 Request sequences extracted from the Calgary Corpus

In the third and last access cost experiment, we extracted request sequences from some files of the Calgary Corpus, basically following the approach of Bentley and McGeoch [6] who extracted request sequences from several Pascal and text files. More precisely, we extracted five sequences from each of the two corpus files `book1` and `geo` which are qualitatively rather different. Whereas `book1` contains pure English text and thus numerous repeating substrings, `geo` is a concatenation of 32-bit numbers. The first extracted sequence is the file itself, i.e. each individual byte is considered a request. For the second sequence, we considered each byte modulo 16, thus obtaining a related sequence producing a shorter list. The remaining sequences were obtained by application of the Burrows-Wheeler transformation [9] (BW transformation for short). For a finite input string, the BW transformation calculates a permutation in which there usually is a strong locality. We obtained the third request sequence by application of the BW transformation on the raw corpus file. The forth sequence was obtained from the third sequence by replacing each run of subsequent requests on the same item by one single request on that item. Finally, the fifth sequence was obtained by taking each byte in the forth sequence modulo 16. Table 2 shows the length and numbers of distinct items in the resulting request sequences. We decided to include the "projected" (modulo

sequence	length	distinct	sequence	length	distinct
book1_1	768771	82	geo_1	102400	256
book1_2	768771	16	geo_2	102400	16
book1_3	768771	82	geo_3	102400	256
book1_4	386264	82	geo_4	65778	256
book1_5	386264	16	geo_5	65778	16

Table 2: Lengths and numbers of distinct items in the request sequences extracted from the Calgary corpus.

16) sequences in the experiment, because the "raw", unprojected sequences produce rather long lists, and this setting is rather untypical for linear-list applications. However, the results indicate that there is relation between the projected and unprojected sequences.

The raw results of the access cost experiment on the request sequences extracted from the Calgary Corpus are summarized in Tables 6 and 7 of Appendix D.

8 Experiment 2: Compression performance

8.1 Description of the compression algorithm

Bentley *et al.* [7] proposed a compression scheme based on online list accessing algorithms. In this scheme, all possible symbols (e.g. characters or words) occurring in the input string are stored in a linear list. Whenever a symbol is to be encoded, a binary encoding of its current list position is transmitted (using some variable length prefix-free code). Before the next symbol is encoded, the list is reorganized by a list accessing algorithm. If the symbol is not in the list, it has to be transmitted specifically. To restore the data, the receiver performs the inverse operations. That is, the receiver can recover each symbol whose location was transmitted and updates its symbol list using the same list accessing algorithm. As the receiver has no knowledge about locations transmitted in the future, the list accessing algorithm has to work online. In the experiment described in this section, we tested the performance of all algorithms in a byte- and a word-level variant of the Bentley *et al.* compression scheme.

In the byte-level experiment, we start with a full list storing all 256 characters of the ASCII character set. Thus, we need a prefix-free code providing 257 codewords for the possible list positions and an additional end-of-file symbol. We performed the byte-level experiment with several variable length prefix-free encodings including Elias, δ-encoding, ω-encoding and several start-step-stop encodings. A brief description of these schemes can be found in Appendix E. The best results were obtained with a $(2, 2, 8)$-start-step-stop code improved by the phasing technique (see Table 8 in Appendix E).

In the word-level experiment, two disjoint word sequences are extracted from the input sequence. We denote all characters of the ASCII character set for which the C-library function `isspace` returns a non-zero value as *space characters*. All other characters are called *non-space characters*. A *space word* then is a maximum length sequence of subsequent space characters between two non-space characters. Analogously, a *non-space word* is a maximum length sequence of subsequent non-space characters between two space characters. We use two initially empty word-lists which can have arbitrary length. The first list stores all space words, the second list stores all non-space words. In the input sequence, space words and non-space words alternate, and the next word is always

encoded by its current position in the corresponding list using some prefix-free code. If a new word is encountered and ℓ is the current length of the list, we encode the list position $\ell+1$, thus switching into a character mode to transmit the characters of the new word. Here again, we use two lists, one for space characters and one for non-space characters. Furthermore, both lists contain an escape character to encode the end of the new word. After encoding the end of the new word, we switch back into word mode, and the new word is inserted in the word list according to the applied list accessing algorithm. To encode the end of the input sequence, we switch into character mode and then immediately back into word mode (this can be viewed as transmitting an empty word). All four lists are managed by the same list accessing algorithm. We use an Elias code to encode the positions in the space word list, and a δ-code to encode the positions in the non-space word list. We developed two special codes to encode the positions in the 7-element space character and the 251-element non-space character list. These two codes are described in Appendix E.

In both variants of the Bentley *et al.* compression scheme, we are only interested in measuring the compression performance and ignore space and time complexity issues. Also none of the algorithms was actually implemented to compress and decompress data. However, both settings are completely realistic from a practical point of view, because they take into account everything that is necessary to compress and decompress the data. In particular, we charge the costs contributed by escape symbols in our settings. This is often neglected, but we consider escape symbols an essential part of the compressed representation of an input sequence.

8.2 The data set

Exact analyses of compression algorithms are difficult, because they require a precise mathematical description of the input source. Real data is usually too complex for such a description.

In the practical field of data compression, an algorithm's performance is usually measured experimentally with respect to a benchmark corpus which is a relatively large data set supposed to be representative for future inputs to the algorithm. Such benchmark data sets exist for various fields of data compression such as text compression, or compression of images or audio data.

Bell, Cleary, and Witten [5] collected the *Calgary Corpus*. This corpus has become the standard corpus for lossless compression algorithms. It consists of (mainly text) files from various application fields including books, papers, numeric data, a picture, programs and object files. Descriptions of the files can be found in [5]. Table 3 lists some properties of the corpus files.

Recently, Arnold and Bell [4] proposed the *Canterbury Corpus* as an alternative corpus for lossless compression algorithms. The motivation for the new corpus was the fear of compression algorithms being tuned for the standard corpus and the possibility of the Calgary Corpus being too out-dated for today's applications. However, after having tested various compression algorithms on the new corpus, the authors admit that there are no considerable qualitative differences in the results obtained on the two data sets, and conclude that the Calgary Corpus is still appropriate to measure the performance of compression algorithms.

8.3 Previous experimental studies

A few empirical tests of the performance of some list accessing algorithms applied to text compression have been conducted. Bentley *et al.* [7] tested the performance of MTF-compression algorithms with various list ("cache") sizes with respect to several text files (containing several C and Pascal programs, book sections and transcripts of terminal sessions). They also compared the algorithms' performance with that of Huffman coding compression and found that for a sufficiently large cache size (e.g. 256) MTF-compression is "competitive" with Huffman coding. Albers and Mitzenmacher

file	characters			non-space words			space words		
	total	distinct	ratio	total	distinct	ratio	total	distinct	ratio
bib	111261	81	1373	19274	4282	4.50	19274	6	3212.33
book1	768771	82	9375	141274	21076	6.70	141274	17	8310.23
book2	610856	96	6363	101221	14974	6.76	101221	25	4048.84
geo	102400	256	400	926	444	2.09	925	10	92.50
news	377109	98	3848	53939	14974	3.60	53939	231	233.50
obj1	21504	256	84	767	541	1.42	767	18	42.61
obj2	246814	256	964	6819	4114	1.66	6818	36	189.39
paper1	53161	95	559	8512	2537	3.35	8512	18	472.89
paper2	82199	91	903	13829	3298	4.19	13829	10	1382.90
pic	513216	159	3227	164	161	1.02	163	6	27.17
progc	39611	92	430	6313	1937	3.26	6313	66	95.65
progl	71646	87	823	9235	2139	4.32	9235	68	135.81
progp	49379	89	554	4847	1530	3.17	4847	122	39.73
trans	93695	99	946	9334	2566	3.64	9333	72	129.62

Table 3: The Calgary Corpus. Total number of symbols, number of distinct symbols and the ratio between them with respect to byte- and word-parsing of the Calgary Corpus files. The corpus files can be downloaded from the ftp site `ftp.cpsc.ucalgary.ca`.

[2] compared the compression performance of TIMESTAMP and MTF with respect to the Calgary Corpus. They considered both word and byte (i.e. character) parsings. Using Elias encoding (see Appendix E), they obtained the following results. TIMESTAMP-compression is significantly better than MTF-compression with respect to byte parsing, but both are not "competitive" with standard Unix compression utilities. With respect to word parsing, TIMESTAMP is often (only marginally) better than MTF compression. The word-based compression performance of both these algorithms is found to be close to that of standard Unix compression utilities. However, note that the results of this experiment do not count the encoding of new words that are not already on the list. A few other studies compare the performance of particular, more sophisticated list accessing compressors that alter the basic scheme. Burrows and Wheeler [9] tested the performance of an MTF-compressor that operates on data that is first transformed via a "block-sorting" transformation. Grinberg *et al.* [10] tested the performance of a MTF-compressor that uses "secondary lists". We note (based on their results and ours) that these more sophisticated schemes achieve in general better compression results than the basic scheme.

From the above, the only comparable studies are those of Bentley *et al.*, and Albers and Mitzenmacher. First we note that our study supports the qualitative results of both these studies. However, they are not comparable quantitatively (the Bentley *et al.* study is incomparable to ours as it used a different corpus; the quantities reported in the Albers-Mitzenmacher study are incomparable to ours as they have not measured the transmission costs of new words).

Compared to the known empirical studies, the results reported here are significantly more comprehensive and provide insights into many algorithms, among which some that have never been tested.

8.4 Results of the compression experiment

The detailed results of the compression experiment can be found in Appendix F. Compression performance is usually measured in terms of the *compression ratio* which is the number of calculated output bits per input character. Tables 10 and 11 in Appendix F give the compression ratios achieved by all algorithms on each corpus file for the byte- and the word-level compression experiment, respectively.

Let us first turn to the byte-level compression experiment. Almost all list accessing algorithms indeed compress all input files, although the compression scheme is very simple and has no preknowledge about the input. Only some members of the MF(k) family expand some of the input files; these algorithms do not reorganize the front part of their lists. However, the compression performance of all algorithms is poor compared with standard compression programs such as `compress` and `gzip`. The best results are obtained by members of the families MF$'$(k) and MHD(k) for small values of k, FC and MRI(0) and PRI(0). Note that none of these algorithms is optimal or even competitive in terms of the competitive ratio. We observe that the members of the families MF$'$(k) and MHD(k) appear ordered by increasing k (modulo some exceptions for small values of k). The same holds for the families MRI(m) and PRI(m) with increasing m. This is remarkable, because these algorithms have more knowledge about past requests (or symbols) for greater values of m, but apparently they cannot take an advantage of this knowledge (w.r.t. this data set). The MRI family seems to be superior to the PRI family: MRI(m) outperforms PRI(m) for all values of m. Another remarkable observation is a considerable decline of the compression performance between MRI(0) and MRI(1), as well as between PRI(1) and PRI(2). The algorithm MTF is among the worst performing algorithms.

We observe that the randomized algorithms perform considerably worse than the best deterministic ones. Algorithm RMHD is by far the best randomized algorithm. A striking fact is that the ranking of the randomized algorithms exactly matches their competitive ratio ranking. For the algorithm COMB, it is important to mention that the best performance was consistently due to the deterministic algorithm TIMESTAMP and not to BIT.

In the word-level compression experiment, we observe again that all algorithms compressed all input files (except for some MF(k) algorithms that could not compress the file `pic`). The compression performance is considerable better than in the byte-level compression experiment. This is because longer substrings are encoded by relatively short codewords on the text files of the Calgary corpus, where our word-parsing is natural. However, we observe that on those inputs where our word-parsing is not natural, namely the binary files of the corpus, the word-level performance may be worse than in the byte-level setting. The relative inadequacy of our setting for these files is supported by the following observation made for all algorithms. For the binary files, most of the output bits were contributed by the character lists, whereas the word lists added only a marginally small fraction (e.g. for algorithm MTF and file `geo`, the character lists produced about 97 % of all output bits). During the compression of the text files however, the word and character lists contributed comparable fractions of bits to the output (e.g. for algorithm MTF and file `bib`, the character lists contributed about 42 % of all output bits).

The best results are obtained by members of the MF$'$(k) family for small values of k, the MRI family (in particular MRI(0)), PRI(0), PRI(1) (i.e. TIMESTAMP) and FC. Again the members of the families MRI(m) and PRI(m) appear ordered by increasing m, and we observe a considerable increase in cost between MRI(0) and MRI(1). For all considered values of m, MRI(m) outperforms PRI(m). In the word-level experiment however, the variants MRI$'$(m) and PRI$'$(m) which insert new items at the front have significantly higher costs than MRI(m) and PRI(m). The reason for this is the

fact that for all corpus files, more than half of the non-space words (and these dominate the costs) occur only once. Thus inserting a new word at the front of the list usually has the only effect of making the encoding of future words more expensive. The members of the MHD family appear in the second part of the field only, with the extrem members MTF and TRANS (but also MHD(2) and MHD(1024)) ranking among the worst algorithms.

For the randomized algorithms, we make similar observations as in the byte-level experiment. Again, RMHD is by far the best randomized algorithm and the ranking of the randomized algorithms almost matches their competitive ratio ranking (with the algorithms BIT and CTR being transposed in that ranking).

Finally we note that the results of this experiment support the qualitative results of the Albers-Mitzenmacher compression experiment [2]. Namely, the compression ratios obtained by algorithm TIMESTAMP were consistently better than those obtained by MTF-compression.

8.5 Some notes on the compression experiment

The word-level compression experiment shows that the Bentley *et al.* compression scheme may yield good results, if it makes use of some prior knowledge about the structure of the input, as our word-level setting did by assuming text inputs and parsing the input file into words. On the other hand, if the input is not of the assumed structure, schemes depending on prior knowledge usually perform poorly. Our compression experiment illustrates this problem by the poor results obtained for the highly compressible corpus file `pic`. Note that if one expects a "universal" compression algorithm, that compresses well on average (i.e. averaged over all the inputs it will ever see), then the compression algorithms in our word-level setting are not acceptable. Examples of universal or more powerful compression algorithms are the Context-tree weighting method, Ziv-Lempel algorithms, the PPM (prediction by partial match) schemes, and the block-sorting algorithm. These algorithms usually achieve better compression ratios than our word-level compression setting, though the latter is already a rather complicated variant of the basic Bentley *et al.* scheme.

9 Concluding remarks

The MRI family presented here exhibits some very attractive features. Nevertheless notice that the implementation of the algorithm MRI(m) is quite expensive in terms of time and memory. For the implementation of MRI(m) we need to maintain, for each item x on the list, an m-ary vector T_x containing the times (indices of requests) of the last m requests for x.

This paper leaves some questions open. For instance, it remains to determine the competitive ratio of the algorithms in the PRI family. We conjecture that for each $m > 0$ PRI(m) is 2-competitive.

Perhaps a more interesting goal would be to identify "the most conservative" algorithms that are still optimal thus expanding the set of optimal list accessing algorithms (with different characteristics) even more. Note that this question as stated is not well defined and in fact, one crucial step towards answering this question would be to define a meaningful measure of "conservatism".

To our knowledge, our experimental study is the first one comparing a large number of different online list accessing algorithms. Even so few interesting algorithm were left aside, mainly because they were introduced to us after we conducted our experiments. Two such interesting families of algorithms were developed by Schulz [20]: SORT-BY-RANK and SORT-BY-DELAY.

Some of the experimental results reported in this paper stand in contrast to various theoretical studies of the list accessing problem. In the case of the access cost experiment, the reader should

be aware that, because of the nature of the examined request sequences, our experimental results are hardly appropriate to support theoretical results obtained by competitive analysis.

In the access cost experiment, we tried to tie on previous experimental studies and to extend their results to our large algorithm set. In contrast to previous studies however, we also tried to cover a wider range of request sequences. The fact that we obtained quite different rankings for different request sequences indicates that, at least for the online list accessing problem, it is dangerous to restrict experimental studies to one particular class of input sequences. This may lead to false conclusions. We used an experimental approach to examine the influence of locality in request sequences on the performance of online list accessing algorithms. Our experiments show that the degree of locality has a considerable influence both on the algorithms' costs and on their ranking. It would be of major importance to devise a meaningful, quantitative measure of locality of reference that could be used to classify request sequences and further investigate the correlation between various algorithms and their performance with respect to sequences. To the best of our knowledge, no such measure has been studied. Also, it would be of great interest to put together an appropriate corpus that could be used to test the performance of data structures and algorithms for dictionary maintenance.

The results concerning compression performance clearly indicate that the list accessing compression scheme by itself will not give compression ratios that are competitive with popular compression algorithms such as those based on Lempel-Ziv schemes. Nevertheless, it is remarkable that a compression scheme as simple as the Bentley *et al.* scheme on byte-level is already capable of performing compression. Burrows and Wheeler [9] used the Bentley *et al.* scheme with MTF-encoding as backend for their very powerful block-sorting compression algorithm. For example, this method is used in the BZIP compression software. Our results in the access cost experiment with sequences generated by the BW scheme suggest that using other list accessing algorithms such as MF instead of MTF might yield even better results. Finally, the results of the access cost experiment suggest that it would be very interesting to experiment with dynamic transitions between different basic list accessing algorithms in order to adapt to changing levels of locality of reference.

Acknowledgments

We thank Susanne Albers, Allan Borodin, Brenda Brown, David Johnson, Steve Ponzio, Jeffrey Westbrook, and the anonymous referees for very useful comments that greatly improved the presentation and content.

References

[1] S. Albers. Improved randomized on-line algorithms for the list update problem. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 412–419, 1995.

[2] S. Albers and M. Mitzenmacher. Average case analyses of list update algorithms, with applications to data compression. Technical Report TR-95-039, International Computer Science Institute, 1995.

[3] S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995.

[4] R. Arnold and T. Bell. A corpus for the evaluation of lossless compression algorithms. In *Data Compression Conference*, pages 201–210, 1997.

[5] T. Bell, J.G. Cleary, and I.H. Witten. *Text Compression.* Prentice Hall, 1990.

[6] J.L. Bentley and C. McGeoch. Amortized analysis of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985.

[7] J.L. Bentley, D.D. Sleator, R.E. Tarjan, and V.K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.

[8] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[9] M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital System Research center, 1994.

[10] D. Grinberg, S. Rajagopalan, R. Venkatesan, and V.K. Wei. Splay trees for data compression. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[11] J.H. Hester and D.S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17(3):295–312, 1985.

[12] S. Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):202–208, June 1991.

[13] S. Irani, N. Reingold, J. Westbrook, and D.D. Sleator. Randomized competitive algorithms for the list update problem. In *Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms*, pages 251–260, 1991.

[14] A.R. Karlin, L. Rudolph, and D.D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):70–119, 1988.

[15] J. McCabe. On serial files with relocatable records. *Operations Research*, 13:609–618, July 1965.

[16] K. Mehlhorn, S. Näher, M. Seel, and C. Uhrig. *The LEDA user manual : version 3.6.* Max-Planck-Institut für Informatik, Saarbrücken, 1998.

[17] N. Reingold and J. Westbrook. Randomized algorithms for the list update problem. Technical Report YALEU/DcS/TR-804, Yale University, June 1990.

[18] N. Reingold, J. Westbrook, and D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11:15–32, 1994.

[19] R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, 19, 2:63–67, February 1976.

[20] F. Schulz. Two new families of list update algorithms. In *ISSAC'98, LCNS 1533*, pages 99–108. Springer, 1998.

[21] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[22] B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47:5–9, 1993.

[23] A. Tenenbaum. Simulations of dynamic sequential search algorithms. *Communications of the ACM*, 28(2):790–791, 1978.

[24] J. Westbrook, 1996. personal communication.

A Proof of Inequality (4) and Lemma 1

Denote by $\text{OPT}^*_{P;xy}(\sigma)$ the total cost incurred by OPT for *paid exchanges* between x and y while serving σ. Then a very similar development to the one leading to identity (3) with the inclusion of the costs incurred by OPT for paid exchanges easily yields

$$\text{OPT}^*(\sigma) = \sum_{\substack{\{x,y\}\subseteq L \\ x\neq y}} \left[\text{OPT}^*_{xy}(\sigma) + \text{OPT}^*_{P;xy}(\sigma)\right] \tag{8}$$

Also, it is not hard to see that OPT does not in general satisfy the Pairwise Property. Nevertheless, it can be easily shown that OPT does satisfy the following inequality for every pair of items, x and y

$$\text{OPT}^*(\sigma_{xy}) \leq \text{OPT}^*_{xy}(\sigma) + \text{OPT}^*_{P;xy}(\sigma), \tag{9}$$

To prove this inequality notice that the right hand side of (9) gives the total cost of some offline (not necessarily optimal) algorithm that is a "projection" of OPT over x and y. Namely, this projected algorithm operates on L_{xy} and serves the request sequence σ_{xy} according to the relative order of x and y in L as maintained by OPT while serving σ. An *optimal* offline algorithm for the two-item list, whose total cost in serving σ_{xy} is the left hand side of (9), surely pays no more than any other offline algorithm so the inequality must hold.

Equation (8) combined with inequality (9) yields inequality (4). That is,

$$\sum_{\substack{\{x,y\}\subseteq L \\ x\neq y}} \text{OPT}^*(\sigma_{xy}) \leq \text{OPT}^*(\sigma).$$

The extension of inequality (4) to the full-cost model can be made using an extended cost function instead of $\text{ALG}(x, r_j)$. Define the following extended cost function $\overline{\text{ALG}}(x, r_j)$,

$$\overline{\text{ALG}}(x, r_j) = \begin{cases} 1 + \frac{1}{\ell-1} & \text{if } x \text{ is in front of } r_j, \\ \frac{1}{\ell-1} & \text{if } r_j \text{ is in front of } x, \\ 0 & \text{otherwise } (x = r_j). \end{cases} \tag{10}$$

This cost function indirectly counts also the last (ith) comparison that must be counted in the full-cost model (that is, when we access the ith item). It is not hard to verify that with this extended cost function the equality (2) holds within the full-cost model. Analogous to $\text{ALG}^*_{xy}(\sigma)$, we now define $\text{ALG}_{xy}(\sigma)$ (using this extended cost function) and accordingly modify the equality (3), Equation (8), and the inequality (9). This proves Lemma 1.

B Proof of Lemma 6

Fix any request sequence σ. We prove by induction on i, the index of the ith request, that the invariant holds. The base case, $i = 1$, trivially holds. Assume the induction hypothesis for all $j < i$. We prove that the invariant holds for the ith request for an item x. (We refer to this request "the current request for x"; the previous request for x is referred to as "the last request for x".) Consider the configuration of the list L (of either MRI(1) or TIMESTAMP). Suppose, by contradiction, that the invariant does not hold. Let y be the first element in L that was requested less than twice since the last request for x. Let z be the last element in L that was requested twice or more since the last for x. By the contradiction assumption y appears in front of z. Let i' be the index of the

last request for x. Let r_{z_1} and r_{z_2} be the first and second requests for z after the i'th request (for x), respectively. (r_{z_1} and r_{z_2} exist by the definition of z). By the definition of y, y was requested at most once between the i'th and ith requests (for x) and therefore, was requested at most once between r_{z_1} and r_{z_2}. By the induction hypothesis for the request r_{z_2} and by the definitions of MRI(1) and TIMESTAMP it must be that z passed y at that time (if z was not already in front of y).

It may be the case though that y was requested once after the second request for z (i.e. r_{z_2}) and before the current request for x (i.e. the ith request). By the above derivation, it must be that just before such a request for y, z appears in front of y. We now prove that y will remain after z. By the induction hypothesis for this request for y and by the definitions of MRI(1) and TIMESTAMP it cannot be that y will pass z since z must have been requested at least twice between this request for y and the previous one. Therefore, z must be in front of y and the invariant holds.

C Proof of Lemma 8

To see the first part of the lemma, we observe that the expected number of subsequent requests in a local set is

$$\sum_{t=1}^{\infty} t \cdot \alpha^{t-1} \cdot (1-\alpha).$$

It is straightforward to verify that the value of this sum is $1/(1-\alpha)$.

To show the second part of the lemma, we make use of a result from the theory of Markov chains. This result tells us that the stationary distribution of a Markov source with transition matrix P is the unique probability vector $(p_1, \ldots, p_m)$ satisfying

$$(p_1, \ldots, p_m) \cdot P = (p_1, \ldots, p_m). \tag{11}$$

We show that, if P is the transistion matrix defined in equation (6) and the probability vector $(p_1, \ldots, p_m)$ is defined according to equation (7), condition (11) is satisfied which implies the second part of the lemma.

Let $(a_1, \ldots, a_m)$ be the vector resulting from the matrix multiplication on the left side of equation (11), i.e.

$$\begin{aligned}(a_1, \ldots, a_m) &:= (p_1, \ldots, p_m) \cdot P \\ &= (\tfrac{q_1}{k}, \ldots, \tfrac{q_n}{k}, \ldots, \tfrac{q_1}{k}, \ldots, \tfrac{q_n}{k}) \cdot \begin{pmatrix} \alpha Q & \frac{1-\alpha}{k-1}Q & \cdots & \frac{1-\alpha}{k-1}Q \\ \frac{1-\alpha}{k-1}Q & \alpha Q & \cdots & \frac{1-\alpha}{k-1}Q \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1-\alpha}{k-1}Q & \frac{1-\alpha}{k-1}Q & \cdots & \alpha Q \end{pmatrix}.\end{aligned}$$

Remember that $m = k \cdot m$ and that Q is the $(n \times n)$-matrix with all lines identical to the vector $(q_1 \ldots, q_n)$. Because of the form of the matrix P and because the entries of $(p_1, \ldots, p_m)$ repeat with period n, we can restrict our attention to the first n entries of $(a_1, \ldots, a_n)$ and the first n columns of P. The entry a_j is the scale product of $(p_1, \ldots, p_m)$ and the jth column of P, i.e.

$$a_j = \sum_{\mu=1}^{n} \frac{q_\mu}{k} \cdot \alpha\, q_j + (k-1) \sum_{\mu=1}^{n} \frac{q_\mu}{k} \cdot \frac{1-\alpha}{k-1}\, q_j.$$

The simplication of the right side yields

$$a_j = \alpha\, \frac{q_j}{k} + (1-\alpha)\, \frac{q_j}{k} = \frac{q_j}{k} = p_j.$$

Thus $a_j = p_j$ and the lemma follows immediately.

D Results of the access cost experiment

This section gives the detailed results of the access cost experiments we performed. The algorithms are ranked in terms of the ratio of the algorithms' cost and the cost of the algorithm that performed best among the examined algorithms. Additionally, each table gives the average access cost per request of the best algorithm. Thus, each algorithm's average access cost can be computed as the product of its cost ratio and the average access cost of the best algorithm.

Table 4 gives the results of the access cost experiment on Zipf sequences with $m = 8, 16$ and 32 distinct items. The results of the Markov experiment are listed in Table 5 for the values $\alpha = 0.1, 0.5$ and 0.9. Finally, Tables 6 and 7 present the results obtained on the request sequences extracted from the Calgary Corpus.

$m = 8$		Average access costs for FC:							2.944558
FC	1.00000	MRI′(1)	1.15123	PRI′(3)	1.18213	COMB	1.20438	PRI′(7)	1.22135
FC′	1.00000	PRI′(1)	1.15123	MRI(5)	1.18688	TS	1.20527	PRI(8)	1.22513
TRANS	1.09380	MRI(2)	1.15128	MRI′(5)	1.18688	PRI(5)	1.20877	PRI′(8)	1.22514
MF′(2)	1.11933	MRI′(2)	1.15129	MF(4)	1.18924	PRI′(5)	1.20878	SPL	1.23019
MRI(0)	1.12173	PRI(2)	1.16100	MHD(4)	1.19339	MRI(8)	1.20911	RST	1.23211
MRI′(0)	1.12174	PRI′(2)	1.16100	MRI(6)	1.19593	MRI′(8)	1.20912	RMTF	1.23827
MF′(4)	1.12440	MF(8)	1.16276	MRI′(6)	1.19594	CTR	1.21443	MTF	1.23859
MHD(2)	1.13362	MRI(3)	1.16301	PRI(4)	1.19789	PRI(6)	1.21612	MF(2)	1.27761
PRI(0)	1.13998	MRI′(3)	1.16301	PRI′(4)	1.19790	PRI′(6)	1.21612	MTL	1.81788
PRI′(0)	1.13998	MRI(4)	1.17591	RMHD	1.19918	MTF2	1.21624		
PRI(1)	1.15123	MRI′(4)	1.17592	MRI(7)	1.20319	BIT	1.21675		
MRI(1)	1.15123	PRI(3)	1.18213	MRI′(7)	1.20319	PRI(7)	1.22134		
$m = 16$		Average access costs for FC:							4.736801
FC′	1.00000	PRI(0)	1.16058	MRI(4)	1.19752	MRI′(7)	1.22941	BIT	1.26165
FC	1.00001	PRI′(0)	1.16058	MRI′(4)	1.19752	MRI(8)	1.23697	PRI(6)	1.26415
TRANS	1.06667	PRI(1)	1.17073	MRI(5)	1.20988	MRI′(8)	1.23697	PRI′(6)	1.26415
MF(4)	1.08473	MRI(1)	1.17073	MRI′(5)	1.20988	PRI(4)	1.24346	PRI(7)	1.26995
MF′(4)	1.09356	PRI′(1)	1.17073	MF(8)	1.21144	PRI′(4)	1.24346	PRI′(7)	1.26995
MHD(2)	1.09706	MRI′(1)	1.17073	RMHD	1.21928	COMB	1.24470	PRI(8)	1.27417
MF(2)	1.10686	MRI(2)	1.17084	MRI(6)	1.22050	TS	1.24570	PRI′(8)	1.27417
MF′(2)	1.12336	MRI′(2)	1.17084	MRI′(6)	1.22050	MF(16)	1.25393	SPL	1.27757
MRI(0)	1.12443	MRI(3)	1.18383	PRI(3)	1.22439	PRI(5)	1.25581	RST	1.28343
MRI′(0)	1.12443	MRI′(3)	1.18383	PRI′(3)	1.22439	PRI′(5)	1.25581	RMTF	1.29226
MF′(8)	1.14066	PRI(2)	1.19473	MHD(8)	1.22648	CTR	1.25882	MTF	1.29229
MHD(4)	1.14899	PRI′(2)	1.19473	MRI(7)	1.22941	MTF2	1.26165	MTL	2.29655
$m = 32$		Average access costs for FC:							7.882057
FC	1.00000	PRI′(0)	1.17340	MRI(5)	1.21923	PRI′(3)	1.25997	PRI′(7)	1.30480
FC′	1.00000	MF(2)	1.17563	MRI′(5)	1.21923	COMB	1.27125	PRI(8)	1.30880
TRANS	1.04362	MF(4)	1.17857	PRI(2)	1.22562	TS	1.27254	PRI′(8)	1.30880
MHD(2)	1.06413	PRI(1)	1.18144	PRI′(2)	1.22563	PRI(4)	1.27956	SPL	1.30888
MF′(4)	1.07744	MRI(1)	1.18144	RMHD	1.22564	PRI′(4)	1.27957	MF(8)	1.31182
MF′(8)	1.08659	MRI′(1)	1.18145	MRI(6)	1.23015	CTR	1.28880	RST	1.31818
MHD(4)	1.10116	PRI′(1)	1.18146	MRI′(6)	1.23016	PRI(5)	1.29148	RMTF	1.32862
MRI(0)	1.11243	MRI(2)	1.18173	MRI(7)	1.23947	PRI′(5)	1.29149	MTF	1.32908
MRI′(0)	1.11248	MRI′(2)	1.18174	MRI′(7)	1.23947	BIT	1.29185	MF(16)	1.44115
MF′(2)	1.12620	MRI(3)	1.19374	MRI(8)	1.24762	MTF2	1.29230	MF(32)	1.45571
MF′(16)	1.15758	MRI′(3)	1.19374	MRI′(8)	1.24762	PRI(6)	1.29933	MTL	2.85741
MHD(8)	1.16194	MRI(4)	1.20706	MHD(16)	1.25063	PRI′(6)	1.29933		
PRI(0)	1.17338	MRI′(4)	1.20706	PRI(3)	1.25997	PRI(7)	1.30479		

Table 4: Costs ratios on request sequences of length 10^6 produced by distributions according to Zipf's law.

$\alpha = 0.1$		Average access costs of FC:							
FC	1.00000	PRI(1)	1.15824	MRI(4)	1.19342	MTF2	1.21387	PRI(5)	1.23545
FC′	1.00006	MRI(1)	1.15824	MRI′(4)	1.19342	PRI(3)	1.21695	PRI′(5)	1.23545
TRANS	1.04413	MRI′(1)	1.15825	MF(2)	1.20160	PRI′(3)	1.21695	MF(8)	1.23712
MF′(4)	1.06489	PRI′(1)	1.15825	TS	1.20295	MRI(7)	1.22078	MF(16)	1.23717
MHD(2)	1.07198	MRI(2)	1.16639	MRI(5)	1.20464	MRI′(7)	1.22078	PRI(6)	1.23911
MF′(8)	1.09867	MRI′(2)	1.16639	MRI′(5)	1.20465	SPL	1.22318	PRI′(6)	1.23911
MRI(0)	1.10224	MRI(3)	1.18004	COMB	1.20692	RMTF	1.22624	PRI(7)	1.24133
MRI′(0)	1.10225	MRI′(3)	1.18004	MF(4)	1.20793	MRI(8)	1.22633	PRI′(7)	1.24133
MF′(2)	1.10948	RMHD	1.18844	CTR	1.20928	MRI′(8)	1.22633	PRI(8)	1.24272
MHD(4)	1.12091	PRI(2)	1.19312	MRI(6)	1.21369	PRI(4)	1.22911	PRI′(8)	1.24273
PRI(0)	1.14681	PRI′(2)	1.19312	MRI′(6)	1.21370	PRI′(4)	1.22911	MTF	1.24598
PRI′(0)	1.14682	MHD(8)	1.19339	BIT	1.21382	RST	1.23180	MTL	1.50408
$\alpha = 0.5$		Average access costs of FC′:							
FC′	1.00000	MRI(3)	1.09185	MRI′(2)	1.09579	PRI′(5)	1.10754	PRI′(0)	1.11356
FC	1.00015	MRI′(3)	1.09185	MRI(6)	1.09646	PRI(6)	1.10967	MTF	1.11488
TRANS	1.02459	MRI(4)	1.09194	MRI′(6)	1.09646	PRI′(6)	1.10967	RMHD	1.12752
MF′(4)	1.03065	MRI′(4)	1.09195	MRI(7)	1.09902	PRI(1)	1.11027	RST	1.13039
MF′(8)	1.03426	MRI(5)	1.09382	MRI′(7)	1.09903	MRI(1)	1.11027	TS	1.13227
MHD(2)	1.03884	MRI′(5)	1.09382	PRI(3)	1.09932	PRI′(1)	1.11027	COMB	1.13382
MF(8)	1.06012	MRI(0)	1.09396	PRI′(3)	1.09933	MRI′(1)	1.11028	BIT	1.13384
MF(4)	1.06124	MRI′(0)	1.09398	MRI(8)	1.10142	PRI(7)	1.11121	MTF2	1.13425
MHD(4)	1.06429	PRI(2)	1.09425	MRI′(8)	1.10142	PRI′(7)	1.11121	SPL	1.13712
MF(2)	1.06585	PRI′(2)	1.09426	PRI(4)	1.10429	PRI(8)	1.11221	CTR	1.13975
MF′(2)	1.06757	MHD(8)	1.09536	PRI′(4)	1.10430	PRI′(8)	1.11222	RMTF	1.15764
MF(16)	1.08914	MRI(2)	1.09578	PRI(5)	1.10754	PRI(0)	1.11355	MTL	1.63188
$\alpha = 0.9$		Average access costs of MTF:							
MTF	1.00000	PRI′(3)	1.05345	BIT	1.11882	MF′(2)	1.16377	MRI′(0)	1.21271
PRI′(8)	1.00875	PRI(3)	1.05347	MTF2	1.11889	MRI′(2)	1.17234	MRI(0)	1.21273
PRI(8)	1.00877	RST	1.06896	MRI′(5)	1.12386	MRI(2)	1.17236	MF′(8)	1.22794
PRI′(7)	1.01150	MRI′(8)	1.08935	MRI(5)	1.12387	RMHD	1.17570	MHD(2)	1.25494
PRI(7)	1.01152	MRI(8)	1.08936	COMB	1.13791	PRI′(1)	1.19557	MF′(4)	1.27570
PRI′(6)	1.01547	PRI′(2)	1.09622	MRI′(4)	1.13793	MRI′(1)	1.19558	TRANS	1.31932
PRI(6)	1.01549	PRI(2)	1.09624	MRI(4)	1.13795	PRI(1)	1.19560	FC′	1.42510
PRI′(5)	1.02198	MRI′(7)	1.09962	TS	1.14060	MRI(1)	1.19560	FC	1.42569
PRI(5)	1.02200	MRI(7)	1.09963	MRI′(3)	1.15392	PRI′(0)	1.19735	MF(4)	1.44681
PRI′(4)	1.03276	SPL	1.10978	MRI(3)	1.15393	PRI(0)	1.19737	MF(8)	1.71888
PRI(4)	1.03278	MRI′(6)	1.11094	MHD(4)	1.15738	RMTF	1.19925	MF(16)	1.86449
MHD(8)	1.05059	MRI(6)	1.11096	CTR	1.15800	MF(2)	1.20043	MTL	2.92394

Table 5: Costs ratios on request sequences of length 10^6 produced by the Markov source of Section 7.3.1 for the values $\alpha = 0.1, 0.5$ and 0.9.

E Variable length prefix-free binary encodings

In the compression experiment, we examined various variable length prefix-free binary coding schemes. We now give short descriptions of some standard coding schemes. Then we explain the phasing technique which can often be applied to improve codeword sets. Finally, we give precise descriptions of the codeword sets used in our experiments.

Standard coding schemes

We start with four coding schemes providing an infinite number of codewords. Such coding schemes are used, if no upper bound on the number of codewords (or alternatively the integers to be encoded) is known.

Elias encoding: To encode the integer $i \geq 1$, we transmit the unary encoding of the length of the binary representation of i, followed by the binary representation of i itself, excluding the most significant bit. Thus i is encoded with $2\lfloor \log_2 i \rfloor + 1$ bits.

δ-encoding: For an integer $i \geq 1$, the length of the binary representation of i is encoded by Elias

encoding, followed by the binary representation of i excluding the most significant bit. Thus i is encoded with $1 + \lfloor \log_2 i \rfloor + 2\lfloor \log_2(1 + \lfloor \log_2 i \rfloor) \rfloor$ bits.

ω-encoding: To encode $i \geq 1$, the binary representation $\beta(i)$ of i is written. Then the binary representation of $|\beta(i)| - 1$ is appended to the left. This process is repeated recursively and halts on the left with 2 bits. A single zero is appended on the right to mark the end of the codeword.

ω'-encoding: This coding scheme is similar to the ω-encoding, but halts with 3 bits.

If only a finite number of codewords is needed, coding schemes like the ones above are usually inefficient, and techniques like the following often yield better encodings.

Start-step-stop-encodings: The start-step-stop family produces a great variety of codes. Each member of the family is specified by three parameters a, b and c. An (a, b, c)-start-step-stop code uses k different binary encodings of lengths $a, a+b, a+2b, \ldots, a+c$. That is, it consists of $k = (c-a)/b + 1$ different binary codes. To indicate which binary code s with $1 \leq s \leq k$ is used, the codeword is prefixed by the unary representation of s. Thus, an (a, b, c)-code provides

$$\sum_{i=0}^{(c-a)/b} 2^{a+ib} = \frac{2^{b+c} - 2^a}{2^b - 1}$$

codewords.

Phasing

If only a finite number of codewords is needed, variable length prefix-free codes can often be improved by the so-called *phasing technique*.

If integers i with $1 \leq i \leq m$ are to be encoded, i.e. m codewords are needed, each integer can be encoded with $k = \lceil \log_2 m \rceil$ bits. However if $m < 2^k$, then $2^k - m$ codewords remain unused. In this case, the coding can be done in the following way. If $i \leq 2^k - m$ then i is encoded in $k-1$ bits by the binary representation of $i - 1$. If $i > 2^k - m$, then $i + 2^k - m$ is encoded in k bits by the binary representation of $i - 1 + 2^k - m$.

Start-step-stop codes can often be improved by the phasing technique, if not all codewords in one of the binary subcodes are needed. The codes used in the compression experiment have been improved in this way.

Codeword sets used in the compression experiment

In the byte-level compression experiment of Section 8, we obtained the best results with a $(2, 2, 8)$-start-step-stop code. By the above formula, this code provides 340 codewords, however only 257 are needed. Therefore, we applied the phasing technique on the last binary code of length 8, thus obtaining the codeword set described in Table 8. The colons in the codewords mark the ends of codeword prefixes and are not transmitted.

Positions in the 7-element space character list are encoded using a 2/3-bit binary encoding optimized by the phasing technique. To encode the 251 positions in the non-space list, we developped a special code described in Table 9 of Appendix E. The first 56 list positions are encoded by a $(3, 1, 5)$-start-step-stop code. The remaining positions are encoded by the prefix 000 followed by an 8-bit binary representation again optimized by the phasing technique.

The word-level compression algorithm of Section 8 works with two word lists of arbitrary length. For these two lists, we therefore need coding schemes providing an infinite number of codewords. In the experiment, we use an Elias encoding to encode locations in the space word list and δ-encoding in the non-space word list. However, the two character lists have only finite length, and we developped special codeword sets for them. We use a simple 2/3-bit binary code improved by the phasing technique for the 7-element space character list. The non-space character list holds 251 characters, thus we need 251 codewords. The first 56 codewords are taken from a $(3, 1, 5)$-start-step-stop code. The remaining codewords are provided by a 7/8-bit binary code (improved by the phasing technique) where each codeword is prefixed by 000. The two character list codeword sets are given in Table 9.

F Results of the compression experiment

Tables 10 and 11 show the results of the compression experiment with the Bentley *et al.* scheme on byte- and word-level, respectively. The tables give the compression ratios, that is the number of produced output bits per input character, achieved on the files of the Calgary corpus. The algorithms appear ordered by their average compression ratio with respect to the entire corpus which is given in the last column.

Sequence book1_1		Average access costs of FC′:									9.095565
FC′	1.00000	MHD(8)	1.16435	MRI(5)	1.23910	MRI′(8)	1.27256	PRI(5)	1.32950	MF′(64)	1.37421
FC	1.00005	MF′(16)	1.17075	MRI′(5)	1.23925	MF(4)	1.27333	PRI′(5)	1.32965	MHD(64)	1.37863
TRANS	1.03647	PRI(1)	1.17541	RMHD	1.24432	TS	1.29060	PRI(6)	1.33857	MTF	1.37932
MHD(2)	1.05572	MRI(1)	1.17541	PRI(2)	1.24643	PRI(3)	1.29110	PRI′(6)	1.33871	MF(8)	1.60552
MF′(4)	1.07189	PRI′(1)	1.17565	PRI′(2)	1.24662	PRI′(3)	1.29127	SPL	1.34041	MF(16)	2.18731
MF′(8)	1.07918	MRI′(1)	1.17565	MRI(6)	1.25152	COMB	1.30828	PRI(7)	1.34502	MF(32)	2.59891
MHD(4)	1.09437	MRI(2)	1.18766	MRI′(6)	1.25167	CTR	1.30985	PRI′(7)	1.34516	MF(64)	2.98101
MRI(0)	1.09557	MRI′(2)	1.18787	MF(2)	1.25422	PRI(4)	1.31529	PRI(8)	1.34957	MF(128)	2.98403
MRI′(0)	1.09600	MRI(3)	1.20698	MRI(7)	1.26234	PRI′(4)	1.31545	PRI′(8)	1.34970	MTL	7.33361
MF′(2)	1.13761	MRI′(3)	1.20716	MRI′(7)	1.26249	MTF2	1.31641	MHD(32)	1.35147		
PRI(0)	1.15628	MRI(4)	1.22438	MHD(16)	1.26838	BIT	1.31715	RST	1.35579		
PRI′(0)	1.15661	MRI′(4)	1.22454	MRI(8)	1.27242	MF′(32)	1.32084	RMTF	1.35673		
Sequence book1_2		Average access costs of FC:									5.856458
FC	1.00000	PRI′(0)	1.15111	MF(2)	1.20769	CTR	1.24776	PRI′(3)	1.27006	PRI(5)	1.29605
FC′	1.00003	MHD(4)	1.15509	MRI(3)	1.22060	COMB	1.25207	SPL	1.27272	PRI′(5)	1.29606
TRANS	1.05967	MF(4)	1.15732	MRI′(3)	1.22061	MRI(5)	1.25485	MRI(7)	1.27452	PRI(6)	1.30116
MHD(2)	1.09470	PRI(1)	1.17651	RMHD	1.22224	MRI′(5)	1.25485	MRI′(7)	1.27453	PRI′(6)	1.30117
MF′(4)	1.09490	MRI(1)	1.17651	PRI(2)	1.23522	MTF2	1.25687	MF(16)	1.28068	PRI(7)	1.30431
MRI(0)	1.10800	MRI′(1)	1.17652	PRI′(2)	1.23523	BIT	1.25723	MRI(8)	1.28217	PRI′(7)	1.30432
MRI′(0)	1.10802	PRI′(1)	1.17652	MRI(4)	1.23998	MRI(6)	1.26598	MRI′(8)	1.28218	PRI(8)	1.30637
MF′(2)	1.13627	MRI(2)	1.19751	MRI′(4)	1.23999	MRI′(6)	1.26599	RST	1.28605	PRI′(8)	1.30638
MF′(8)	1.13871	MRI′(2)	1.19752	TS	1.24061	RMTF	1.26778	PRI(4)	1.28717	MTF	1.31255
PRI(0)	1.15109	MF(8)	1.19787	MHD(8)	1.24381	PRI(3)	1.27005	PRI′(4)	1.28718	MTL	1.59018
Sequence book1_3		Average access costs of MF′(2):									3.285502
MF′(2)	1.00000	PRI′(4)	1.01111	PRI(1)	1.01896	MHD(4)	1.02348	MHD(2)	1.05795	MF(4)	1.86620
PRI′(2)	1.00071	MRI(5)	1.01284	MRI(1)	1.01896	PRI(8)	1.02529	MTF2	1.06190	MF(8)	2.72949
PRI(2)	1.00079	MRI′(5)	1.01303	PRI(0)	1.01899	PRI′(8)	1.02530	BIT	1.06213	FC′	2.80342
PRI(3)	1.00529	MRI(0)	1.01356	MRI′(1)	1.01931	MF′(4)	1.03040	RST	1.06268	FC	2.80440
PRI′(3)	1.00532	MRI′(0)	1.01425	PRI′(0)	1.01950	MHD(16)	1.03885	SPL	1.06660	MF(16)	4.18253
MRI(3)	1.00826	MRI(6)	1.01541	PRI(6)	1.01962	MHD(32)	1.04646	MF′(8)	1.07086	MF(32)	5.67824
MRI′(3)	1.00850	MRI′(6)	1.01558	PRI′(6)	1.01964	MHD(64)	1.04796	CTR	1.07303	MF(64)	6.47293
MRI(2)	1.00883	PRI(5)	1.01600	MRI(8)	1.01999	MTF	1.04799	MF′(32)	1.08724	MF(128)	6.48420
MRI′(2)	1.00911	PRI′(5)	1.01603	MRI′(8)	1.02015	COMB	1.05136	MF′(16)	1.09157	MTL	22.64814
MRI(4)	1.01016	MRI(7)	1.01771	MHD(8)	1.02157	MF′(64)	1.05219	RMTF	1.11358		
MRI′(4)	1.01038	MRI′(7)	1.01788	PRI(7)	1.02290	RMHD	1.05329	TRANS	1.12857		
PRI(4)	1.01109	PRI′(1)	1.01842	PRI′(7)	1.02292	TS	1.05705	MF(2)	1.27201		
Sequence book1_4		Average access costs of MRI(0):									5.409122
MRI(0)	1.00000	PRI(2)	1.05432	RMHD	1.07675	PRI′(5)	1.09649	PRI′(8)	1.11045	MF(4)	1.53544
MRI′(0)	1.00129	PRI′(2)	1.05464	MRI(6)	1.08117	MF′(8)	1.09974	SPL	1.11601	MF(8)	1.97297
PRI(0)	1.01590	MHD(4)	1.05870	MRI′(6)	1.08171	PRI(6)	1.10201	MHD(16)	1.11969	FC′	2.03327
PRI′(0)	1.01696	MRI(3)	1.05892	MHD(8)	1.08262	PRI′(6)	1.10237	RST	1.12550	FC	2.03420
MF′(2)	1.01945	MRI′(3)	1.05954	MRI(7)	1.08610	COMB	1.10280	MHD(32)	1.13373	MF(16)	2.89527
PRI′(1)	1.04011	MHD(2)	1.06304	MRI′(7)	1.08663	TRANS	1.10499	MHD(64)	1.13672	MF(32)	3.74048
PRI(1)	1.04040	MRI(4)	1.06789	PRI(4)	1.08707	PRI(7)	1.10658	MTF	1.13681	MF(64)	4.21664
MRI(1)	1.04040	MRI′(4)	1.06847	PRI′(4)	1.08744	BIT	1.10677	MF′(64)	1.14082	MF(128)	4.22191
MRI′(1)	1.04121	PRI(3)	1.07381	MRI(8)	1.09038	PRI′(7)	1.10694	RMTF	1.14943	MTL	13.52619
MF′(4)	1.04235	PRI′(3)	1.07422	MRI′(8)	1.09089	MTF2	1.10717	MF′(16)	1.15453		
MRI(2)	1.04883	MRI(5)	1.07513	TS	1.09443	CTR	1.10967	MF′(32)	1.16771		
MRI′(2)	1.04952	MRI′(5)	1.07568	PRI(5)	1.09613	PRI(8)	1.11010	MF(2)	1.18297		
Sequence book1_5		Average access costs of MRI(0):									3.989589
MRI(0)	1.00000	MRI′(1)	1.04971	MRI′(4)	1.08894	MRI′(6)	1.10531	PRI(5)	1.12040	PRI′(8)	1.13483
MRI′(0)	1.00005	MF′(4)	1.05593	RMHD	1.09001	CTR	1.10942	PRI′(5)	1.12042	RMTF	1.14043
PRI(0)	1.01748	MRI(2)	1.06326	COMB	1.09249	PRI(4)	1.11069	SPL	1.12427	MTF	1.15526
PRI′(0)	1.01752	MRI′(2)	1.06329	PRI(3)	1.09608	PRI′(4)	1.11072	PRI(6)	1.12670	MF(2)	1.26173
TRANS	1.03313	PRI(2)	1.07402	PRI′(3)	1.09611	MRI(7)	1.11118	PRI′(6)	1.12672	MF(4)	1.42862
MF′(2)	1.03793	PRI′(2)	1.07405	MRI(5)	1.09792	MRI′(7)	1.11121	MHD(8)	1.12890	FC′	1.67230
MHD(2)	1.04293	MRI(3)	1.07727	MRI′(5)	1.09795	BIT	1.11230	PRI(7)	1.13123	FC	1.67238
PRI(1)	1.04968	MRI′(3)	1.07730	TS	1.09898	MTF2	1.11252	PRI′(7)	1.13125	MF(8)	1.80266
MRI(1)	1.04968	MHD(4)	1.07842	MF′(8)	1.10194	MRI(8)	1.11628	RST	1.13464	MF(16)	2.12189
PRI′(1)	1.04971	MRI(4)	1.08891	MRI(6)	1.10528	MRI′(8)	1.11630	PRI(8)	1.13480	MTL	3.31364

Table 6: Costs ratios on request sequences extracted from file `book1` of the Calgary Corpus.

Sequence geo_1		Average access costs of FC′:									37.89396
FC′	1.00000	MHD(32)	1.05588	MRI(1)	1.13569	MRI(7)	1.14853	PRI′(3)	1.22252	PRI(8)	1.23618
FC	1.00125	MF′(2)	1.08716	MRI(4)	1.13628	MF′(128)	1.15094	PRI(4)	1.22664	PRI′(7)	1.23675
MHD(8)	1.02020	TRANS	1.08753	MRI′(2)	1.13667	MRI′(7)	1.15102	PRI′(4)	1.22876	PRI′(8)	1.23825
MF′(8)	1.02224	MF(4)	1.08938	MRI′(3)	1.13680	MRI(8)	1.15250	MTF2	1.22987	SPL	1.24816
MF′(16)	1.02372	MF′(64)	1.09337	MRI′(4)	1.13906	MRI′(8)	1.15491	PRI(5)	1.23027	RST	1.25364
MHD(4)	1.02461	MF(2)	1.10582	MRI′(1)	1.13967	RMHD	1.16351	MF(16)	1.23186	MTF	1.25726
MHD(16)	1.02891	MHD(64)	1.10913	MF(8)	1.14037	MHD(128)	1.19197	PRI′(5)	1.23237	RMTF	1.27518
MRI(0)	1.03526	PRI(0)	1.13093	MRI(5)	1.14055	COMB	1.19623	BIT	1.23240	MF(32)	1.36614
MF′(4)	1.03747	MRI(2)	1.13310	PRI′(1)	1.14137	PRI(2)	1.20541	CTR	1.23244	MF(64)	1.52161
MHD(2)	1.04563	MRI(3)	1.13371	MRI′(5)	1.14322	PRI′(2)	1.20774	PRI(6)	1.23277	MF(128)	1.70587
MF′(32)	1.04783	PRI′(0)	1.13536	MRI(6)	1.14452	TS	1.21624	PRI(7)	1.23467	MTL	5.43809
MRI′(0)	1.05500	PRI(1)	1.13569	MRI′(6)	1.14707	PRI(3)	1.22035	PRI′(6)	1.23485		

Sequence geo_2		Average access costs of FC′:									4.035117
FC′	1.00000	PRI′(1)	1.08736	PRI(2)	1.13366	MRI(6)	1.17354	MHD(8)	1.20356	PRI(8)	1.23444
FC	1.00012	MRI′(1)	1.08739	MRI′(4)	1.14665	MRI′(7)	1.18419	PRI′(5)	1.20638	SPL	1.23580
MRI′(0)	1.02396	PRI(1)	1.08744	MRI(4)	1.14670	MRI(7)	1.18424	PRI(5)	1.20644	RMTF	1.24522
MRI(0)	1.02401	MRI(1)	1.08744	MF′(8)	1.14699	TS	1.18489	MTF2	1.20811	RST	1.24857
TRANS	1.03402	MRI′(2)	1.10975	RMHD	1.15030	COMB	1.18680	BIT	1.20973	MF(2)	1.25793
PRI′(0)	1.04047	MRI(2)	1.10981	MRI′(5)	1.16120	PRI′(4)	1.18983	PRI′(6)	1.21805	MTF	1.28008
PRI(0)	1.04049	MHD(4)	1.12475	MRI(5)	1.16125	PRI(4)	1.18989	PRI(6)	1.21811	MF(4)	1.43517
MF′(2)	1.06572	MRI′(3)	1.12911	PRI′(3)	1.16654	MRI′(8)	1.19307	PRI′(7)	1.22723	MF(8)	1.68729
MHD(2)	1.06968	MRI(3)	1.12917	PRI(3)	1.16660	MRI(8)	1.19312	PRI(7)	1.22729	MF(16)	1.81082
MF′(4)	1.07639	PRI′(2)	1.13360	MRI′(6)	1.17348	CTR	1.20212	PRI′(8)	1.23438	MTL	2.93022

Sequence geo_3		Average access costs of MF′(4):									32.67832
MF′(4)	1.00000	MHD(4)	1.06087	MRI′(8)	1.07042	PRI′(2)	1.11119	PRI(6)	1.12348	RMTF	1.15775
MF′(8)	1.00525	MRI(5)	1.06861	MRI(2)	1.07313	PRI(2)	1.11128	PRI′(7)	1.12423	FC′	1.16500
MHD(16)	1.00538	MRI′(5)	1.06880	MRI′(2)	1.07357	PRI′(3)	1.11769	PRI(7)	1.12435	FC	1.16817
MRI(0)	1.01160	MRI(6)	1.06886	PRI′(1)	1.07699	PRI(3)	1.11781	PRI′(8)	1.12492	MF′(128)	1.18917
MHD(32)	1.01541	MRI(4)	1.06896	PRI(1)	1.07731	TS	1.11981	PRI(8)	1.12505	MF′(64)	1.20967
MHD(8)	1.01872	MRI′(6)	1.06902	MRI(1)	1.07731	PRI′(4)	1.12063	BIT	1.12827	MF(16)	1.22030
MRI′(0)	1.02223	MRI′(4)	1.06916	MRI′(1)	1.07787	PRI(4)	1.12076	CTR	1.12948	TRANS	1.26569
MF′(2)	1.03011	MRI(7)	1.06949	PRI(0)	1.07792	COMB	1.12134	MTF2	1.12995	MF(32)	1.45763
MF′(16)	1.03920	MRI′(7)	1.06965	PRI′(0)	1.07870	MF′(32)	1.12214	SPL	1.13045	MF(64)	1.85043
MF(2)	1.03921	MRI(3)	1.07002	RMHD	1.09249	PRI′(5)	1.12232	MTF	1.13144	MF(128)	2.22865
MF(4)	1.04079	MRI(8)	1.07028	MHD(128)	1.09540	PRI(5)	1.12244	RST	1.13529	MTL	6.58295
MHD(64)	1.04583	MRI′(3)	1.07029	MF(8)	1.10199	PRI′(6)	1.12335	MHD(2)	1.14430		

Sequence geo_4		Average access costs of MF′(4):									49.673371
MF′(4)	1.00000	MHD(4)	1.05714	MRI(7)	1.08145	MF′(32)	1.11639	PRI(5)	1.13785	RST	1.14745
MF′(8)	1.00130	MRI(4)	1.07886	MRI′(7)	1.08156	PRI′(2)	1.12493	CTR	1.13885	MTF	1.14753
MHD(16)	1.00572	MRI(3)	1.07889	PRI(0)	1.08242	PRI(2)	1.12496	PRI′(6)	1.13902	RMTF	1.16656
MRI(0)	1.00589	MRI′(4)	1.07902	MRI(8)	1.08306	COMB	1.12776	PRI(6)	1.13909	MF(16)	1.17879
MHD(8)	1.01398	MRI′(3)	1.07913	MRI′(8)	1.08314	TS	1.12871	FC′	1.13964	MF′(128)	1.19836
MRI′(0)	1.01532	MF(8)	1.07936	PRI′(1)	1.08315	PRI′(3)	1.13252	PRI′(7)	1.13990	MF′(64)	1.20859
MHD(32)	1.02095	MRI(5)	1.07947	PRI′(0)	1.08321	PRI(3)	1.13258	PRI(7)	1.13997	TRANS	1.26217
MF(4)	1.03303	MRI′(5)	1.07961	PRI(1)	1.08331	PRI′(4)	1.13589	MHD(2)	1.13998	MF(32)	1.37780
MF′(2)	1.03494	MRI(6)	1.08031	MRI(1)	1.08331	PRI(4)	1.13596	PRI′(8)	1.14062	MF(64)	1.70690
MF′(16)	1.03675	MRI′(6)	1.08042	MRI′(1)	1.08389	MTF2	1.13695	PRI(8)	1.14069	MF(128)	2.01060
MF(2)	1.04265	MRI(2)	1.08061	RMHD	1.09915	PRI′(5)	1.13778	SPL	1.14071	MTL	3.99686
MHD(64)	1.05565	MRI′(2)	1.08104	MHD(128)	1.10887	BIT	1.13785	FC	1.14239		

Sequence geo_5		Average access costs of TRANS:									4.906549
TRANS	1.00000	PRI(1)	1.01894	PRI(2)	1.02301	MRI(8)	1.02893	PRI(7)	1.03685	SPL	1.04520
MRI′(0)	1.00490	MRI(1)	1.01894	PRI′(2)	1.02305	MRI′(8)	1.02897	PRI′(7)	1.03690	RST	1.04690
MRI(0)	1.00499	MRI′(1)	1.01894	MRI(5)	1.02487	PRI(4)	1.03136	TS	1.03695	MTF	1.04865
MF′(4)	1.00577	PRI′(1)	1.01902	MRI′(5)	1.02491	PRI′(4)	1.03140	PRI(8)	1.03769	RMTF	1.05172
MHD(2)	1.00756	MRI(2)	1.01941	MRI(6)	1.02610	RMHD	1.03339	PRI′(8)	1.03774	FC	1.10094
MF′(2)	1.00811	MRI′(2)	1.01946	MRI′(6)	1.02615	PRI(5)	1.03358	COMB	1.03952	FC′	1.10107
PRI′(0)	1.01493	MRI(3)	1.02104	MRI(7)	1.02746	PRI′(5)	1.03363	CTR	1.04122	MF(4)	1.16902
PRI(0)	1.01496	MRI′(3)	1.02108	MRI′(7)	1.02751	MHD(8)	1.03388	BIT	1.04139	MF(8)	1.43901
MHD(4)	1.01739	MRI(4)	1.02286	PRI(3)	1.02784	PRI(6)	1.03531	MTF2	1.04319	MF(16)	1.64869
MF′(8)	1.01749	MRI′(4)	1.02291	PRI′(3)	1.02789	PRI′(6)	1.03535	MF(2)	1.04348	MTL	2.41602

Table 7: Costs ratios on request sequences extracted from file geo of the Calgary Corpus.

character list					
pos.	codeword	length	pos.	codeword	length
1	1:00	3	85	000:0000000	10
	...			...	
4	1:11	3	167	000:1010010	10
5	01:0000	6	168	000:10100110	11
	...			...	
20	01:1111	6	256	000:11111110	11
21	001:000000	9	EOF	000:11111111	11
	...				
84	001:111111	9			

Table 8: The $(2, 2, 8)$-start-step-stop code (optimized by the phasing technique) used in the byte-level compression experiment.

space character list					
pos.	codeword	length	pos.	codeword	length
1	00	2	5	101	3
2	010	3	6	110	3
3	011	3	7	111	3
4	100	3			

non-space character list					
pos.	codeword	length	pos.	codeword	length
1	1:000	4	57	000:0000000	10
	...			...	
8	1:111	4	117	000:0111100	10
9	01:0000	6	118	000:01111010	11
	...			...	
24	01:1111	6	251	000:11111111	11
25	001:00000	8			
	...				
56	001:11111	8			

Table 9: The codes used to transmit list positions in the character lists of the word-level compression experiment.

Bentley *et al.* scheme on byte-level															
algorithm	bib	book1	book2	geo	news	obj1	obj2	paper1	paper2	pic	progc	progl	progp	trans	average
MF′(4)	6.016	5.289	5.377	5.954	5.689	6.049	6.208	5.524	5.327	3.370	5.722	5.251	5.465	5.956	5.5141
MF′(2)	6.084	5.394	5.472	5.983	5.768	5.969	6.220	5.606	5.442	3.378	5.803	5.314	5.520	5.953	5.5647
MRI(0)	6.096	5.364	5.449	5.995	5.786	6.008	6.209	5.604	5.417	3.382	5.847	5.335	5.522	5.951	5.5689
MHD(2)	5.990	5.293	5.403	5.992	5.721	6.511	6.373	5.536	5.330	3.390	5.774	5.266	5.502	5.994	5.5768
MRI′(0)	6.100	5.365	5.450	6.006	5.787	6.071	6.215	5.613	5.421	3.385	5.858	5.343	5.533	5.955	5.5787
MF(2)	6.093	5.390	5.468	6.036	5.761	6.076	6.363	5.608	5.435	3.408	5.805	5.310	5.525	5.974	5.5894
TRANS	5.958	5.251	5.371	6.056	5.709	6.788	6.426	5.509	5.278	3.407	5.766	5.264	5.504	6.046	5.5952
FC′	5.902	5.171	5.355	6.003	5.734	6.404	6.601	5.561	5.211	3.381	5.804	5.411	5.514	6.302	5.5967
FC	5.904	5.171	5.355	6.006	5.734	6.415	6.604	5.562	5.211	3.381	5.805	5.411	5.514	6.301	5.5981
MHD(4)	6.046	5.380	5.482	5.968	5.767	6.334	6.388	5.608	5.432	3.386	5.806	5.330	5.551	5.997	5.6054
MF′(8)	6.056	5.406	5.514	5.959	5.803	6.248	6.454	5.630	5.456	3.385	5.855	5.360	5.607	6.072	5.6289
MF(4)	6.028	5.247	5.348	6.974	5.662	6.461	6.751	5.514	5.279	3.487	5.707	5.251	5.483	6.090	5.6630
PRI(0)	6.277	5.471	5.574	6.106	5.893	6.021	6.288	5.733	5.549	3.396	5.957	5.437	5.630	6.041	5.6695
PRI′(0)	6.280	5.471	5.575	6.111	5.894	6.075	6.290	5.739	5.552	3.398	5.965	5.442	5.635	6.043	5.6764
MHD(8)	6.160	5.534	5.632	5.971	5.870	6.225	6.457	5.734	5.604	3.393	5.914	5.431	5.651	6.029	5.6861
PRI(1)	6.326	5.512	5.635	6.113	5.927	6.058	6.458	5.785	5.601	3.400	6.024	5.487	5.707	6.135	5.7263
MRI(1)	6.326	5.512	5.635	6.113	5.927	6.058	6.458	5.785	5.601	3.400	6.024	5.487	5.707	6.135	5.7263
PRI′(1)	6.328	5.512	5.635	6.117	5.927	6.088	6.457	5.790	5.603	3.401	6.028	5.489	5.711	6.135	5.7301
MRI′(1)	6.329	5.512	5.635	6.115	5.927	6.095	6.459	5.788	5.603	3.401	6.029	5.491	5.710	6.136	5.7307
MRI(2)	6.319	5.545	5.669	6.089	5.947	6.106	6.521	5.803	5.635	3.400	6.041	5.499	5.724	6.130	5.7449
MRI′(2)	6.321	5.546	5.670	6.091	5.947	6.126	6.523	5.806	5.636	3.401	6.044	5.503	5.727	6.131	5.7480
MRI(3)	6.344	5.598	5.715	6.091	5.982	6.112	6.548	5.844	5.692	3.402	6.058	5.518	5.763	6.159	5.7733
MRI′(3)	6.346	5.599	5.716	6.094	5.982	6.131	6.549	5.847	5.694	3.403	6.061	5.521	5.766	6.160	5.7764
MF′(16)	6.167	5.630	5.721	5.996	5.981	6.479	6.596	5.825	5.683	3.407	6.015	5.499	5.777	6.156	5.7809
MRI(4)	6.378	5.645	5.754	6.098	6.016	6.132	6.567	5.883	5.740	3.405	6.090	5.538	5.792	6.174	5.8009
MRI′(4)	6.380	5.645	5.755	6.100	6.017	6.149	6.568	5.886	5.742	3.405	6.092	5.541	5.794	6.175	5.8035
MHD(16)	6.321	5.749	5.813	5.993	6.024	6.195	6.516	5.923	5.825	3.411	6.069	5.581	5.798	6.109	5.8091
MRI(5)	6.412	5.684	5.785	6.106	6.047	6.152	6.585	5.918	5.780	3.407	6.117	5.554	5.810	6.186	5.8245
MRI′(5)	6.413	5.684	5.785	6.108	6.048	6.167	6.586	5.921	5.782	3.408	6.119	5.557	5.813	6.186	5.8269
PRI(2)	6.427	5.667	5.766	6.167	6.056	6.179	6.591	5.921	5.758	3.410	6.122	5.564	5.815	6.169	5.8294
PRI′(2)	6.428	5.667	5.766	6.168	6.057	6.197	6.592	5.924	5.760	3.411	6.123	5.566	5.819	6.170	5.8320
MRI(6)	6.442	5.715	5.809	6.113	6.070	6.173	6.599	5.947	5.811	3.409	6.135	5.569	5.830	6.199	5.8444
MRI′(6)	6.444	5.716	5.809	6.114	6.071	6.187	6.600	5.949	5.812	3.410	6.136	5.571	5.833	6.200	5.8466
MRI(7)	6.465	5.741	5.828	6.118	6.090	6.181	6.611	5.968	5.834	3.410	6.150	5.584	5.852	6.205	5.8598
MRI′(7)	6.466	5.742	5.829	6.119	6.090	6.197	6.611	5.970	5.836	3.411	6.152	5.585	5.854	6.206	5.8620
MRI(8)	6.483	5.764	5.843	6.122	6.108	6.190	6.621	5.987	5.853	3.412	6.161	5.597	5.865	6.212	5.8727
MRI′(8)	6.484	5.764	5.844	6.123	6.108	6.206	6.622	5.989	5.854	3.413	6.162	5.598	5.868	6.213	5.8749
PRI(3)	6.505	5.770	5.848	6.199	6.141	6.228	6.658	6.006	5.854	3.418	6.186	5.618	5.888	6.226	5.8961
PRI′(3)	6.507	5.771	5.848	6.200	6.141	6.246	6.659	6.008	5.856	3.419	6.188	5.619	5.891	6.227	5.8986
MF′(32)	6.481	5.901	5.914	6.044	6.148	6.575	6.664	6.032	5.951	3.432	6.138	5.669	5.902	6.167	5.9299
PRI(4)	6.553	5.827	5.891	6.214	6.188	6.269	6.695	6.049	5.903	3.423	6.225	5.645	5.924	6.248	5.9324
MHD(32)	6.513	5.933	5.958	6.038	6.188	6.212	6.606	6.088	5.988	3.430	6.234	5.703	5.973	6.214	5.9341
PRI′(4)	6.555	5.827	5.891	6.215	6.188	6.283	6.695	6.051	5.905	3.424	6.227	5.646	5.927	6.248	5.9344
PRI(5)	6.584	5.860	5.917	6.224	6.214	6.292	6.718	6.076	5.933	3.426	6.247	5.662	5.948	6.261	5.9544
PRI′(5)	6.585	5.860	5.917	6.225	6.215	6.306	6.718	6.078	5.935	3.427	6.249	5.662	5.951	6.261	5.9564
PRI(6)	6.603	5.882	5.933	6.229	6.232	6.306	6.734	6.094	5.952	3.428	6.265	5.673	5.967	6.271	5.9692
PRI′(6)	6.604	5.882	5.933	6.231	6.232	6.319	6.734	6.096	5.953	3.428	6.267	5.673	5.970	6.271	5.9709
PRI(7)	6.619	5.897	5.944	6.233	6.245	6.316	6.745	6.106	5.964	3.429	6.277	5.683	5.978	6.276	5.9794
MTF2	6.660	5.835	5.897	6.435	6.205	6.382	6.759	6.079	5.891	3.439	6.285	5.679	5.915	6.272	5.9809
PRI′(7)	6.620	5.897	5.944	6.234	6.245	6.329	6.745	6.108	5.966	3.430	6.278	5.684	5.981	6.276	5.9812
PRI(8)	6.632	5.908	5.952	6.235	6.254	6.324	6.755	6.116	5.973	3.430	6.286	5.693	5.985	6.281	5.9874
PRI′(8)	6.633	5.908	5.952	6.236	6.254	6.337	6.755	6.118	5.974	3.431	6.287	5.693	5.987	6.282	5.9891
MHD(64)	6.713	5.982	6.007	6.102	6.297	6.252	6.701	6.164	6.030	3.442	6.333	5.732	6.036	6.300	6.0065
MF(8)	6.026	5.230	5.342	7.438	5.706	7.337	7.317	5.520	5.240	6.149	5.747	5.428	5.558	6.230	6.0191
MHD(128)	6.717	5.982	6.011	6.184	6.311	6.326	6.780	6.170	6.034	3.445	6.349	5.733	6.038	6.311	6.0279
MF′(64)	6.703	5.979	6.000	6.123	6.287	6.588	6.757	6.148	6.018	3.450	6.309	5.729	6.021	6.302	6.0296
MTF	6.717	5.982	6.011	6.247	6.311	6.409	6.810	6.170	6.034	3.445	6.349	5.733	6.038	6.311	6.0405
MF′(128)	6.717	5.982	6.011	6.175	6.311	6.559	6.825	6.170	6.034	3.452	6.349	5.733	6.038	6.311	6.0476
MF(16)	6.106	5.318	5.415	7.598	5.794	7.716	7.813	5.571	5.309	6.262	5.878	5.580	5.691	6.459	6.1793
MF(32)	6.271	5.346	5.451	9.102	5.879	8.893	8.806	5.631	5.339	9.004	5.993	6.038	5.825	6.893	6.7479
MF(64)	6.271	5.346	5.451	9.146	5.882	8.964	8.887	5.633	5.340	9.028	5.996	6.040	5.826	6.896	6.7647
MF(128)	6.277	5.354	5.460	9.606	5.906	9.432	9.325	5.641	5.344	9.997	6.059	6.076	5.842	7.011	6.9521
MTL	10.997	10.999	10.999	10.669	10.998	10.790	10.906	10.992	10.995	10.999	10.989	10.995	10.992	10.995	10.9511
RMHD	6.385	5.640	5.720	6.142	6.011	6.175	6.471	5.872	5.704	3.410	6.055	5.531	5.741	6.139	5.7859
COMB	6.555	5.735	5.862	6.358	6.168	6.303	6.741	5.980	5.818	3.432	6.243	5.662	5.878	6.261	5.9287
TS	6.627	5.785	5.851	6.370	6.160	6.357	6.709	6.024	5.846	3.434	6.237	5.643	5.882	6.253	5.9418
RST	6.656	5.813	5.874	6.447	6.190	6.385	6.746	6.050	5.866	3.436	6.257	5.656	5.890	6.265	5.9669
CTR	6.655	5.816	5.876	6.451	6.195	6.392	6.754	6.054	5.866	3.440	6.268	5.670	5.906	6.272	5.9730
BIT	6.659	5.836	5.894	6.443	6.203	6.389	6.758	6.069	5.891	3.440	6.277	5.675	5.914	6.271	5.9804
SPL	6.691	5.878	5.934	6.481	6.240	6.418	6.799	6.111	5.934	3.444	6.320	5.710	5.956	6.299	6.0159
RMTF	6.730	5.881	5.939	6.599	6.268	6.482	6.849	6.124	5.926	3.451	6.339	5.742	5.990	6.331	6.0469

Table 10: Results of the byte-level compression experiment.

Bentley *et al.* scheme on word-level															
Regel	bib	book1	book2	geo	news	obj1	obj2	paper1	paper2	pic	progc	progl	progp	trans	average
MF′(2)	4.104	3.730	3.390	4.127	4.401	6.064	5.486	4.265	3.925	4.189	4.349	3.172	3.637	3.887	4.1947
MRI(0)	4.012	3.697	3.415	4.121	4.420	6.122	5.499	4.277	3.925	4.189	4.357	3.194	3.638	3.915	4.1986
PRI(0)	4.103	3.767	3.426	4.199	4.465	6.094	5.531	4.309	3.980	4.197	4.379	3.173	3.670	3.907	4.2286
MF′(4)	4.100	3.714	3.424	4.104	4.424	6.172	5.510	4.306	3.939	4.182	4.422	3.257	3.689	3.959	4.2287
FC′	3.917	3.634	3.427	4.091	4.399	6.420	5.845	4.254	3.845	4.193	4.408	3.272	3.717	4.058	4.2486
MF(2)	4.018	3.730	3.625	4.136	4.396	6.114	5.556	4.279	3.937	4.203	4.599	3.376	3.663	3.908	4.2529
PRI(1)	4.183	3.775	3.439	4.201	4.478	6.120	5.604	4.322	3.990	4.199	4.418	3.200	3.724	3.940	4.2566
MRI(1)	4.183	3.775	3.439	4.201	4.478	6.120	5.604	4.322	3.990	4.199	4.418	3.200	3.724	3.940	4.2566
MRI(2)	4.212	3.778	3.441	4.196	4.476	6.130	5.619	4.316	3.986	4.199	4.416	3.210	3.735	3.944	4.2613
FC	3.924	3.638	3.439	4.093	4.415	6.446	5.857	4.283	3.863	4.193	4.441	3.301	3.744	4.080	4.2655
MRI(3)	4.237	3.786	3.446	4.197	4.480	6.136	5.639	4.325	3.997	4.200	4.424	3.217	3.748	3.955	4.2705
MRI(4)	4.253	3.795	3.452	4.201	4.484	6.140	5.656	4.334	4.008	4.202	4.435	3.223	3.755	3.967	4.2789
MRI(5)	4.268	3.804	3.458	4.205	4.490	6.142	5.671	4.341	4.018	4.203	4.442	3.228	3.762	3.973	4.2861
MF′(8)	4.118	3.731	3.472	4.110	4.464	6.348	5.574	4.388	3.994	4.187	4.504	3.351	3.758	4.046	4.2889
MRI(6)	4.282	3.813	3.463	4.210	4.494	6.150	5.679	4.348	4.027	4.204	4.447	3.235	3.767	3.975	4.2924
PRI(2)	4.288	3.871	3.473	4.255	4.487	6.162	5.675	4.350	4.042	4.207	4.423	3.201	3.755	3.947	4.2954
MRI(7)	4.293	3.825	3.468	4.212	4.498	6.151	5.685	4.354	4.035	4.205	4.454	3.241	3.772	3.976	4.2978
MRI(8)	4.302	3.839	3.474	4.215	4.503	6.159	5.694	4.361	4.042	4.206	4.459	3.245	3.778	3.978	4.3039
PRI′(1)	4.354	3.962	3.518	4.213	4.519	6.136	5.589	4.389	4.109	4.200	4.480	3.182	3.738	3.938	4.3091
PRI(3)	4.338	3.903	3.497	4.276	4.510	6.200	5.731	4.376	4.074	4.213	4.449	3.214	3.774	3.974	4.3235
MF(4)	4.016	3.749	3.686	4.139	4.425	6.329	5.759	4.345	3.976	4.225	4.684	3.462	3.760	4.025	4.3271
MRI′(2)	4.376	3.954	3.543	4.207	4.566	6.149	5.617	4.418	4.123	4.200	4.520	3.234	3.772	3.975	4.3324
MRI′(3)	4.386	3.949	3.540	4.207	4.563	6.152	5.636	4.414	4.119	4.200	4.517	3.237	3.779	3.983	4.3344
PRI′(0)	4.361	3.990	3.569	4.216	4.586	6.142	5.533	4.453	4.165	4.198	4.542	3.229	3.752	3.958	4.3353
PRI′(2)	4.393	3.983	3.534	4.263	4.530	6.172	5.668	4.411	4.125	4.208	4.490	3.206	3.774	3.959	4.3369
MRI′(4)	4.390	3.948	3.540	4.210	4.561	6.154	5.652	4.413	4.121	4.202	4.519	3.242	3.786	3.992	4.3379
MRI′(1)	4.372	3.970	3.554	4.216	4.580	6.151	5.603	4.437	4.144	4.200	4.533	3.231	3.768	3.978	4.3384
PRI(4)	4.360	3.922	3.512	4.285	4.522	6.221	5.757	4.393	4.092	4.216	4.464	3.227	3.783	3.987	4.3386
MRI′(5)	4.395	3.949	3.542	4.214	4.563	6.155	5.667	4.416	4.125	4.204	4.520	3.243	3.788	3.995	4.3411
MRI′(6)	4.402	3.952	3.544	4.219	4.563	6.163	5.675	4.419	4.129	4.205	4.522	3.247	3.791	3.995	4.3447
MRI′(7)	4.407	3.959	3.547	4.220	4.564	6.165	5.680	4.422	4.133	4.206	4.526	3.253	3.796	3.995	4.3481
PRI(5)	4.373	3.935	3.520	4.290	4.533	6.237	5.772	4.402	4.104	4.218	4.476	3.233	3.789	3.993	4.3482
MRI′(8)	4.412	3.968	3.550	4.223	4.567	6.172	5.689	4.426	4.136	4.207	4.527	3.256	3.801	3.997	4.3522
PRI(6)	4.383	3.945	3.526	4.293	4.538	6.245	5.781	4.408	4.111	4.219	4.483	3.240	3.795	3.997	4.3546
PRI′(3)	4.426	4.000	3.547	4.283	4.548	6.209	5.724	4.427	4.146	4.213	4.508	3.216	3.792	3.984	4.3588
PRI(7)	4.390	3.955	3.531	4.295	4.543	6.253	5.787	4.414	4.117	4.220	4.489	3.245	3.800	4.000	4.3599
MHD(8)	4.223	3.847	3.614	4.115	4.548	6.367	5.645	4.478	4.107	4.194	4.594	3.446	3.794	4.097	4.3621
PRI(8)	4.395	3.968	3.536	4.297	4.547	6.256	5.791	4.419	4.121	4.220	4.495	3.249	3.803	4.002	4.3642
MHD(16)	4.276	3.868	3.612	4.124	4.563	6.273	5.682	4.484	4.129	4.202	4.597	3.432	3.808	4.103	4.3681
PRI′(4)	4.441	4.011	3.556	4.292	4.558	6.230	5.750	4.439	4.158	4.216	4.518	3.227	3.800	3.996	4.3709
MHD(4)	4.190	3.846	3.627	4.122	4.547	6.511	5.642	4.490	4.104	4.194	4.604	3.469	3.826	4.116	4.3777
PRI′(5)	4.451	4.018	3.561	4.296	4.567	6.244	5.765	4.445	4.167	4.218	4.526	3.233	3.805	4.001	4.3784
MHD(32)	4.325	3.892	3.611	4.149	4.585	6.223	5.737	4.499	4.141	4.216	4.597	3.413	3.835	4.086	4.3792
PRI′(6)	4.458	4.025	3.566	4.300	4.571	6.253	5.774	4.450	4.172	4.220	4.531	3.239	3.811	4.005	4.3839
MTF2	4.380	3.996	3.574	4.309	4.598	6.289	5.765	4.439	4.145	4.221	4.534	3.288	3.808	4.037	4.3845
PRI′(7)	4.464	4.033	3.569	4.302	4.575	6.259	5.779	4.454	4.176	4.220	4.537	3.244	3.815	4.008	4.3882
MHD(64)	4.375	3.906	3.602	4.194	4.602	6.218	5.785	4.489	4.142	4.225	4.600	3.387	3.842	4.071	4.3884
MRI′(0)	4.449	4.153	3.707	4.149	4.686	6.187	5.504	4.529	4.267	4.191	4.598	3.290	3.747	3.997	4.3896
MHD(128)	4.391	3.915	3.589	4.256	4.601	6.248	5.829	4.468	4.136	4.226	4.575	3.354	3.831	4.046	4.3904
MHD(512)	4.419	3.945	3.569	4.321	4.592	6.295	5.829	4.446	4.141	4.227	4.539	3.301	3.820	4.021	4.3904
PRI′(8)	4.468	4.044	3.573	4.304	4.578	6.263	5.784	4.458	4.179	4.221	4.542	3.247	3.818	4.010	4.3921
MHD(256)	4.406	3.928	3.577	4.319	4.596	6.301	5.842	4.452	4.134	4.227	4.550	3.321	3.821	4.028	4.3930
MHD(1024)	4.437	3.965	3.566	4.321	4.588	6.295	5.819	4.455	4.155	4.227	4.555	3.287	3.826	4.020	4.3940
MF′(16)	4.244	3.803	3.556	4.133	4.545	6.573	5.755	4.498	4.115	4.201	4.623	3.472	3.886	4.146	4.3964
MHD(2)	4.177	3.858	3.644	4.156	4.559	6.725	5.689	4.512	4.119	4.206	4.634	3.491	3.909	4.151	4.4164
MTF	4.502	4.086	3.606	4.321	4.612	6.295	5.813	4.486	4.213	4.227	4.579	3.290	3.837	4.029	4.4211
MF′(1024)	4.441	4.006	3.706	4.321	4.659	6.295	5.866	4.514	4.199	4.227	4.601	3.392	3.849	4.071	4.4391
MF′(512)	4.428	4.003	3.732	4.321	4.662	6.296	5.875	4.569	4.215	4.227	4.705	3.436	3.848	4.127	4.4603
TRANS	4.157	3.885	3.661	4.229	4.577	7.040	5.769	4.538	4.148	4.232	4.692	3.510	4.029	4.203	4.4764
MF′(256)	4.426	4.012	3.743	4.318	4.671	6.334	5.884	4.587	4.239	4.227	4.705	3.526	3.875	4.176	4.4802
MF′(32)	4.332	3.894	3.634	4.191	4.619	6.762	5.863	4.589	4.217	4.226	4.709	3.553	3.979	4.199	4.4834
MF′(128)	4.444	4.009	3.737	4.263	4.687	6.594	5.928	4.623	4.255	4.241	4.739	3.558	4.007	4.226	4.5222
MF(8)	4.059	3.839	3.799	4.609	4.498	6.785	6.258	4.478	4.086	4.322	4.824	3.601	3.918	4.291	4.5262
MF′(64)	4.420	3.966	3.695	4.250	4.667	6.786	5.926	4.618	4.250	4.246	4.743	3.579	4.038	4.230	4.5296
MF(16)	4.112	3.915	3.883	4.854	4.575	7.488	6.625	4.564	4.175	6.132	4.951	3.712	4.058	4.500	4.8246
MF(32)	4.190	3.999	3.979	5.578	4.675	8.469	7.441	4.662	4.271	8.023	5.065	3.844	4.214	4.882	5.2351
MF(64)	4.252	4.066	4.050	6.173	4.740	9.393	8.266	4.705	4.335	9.966	5.128	3.884	4.344	5.247	5.6106
MF(128)	4.273	4.165	4.097	6.279	4.779	9.522	8.457	4.732	4.360	10.046	5.165	3.907	4.392	5.260	5.6739
MF(256)	4.274	4.191	4.113	6.279	4.785	9.522	8.457	4.734	4.364	10.046	5.168	3.909	4.393	5.264	5.6785
MF(512)	4.275	4.202	4.121	6.279	4.786	9.522	8.457	4.734	4.366	10.046	5.169	3.911	4.393	5.265	5.6804
MF(1024)	4.275	4.204	4.123	6.279	4.787	9.522	8.457	4.735	4.366	10.046	5.170	3.911	4.393	5.265	5.6809
MTL	7.355	7.379	6.731	6.965	8.969	10.741	9.782	7.708	7.086	10.968	8.492	6.444	7.238	7.713	8.1122
RMHD	4.220	3.831	3.474	4.219	4.499	6.181	5.624	4.358	4.031	4.207	4.447	3.239	3.721	3.974	4.2875
COMB	4.385	3.989	3.568	4.291	4.594	6.261	5.731	4.443	4.146	4.217	4.541	3.269	3.796	4.020	4.3751
TS	4.402	4.010	3.583	4.301	4.608	6.267	5.733	4.455	4.161	4.217	4.553	3.279	3.802	4.028	4.3856
RST	4.369	3.991	3.579	4.314	4.603	6.299	5.762	4.448	4.142	4.219	4.541	3.293	3.801	4.041	4.3859
BIT	4.436	4.042	3.600	4.314	4.623	6.296	5.763	4.473	4.185	4.222	4.572	3.287	3.814	4.040	4.4048
CTR	4.430	4.040	3.607	4.318	4.634	6.310	5.764	4.481	4.187	4.220	4.583	3.297	3.820	4.050	4.4101
SPL	4.459	4.058	3.608	4.323	4.633	6.316	5.789	4.484	4.197	4.226	4.585	3.302	3.831	4.056	4.4191
RMTF	4.487	4.095	3.653	4.365	4.686	6.396	5.844	4.534	4.234	4.229	4.647	3.341	3.865	4.095	4.4622

Table 11: Results of the word-level compression experiment.