Two New Families of List Update Algorithms

Frank Schulz*

Universität des Saarlandes, FB 14, Lehrstuhl Prof. Dr. G. Hotz,
Postfach 151150, 66041 Saarbrücken, Germany.
Email: schulz@cs.uni-sb.de

Abstract. We consider the online list accessing problem and present a new family of competitive-optimal deterministic list update algorithms which is the largest class of such algorithms known to-date. This family, called Sort-by-Rank (sbr), is parametrized with a real $0 \leq \alpha \leq 1$, where sbr(0) is the Move-to-Front algorithm and sbr(1) is equivalent to the Timestamp algorithm. The behaviour of sbr(α) mediates between the eager strategy of Move-to-Front and the more conservative behaviour of Timestamp.
We also present a family of algorithms Sort-by-Delay (sbd) which is parametrized by the positive integers, where sbd(1) is Move-to-Front and sbd(2) is equivalent to Timestamp. In general, sbd(k) is k-competitive for $k \geq 2$. This is the first class of algorithms that is asymptotically optimal for independent, identically distributed requests while each algorithm is constant-competitive.
Empirical studies with with both generated and real-world data are also included.

1 Introduction

Consider a set $\{1, \ldots, n\}$ of n items that are to be stored as an unsorted linear list. A request for an item x is processed by searching the list for x starting from the head of the list. The cost of the retrieval is assumed to be the number of comparisons necessary to identify the item, i.e. the number of the position at which the item was found. After the retrieval, a list update algorithm may move the requested element to any position closer to the front without additional costs (*free exchanges*). Any other swap of two consecutive items incurs cost 1 (*paid exchanges*). The goal of such exchanges is to minimize the cost for future accesses. When the list update algorithm processes the requests without knowing future queries, it is said to work *online*.
In the sequel we will only consider requests to items and no insertions or deletions (static model). It is possible to extend the analyses to the dynamic model where the set of list items may change over time, the results carry over to this situation. List update algorithms are interesting because they are frequently used in practice. The most direct application is the implementation of a dictionary as a

* supported by a Graduiertenkolleg fellowship from the Deutsche Forschungsgemeinschaft (DFG), Germany.

singly-linked list. As online algorithms constantly try to adapt to the request sequence, they can approximate the process that generated the requests. Moreover, this is achieved in a computationally inexpensive way. As such, list update schemes have been successfully applied in data compression algorithms [1, 5] Surprisingly, online algorithms also gain importance in geometric applications such as computing point maxima or convex hulls [9].

In competitive analysis, an online algorithm ALG is compared to an optimal offline algorithm OPT. Let σ be any sequence of requests to items in the list (we assume that all queries are successful, i.e. every requested item is actually in the list). Denote by $\text{ALG}(\sigma)$ the total cost incurred by ALG to serve σ, and $\text{OPT}(\sigma)$ the total cost incurred by an optimal offline algorithm. Then ALG is said to be c-competitive (having a competitive ratio of c) if there exists a constant a such that $\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + a$ for all request sequences σ.
In average case analysis, the request sequence is generated according to some probability model. In the independent model, a probability distributions over the list items is given. The requests are the realization of a sequence of independent, identically distributed random variables with this distribution (the requests are generated by some discrete memoryless source). In the markov model, the requests are generated by a (first order) Markov chain. In those models, the asymptotic expected costs of online algorithms and of optimal offline algorithms and their ratios are of main interest.

1.1 On previous Work

The most famous list update algorithm is MOVE-TO-FRONT, it has been studied since 1965 [12].

> MOVE-TO-FRONT (MTF): Upon a request for item x, move x to the front of the list.

In their seminal work on competitive analysis [14], Sleator and Tarjan showed that MTF ist 2-competitive. Raghavan and Karp proved a lower bound of $2 - 2/(n+1)$ on the competitive ratio of any deterministic list update scheme and list size n. Hence, MTF is a competitive-optimal algorithm[1].

The important TIMESTAMP algorithm discovered recently by Albers [2] is 2-competitive as well.

> TIMESTAMP (TS): Upon a request for item x, insert x in front of the first item y that precedes x and that 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.

Both algorithms have optimal competitive ratio, nevertheless they are rather different. This difference is blurred by competitive analysis, but it is revealed

[1] All algorithms with competitive ratio 2 will be called optimal. As MTF does indeed have a competitive ratio of $2 - 2/(n+1)$, it is sometimes called *strongly-optimal*.

when average case analysis is employed. In the independent model, the ratio between the expected cost of the MTF algorithm and a static optimal algorithm can be arbtrarily close to $\pi/2 = 1.5707\ldots$ [10]. For TS, this ratio is less than 1.34 on any access distribution [1]. On the other hand, for Markov sources with the phenomenon of locality, MTF fares better. This demonstrates that TIMESTAMP reacts more conservatively, and MTF more eagerly.

Recently, a new family of list update algorithms has been introduced by El-Yaniv [8]. The MOVE-TO-RECENT-ITEM(k) algorithms are located "between" the MTF and TS strategies.

> MOVE-TO-RECENT-ITEM(k), MRI(k): Upon a request for item x, move x forward just after the last item y that is in front of x and that was requested at least $k+1$ times since the last request for x. If there is no such item y of if this is the first request for x, move x to the front.

It has been shown that MRI(1) is equivalent to the TIMESTAMP algorithm, and that MOVE-TO-FRONT is the limit element of that family, as $k \to \infty$. Moreover, for all $k \geq 1$ MRI(k) is 2-competitive. Hence there is an infinite (countable) class of competitive-optimal list update algorithms.

1.2 New Results

We define a new infinite (uncountable) family of list update schemes that turn out to be competitive-optimal. Let $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_m$ be any request sequence. Consider any point t, $1 \leq t \leq m$, in time, and define for each list item x the last access time $w_1(x,t)$,

$$w_1(x,t) = \max(\{t' \leq t : \sigma_{t'} = x\} \cup \{0\}) \; .$$

For $k > 1$ define the kth last access time $w_k(x,t)$ to x at time t,

$$w_k(x,t) = \max(\{t' < w_{k-1}(x,t) : \sigma_{t'} = x\} \cup \{0\}) \; .$$

Now consider for $k \geq 1$ the rank functions

$$s_k(x,t) = t - w_k(x,t) \; .$$

Lemma 1. *(a) For all t, such that each list item has been requested at least once, we have: When the list is maintained by* MOVE-TO-FRONT*, the current list ordering is* $[x_1, \ldots, x_n]$ *iff*

$$s_1(x_1,t) < s_1(x_2,t) < \cdots < s_1(x_n,t).$$

(b) For all t, such that each list item has been requested at least twice, we have: When the list is maintained by TIMESTAMP*, the current list ordering is* $[x_1, \ldots, x_n]$ *iff*

$$s_2(x_1,t) < s_2(x_2,t) < \cdots < s_2(x_n,t).$$

Due to lack of space, the proof is omitted. Note that this approach gives a very natural interpretation of the TIMESTAMP rule.

For $\alpha \geq 0$, define a new rank function as the convex combination of $s_1(x,t)$ and $s_2(x,t)$:

$$r_t(x) = r_t(x,\alpha) \;=\; (1-\alpha)\cdot s_1(x,t) + \alpha \cdot s_2(x,t).$$

Then we define the new family of list update algorithms.

> SORT-BY-RANK(α), SBR(α): Upon a request for item x at time t, insert x just after the last item y in front of x with $r_t(y) < r_t(x)$. Insert x in front of the list if there is no such item y, or when x is requested for the first time.

Note that SBR(0) = MTF and SBR(1) is equivalent to TS modulo the handling of first requests. This means that SBR(1) and TS acted similarly if TS would be changed such that it moves an item to the front of the list upon its first request.

Theorem 1. *For $0 \leq \alpha \leq 1$, the algorithm* SBR(α) *is 2-competitive.*

This result provides the first example of an uncountable class of competitive-optimal list update algorithms. The SBR(α) schemes use $2n$ timestamps for all α. In contrast, the MRI(k) rules need nk timestamps which is significant when considering rules "close" to MTF. We remark that in general, the SBR(α) algorithms do not share the *pairwise independence property* that is satisfied by MTF and TS. Hence the proof is based on list factoring and a potential function argument similar to the one used in [8]. Note that for $0 \leq \alpha \leq 1$, the SBD(α) schemes interpolate between MTF and TS. For $\alpha > 1$, SBD(α) "extrapolates", and in the limit $\alpha \to \infty$, the list is sorted by increasing time *differences* $s_2(x,t) - s_1(x,t)$ between the second last and the last access to x. However, in these cases, the algorithms are no longer 2-competitive, as can be easily shown by example.

The second class of list update schemes that we examine is

> SORT-BY-DELAY(k), SBD(k): Upon a request for item x at time t, insert x just after the last item y in front of x with $s_k(y,t) < s_k(x,t)$. Insert x in front of the list if there is no such item y, or when x is requested for the first time.

Here we have SBD(1) = MTF and SBD(2) is equivalent to TS in the above sense. Thus, SBD(k) turns out to be a natural generalization of MTF and TS.

Theorem 2. *For every $k \geq 2$,* SBD(k) *is k-competitive.*

Lemma 2. *An equivalent definition of* SBD(k) *is the following:*

> *Upon a request for item x, insert x in front of the first item y that precedes x and that was requested at most $k-1$ times since the $(k-1)$th last access to x. Insert x in front of the list if there is no such item y, or if x has been requested for the first time.*

It turns out that the SBD(k) rules do satisfy the pairwise independence property. This makes it easy to achieve an average case analysis.

Theorem 3. *For any probability distribution $p = (p_1, \ldots, p_n)$, the asymptotic expected costs per request for* SBD*(k) are*

$$E_{\mathrm{SBD}(k)}(p) = \sum_{1 \le i \le j \le n} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_i^{\ell+1} p_j^{2k-1-\ell} + p_i^{2k-1-\ell} p_j^{\ell+1}}{(p_i + p_j)^{2k-1}} .$$

For $k = 1$, this has been found in [12], and for $k = 2$ see [1]. We compare this expected cost with the optimal cost $M(p)$ incurred by an optimal static adversary. For $p_1 \ge p_2 \ge \cdots \ge p_n$, we have $M(p) = \sum_{i=1}^{n} i p_i$.

Theorem 4. *The family* SBD*(k) is asymptotically optimal, i.e. for all distributions p we have*

$$E_{\mathrm{SBD}(k)}(p) \longrightarrow M(p) \qquad \text{for } k \to \infty.$$

In particular, the cost ratios are bounded as follows

$$\begin{aligned} E_{\mathrm{SBD}(1)}(p)/M(p) &\le 1.5708 \\ E_{\mathrm{SBD}(2)}(p)/M(p) &\le 1.3391 \\ E_{\mathrm{SBD}(3)}(p)/M(p) &\le 1.3044 \\ E_{\mathrm{SBD}(4)}(p)/M(p) &\le 1.2904 \\ E_{\mathrm{SBD}(5)}(p)/M(p) &\le 1.2828. \end{aligned}$$

For $k = 1$ (MTF), the ratio is bounded by $\pi/2 \le 1.5708$, as proved in [7], and for $k = 2$ (TS), the ratio has been bounded in [1].

While the SBD(k) algorithms are not competitive-optimal, they are constant-competitive and, under the independent model, achieve an expected search time that is arbirtrarily close to the optimal expected search time of an static adversary. To the best of our knowledge, all previous list update algorithms with this property, e.g. FREQUENCY-COUNT, SIMPLE(k)-MTF or BATCHED(k)-MTF have a competitive ratio that is linear in the length n of the list, for all $k \ge 2$.

The constant-competitiveness makes the SBD(k) schemes attractive for use with request sequences with a low degree of locality. In particular, the empirical studies suggest that there is a parameter k^* that suits best to the degree of locality of a given sequence, i.e. SBD(k^*) achieves the lowest cumulative access costs among all SBD(k) schemes.

In *section 2*, the SORT-BY-RANK algorithms are investigated and theorem 1 is proved. *Section 3* outlines the analysis of the SORT-BY-DELAY rules. In *section 4*, empirical results for both families are reported. They were obtained from streams of independent identically distributed requests and from request sequences obtained from the Calgary Compression Corpus [15], a standard benchmark collection for file compression. *Section 5* concludes with some remarks and open questions.

2 Analysis of the Sort-by-Rank algorithms

First we give an example showing that the SBR(α) rules do not in general satisfy the pairwise independence property. This property means that for every request sequence σ and two arbitrary items x and y in the list L, their relative order is the same when σ is served as their relative order in L_{xy} (the two-element list holding x and y) when the projected sequence σ_{xy} is served [4]. The projeted sequence σ_{xy} is defined as σ after deletion of all requests for items other than x and y. Consider the request sequences $\sigma_1 = x, y, y, a, x$ and $\sigma_2 = x, a, y, y, x$. The projection onto $\{x, y\}$ gives in both cases $\sigma_{xy} = x, y, y, x$. But for $\alpha = 1/2$ and $t = 5$, we have in the first case $r_t(x) = 2 < 2.5 = r_t(y)$ and in the second case $r_t(x) = 2 > 1.5 = r_t(y)$.

For the proof of theorem 1, we need some preliminaries on list factoring. This technique provides lower bounds on the cost OPT(σ) of the optimal algorithm on the request sequence σ by considering a collection of optimally maintained two-element lists. Let FOPT be an optimal "factored" offline algorithm that maintains the collection of all $\binom{n}{2}$ two-element lists L_{xy} for all sets $\{x, y\} \subseteq L$. Hence, each such list L_{xy} is controlled by an optimal offline algorithm that serves the request sequence σ_{xy}. Denote by FOPT_t the cost of serving σ_t which is incurred by FOPT, i.e. the number of two-element lists with σ_t in the second position. Then, for any request sequence $\sigma = \sigma_1, \ldots, \sigma_m$, we have the following lower bound for the optimal offline cost:

$$\sum_{t=1}^{m} \text{FOPT}_t \leq \text{OPT}(\sigma). \tag{1}$$

Further details are omitted due to the lack of space, they may be found in the full version or in [8].

Let L be the list maintained by SBR(α). When item x is currently in front of item y in L, we say $\langle x, y \rangle$ in L. When in the same situation y is in front of x in the list L_{xy} maintained by FOPT, then $\langle x, y \rangle$ is called an inversion.
For each pair of items $x \neq y$, we define the weight function

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

Then we define the potential function $\Phi = \sum_{x \neq y} w(x, y)$.

The potential is always non-negative. Fix a request sequence $\sigma = \sigma_1, \ldots, \sigma_m$. The *amortized cost* a_t for SBR(α) to serve σ_t is defined as $a_t = \text{SBR}(\alpha)_t + \Phi_t - \Phi_{t-1}$, where $\text{SBR}(\alpha)_t$ is the actual cost incurred by SBR(α) in this step, and Φ_t is the potential at time t. The proof demonstrates that there is a constant $c = 2$ such that $\text{SBR}(\alpha)_t \leq c \cdot \text{FOPT}_t$ for all requests $t = 1, \ldots, m$. Then it follows from equation (1) that SBR(α) is 2-competitive.

Lemma 3. *Let y and z be two items such that $\langle z, y \rangle$ in L is an inversion with weight $w(z, y) = 1$. Let x be any item other than y or z. Then $w(z, y)$ remains 1 after a request for x is served by* $\text{SBR}(\alpha)$.

Let the current request be for item $\sigma_t = x$. Without loss of generality we assume that each element has been requested at least once. Define the set $\mathcal{A}$ as the collection of all items z such that $\langle z, x \rangle$ in L is not an inversion; the set $\mathcal{B}$ as the set of all z such that $\langle z, x \rangle$ in L is an inversion, and the set $\mathcal{C}$ as the collection of all elements z such that $\langle x, z \rangle$ in L is an inversion. Note that these three sets are pairwise disjoint. Set $A = |\mathcal{A}|$, $B = |\mathcal{B}|$ and $C = |\mathcal{C}|$.
Let the actual cost at time t be $\text{SBR}(\alpha)_t$. This is equal to $A + B + 1$, as the elements in front of x are exactly the items from $\mathcal{A} \cup \mathcal{B}$. The cost incurred in the optimally maintained factored list is $\text{FOPT}_t = A + C + 1$, because exactly the sets $\{z, x\}$ with $z \in \mathcal{A} \cup \mathcal{C}$ form a contribution to this cost.
Now we consider the change in potential due to the action of $\text{SBR}(\alpha)$ when serving $x = \sigma_t$. If an element $z \in \mathcal{B}$ is passed by x, the weight of the inversion $\langle z, x \rangle$ in L was 1 before the move, and it is now eliminated. If an element $z \in \mathcal{B}$ is not passed by x, the weight of the inversion $\langle z, x \rangle$ in L was 2 before the move, and is now reduced to 1. Hence there is a decrease in potential of exactly B due to items in $\mathcal{B}$. On the other hand, due to $\text{SBR}(\alpha)$'s move, the weight of each inversion $\langle x, z \rangle$ with $z \in \mathcal{C}$ may increase by one which results in a total increase of at most C.
Next we examine the change in potential due to the action of the optimal offline algorithm FOPT maintaining the factored list when serving $x = \sigma_t$. Suppose that a new inversion $\langle x, z \rangle$ in L is created due to the fact that x has passed z in L, but not in L_{xz}. Then $z \in \mathcal{A}$ before the request and we use the following lemma:

Lemma 4. *Let $0 \le \alpha \le 1$. If, at time t in the list L, item $\sigma_t = x$ has passed an element $z \in \mathcal{A}$, then $s_1(z, t) \le s_2(x, t)$. If this request created an inversion $\langle x, z \rangle$ in L, and the next request is for $\sigma_{t+1} = z$, then z will pass x. In other words, this inversion has weight $w(x, z) = 1$.*

Next suppose that a new inversion $\langle x, z \rangle$ in L is created due to the fact that x has passed z in L_{xz}, but not in L. Then $z \in \mathcal{A}$ before the request, and the weight of the inversion can only be one, since a subsequent request for $x = \sigma_{t+1}$ would move x to the front and remove this inversion. Hence, the weight of the new inversions is at most A.
In total, the change in potential after serving the request σ_t can be bounded as $\Phi_t - \Phi_{t-1} \le C - B + A$. The amortized costs for $\text{SBR}(\alpha)$ to serve σ_t are

$$\begin{aligned} a_t &= \text{SBR}(\alpha)_t \ + \ \Phi_t - \Phi_{t-1} \\ &\le A + B + 1 \ + \ C - B + A \\ &= (2 - (C+1)/(A+C+1)) \cdot (A + C + 1) \\ &\le (2 - 1/n) \cdot \text{FOPT}_t. \end{aligned}$$

This completes the proof that $\text{SBR}(\alpha)$ is 2-competitive for $0 \le \alpha \le 1$.

3 Analysis of the Sort-by-Delay algorithms

The proof of SBD(k) being k-competitive for $k \geq 2$ is a modification of the proof of theorem 1 with a different potential function; it is based on the following weight function

$$w(x,y) = \left\{ \begin{array}{l} 0 : \text{if } \langle x,y\rangle \text{ is not an inversion} \\ \ell : \text{if } \langle x,y\rangle \text{ is an inversion and } y \text{ will pass } x \text{ in } L \\ \qquad\qquad\quad \text{iff } y \text{ is now requested } \ell \text{ times, } \ell = 1,\ldots,k \end{array} \right\}.$$

The pairwise independence property of the SBD(k) rules is an immediate consequence of lemma 5.

Lemma 5. *Upon a request for item x, all items that were requested at least k times since the $(k-1)$th last request for x are in front of all items that were requested less than k times since the $(k-1)$th last request for x.*

Lemma 6. *Consider any point in the request sequence where there have been at least $2k-1$ requests to x or y. Then y precedes x if and only if the majority (i.e. at least k) of the $2k-1$ last requests to x or y have been for y.*

The proof of theorem 3 is based on lemma 6. Similarly, lemma 6 can be used to obtain explicit formulæ for the expected search cost of SBD(k) for Markov dependent request sequences. The first part of theorem 4 follows with a Chernoff bound argument, in the second part the calculations are similar to [7].

4 Empirical Results

In figure 1, the simulation results of SBR(α) for the Zipf distribution ($p_i \propto 1/i$) are presented for $0 \leq \alpha \leq 1$, and the average search cost on a list with $n = 50$ items is depicted. The SBR(α) rules provide a gradual transition from MTF to TS. This confirms our intuition in the average case setting where analytical results are difficult to achieve.

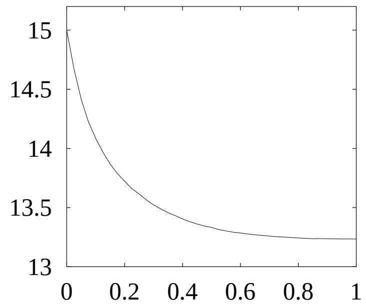

figure 1

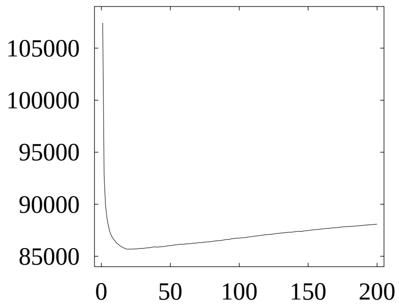

figure 2

We conducted compression experiments with the SBD(k) rules on the Calgary/ Canterbury Compression Corpus [15]. The list update schemes work as "engines" that dynamically define a statistical model for the data. Initially the list is sorted by decreasing access frequencies of the characters. Then the characters are encoded sequentially. When a character has been found on position i of the list, its position is encoded with Elias encoding, a standard variable length prefix-free code for positive integers, using $1 + 2\lfloor \log_2(i) \rfloor$ bits.

For example, the compression performance of SBD(k) on the file *geo* is depicted in figure 2 for $1 \leq k \leq 200$. The value k^* shall describe the rule with the best compression among all SBD(k) schemes. Hence the adaptive discipline SBD(k^*) fits best to the particular degree of locality found in the input stream. Sometimes however, the value k^* is not sharply distinguished. Therefore, k^* can not give a precise quantification of locality in general.

In the following table, the results are collected and compared with the sizes of the original file and the results achieved by MOVE-TO-FRONT compression (MTF) and by FREQUENCY COUNT compression (FC). The latter uses as statistical model a static list that is ordered by decreasing access frequencies. The results of SBD(k^*) are significantly better than MTF compression, with typical savings around 15 %. However, in comparison with FREQUENCY COUNT, SBD(k^*) hardly manages to break even. The good performance of FC on the Calgary Compression Corpus has already been observed in [3].

File	Original	FC	MTF	k^*	SORT-BY-DELAY(k^*)			
	Bytes	Bytes	Bytes		Bytes	% Orig	% FC	% MTF
bib	111261	88641	106468	73	89145	80.1	100.6	83.7
book1	768771	516198	644418	153	516966	67.2	100.1	80.2
book2	610856	434798	515255	23	432940	70.9	99.6	84.0
geo	102400	85533	107422	19	85687	83.7	100.2	79.8
news	377109	292800	333729	18	289925	76.9	99.0	86.9
obj1	21504	19304	19351	3	17802	82.8	92.2	92.0
obj2	246814	237165	250952	7	219668	89.0	92.6	87.5
paper1	52161	39617	46140	25	39066	73.5	98.6	84.7
paper2	82199	56356	69437	54	56539	68.8	100.3	81.4
paper3	46526	32497	39373	35	32697	70.3	100.6	83.0
paper4	13286	9229	11273	30	9327	70.2	101.1	82.7
paper5	11954	8741	10195	18	8759	73.3	100.2	85.9
paper6	38105	28487	32073	13	27320	71.7	95.9	85.2
pic	513216	111607	119125	7	109733	21.4	98.3	92.1
progc	39611	30779	39150	13	30414	76.8	98.8	77.7
progl	71646	51260	55178	14	48776	68.1	95.2	88.4
progp	49379	35066	40041	9	34903	70.7	99.5	87.2
trans	93695	82297	82055	5	76895	82.1	93.4	93.7

We are optimistic that the possibility to adapt to the degree of locality is fruitful in other applications. Similarly, the SBR(α) schemes could be used for fine-tuning to data streams with very high level of locality, such as the ouput of the Burrows-Wheeler-transform [6].

5 Conclusion

We introduced and analysed two families of deterministic list update algorithms, called Sort-by-Rank(α) and Sort-by-Delay(k). Empirical studies with sequences from the Calgary Compression Corpus suggest that the parameter k^* which denotes the rule SBD(k^*) with the best compression performance could be considered as a crude measure of the degree of locality of that sequence. The question of quantifying the locality of reference in data streams remains a challenge of major importance.
It would also be interesting to experiment with non-linear rank functions, for example $r_t(x) = (s_1(x,t)^p + s_2(x,t)^p)^{1/p}$. Finally, an open question is to improve the tradeoff between competitive ratio and expected search cost on memoryless sources as exhibited by the SBD(k) family, or to prove a lower bound.

Acknowledgements
The discussions with Günter Hotz and the comments by Susanne Albers are greatly appreciated.

References

1. S. Albers and M. Mitzenmacher. Average case analyses of list update algorithms. In *ICALP'96*, LNCS 1099, pages 514–525. Springer, 1996.
2. S. Albers. Improved randomized on-line algorithms for the list update problem. In *SODA'95*, pages 412–419, 1995.
3. Ran Bachrach and Ran El-Yaniv. Online list accessing algorithms and their applications: Recent empirical evidence. In *SODA'97*, pages 53–62, 1997.
4. J. L. Bentley and C. C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985.
5. J. L. Bentley, D. Sleator, R. E. Tarjan, and V. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29:320–330, 1986.
6. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report 124, DEC Systems Research Center, May 1994.
7. F. R. K. Chung, D. J. Hajela, and P. D. Seymour. Self-organizing sequential search and Hilbert's inequalities. In *Proc. 17th ACM STOC*, pages 217–223, 1985.
8. Ran El-Yaniv. There are infinitely many competitive-optimal online list accessing algorithms. Manuscript, 1997.
9. M. J. Golin. A provably fast linear-expected-time maxima-finding algorithm. *Algorithmica*, 11:501–524, 1994.
10. G. H. Gonnet, J. I. Munro, and H. Suwanda. Exegesis of self-organizing linear search. *SIAM Journal of Computing*, 10(3):613–637, 1981.
11. K. Lam, M. Y. Leung, and M. K. Siu. Self-organizing files with dependent accesses. *Journal of Applied Probability*, 21:343–359, 1984.
12. J. McCabe. On serial files with relocatable records. *Oper. Res.*, 12:609–618, 1965.
13. F. Schulz and E. Schömer. Self-organizing data structures with dependent accesses. In *ICALP'96*, LNCS 1099, pages 526–537. Springer, 1996.
14. D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.
15. I. H. Witten and T. Bell. The calgary text compression corpus. Anonymous ftp from `ftp.cpsc.ucalgary.ca:/pub/projects/text.compression.corpus/`.