A Survey of Algorithms and Models for List Update

Shahin Kamali and Alejandro López-Ortiz

School of Computer Science, University of Waterloo
Waterloo, Ont., N2L 3G1, Canada
{s3kamali,alopez-o}@uwaterloo.ca

Abstract. The list update problem was first studied by McCabe [47] more than 45 years ago under distributional analysis in the context of maintaining a sequential file. In 1985, Sleator and Tarjan [55] introduced the competitive ratio framework for the study of worst case behavior on list update algorithms. Since then, many deterministic and randomized online algorithms have been proposed and studied under this framework. The standard model as originally introduced has a peculiar cost function for the rearrangement of the list after each search operation. To address this, several variants have been introduced, chiefly *the MRM model* (Martínez and Roura, [46]; Munro, [49]), *the paid exchange model*, and *the compression model*. Additionally, the list update problem has been studied under locality of reference assumptions, and several models have been proposed to capture locality of input sequences. This survey gives a brief overview of the main list update algorithms, the main alternative cost models, and the related results for list update with locality of reference. Open problems and directions for future work are included.

1 Introduction

List update is a fundamental problem in the context of online computation. Consider an unsorted list of l items. The input to the algorithm is a sequence of n requests that must be served in an online manner. Let $\mathcal{A}$ be an arbitrary online list update algorithm. To serve a request to an item x, $\mathcal{A}$ linearly searches the list until it finds x. If x is the ith item in the list, $\mathcal{A}$ incurs a cost i to access x. Immediately after this access, $\mathcal{A}$ can move x to any position closer to the front of the list at no extra cost; this is called a *free exchange*. Also, $\mathcal{A}$ can exchange any two consecutive items at a cost of 1; these are called *paid exchanges*. An efficient algorithm can thus use free and paid exchanges to minimize the overall cost of serving a sequence. This model is called the *standard cost model* [7].

The competitive ratio, first introduced formally by Sleator and Tarjan [55], has served as a practical measure for the study and classification of online algorithms in general and list update algorithms in particular. An algorithm is said to be α-competitive (assuming a cost-minimization problem) if the cost of serving any specific request sequence never exceeds α times the optimal cost (up to some additive constant) of an *offline* algorithm which knows the entire request sequence in advance.

Notwithstanding its wide applicability, competitive analysis has some drawbacks. For certain problems, it gives unrealistically pessimistic performance ratios and fails to distinguish between algorithms that have vastly differing performance in practice. Such anomalies have led to the introduction of many alternatives to competitive analysis of online algorithms. For a comprehensive survey of alternative models, see [31].

As well, a common objection to competitive analysis is that it relies on an optimal offline algorithm, OPT, as a baseline for comparing online algorithms. While this may be convenient, it is rather indirect: One could argue that in comparing two online algorithms $\mathcal{A}$ and $\mathcal{B}$, all the information we should need is the cost incurred by the algorithms on each request sequence. For example, for some problems, OPT is too powerful, causing all online algorithms to seem equally bad. Certain alternative measures allow direct comparison of online algorithms, for example, the *Max-Max Ratio* [18], the *Relative Worst Order Analysis* [23,24], and the *Bijective Analysis* [12,13,14]. These measures have been applied mostly to the paging problem as well as some other online problems.

To the best of our knowledge, relative worst order analysis [33] and bijective analysis [13,14] are the only alternative measures which have already been applied to the list update problem. As mentioned above, both these measures directly compare two online algorithms $\mathcal{A}$ and $\mathcal{B}$. For a given sequence, the relative worst order analysis considers the worst ordering of the sequence for both $\mathcal{A}$ and $\mathcal{B}$ and compares their outcome on these orderings. Then, among all sequences, it considers the one that maximizes the worst case performance. For a precise definition, see [24]. Besides the list update problem, relative worst order analysis has been applied to other online problems including bin packing [23,36], paging [24,25], scheduling [35], and seat reservation [26].

Under bijective analysis, we say $\mathcal{A}$ is no worse than $\mathcal{B}$ if we can define a bijection f on the input sequences of the same length, such that the cost of $\mathcal{A}$ for any sequence σ is not more than that of $\mathcal{B}$ for $f(\sigma)$, where $f(\sigma)$ is the bijected sequence of σ. Bijective analysis proved successful in proving that LRU and MTF are the unique optimal algorithms for respectively paging and list update problems under a locality of reference assumption, while all other measures have failed to show this empirically observed separation.

The list update problem is closely related to the *paging problem.* For paging, there is a small memory (cache) of size k and a large memory of unbounded size. The input is a sequence of page requests. If a requested page a is already in the cache the algorithm pays no cost; otherwise, it pays a cost of one unit to bring a into the cache. On bringing a page to the cache, an algorithm might need to *evict* some pages from the cache to make room for the incoming page. Paging algorithms are usually described by their eviction strategy; for example, Least-Recently-Used (LRU) evicts the page in the cache that was least recently used, and First-In-First-Out (FIFO) evicts the page that was first brought to the cache. Paging can be seen as a type of list update problem with an alternative cost model [53,55]: Each page is equivalent to an item in the list and the access cost for the first k items (k being the size of the cash) is 0, while the cost for other items is 1. The only difference between the two problems is that when a requested page is not in the cache, a paging algorithm should bring the page to the cache. This is optional for a list update algorithm.

1.1 Outline

In this survey, we give an overview of selected results regarding list update in the standard and alternative cost models. We do not aim to be exhaustive, but rather give highlights of the field. For other surveys on the list update problem, we refer the reader to [2,4,48]. In Section 2, we review a selection of existing results on list update for the standard model. In particular, we consider the classical deterministic and randomized algorithms for the problem. While list update algorithms with a better competitive ratio tend to have better performance in practice, the validity of the cost model has been debated, and a few other models have been proposed for the problem. In Section 3, we review the main results related to these alternative cost models which include the MRM cost model, the paid exchange model, the compression model, and the free exchange model. In Section 4, we discuss the list update problem with locality of reference and review the proposed models which measure this locality.

2 Algorithms

List update algorithms were among the first algorithms studied using competitive analysis. Three well-known deterministic online algorithms are *Move-To-Front* (MTF), *Transpose*, and *Frequency-Count* (FC). MTF moves the requested item to the front of the list; whereas Transpose exchanges the requested item with the item that immediately precedes it. FC maintains an access count for each item ensuring that the list always contains items in non-increasing order of frequency count. *Timestamp* is an efficient list update algorithm introduced by Albers [1]: After accessing an item x, Timestamp inserts x in front of the first item y that is before x in the list and was requested at most once since the last request for x. If there is no such item y, or if this is the first access to x, Timestamp does not reorganize the list. Sleator and Tarjan showed that MTF is 2-competitive, while Transpose and FC do not have constant competitive ratios [55]. Karp and Raghavan proved a lower bound of $2 - 2/(l + 1)$ (reported in [40]), and Irani proved that MTF gives a matching upper bound [40]. It is known that Timestamp is also optimum under competitive analysis [1].

Besides MTF and Timestamp which have optimum competitive ratio, El-Yaniv showed that there are infinitely many algorithms which have optimum ratio [34]. In doing so, he introduced a family of algorithms called Move-to-Recent-Item (MRI): A member of this family has an integer parameter $k \geq 1$, and it inserts an accessed item x just after the last item y in the list which precedes x and is requested at least $k+1$ times since the last request to x. If such item y does not exist, or if this is the first access to x, the algorithm moves x to front of the list. It is known that any member of MRI family of algorithms is 2-competitive [34].

Note that when $k = 0$, MRI is the same as Timestamp, and when k tends to infinity, it is the same as MTF. Schulz proposed another family of algorithm called Sort-By-Rank (SBR) [54], which is parameterized by a real value α where $0 \leq \alpha \leq 1$. The extreme values of α result in MTF (when $\alpha = 0$) and Timestamp (when $\alpha = 1$). It is known that any member of SBR family is also 2-competitive [54].

Classical list update algorithms have also been studied under relative worst order analysis [33]. It is known that MTF, Timestamp, and FC perform identically according to the relative worst order ratio, while Transpose is worse than all these algorithms. Note that these results are almost aligned with the results under competitive analysis.

In terms of the optimum offline algorithm for the list update problem, Manasse et al. presented an offline optimal algorithm which computes the optimal list ordering at any step in time $\Theta(n \times (l!)^2)$ [45], where n is the length of the request sequence. This time complexity was improved to $\Theta(n \times 2^l(l-1)!)$ by Reingold and Westbrook [51]. Hagerup proposed another offline algorithm which runs in time $O(2^l \ l! \ f(l) + l \times n)$, where $f(l) \leq l! \ 3^{l!}$ [38]. Note that the time complexity of these algorithms is incomparable to one another. As pointed out in [38]: "[Hagerup's algorithm] is probably the best algorithm known for $l = 4$, but it loses out rapidly for larger values of l due to the growth of $f(l)$." Another algorithm by Pietzrak is reported to run in time $\Theta(n \ l^3 \ l!)$ [48]. It should be mentioned that Ambühl claims that the offline list update problem is NP-hard [8], although a full version of the proof remains to be published.

2.1 Paid Exchanges vs Free Exchanges

All the main existing online algorithms for the list update problem are *economical*, i.e., they only use free exchanges (look at Table 1). In fact, there is only one known non-trivial class of algorithms that uses paid exchanges [37]. This raises the question of how important it is to make use of paid exchanges. In their seminal paper, Sleator and Tarjan claimed that free exchanges are sufficient to achieve an optimal solution [55]. The following example by Reingold and Westbrook shows that this is not the case, and that paid exchanges are sometimes necessary [51].

Consider the initial list configuration (1,2,3) and the request sequence $\langle 3, 2, 2, 3 \rangle$. One can verify that any algorithm relying solely on free exchanges must incur a cost of at least 9. On the other hand, if we apply two paid exchanges (at a cost of 2) before any access is made and refashion the list into (2,3,1), the accesses $\langle 3, 2, 2, 3 \rangle$ can be served on that list at a cost of 2,1,1,2 respectively, for a total access cost of 6 and an overall cost of $2 + 6 = 8$ once we include the cost of the paid accesses.

Considering the above example, one might ask: what is the approximation ratio of the best (offline) list update algorithm which is restricted to only use free exchanges? This question is still open and remains to be answered. We conjecture this ratio to be strictly larger than 1 and smaller than or equal to 4/3.

Table 1. A review of online strategies for list update problem

Algorithm	Competitive Ratio	deterministic	Projective	Economical
MTF	2 [55,40]	✓	✓	✓
Transpose	non-constant [55]	✓	x	✓
Frequency Count	non-constant [55]	✓	✓	✓
Random-MTF (RMTF)	2 [21]	x	✓	✓
Move-Fraction (MF$_k$)	$2k$ [55]	✓	x	✓
Timestamp	2 [1]	✓	✓	✓
MRI family	2 [34]	✓	✓	✓
SBR family	2 [54]	✓	✓	✓
Timestamp family (randomized)	1.618 [1]	x	✓	✓
SPLIT	1.875 [40,42]	x	x	✓
BIT	1.75 [52]	x	✓	✓
Random Reset (RST)	1.732 [52]	x	✓	✓
COMB	1.60 [6]	x	✓	✓

Reingold and Westbrook showed that there are optimal offline algorithms which only make use of paid exchanges [51]. Note that this is the exact opposite of the situation as claimed by Sleator and Tarjan. It is still not clear how an online algorithm can make good use of paid exchanges. Almost all existing optimal online algorithms only use free exchanges, while there are optimal offline algorithms which only use paid exchanges. This calls into question the validity of the standard model (see Section 3.4).

2.2 Randomization

As mentioned earlier, any deterministic list update algorithm has a competitive ratio of at least $2 - 2/(l + 1)$. In order to go past this lower bound, a few randomized algorithms have been proposed. However, it is important to observe that the competitive ratio of a randomized algorithm is not a worst case measure, and in that sense, it is closer in nature to the *average* competitive ratio of a deterministic algorithm.

Randomized online algorithms are usually compared against an *oblivious* adversary which has no knowledge of the random bits used by the algorithm. To be more precise, an oblivious adversary generates a request sequence before the online algorithm starts serving it, and in doing so, it does not look at the random bits used by the algorithm. Note that the oblivious adversary knows the algorithm code. Two stronger types of adversaries are *adaptive online* and *adaptive offline* adversaries. An adaptive online adversary generates the tth requests of an input sequence by looking at the actions of the algorithm for serving the last $t - 1$ requests. An adaptive offline adversary is even more powerful and knows the random bits used by the algorithm, i.e., before giving the input sequence to the online algorithm, it can observe how the algorithm serves the sequence. The definition of the competitive ratio of an online algorithm is slightly different when compared with different adversaries, and is based on the expectations over the random choices made by the online algorithm (and adaptive adversaries). For a precise definition of competitiveness for randomized algorithms, we refer the reader to [52].

Ben-David et al. proved that if there is a randomized algorithm which is c-competitive against an adaptive offline adversary, then there exist deterministic algorithms which are also c-competitive [19]. In this sense, randomization does not help to improve the competitive ratio of online algorithms against adaptive offline adversaries. In fact, for the list update problem, the adaptive online and adaptive offline adversaries are equally powerful, and the lower bound $2-2/(l+1)$ for deterministic algorithms extends to adaptive adversaries [50,52]. This implies that there is no randomized algorithm which achieves a competitive ratio better than $2-2/(l+1)$ when compared against adaptive adversaries. So, randomization can only help in obtaining a better competitive ratio when compared against an oblivious adversary. In the following review of randomized algorithms, by the notion of c-competitiveness, we mean c-competitiveness against an oblivious adversary.

Random-MTF (RMTF) is a simple randomized algorithm for the list update problem: after accessing an item, RMTF moves it to the front with probability 0.5. The competitive ratio of RMTF is 2 [21], which is no better than the best deterministic algorithms. The first randomized algorithm that beats the deterministic lower bound was introduced by Irani in 1991 [40]. In this algorithm, called SPLIT, each item x has a pointer to another item which precedes it in the list, and after each access to x, the algorithm moves x to either the front of the list or front of the item that x points to (we omit the details here). This randomized algorithm has a competitive ratio of 1.875 [40,42]. Reingold et al. proposed another randomized algorithm named BIT [52]: Before serving the sequence, the algorithm assigns a bit $b(x)$ to each item x which is randomly set to be 0 or 1. At the time of an access to an element x, the content of $b(x)$ is complemented. Then, if $b(x) = 1$, the algorithm moves x to the front; otherwise (when $b(x) = 0$), it does nothing. Note that BIT uses randomness only in the initialization phase, and after that it runs deterministically; in this sense the algorithm is *barely random*. It is known that BIT has a competitive ratio of 1.75 [52].

BIT is a member of a more generalized family of online algorithms called COUNTER[52]. A COUNTER algorithm has two parameters: an integer s and a fixed subset S of $\{0, 1, ..., s-1\}$. The algorithm keeps a counter modulo s for each item. With each access to an item x, the algorithm decrements the counter of x and moves it to the front of the list if the new value of the counter is a member of S. With good assignments of s and S, a COUNTER algorithm can be better than BIT. For example, with $s = 7, S = \{0, 2, 4\}$, the ratio will be 1.735, which is better than the 1.75 of BIT [52].

It is also known that a random reset policy can improve the ratio even further: The algorithm Random Reset maintains a counter $c(x)$ for each item x in the list. The counter is initially set randomly to be a number i between 0 and s

with probability π_i. When the requested item has a counter larger than 1, the algorithm makes no move and decrements the counter. If the counter is 1, it moves the item to front and resets the item counter to $i < s$ with probability π_i. Unlike COUNTER algorithms, RANDOM RESET algorithms are not barely random. The best values of s and D result in an algorithm with a competitive ratio of $\sqrt{3} \approx 1.732$ [52].

The deterministic Timestamp algorithm described earlier is indeed a special case of a family of randomized algorithms introduced by Albers [1]. A randomized Timestamp(p) algorithm has a parameter p. Upon a request to an item, the algorithm applies the MTF strategy with probability p and (deterministic) Timestamp with probability $1 - p$. The competitive ratio of Timestamp(p) is $\max\{2 - p, 1 + p(2 - p)\}$ which achieves its minimum when $p = (3 - \sqrt{5})/2$; this gives a competitive ratio of 1.618. Albers et al. proposed another hybrid algorithm which randomly chooses between two other algorithms [6]. This algorithm is called COMB. Upon a request to an item, the algorithm applies BIT strategy with probability 0.8 and (deterministic) Timestamp with probability 0.2. COMB has a competitive ratio of 1.6 [6], which is the best competitive ratio among existing randomized online algorithm for the list update problem.

There has been some research for finding lower bounds for competitive ratio of randomized list update algorithms against an oblivious adversary [40,52,56]. The best existing lower bound is 1.5 proven by Teia, assuming the list is sufficiently large [56]. Under the partial cost model, where an algorithm pays $i - 1$ units to access an item in the ith position, Ambühl et al. proved a lower bound of 1.50084 for a randomized online algorithm. Note that there is still a gap between the best upper and lower bounds [10].

Although randomized algorithms achieve better competitive ratios than deterministic algorithms, the validity of such comparisons is under question. As mentioned earlier, the competitive ratio of a randomized algorithm (against an oblivious adversary) is defined by the expectation over the random choices made by the online algorithm. In that sense, the competitive ratio does not capture the worst-case behavior of randomized algorithms, instead, it captures the expected cost over random bits for the worst sequence generated by the adversary. A better comparison would be to compare the worst cost over random bits, which is captured by adaptive adversaries. As mentioned earlier, randomized online algorithms cannot achieve improved ratios against adaptive adversaries. To conclude, the improved competitive ratios of randomized online algorithms are mostly due to the decreased power of adversary rather than enhanced power or smarts of online algorithms. There is empirical evidence supporting this, as in real life sequences (e.g., when there is locality of reference) deterministic algorithms outperform their randomized counterparts [15,13]. Besides randomized competitive analysis, randomized algorithms have also been studied under the relative worst order ratio. It is known that under this framework RMTF and BIT are not comparable [33].

2.3 Projective Property

Most of the existing algorithms for the list update problem satisfy the *projective property*. Intuitively, an algorithm is projective if the relative position of any two items x, y in the list maintained by the algorithm only depends on their relative position in the initial configuration and also the requests to x and y in the input sequence. The algorithms with the projective property are usually studied under the partial cost model, where an algorithm pays $i-1$ units to access an item in the ith position.

In order to achieve an upper bound for the competitive ratio of an algorithm $\mathcal{A}$ with the projective property, it suffices to compare the cost of $\mathcal{A}$ when applied to sequences of two items with the cost of an optimal algorithm OPT_2 for serving those sequences. Fortunately, the nature of OPT_2 is well-understood and there are efficient optimal offline algorithms for lists of size two [51]. This opens the door for deriving upper bounds for the competitive ratio of projective algorithms under the partial cost model, which also extend to the full cost model.

Ambühl et al. showed that COMB is the optimum randomized projective algorithm under competitive analysis [9,11]. Consequently, if one wants to improve on the randomized competitive ratio 1.6 of COMB, they should introduce a new algorithm which is not projective.

3 Alternative Models

The validity of the standard cost model for the list update has been debated, and a few other models have been proposed for this problem. In this section, we review these models and the main relevant results.

3.1 MRM Model

Martínez and Roura [46], and also Munro [49], independently addressed the drawbacks of the standard cost model. The result is a model that we refer to as the MRM model. The standard model is not realistic in some practical settings such as when the list is represented by an array or linked list. Martínez and Roura argued that, in a realistic setting, a complete rearrangement of all items in the list which precede the requested item at position i would require a cost proportional to i, while this has cost proportional to i^2 in the standard cost model. Munro provided the example of accessing the last item of the list of size l and then reversing the entire list. The real cost of this operation in an array or a linear link list should be $O(l)$, while it costs about $l^2/2$ in the standard cost model. As a consequence, their main objection to the standard model is that it prevents online algorithms from using their true power. They instead proposed the MRM model in which the cost of accessing the ith item of the list plus the cost of reorganizing the first i items is linear in i.

Surprisingly, it turns out that the offline optimum benefits substantially more from this realistic adjustment than the online algorithms do. Under the MRM model, every online algorithm has an amortized cost of $\Theta(l)$ per access for some arbitrary long sequences, while there are optimal algorithms which incur a cost of $\Theta(\lg l)$ on every sequence. Hence, all online list update algorithms have a competitive ratio of $\Omega(l/\lg l)$. Among offline algorithms which have an amortized access cost of $\Theta(\lg l)$ per request, we might mention Order_By_Next_Request (OBNR) proposed by Munro [49]: After accessing an item x at position i, OBNR reorders the elements from position 1 to position p in order of next access (i.e., the items which will be requested earlier appear closer to front). Here p is the first position at or beyond i which is a power of 2, i.e., $p = 2^{\lceil \lg i \rceil}$.

We would like to observe that randomization does not help to improve the competitive ratio under the MRM model, and the same lower bound holds for randomized online algorithms, i.e., there is no randomized algorithm with a constant competitive ratio under the MRM model [46]. One may be tempted to argue that this is proof that the new model makes the offline optimum too powerful and hence this power should be removed; however, this is not correct as in real life online algorithms can rearrange items at the cost indicated. Note that the ineffectiveness of this power for improving the worst case competitive ratio does not preclude the possibility that under certain realistic input distributions (or other similar assumptions on the input) this power might be of use. Martínez and Roura observed this and posed the question [46]: "[An] important open question is whether there exist alternative ways to define competitiveness such that MTF and other good online algorithms for the list update problem would be competitive, even for the [modified] cost model". This question was answered by Angelopoulos et al. who showed that MTF is the unique optimal algorithm under bijective analysis for input sequences which have locality of reference [13,14].

Unlike the standard model, under which the offline problem is NP-hard, Golynski and López-Ortiz introduced an offline algorithm which computes the optimal arrangement in time $O(n^3)$ (n is the length of the input sequence) and serves any input sequence optimally under the MRM model [37]. It remains open to investigate whether an optimal offline algorithm with a better running time exists. Kamali et al. did an empirical study of the performance of the list update algorithms under the MRM model and observed that a *context-based* algorithm, initially applied for compression purposes, outperforms other algorithms [43].

3.2 Paid Exchange Model

Reingold et al. considered another variant of the standard model in which the access cost is similar to the standard model, but the cost for paid exchanges is scaled up by a value $d \geq 1$ [52]. As pointed out in [52], "This is a very natural extension, because there is no a priori reason to assume that the execution time of the program that swaps a pair of adjacent elements is the same as that of the program that does one iteration of the search loop". In the new model, the cost involved in a paid exchange is d, while free exchanges are not allowed. We refer to this model as the *d-paid exchange model.* Reingold et al. suggested a family of randomized COUNTER algorithms for this model [52]. Each algorithm in this family has a parameter s and maintains a modulo s counter for each item.

The counters are initially set independently and uniformly at random. When there is a request to an item x the counter for x is decremented. In case the counter becomes $s - 1$, x is moved to front. The competitive ratio of these algorithms improves as d increases. For the best selection of s, the ratio will be 2.75 when $d = 1$. This decreases to $(5 + \sqrt{17})/4 \approx 2.28$ when d tends to infinity.

On the other hand, for any value of $d \geq 1$, no deterministic algorithm can be better than 3-competitive under the d-paid exchange model [52]. The proof is based on a cruel adversary which requests the last item in the list maintained by the online algorithm $\mathcal{A}$. If the cost paid by $\mathcal{A}$ for paid exchanges is more than half of the cost it pays for the accesses, the adversary takes the optimal static approach. (It sorts items in decreasing order of their frequencies in the sequence and does not move any item.) Otherwise, the adversary maintains a list which is the same list as the one maintained by the algorithm, but in reverse order. Hence, the total access cost that the adversary pays is n, compared to the $n \times l$ that the online algorithm pays (l being the size of the list), while the cost involved in exchanges remain the same in both. A precise analysis gives the lower bound of 3. This lower bound extends to randomized algorithms when compared against adaptive adversaries [52]. Westbrook showed that a deterministic version of the COUNTER family has a competitive ratio of at most $(5 + \sqrt{17})/2 \approx 4.56$ (reported in [7]). These algorithms perform similar to randomized COUNTER algorithms, except that the counters are initially set to be 0. Note that there is still a gap between the best upper and lower bounds.

Sleator and Tarjan studied the list update problem under the standard model when no free exchanges are allowed [55]. We refer to this model as the *paid exchange model*. Note that this model is equivalent to the d-paid exchange model with $d = 1$. Recall that almost all existing algorithms only make use of free exchanges. Under the paid exchange model, free exchanges can be replaced by a set of paid exchanges (paying an additional cost). For example, MTF pays almost twice for each request, since after accessing an element at index i, the algorithm pays another $i - 1$ units to move the item to the front using paid exchanges. In fact, any algorithm with a competitive ratio i under the standard model has a competitive ratio of at most $2i$ under the paid exchange model. This holds because OPT pays the same cost under both the standard and paid exchange models. In particular, MTF has a competitive ratio of 4 under the paid exchange model [55]. So, there is a gap between the lower bound 3 and the upper bound of 4 given by MTF. To close this gap, one might consider the algorithm MTF-Every-Other-Access which moves a requested item to the front of the list on every even request for the item. Note that this is equal to (deterministic) COUNTER algorithm with $s = 2$. A detailed analysis shows that this algorithm is 2-competitive under the standard model [21], and 3-competitive under the paid exchange model (we skip the details in this review). Hence, the algorithm is optimal under both models, and the lower bound 3 is tight for the paid exchange model.

3.3 Compression Model

An important application of the list update problem is in data compression. Such an application was first reported by Bentley et al. who suggested that an online list update algorithm can be used as a subroutine for a compression scheme [20]. Consider each character of a text as an item in the list, and the text as the input sequence. A compression algorithm writes an arbitrary initial configuration in the compressed file, as well as the access cost of ALG for serving each character in unary. Hence, the size of the compressed file is equal to the access cost of the list update algorithm. The initial scheme proposed in [20] used MTF as its subroutine. Albers and Mitzenmacher [5] used Timestamp and showed that in some cases it outperforms MTF. Bachrach et al. studied compressions schemes for a large family of list update algorithms which includes MTF and Timestamp [16]. In order to enhance the performance of the compression schemes, the Burrows-Wheeler Transform (BWT) can be applied to the input string to increase the amount of locality [27]. Dorrigiv et al. observed that after applying the BWT, the schemes which use MTF outperform other schemes in most cases [32].

All the above studies adopt the standard cost model for analysis of compressions schemes. More formally, when an item is accessed in the ith position of the list, the value of i is written in unary format on the compressed file. In practice, however, the value of i is written in binary format using $\Theta(\lg i)$ bits. Hence the true "cost" of the access is logarithmic in what the standard model assumes. This was first observed in the literature by Dorrigiv et al. in [32] where they proposed a new model for the list update problem which is more appropriate for compression purposes. We refer to this model as the *compression model.* Under this model, the cost of accessing an item in the ith position is $\Theta(\lg i)$. They observed that there is a meaningful difference between the standard model and compression model: Consider the Move-Fraction (MF) family of list update algorithms proposed by Sleator and Tarjan [55]. An algorithm in this family has a parameter k ($k \geq 2$) and upon a request to an item in the ith position, moves that item $\lceil i/k \rceil - 1$ positions towards the front. While MF_k is known to be $2k$-competitive under the standard model [55], it is not competitive under the compression model [32]. For example, MF_2 is 4-competitive under the standard model, and $\Omega(\lg l)$ competitive under the compression model. A precise analysis of MTF under the compression model shows that it has a competitive ratio of 2 under the compression model, which makes it an optimal algorithm under this model. We skip the details in this review.

A randomized algorithm can also be applied for text compression if the random bits used by the algorithm are included in the compressed file. The number of random bits does not change the size of the file dramatically, specially for barely random algorithms like BIT. So it is worthwhile to study these algorithms under the compression model.

3.4 Free Exchange Model

Recall that all well known existing online algorithms for the list update problem only use free exchanges, while an optimal offline can restrict itself to using paid exchanges only. This suggests that it is more appropriate to consider a model which does not allow paid exchanges, i.e., after an access to an element an algorithm can only move the item to somewhere closer to the front. This is a natural variant of the standard model which we call the *free exchange model.* Since most of the existing algorithms for list update only use free exchanges, they have the same cost under this model. As mentioned earlier, under the standard model, OPT needs to use paid exchanges. Hence, restricting the model to only allow free exchanges decreases the power of OPT. However, the lower bound of 2 (the lower bound for the competitive ratio of any deterministic online algorithm under the standard model) can be extended to the free exchange model, i.e., under the free exchange model, any deterministic online algorithm has a competitive ratio of at least 2. The proof is straightforward and omitted in this review.

4 Locality of Reference

Another issue in the analysis of online algorithms is that "real-life" sequences usually exhibit *locality of reference*. Informally, this property suggests that the currently requested item is likely to be requested again in the near future. For the paging problem, several models for capturing locality of reference have been proposed [22,57,3,17].

Borodin et al. suggested the notion of *access graph* to model locality for paging [22]. This model assumes that input sequences are consistent with an access graph G, which is known to the online algorithm. Each page is associated to a vertex of G. In a consistent sequence, there is an edge between vertices associated with two consecutive requests. The quality of algorithms is then compared using competitive analysis [22,41,28] or any other measure, e.g., the relative worst order ratio [25]. It is known that an algorithm FAR, which brings the structure of the access graph into account, is a *uniformly competitive* algorithm, i.e., it is within a constant factor of the best attainable competitive ratio for any access graph [41]. In that sense, FAR outperforms both FIFO and LRU, which are k-competitive for some graph families (e.g., cycles of length $k + 1$). Among classical paging algorithms, most results show an advantage for LRU over other algorithms, in particular FIFO [28,25].

Other models defined for capturing locality of reference in paging include that of Karlin et al., which assumes the sequences are generated by a Markov chain [44], and that of Torng, which compares the average length of sequences in a window containing m different pages [57]. The model considered by Torng is related to the concept of *working set* defined by Denning [29,30]. At any time t, and for a time window of size τ, the working set $W(t, \tau)$ is the number of pages accessed by a process in the time window τ. Denning shows that in practical scenarios the size of the working set is a concave function of τ. Inspired by this, Albers et al. defined a model for locality, called *concave analysis*, in which the sequences are consistent with a concave function f so that the maximum number of distinct requests in a window of size τ is at most $f(\tau)$ [3]. Measuring the performance of online algorithms by the page fault rate (the ratio between the number of faults and all requests), they showed that LRU is the optimal algorithm for sequences which are consistent with a concave function, while this is not the case for FIFO. Later, Angelopoulos et al. showed that under the same model of locality, LRU is the unique optimal algorithm for paging with respect to bijective analysis, when the quality is measured by the number of faults (rather than the fault rate) [12,14].

In practical scenarios, input sequences for the list update problem have a high degree of locality. This is particularly the case when list update is used for compression purposes after BWT (see Section 3.3). Hester and Hirschberg claim [39]: "Move-To-Front performs best when the list has a high degree of locality". Angelopoulos et al. formalized this claim by showing that MTF is the unique optimal solution under bijective analysis for sequences that have locality of reference with respect to concave analysis [13,14].

Albers and Lauer further studied the problem under locality of reference assumption [4]. They defined a new model which is based on the number of *runs* in input sequences: For an input sequence $\sigma_{x,y}$ involving two elements only, a run is a maximal subsequence of requests to the same item. A run is *long* if it contains at least two requests; otherwise, it is *short*. Consider the items in the list to be x and y. Consider a long run R of requests to x. Let R' denote the next long run which comes after R in the sequence. Note that there might be short runs between R and R'. If R' is formed by requests to y, then a *long run change* happens. Also, a single extra long run change happens when the first long run and the first request of the sequence reference the same item. For an arbitrary sequence σ, define the number of runs (resp. long run changes) to be the total number of runs (resp. long run changes) of the projected sequences over all pairs (x, y). Let $r(\sigma)$ and $l(\sigma)$ respectively denote the total number of runs and long run changes in σ. Define $\lambda = \frac{l(\sigma)}{r(\sigma)}$, i.e., λ represents the fraction of long run changes among all the runs. Note that we have $0 \leq \lambda \leq 1$. The larger values for λ imply a higher locality of the sequence, e.g., when all runs are long, we get $\lambda = 1$. Also, note that the length of long runs does not affect the value of λ. Using this notion of locality, the competitive ratio of MTF is at most $\frac{2}{1+\lambda}$, i.e., for sequences with high locality MTF is 1-competitive. The ratio of Timestamp does not improve on request sequences satisfying λ-locality, i.e., it remains 2-competitive. The same holds for algorithm COMB, i.e., it remains 1.6-competitive. However, for the algorithm BIT, the competitive ratio improves to $\min\{1.75, \frac{2+\lambda}{1+\lambda}\}$.

5 Concluding Remarks

The standard model for the list update problem has been found wanting for certain applications and alternative models have been proposed. These models are not yet fully-studied, and some aspects of them remain to be settled. In order to obtain meaningful results for new models, one might require alternative analysis methods which act as substitutes for competitive analysis. These methods can also be applied for comparing deterministic online algorithms with randomized algorithms. As discussed earlier, such comparison is not valid under competitive analysis. There are also open questions regarding the list update problem with locality of reference, e.g., whether the access graph model can be applied for the list update problem, and in case it can, how one can devise a reasonable online algorithm which brings the graph structure into account. (Recall that such an algorithm exists for paging.)

References

1. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM J. Comput. 27, 682–693 (1998)
2. Albers, S.: Online algorithms: a survey. Math. Program. 97(1-2), 3–26 (2003)
3. Albers, S., Favrholdt, L.M., Giel, O.: On paging with locality of reference. J. Comput. Systems Sci. 70(2), 145–175 (2005)
4. Albers, S., Lauer, S.: On list update with locality of reference. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 96–107. Springer, Heidelberg (2008)
5. Albers, S., Mitzenmacher, M.: Average case analyses of list update algorithms, with applications to data compression. Algorithmica 21(3), 312–329 (1998)
6. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Inform. Process. Lett. 56, 135–139 (1995)
7. Albers, S., Westbrook, J.: Self-organizing data structures. In: Fiat, A. (ed.) Online Algorithms 1996. LNCS, vol. 1442, pp. 13–51. Springer, Heidelberg (1998)
8. Ambühl, C.: Offline list update is NP-hard. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 42–51. Springer, Heidelberg (2000)
9. Ambühl, C., Gärtner, B., von Stengel, B.: Optimal projective algorithms for the list update problem. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 305–316. Springer, Heidelberg (2000)
10. Ambühl, C., Gärtner, B., von Stengel, B.: A new lower bound for the list update problem in the partial cost model. Theoret. Comput. Sci. 268, 3–16 (2001)
11. Ambühl, C., Gärtner, B., von Stengel, B.: Optimal projective algorithms for the list update problem. CoRR, abs/1002.2440 (2010)
12. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: Proc. 18th Symp. on Discrete Algorithms (SODA), pp. 229–237 (2007)
13. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: List update with locality of reference. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 399–410. Springer, Heidelberg (2008)
14. Angelopoulos, S., Schweitzer, P.: Paging and list update under bijective analysis. In: Proc. 20th Symp. on Discrete Algorithms (SODA), pp. 1136–1145 (2009)
15. Bachrach, R., El-Yaniv, R.: Online list accessing algorithms and their applications: Recent empirical evidence. In: Proc. 8th Symp. on Discrete Algorithms (SODA), pp. 53–62 (1997)
16. Bachrach, R., El-Yaniv, R., Reinstadtler, M.: On the competitive theory and practice of online list accessing algorithms. Algorithmica 32(2), 201–245 (2002)
17. Becchetti, L.: Modeling locality: A probabilistic analysis of LRU and FWF. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 98–109. Springer, Heidelberg (2004)

18. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica 11, 73–91 (1994)
19. Ben-David, S., Borodin, A., Karp, R.M., Tardos, G., Wigderson, A.: On the power of randomization in on-line algorithms. Algorithmica 11, 2–14 (1994)
20. Bentley, J.L., Sleator, D., Tarjan, R.E., Wei, V.K.: A locally adaptive data compression scheme. Commun. ACM 29, 320–330 (1986)
21. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
22. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. J. Comput. Systems Sci. 50, 244–258 (1995)
23. Boyar, J., Favrholdt, L.M.: The relative worst order ratio for online algorithms. ACM Trans. Algorithms 3(2) (2007)
24. Boyar, J., Favrholdt, L.M., Larsen, K.S.: The relative worst-order ratio applied to paging. J. Comput. Systems Sci. 73(5), 818–843 (2007)
25. Boyar, J., Gupta, S., Larsen, K.S.: Access graphs results for LRU versus FIFO under relative worst order analysis. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 328–339. Springer, Heidelberg (2012)
26. Boyar, J., Medvedev, P.: The relative worst order ratio applied to seat reservation. ACM Trans. Algorithms 4(4) (2008)
27. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. Technical Report 124, DEC SRC (1994)
28. Chrobak, M., Noga, J.: LRU is better than FIFO. Algorithmica 23(2), 180–185 (1999)
29. Denning, P.J.: The working set model for program behaviour. Commun. ACM 11(5), 323–333 (1968)
30. Denning, P.J.: Working sets past and present. IEEE Trans. Softw. Eng. 6, 64–84 (1980)
31. Dorrigiv, R., López-Ortiz, A.: A survey of performance measures for on-line algorithms. SIGACT News 36, 67–81 (2005)
32. Dorrigiv, R., López-Ortiz, A., Munro, J.I.: An application of self-organizing data structures to compression. In: Vahrenhold, J. (ed.) SEA 2009. LNCS, vol. 5526, pp. 137–148. Springer, Heidelberg (2009)
33. Ehmsen, M.R., Kohrt, J.S., Larsen, K.S.: List factoring and relative worst order analysis. Algorithmica 66(2), 287–309 (2013)
34. El-Yaniv, R.: There are infinitely many competitive-optimal online list accessing algorithms. Manuscript (1996)
35. Epstein, L., Favrholdt, L.M., Kohrt, J.S.: Separating online scheduling algorithms with the relative worst order ratio. J. Comb. Optim. 12(4), 363–386 (2006)
36. Epstein, L., Favrholdt, L.M., Kohrt, J.S.: Comparing online algorithms for bin packing problems. J. Sched. 15(1), 13–21 (2012)
37. Golynski, A., López-Ortiz, A.: Optimal strategies for the list update problem under the MRM alternative cost model. Inform. Process. Lett. 112(6), 218–222 (2012)
38. Hagerup, T.: Online and offline access to short lists. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 691–702. Springer, Heidelberg (2007)
39. Hester, J.H., Hirschberg, D.S.: Self-organizing linear search. ACM Computing Surveys 17, 295–312 (1985)
40. Irani, S.: Two results on the list update problem. Inform. Process. Lett. 38, 301–306 (1991)
41. Irani, S., Karlin, A.R., Phillips, S.: Strongly competitive algorithms for paging with locality of reference. SIAM J. Comput. 25, 477–497 (1996)

42. Irani, S., Reingold, N., Sleator, D., Westbrook, J.: Randomized competitive algorithms for the list update problem. In: Proc. 2nd Symp. on Discrete Algorithms (SODA), pp. 251–260 (1991)
43. Kamali, S., Ladra, S., López-Ortiz, A., Seco, D.: Context-based algorithms for the list-update problem under alternative cost models. In: Proc. Data Compression Conf., (DCC) (to appear, 2013)
44. Karlin, A., Phillips, S., Raghavan, P.: Markov paging. In: Proc. 33rd Symp. on Foundations of Computer Science (FOCS), pp. 208–217 (1992)
45. Manasse, M., McGeoch, L.A., Sleator, D.: Competitive algorithms for online problems. In: Proc. 20th Symp. on Theory of Computing (STOC), pp. 322–333 (1988)
46. Martínez, C., Roura, S.: On the competitiveness of the move-to-front rule. Theoret. Comput. Sci. 242(1-2), 313–325 (2000)
47. McCabe, J.: On serial files with relocatable records. Oper. Res. 12, 609–618 (1965)
48. Mohanty, R., Narayanaswamy, N.S.: Online algorithms for self-organizing sequential search - a survey. Elect. Coll. on Comput. Complexity 16, 97 (2009)
49. Munro, J.I.: On the competitiveness of linear search. In: Paterson, M. (ed.) ESA 2000. LNCS, vol. 1879, pp. 338–345. Springer, Heidelberg (2000)
50. Reingold, N., Westbrook, J.: Randomized algorithms for the list update problem. Technical Report YALEU/DCS/TR-804, Yale University (1990)
51. Reingold, N., Westbrook, J.: Off-line algorithms for the list update problem. Inform. Process. Lett. 60(2), 75–80 (1996)
52. Reingold, N., Westbrook, J., Sleator, D.D.: Randomized competitive algorithms for the list update problem. Algorithmica 11, 15–32 (1994)
53. Rivest, R.: On self-organizing sequential search heuristics. Commun. ACM 19, 63–67 (1976)
54. Schulz, F.: Two new families of list update algorithms. In: Chwa, K.-Y., Ibarra, O.H. (eds.) ISAAC 1998. LNCS, vol. 1533, pp. 99–108. Springer, Heidelberg (1998)
55. Sleator, D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM 28, 202–208 (1985)
56. Teia, B.: A lower bound for randomized list update algorithms. Inform. Process. Lett. 47, 5–9 (1993)
57. Torng, E.: A unified analysis of paging and caching. Algorithmica 20(2), 175–200 (1998)