Adaptive Analysis of On-line Algorithms

Reza Dorrigiv [*] Alejandro López-Ortiz[*]

Abstract

On-line algorithms are usually analyzed using competitive analysis, in which the performance of on-line algorithm on a sequence is normalized by the performance of the optimal off-line algorithm on that sequence. In this paper we introduce adaptive/cooperative analysis as an alternative general framework for the analysis of on-line algorithms. This model gives promising results when applied to two well known on-line problems, paging and list update. The idea is to normalize the performance of an on-line algorithm by a measure other than the performance of the off-line optimal algorithm OPT. We show that in many instances the perform of OPT on a sequence is a coarse approximation of the difficulty or complexity of a given input. Using a finer, more natural measure we can separate paging and list update algorithms which were otherwise undistinguishable under the classical model. This createas a performance hierarchy of algorithms which better reflects the intuitive relative strengths between them. Lastly, we show that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the adaptive case. This confirms that the ability of the on-line adaptive algorithm to ignore pathological worst cases can lead to algorithms that are more efficient in practice.

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

Dagstuhl Seminar Proceedings 06421
Robot Navigation
http://drops.dagstuhl.de/opus/volltexte/2007/869

1 Introduction

There has been extensive research on the analysis of on-line algorithms. The competitive ratio, first introduced formally by Sleator and Tarjan [ST85], has served as a practical measure for the study and classification of on-line algorithms. An algorithm (assuming a cost-minimization problem) is said to be α-competitive if the cost of serving any specific request sequence never exceeds α times the optimal cost (up to some additive constant) of an optimal *off-line* algorithm which knows the entire sequence. The competitive ratio has been applied to a variety of on-line problems and settings: is relatively simple measure to apply yet powerful enough to quantify, to a large extent, the performance of many an on-line algorithm.

Paging and List update are two important and well-studied on-line problems. They were among the first problems to be analyzed using the competitive ratio. A paging algorithm mediates between a slower and a faster memory. Assuming a cache of size k, it decides which k memory pages to keep in the cache without the benefit of knowing in advance the *sequence* of upcoming page requests. After receiving the i^{th} page request the on-line algorithm must decide which page to evict, in the event the request results in a fault and the cache is full. The objective is to design efficient on-line algorithms in the sense that on a given request sequence the total cost, namely the total number of faults, is kept low. Three well known paging algorithms are *Least-Recently-Used* (LRU), *First-In-First-Out* (FIFO), and *Flush-When-Full* (FWF). On a fault, if the cache is full, LRU evicts the page that is least recently requested, FIFO evicts the page that is first brought to the cache, and FWF empties the cache. All these paging algorithms have competitive ratio k, which is the best among all deterministic on-line paging algorithms [BEY98].

In the list update or the list accessing problem, we have an unsorted list of m items. The input is a sequence of n requests that should be served in an on-line manner. Let $\mathcal{A}$ be an arbitrary on-line list update algorithm. To serve a request to an item x, $\mathcal{A}$ should linearly search the list until it finds x. If x is i^{th} item in the list, $\mathcal{A}$ incurs cost i to access x. Immediately after accessing x, $\mathcal{A}$ 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*. The idea is to use free and paid exchanges to minimize the overall cost of serving a sequence. This is called the *standard cost model*. Three well known deterministic on-line algorithms are *Move-To-Front* (MTF), *Transpose*, and *Frequency-Count* (FC). MTF moves the requested item to the front of the list and Transpose exchanges the requested item with the item that immediately precedes it. FC maintains a frequency count for each item, updates this count after each access, and makes necessary moves so that the list always contains items in non-increasing order of frequency count. In their seminal paper on competitive analysis, Sleator and Tarjan showed that MTF is 2-competitive, while Transpose and FC do not have constant competitive ratios [ST85].

Shortcomings of competitive analysis It has been observed by numerous researchers [BDB94, BIRS95b, KP00, You94, BF03, PS06] that in certain settings the competitive ratio produces results that are too pessimistic or otherwise found wanting. For example, in the case of paging experimental studies show that LRU has a performance ratio at most four times the optimal off-line [You94, SA88], as opposed to the competitive ratio k. Furthermore, it has been empirically well established that LRU (and/or variants thereof) are, in practice, preferable paging strategies to all other known paging algorithms [SGG02]. This is in contrast to competitive analysis in which the competitive ratio of LRU is the same as FWF and worse than some randomized algorithms. Competitive analysis of list update algorithms has similar drawbacks. While algorithms can generally be more easily distinguished that in the paging case, the experimental study of list update algorithms by Bachrach and El-Yaniv suggests that the relative performance hierarchy as computed by the competitive ratio does not correspond to the observed relative performance of the algorithms in practice [BEY97]. As well, the validity of standard cost model has been debated. Let $(a_1, a_2, \ldots, a_m)$ be the list currently maintained by an algorithm $\mathcal{A}$. Martínez and Roura argued that in a realistic setting a complete rearrangement of all items of the list before a_i would require time proportional to i, while $\mathcal{A}$ incurs a cost proportional to i^2 in the standard cost model to do this [MR00]. Munro argued that, for example, in the case of an access to the last item in a list the true cost of reversing the entire list in an array or a linear link list is $O(m)$, while it costs about $m^2/2$ in the standard cost model. They introduced a new model, which we term the modified cost model, in which the cost of accessing the i^{th} item of the list plus the cost of reorganizing the first i items is linear in i. In this setting, every on-line algorithm has amortized cost of $\theta(m)$ per access for some arbitrary long sequences, while there are some off-line algorithms with amortized cost of $\theta(\log m)$ on every sequence. Therefore no on-line list update algorithm has constant competitive ratio in the modified cost model.

A careful study of the competitive ratio reveals the nature of the shortcomings. Chiefly among them are its focus on worst case behaviour and indirect comparison of online algorithms via an off-line optimal algorithm. In the former case, this might lead, as observed above, to the competitive ratio declaring two wildly differing algorithms "equal" if they happen to err in the same way in the worst possible input, even though in most other inputs one is superior to the other (e.g. LRU versus FWF). In the latter case, the indirect comparison to an off-line optimal can introduce spurious artifacts due to the comparison of two objects of different types, namely an online and an off-line algorithm.[1] Such anomalies have lead to the introduction of many alternatives to competitive analysis of on-line algorithms (see [DLO05] for a comprehensive survey). In this paper we introduce adaptive analysis to on-line algorithms and overcome many of these problems.

Adaptive analysis Standard algorithm analysis expresses performance in terms of the input size. Adaptive analysis takes into account the difficulty of input instances as well. This means that an algorithm has good performance according to adaptive analysis if it performs well on "easy" instances and not too poorly on "difficult" ones. We define adaptive performance of an algorithm by normalizing its traditional performance by the difficulty of input. The two main challenges of adaptive analysis are to find a realistic difficulty measure for input instances and to propose algorithms that perform well under such a measure. Observe that the competitive ratio can be seen as a special case of adaptive analysis, namely the case where the measure of difficulty is the performance of the off-line OPT. Adaptive analysis brings to on-line algorithms the ability to use a finer measure of difficulty. For each problem, we can choose the measure that best reflects the difficulty of the input. It is unlikely that the off-line OPT is such a measure for all cases.

Cooperative Analysis The idea behind *cooperative analysis* is to give more weight to "well-behaved" input sequences. Informally, an on-line algorithm has good *cooperative* ratio if it performs well on good sequences and not too poorly on bad sequences. For example, input sequences for the paging problem have *locality of reference* in practice, therefore one possibility is to relate goodness of sequences to their amount of locality. Assuming there is a "badness" value for each input sequence, then an algorithm for a cost-minimization problem is said to have cooperative ratio α if the cost of serving any specific request sequence never exceeds α times the badness of that sequence. Note that if we consider the difficulty of the input as a particular form of badness, then adaptive analysis is a particular type of cooperative analysis. Another feature of the cooperative

[1] To illustrate, consider a consumer wishing to purchase a mountain bike. There are two choices which the user evaluates indirectly by comparing them to an "optimal" racing bike. While in general good racing bikes and mountain bikes have common characteristics, such a comparison would award no points for shock absorbers. Similarly, lightness, which is essential in a racing bike is secondary to sturdiness in the case of the mountain bike, and so on and so forth.

ratio is that in certain online problems, the competitive ratio measure might force the algorithm to make a move that is suboptimal in most cases except for a pathological worst case scenario. If the application is such that these pathological cases are agreed to be of lesser importance, then the online strategy can perform somewhat more poorly in these and make the choice that is best for the common case. Observe that the input is no longer assumed to be adversarially constructed. This better reflects the case of paging, in which programmers, compilers and optimized virtual machines (such as JVMs) go to great lengths to maintain and increase locality of reference in the code. The same observation has been made in scenarios such as online robot exploration and network packet switching, in which a robot vacuuming a room or a router serving a packet sequence need only concentrate in well behaved cases. A vacuuming robot need not efficiently vacuum a maze, neither does the router have to keep up with denial-of-service floods. Indeed in the latter case the router might actively choose to drop packets from a DoS stream [DLO06].

Our results We propose cooperative analysis as a new framework for the analysis of on-line algorithms and apply it to paging and list update problems. For paging, we suggest two cooperative measures, the phase-cooperative ratio and the locality-cooperative ratio. We show that phase-cooperative ratio gives results similar to the competitive ratio, while locality-cooperative ratio leads to better separation. We obtain tight bounds on the locality-cooperative ratio of several well known paging algorithms and show that LRU is the unique optimum among them. Then we propose a cooperative measure for the list update problem that is based on the locality of reference. We obtain bounds on the performance of well known on-line algorithms and prove the superiority of MTF. Our results apply to both the standard and modified cost models. We also apply our measures to randomized paging and list update algorithms and show that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the adaptive case.

2 Cooperative Analysis of Paging Algorithms

In this section we consider several badness measures for the cooperative ratio of a specific problem. The *fault rate* of a paging algorithm on a sequence is the number of faults that it incurs on that sequence divided by the length of the sequence. Therefore we can think of the fault rate as a cooperative ratio that considers length of a sequence as its badness. However, observe that we can have sequences of different badness among sequences of the same length and therefore the fault rate is not an ideal measure. Next we describe two more elaborate measures of badness.

2.1 Phase-cooperative ratio

An input sequence can be decomposed into phases. Each phase starts right after the last request of the previous phase and contains a maximal subset of requests that has at most k distinct pages. Let $D(\sigma)$ denote the number of phases in the decomposition of a sequence σ.

Definition 1 *We say that an on-line paging algorithm $\mathcal{A}$ has* phase-cooperative ratio α *if there is a constant β so that for every sequence σ: $\mathcal{A}(\sigma) \leq \alpha \times D(\sigma) + \beta$. We define phase-cooperative ratio of $\mathcal{A}$, $PCR(\mathcal{A})$, as the smallest number α so that $\mathcal{A}$ has phase-cooperative ratio α.*

First we define the standard paging algorithms. On a fault, *Last-In-First-Out* (LIFO) evicts the page that is most recently brought to the cache, and *Least-Frequently-Used* (LFU) evicts the page that has been requested the least since entering the cache. LFU and LIFO do not have a constant competitive ratio [BEY98]. A paging algorithm is called *conservative* if it incurs at most k faults on any sequence that contains at most k distinct pages. A marking algorithm $\mathcal{A}$ works in phases: all the pages in the cache are unmarked at the beginning of each phase. We mark any page just after the first request to it. When an eviction is necessary, $\mathcal{A}$ should evict an unmarked page. LRU and FIFO are conservative algorithms, while LRU and FWF are marking algorithms.

Theorem 1 *Every deterministic on-line paging algorithm has PCR at least k. All marking and conservative algorithms achieve the optimal PCR of k. LFU and LIFO do not have constant PCR.*

Proof: Let $\mathcal{A}$ be an arbitrary deterministic paging algorithm. Consider an adversary that maintains $k+1$ pages and at each time requests the page that is not in $\mathcal{A}$'s cache. $\mathcal{A}$ incurs a fault on each request of the corresponding sequence. Since the length of each phase is at least k, we get the desired lower bound. Let σ be a sequence and consider an arbitrary phase φ in its decomposition. Let $\mathcal{A}$ be a marking algorithm and $\mathcal{B}$ be a conservative algorithm. φ is also a phase of $\mathcal{A}$, therefore $\mathcal{A}$ incurs at most k faults in φ. B also incurs at most k faults in φ, because φ contains k distinct pages. Thus, $\mathcal{A}$ and $\mathcal{B}$ have PCR at most k.

For LIFO, consider the sequence $\sigma = p_1 p_2 \dots p_k p_{k+1} \{p_k p_{k+1}\}^n$ for an arbitrary integer n. LIFO incurs a fault on all requests of σ, while we have $D(\sigma) = 2$. Therefore LIFO does not have constant PCR. For LFU, consider the sequence $\sigma = p_1^n p_2^n \dots p_{k-1}^n \{p_k p_{k+1}\}^n$ for an arbitrary integer n. We have $D(\sigma) = 2$ and LFU incurs a fault on all last $2n$ requests. Since we can select an arbitrarily large n, LFU does not have a constant phase-cooperative ratio. □

2.2 Locality-cooperative ratio

It has been long well established that input sequences for paging show *locality of reference* in practice. This means that when a page is requested it is more likely to be requested in the near future. One apparent reason for the shortcomings of competitive analysis of paging algorithms is that it does not incorporate locality of references assumptions. Several models have been suggested for paging with locality of reference (e.g. [BIRS95a, IKP96, Tor98, AFG05, PS06, ADLO07]). In our case we need to relate badness of input sequences to their amount of locality. Unfortunately, none of the above models give a quantitative measure of locality.

Using ideas from the *characteristic vector* introduced by Panagiotou and Souza [PS06], we define a quantitative measure for non-locality of paging instances. Consider an input sequence σ. We call a request "non-local" if it is the first request to a page or at least k distinct pages have been requested since the previous request to this page in σ. The non-locality of σ, $\overline{\ell(\sigma)}$, is defined as the number of non-local requests in it. If a sequence has high locality of reference, there are not many distinct pages between two consecutive requests to a page. Therefore there are not many non-local requests and the sequence has small non-locality.

Definition 2 *We say that an on-line paging algorithm $\mathcal{A}$ has locality-cooperative ratio α if there is a constant β so that for every sequence σ: $\mathcal{A}(\sigma) \le \alpha \times \overline{\ell(\sigma)} + \beta$. We define locality-cooperative ratio of $\mathcal{A}$, $LCR(\mathcal{A})$, as the smallest number α so that $\mathcal{A}$ has locality-cooperative ratio α.*

First we show that LRU is an optimal algorithm according to locality-cooperative ratio.

Theorem 2 *LCR(LRU)=1.*

Proof: LRU always maintains in its cache the last k distinct pages that are requested. Therefore a request is a fault for LRU if and only if it is a non-local request. Thus we have $LRU(\sigma) = \overline{\ell(\sigma)}$, which implies LCR(LRU)=1. □

Lemma 1 *For any on-line paging algorithm $\mathcal{A}$, $LCR(\mathcal{A}) \geq 1$.*

Proof: Consider a sequence σ of length n obtained by requesting an item that is not $\mathcal{A}$'s cache at each time. We have $\mathcal{A}(\sigma) = n$. On the other hand, each sequence of length n has non-locality at most n. Therefore $\mathcal{A}(\sigma)/\overline{\ell(\sigma)} \geq n/n = 1$. □

The following lemma shows that marking algorithms are a reasonable choice in general, even if not always optimal.

Lemma 2 *Let $\mathcal{A}$ be a conservative or marking algorithm. We have $LCR(\mathcal{A}) \leq k$.*

Proof: Let σ be an arbitrary sequence and let φ be an arbitrary phase of the decomposition of σ. We know that $\mathcal{A}$ incurs at most k faults on φ. We claim that the first request of φ is always non-local. If this is the first phase, then this is the first request to a page and is non-local by definition. Otherwise, it should be different from k distinct pages that are requested in the previous phase. Therefore it is not requested in the previous phase and at least k distinct pages are requested since the last request to this page. Thus we have at most k faults and at least one non-local request in each phase and this proves the desired upper bound. □

Other well known algorithms are not optimal under the locality-cooperative ratio.

Lemma 3 *$LCR(FIFO) = k$.*

Proof: $LCR(FIFO) \leq k$ follows from Lemma 2. For $LCR(FIFO) \geq k$ consider an arbitrary long sequence σ that consists of $k+1$ pages. σ starts with $\sigma_0 = p_1 p_2 \ldots p_k p_1 p_2 \ldots p_{k-1} p_{k+1} p_1 p_2 \ldots p_{k-1}$. After this initial subsequence, σ consists of several blocks. Each block starts right after the previous block and contains $2k-1$ requests to k distinct pages. Let p be the page that is not in the cache at the beginning of a block B, q be the page that is requested just before B, and P be the set of $k-1$ pages that are requested in the previous block and are different from q. B starts by an arbitrary permutation π of P, then has a request to page p, and finally ends by another copy of π. It is easy to verify that FIFO incurs a fault on the last k requests of each block while only the middle request of every block is non-local. Therefore $LCR(FIFO) \geq k$. □

We can obtain a similar lower bound for FWF by considering the sequence obtained by sufficient repetition of pattern $p_1 p_2 \ldots p_k p_{k+1} p_k p_{k-1} \ldots p_2$.

Lemma 4 *$LCR(FWF) = k$.*

Lemma 5 *LFU and LIFO do not have constant LCR.*

Proof: Consider the sequence $\sigma = p_1^n p_2^n \ldots p_{k-1}^n \{p_k p_{k+1}\}^n$ for some arbitrary integer n. LFU incurs a fault on the last $2n$ requests of σ. Only the first request to a page is non-local in σ and we have $\overline{\ell(\sigma)} = k+1$. Since n can be selected arbitrary larger than k, LFU does not have constant LCR. For LIFO, consider the sequence $p_1 p_2 \ldots p_k p_{k+1} \{p_k p_{k+1}\}^n$ for some arbitrary integer n. LIFO incurs a fault on all requests of σ, while we have $\overline{\ell(\sigma)} = k+1$. n can be arbitrary large and therefore

LIFO does not have constant LCR. □

LRU-2 is another paging algorithm proposed by O'Neil et al. for database disk buffering [OOW93]. On a fault, LRU-2 evicts the page whose second to the last request is least recent. If there are pages in the cache that have been requested only once so far, LRU-2 evicts the least recently used among them. O'Neil et al. provided experimental results supporting that LRU-2 performs better than LRU in database systems. Recently, Boyar et al. theoretically analyzed LRU-2 using the competitive ratio and the relative worst order ratio [BEL06]. The relative worst order ratio [BF03, BFL05] allows for the direct comparison of on-line algorithms. Informally, for a given request sequence the measure considers the worst-case ordering (permutation) of the sequence, for each of the two algorithms, and compares their behaviour on these orderings. Boyar et al. proved that LRU-2 has competitive ratio $2k$, which is worse than FWF. Using the relative worst order ratio, they showed that LRU-2 and LRU are asymptotically comparable in LRU-2's favor.

In contrast, in a forthcoming paper in SODA'07, Angelopoulos et al. proved that LRU is the unique optimal paging algorithm under the locality of reference assumptions [ADLO07]. Therefore when we have locality of reference, their results suggest that LRU performs better than LRU-2.

In what follows, we show that LRU-2 has locality-cooperative ratio k. Therefore although LRU-2 is worse than LRU, it is not worse than FWF. This refinement of the competitive ratio using the cooperative ratio is consistent with the results of [ADLO07] which incorporates locality of reference, but inconsistent with the results of [BEL06] which do not consider locality of reference.

Theorem 3 *LCR(LRU-2)* $= k$.

Proof: [Lower bound] Let σ be the sequence obtained by n repetitions of the block $b = p_1p_2 \dots p_{k-1}p_kp_kp_{k-1} \dots p_1p_{k+1}p_{k+1}$ for some arbitrary integer n. The first block of σ contains $k+1$ non-local requests. In each subsequent block, only two requests are non-local, namely the first request to p_k and the first request to p_{k+1}. Consider a page p_i for $1 \leq i \leq k-1$ and a block b_j for $2 \leq j \leq n$. There are at most $k-1$ distinct pages between the first request to p_i in b_j and the previous request to p_i (which is in the previous block), since p_k is not requested in this period. Thus the first request to p_i in b_j is not non-local. Also p_{k+1} is not requested between the two requests to p_i in b_j. Therefore the second request to p_i in b_j is not non-local either. The first request to p_k and the first request to p_{k+1} in b_j are non-local. Thus we have $\overline{\ell(\sigma)} = k+1+2(n-1)$. LRU-2 incurs $k+1$ faults in the first block and evicts p_1 on the first access to p_{k+1}. At the beginning of each subsequent block, p_1 is missing from the cache. Then for $1 \leq i \leq k-1$, LRU-2 incurs a fault on p_i and evicts p_{i+1}. On the first request to p_k, LRU-2 incurs a fault and evicts p_{k-1}. It has a hit on the second request to p_k. Then it faults on p_{k-1} and evicts p_{k-2}, faults on p_{k-2} and evicts $p_{k-3}, \dots$, faults on p_2 and evicts p_1, faults on p_1 and evicts p_{k+1}, faults on p_{k+1} and evicts p_1. Finally it has a hit on the last request to p_{k+1}. Thus it incurs $2k$ faults in each block other than the first one and we have LRU-2$(\sigma) = k+1+2k(n-1)$. Therefore

$$\frac{\text{LRU-2}(\sigma)}{\overline{\ell(\sigma)}} = \frac{k+1+2k(n-1)}{k+1+2(n-1)}$$

As n grows, this ratio becomes arbitrary close to k and we have LCR(LRU-2) $\geq k$.

[Upper bound] Let σ be an arbitrary sequence of page requests. Partition σ into a set of consecutive blocks so that each block consists of a maximal sequence that contains exactly one non-local request. Note that each block starts with a non-local request and all other requests of the block are local. We prove that LRU-2 incurs at most k faults in each block. Let $B_1, B_2, \dots, B_m$ be the blocks of

σ. B_1 contains requests to one page and LRU-2 incurs one fault on it. Consider an arbitrary block B_i for $i > 1$, let p be the first request of B_i, and let p_1, p_2, ..., p_{k-1} be the $k-1$ most recently used pages before the block B_i in this order. We have $p \notin P = \{p_1, p_2, \ldots, p_{k-1}\}$, because p is a non-local request. We claim that each request of B_i is either to p or to a page of P. Assume for the sake of contradiction that B_i contains a request to a page $q \notin \{p\} \cup P$ and consider the first request to q in B_i. All pages $p, p_1, p_2, \ldots, p_{k-1}$ are requested since the previous request to q. Therefore at least k distinct pages are requested since the last request to q and q is non-local. This contradicts the definition of a block. Therefore B_i contains at most k distinct pages.

We claim that LRU-2 incurs at most one fault on every page q in phase B_i. Assume that this is not true and LRU-2 incurs two faults on a page q in B_i. Therefore q is evicted after the first request to it in B_i. Assume that this eviction happened on a fault on a page r and consider the pages that are in LRU-2's cache just before that request. Since $r \in \{p\} \cup P$ is not in the cache and $|\{p\} \cup P| = k$, there is a page $s \notin \{p\} \cup P$ in the cache. The last request to s is before the last request to p_{q-1} before the block B_i, while the second last request to q is after this request. Therefore LRU-2 does not evict q on this fault, which is a contradiction. Thus, LRU-2 contains at most k distinct pages in each block and incurs at most one fault on each page. Hence, LRU-2(σ) $\leq km$, and LRU-2(σ) / $\overline{\ell(\sigma)} \leq km / m = k$. □

We can extend the definition of locality-cooperative ratio to randomized paging algorithms by considering their expected cost. A randomized paging algorithm $\mathcal{A}$ has locality-cooperative ratio α if there is a constant β so that the expected cost of $\mathcal{A}$ on each sequence σ, denoted by $E(\mathcal{A}(\sigma))$, is at most $\alpha \times \overline{\ell(\sigma)} + \beta$. While no deterministic on-line paging algorithm can have competitive ratio better than k, there are randomized algorithms with better competitive ratio. The algorithm MARK, introduced by Fiat et al. [FKL+91], is $2H_k$-competitive, where H_k is the k^{th} harmonic number. MARK also reles on phases as defined above. On a fault, MARK evicts a page chosen uniformly at random from among the unmarked pages. Let σ be a sequence and $\varphi_1, \varphi_2, \ldots, \varphi_m$ be its phases. A page requested in phase φ_i is called *clean* if it was not requested in phase φ_{i-1} and *stale* otherwise. Let c_i be the number of clean pages requested in phase φ_i. Fiat et al. proved that the expected number of faults MARK incurs on phase φ_i is $c_i(H_k - H_{c_i} + 1)$.

Theorem 4 *$LCR(MARK) = H_k$.*

Proof: [Lower bound] Consider the sequence $\sigma = \{p_1 p_2 \ldots p_k p_{k+1} p_k p_{k-1} \ldots p_2\}^n$ for some integer n. σ has $2n$ phases, each odd phase has the form $p_1 p_2 \ldots p_k$ and each even phase has the form $p_{k+1} p_k \ldots p_2$. Also each phase has only one clean page, namely its first request. Therefore we have $c_i = 1$ for $1 \leq i \leq 2n$ and the expected number of faults MARK incurs on each phase is $1 \times (H_k - H_1 + 1) = H_k$. Thus $E(MARK(\sigma)) = 2nH_k$. Only the first request of each phase is non-local and we have $\overline{\ell(\sigma)} = 2n$. Hence $E(MARK(\sigma)) / \overline{\ell(\sigma)} = 2nH_k / 2n = H_k$.

[Upper bound] Consider an arbitrary sequence σ and let $\varphi_1, \varphi_2, \ldots, \varphi_m$ be its phases. Suppose that the i^{th} phase has c_i clean pages. Therefore the expected cost of MARK on σ is $\sum_{i=1}^{n} c_i(H_k - H_{c_i} + 1) \leq \sum_{i=1}^{n} c_i H_k$. The first request to a clean page in a phase is non-local because it is not among the k distinct pages that are requested in the previous phase. Therefore we have $\overline{\ell(\sigma)} \geq \sum_{i=1}^{n} c_i$. We have

$$\frac{E(MARK(\sigma))}{\overline{\ell(\sigma)}} \leq \frac{\sum_{i=1}^{n} c_i H_k}{\sum_{i=1}^{n} c_i} = \frac{H_k \sum_{i=1}^{n} c_i}{\sum_{i=1}^{n} c_i} = H_k.$$

Since this holds for every sequence σ, we have $LCR(MARK) \leq H_k$. □

3 Cooperative Analysis of List Acess Algorithms

In this section we apply cooperative analysis to the list update problem. Our results hold for both standard and modified cost models. For the sake of simplicity, in this paper we only consider the *static* list update problem. This means that we only have accesses and do not have any insert or delete operations. In particular, we have a set $S = \{a_1, a_2, \ldots, a_m\}$ of m items which are initially organized as a list $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$. Results of this paper can be easily extended to the dynamic version of the problem.

We first propose a notion of badness for the list update problem. Several authors have pointed out that input sequences of list update algorithms in practice show locality of reference [HH85, Sch98, BEY98] and indeed on-line list update algorithms try to take advantage of this property [HH85, RWS94]. Therefore we can consider locality as a possible definition of goodness. For a sequence σ of length n, we define $d_\sigma[i]$ for $1 \leq i \leq n$ as either 0 if this is the first request to item $\sigma[i]$, or otherwise, the number of distinct items that are requested since the last request to $\sigma[i]$ (including $\sigma[i]$). Now we define $\overline{\ell(\sigma)}$, the non-locality of a sequences σ, as $\overline{\ell(\sigma)} = \sum_{1 \leq i \leq n} d_\sigma[i]$. We also slightly modify the cost model: We do not charge algorithms for their first access to an item. This causes only a constant change in the total cost. Now we are ready to define *locality-cooperative ratio.*

Definition 3 *We say that an on-line list update algorithm $\mathcal{A}$ has locality-cooperative ratio α if there is a constant β so that for every sequence σ, $\mathcal{A}(\sigma) \leq \alpha \times \overline{\ell(\sigma)} + \beta$. We define locality-cooperative ratio of $\mathcal{A}$, $LCR(\mathcal{A})$, as the smallest number α so that $\mathcal{A}$ has locality-cooperative ratio α.*

Next we study the performance of on-line list update algorithms under this new measure. Observe that if an on-line algorithm does not use paid exchanges (in the standard cost model), then it incurs the same cost on each sequence in both models. Almost all well known on-line list update algorithms, and in particular all algorithms we consider in this paper, do not use paid exchanges. Therefore all our results apply to both models. Note that the performance of off-line algorithms changes dramatically in the modified model and that is why no on-line algorithm has constant competitive ratio in this model. However, we do not compare on-line algorithms to off-line algorithms in our measure.

Theorem 5 *For any on-line list update algorithm $\mathcal{A}$, $1 \leq LCR(\mathcal{A}) \leq m$.*

Proof: [Upper bound] Consider an arbitrary sequence σ of length n. Since the maximum cost that $\mathcal{A}$ incurs on a request is m, we have $\mathcal{A}(\sigma) \leq n \times m$. We have $d_\sigma[i] \geq 1$ for at least $n - m$ values of i (at most m values can be 0). Thus $\overline{\ell(\sigma)} \geq n - m$. Therefore $\frac{\mathcal{A}(\sigma)}{\overline{\ell(\sigma)}} \leq \frac{n \times m}{n - m}$. The right hand side of this inequality can become arbitrary close to m by selecting a large enough n. Therefore we have $LCR(\mathcal{A}) \leq m$.

[Lower bound] Consider a sequence σ of length n obtained by requesting the item that is in the last position of list maintained by $\mathcal{A}$ at each time. We have $\mathcal{A}(\sigma) \geq (n - m) \times m$. Also we have $d_\sigma[i] \leq m$ for $1 \leq i \leq n$, because we have at most m distinct items. Therefore $\overline{\ell(\sigma)} \leq n \times m$, and $\frac{\mathcal{A}(\sigma)}{\overline{\ell(\sigma)}} \geq \frac{(n-m) \times m}{n \times m}$. Since n can be arbitrarily larger than m, we get $LCR(\mathcal{A}) \geq 1$. □

The following lemma shows that MTF is an optimal algorithm according to locality-cooperative ratio.

Lemma 6 $LCR(MTF) = 1$.

Proof: The cost of MTF on the i^{th} request of σ is $d_\sigma[i]$. Therefore $MTF(\sigma) = \sum_{1 \leq i \leq n} d_\sigma[i] = \overline{\ell(\sigma)}$ and $LCR(MTF) = 1$. □

The following lemmas show that other well known list update algorithms do not have the optimal locality-cooperative ratio.

Lemma 7 $LCR(Transpose) \geq m/2$.

Proof: Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$ be the initial list. Consider a sequence σ of length n obtained by several repetitions of pattern $a_m a_{m-1}$. We have $Transpose(\sigma) = (n-2) \times m$. Also we have $d_\sigma[i] = 0$ for $1 \leq i \leq 2$ and $d_\sigma[i] = 2$ for $2 < i \leq n$. Therefore $\overline{\ell(\sigma)} = \sum_{i=3}^{n} 2 = (n-2) \times 2$, and $\frac{Transpose(\sigma)}{\overline{\ell(\sigma)}} = \frac{(n-2)\times m}{(n-2)\times 2} = m/2$. Since σ can be arbitrarily long, we get $LCR(Transpose) \geq m/2$. □

Lemma 8 $LCR(FC) \geq \frac{m+1}{2} \approx m/2$.

Proof: Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$ be the initial list and n be an arbitrary integer. Consider the following sequence: $\sigma = a_1^n a_2^n a_3^n \ldots a_m^n$. On serving σ, FC does not change the order of items in its list and incurs cost $\sum_{i=1}^{l} (n-1) \times i = \frac{(n-1)m(m+1)}{2}$. We have $\overline{\ell(\sigma)} = (n-1) \times 1 + (n-1) \times 1 + \cdots + (n-1) \times 1 = (n-1) \times m$. Therefore $\frac{FC(\sigma)}{\overline{\ell(\sigma)}} = \frac{(n-1)m(m+1)/2}{(n-1)m} = \frac{m+1}{2}$. Hence $LCR(FC) \geq \frac{m+1}{2}$. □

Albers introduced the algorithm *Timestamp* (TS) and showed that it has competitive ratio 2 in the standard cost model [Alb98]. After accessing an item a, TS inserts a in front of the first item b that is before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorganize the list.

Lemma 9 $LCR(TS) \geq \frac{2m}{m+1} \approx 2$.

Proof: Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$ be the initial list and n be an arbitrary integer. Consider the sequence σ obtained by the repetition of the block $a_m^2 a_{m-1}^2 \ldots a_1^2$ n times. Let B be an arbitrary block of σ. Each item a is accessed twice in B. TS does not move a after its first access in B, because each item has been accessed twice since the last access to a. After the second access, TS moves the item to the front of the list. Therefore each access is to the last item of the list and TS incurs a cost of m on each access. Considering the zero cost of first access to an item, we have $TS(\sigma) = m \times m + (n-1) \times 2m \times m = m^2 + 2(n-1)m^2$. Next we compute $\overline{\ell(\sigma)}$. The first and second access to a in block B contributes m and 1 to $\overline{\ell(\sigma)}$, respectively. Considering the special case of the first block, we have $\overline{\ell(\sigma)} = m + (n-1) \times m(m+1)$. Therefore $\frac{TS(\sigma)}{\overline{\ell(\sigma)}} = \frac{m^2+2(n-1)m^2}{m+(n-1)\times m(m+1)}$, which becomes arbitrarily close to $\frac{2m}{m+1}$ as n grows. □

Observe that the adaptive measure by virtue of its finer partition of the input space resulted in the separation of several of these strategies which are not separable under the classical model. This introduces a hierarchy of algorithms better reflecting the relative strengths of the strategies considered above. We can also extend the definition of the locality-cooperative ratio to randomized list update algorithms by considering their expected cost. A randomized list update algorithm $\mathcal{A}$ has locality-cooperative ratio α if there is a constant β so that the expected cost of $\mathcal{A}$ on each sequence σ, denoted by $E(\mathcal{A}(\sigma))$, is at most $\alpha \times \overline{\ell(\sigma)} + \beta$.

In the next theorem we show that, surprisingly and quite remarkably, certain randomized algorithms which are superior to MTF in the standard model are not so in the adaptive case. Observe that in the competitive ratio model a deterministic algorithm must serve a pathological, rare worst case even if at the expense of a more common but not critical case, while a randomized algorithm can hedge between the two cases, hence in the classical model the randomized algorithm is superior to the deterministic one. In contrast, in the adaptive model the rare worst case has a larger badness measure if it is pathological, leading to a larger denominator. Hence such a cases can safely be ignored, with a resulting overall increase in the measured quality of the algorithm. The algorithm *Bit*, introduced by Reingold and Westbrook [RW90], is a simple randomized algorithm that achieves competitive ratio 1.75 in the standard cost model, thus beating any deterministic algorithm. Bit considers a bit $b(a)$ for each item a and initializes these bits uniformly and independently at random. Upon an access to a, it first complement $b(a)$, then if $b(a) = 0$ it moves a to the front, otherwise it does nothing.

Theorem 6 $LCR(Bit) \geq \frac{3m+1}{2m+2} \approx 3/2$.

Proof: Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$ be the initial list and n be an arbitrary integer. Consider the sequence $\sigma = \{a_m^2 a_{m-1}^2 \ldots a_1^2\}^n$. Let σ_i and σ_{i+1} be two consecutive accesses to a. After two consecutive accesses to each item, it will be moved to the front of the list with probability 1. Therefore a is in the last position of the list maintained by Bit at the time of request σ_i and Bit incurs cost m on this request. After this request, Bit moves a to the front of the list if and only if $b(a)$ is initialized to 1. Since $b(a)$ is initialized uniformly and independently at random, this will happen with probability 1/2. Therefore the expected cost of Bit on σ_{i+1} is $\frac{1}{2}(m+1)$ and the expected cost of Bit on σ is $m(\frac{m+1}{2}) + (n-1) \times m(m + \frac{m+1}{2})$. We have $\overline{\ell(\sigma)} = m + (n-1) \times m(m+1)$. Therefore $\frac{Bit(\sigma)}{\overline{\ell(\sigma)}} = \frac{m(\frac{m+1}{2})+(n-1)\times m(m+\frac{m+1}{2})}{m+(n-1)\times m(m+1)}$, which becomes arbitrary close to $\frac{3m+1}{2m+2}$ as n grows. □

4 Conclusions

We propose cooperative analysis as a new framework for the analysis of on-line algorithms and apply it to paging and list update problems. This model gives promising results when applied to two well known on-line problems, paging and list update. The plurality of results shows that the new model is effective in that we can readily analyze well known strategies. Using a finer, more natural measure we separated paging and list update algorithms which were otherwise undistinguishable under the classical model. We showed that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the adaptive case. This confirms that the ability of the on-line adaptive algorithm to ignore pathological worst cases can lead to algorithms that are more efficient in practice. We obtained a hierarchy of strategies.

References

[ADLO07] S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. On the separation and equivalence of paging strategies. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*, 2007. To appear.

[AFG05] S. Albers, L. M. Favrholdt, and O. Giel. On paging with locality of reference. *Journal of Computer and System Sciences*, 70(2):145–175, 2005.

[Alb98] S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, June 1998.

[BDB94] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11:73–91, 1994.

[BEL06] J. Boyar, M. R. Ehmsen, and K. S. Larsen. Theoretical evidence for the superiority of LRU-2 over LRU for the paging problem. In *Proceedings of the 4th Workshop on Approximation and Online Algorithms (WAOA '06)*, 2006. to appear.

[BEY97] R. Bachrach and R. El-Yaniv. Online list accessing algorithms and their applications: Recent empirical evidence. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, pages 53–62, 1997.

[BEY98] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BF03] J. Boyar and L. M. Favrholdt. The relative worst order ratio for on-line algorithms. In *Proceedings of the 5th Italian Conference on Algorithms and Complexity*, 2003.

[BFL05] J. Boyar, L. M. Favrholdt, and K. S. Larsen. The relative worst order ratio applied to paging. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA '05)*, pages 718–727, 2005.

[BIRS95a] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50:244–258, 1995.

[BIRS95b] A. Borodin, S. Irani, P. Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50:244–258, 1995.

[DLO05] R. Dorrigiv and A. López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 36(3):67–81, September 2005.

[DLO06] R. Dorrigiv and A. López-Ortiz. Cooperative ratio for online packet switching, 2006. manuscript.

[FKL$^+$91] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[HH85] J. H. Hester and D. S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17(3):295, September 1985.

[IKP96] S. Irani, A. R. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing*, 25:477–497, 1996.

[KP00] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30:300–317, 2000.

[MR00] C. Martínez and S. Roura. On the competitiveness of the move-to-front rule. *Theoretical Computer Science*, 242(1–2):313–325, July 2000.

[OOW93] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 297–306, 1993.

[PS06] K. Panagiotou and A. Souza. On adequate performance measures for paging. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*, pages 487–496, 2006.

[RW90] N. Reingold and J. Westbrook. Randomized algorithms for the list update problem. Technical Report YALEU/DCS/TR-804, Department of Computer Science, Yale University, June 1990.

[RWS94] N. Reingold, J. Westbrook, and D. D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11:15–32, 1994.

[SA88] R. L. Sites and A. Agarwal. Multiprocessor cache analysis using ATUM. In *Proceedings of the fifteenth Annual International Symposium on Computer Architecture (ISCA '88)*, pages 186–195, 1988.

[Sch98] F. Schulz. Two new families of list update algorithms. In *Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC '98)*, volume 1533 of *Lecture Notes in Computer Science*, pages 99–108. Springer, 1998.

[SGG02] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. John Wiley & Sons, 2002.

[ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[Tor98] E. Torng. A unified analysis of paging and caching. *Algorithmica*, 20(2):175–200, 1998.

[You94] N. E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, June 1994.