Alcom-FT Technical Report Series
ALCOMFT-TR-03-32 (revised February 4, 2004)

The Relative Worst Order Ratio Applied to Paging*

Joan Boyar, Lene M. Favrholdt**, and Kim S. Larsen

Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark,
{joan,lenem,kslarsen}@imada.sdu.dk

Abstract. The relative worst order ratio, a new measure for the quality of on-line algorithms, was recently defined and applied to two bin packing problems. Here, we extend the definition and apply it to the paging problem. Work in progress by other researchers shows that the measure gives interesting results and new separations for bin coloring and scheduling problems as well.

Using the relative worst order ratio, we obtain the following results: We devise a new deterministic paging algorithm, Retrospective-LRU, and show that it performs better than LRU. This is supported by experimental results, but contrasts with the competitive ratio. All deterministic marking algorithms have the same competitive ratio, but here we find that LRU is better than FWF. Furthermore, LRU is better than PERM_π. No deterministic marking algorithm can be significantly better than LRU, but the randomized algorithm MARK is better than LRU. Finally, look-ahead is shown to be a significant advantage, in contrast to the competitive ratio, which does not reflect that look-ahead can be helpful.

1 Introduction

The standard measure for the quality of on-line algorithms is the competitive ratio [15, 26, 18], which is, roughly speaking, the worst-case ratio, over all possible input sequences, of the on-line performance to the optimal off-line performance. The definition of the competitive ratio has been taken rather directly from that of the approximation ratio. This seems natural in that on-line algorithms can be viewed as a special class of approximation algorithms. However, for approximation algorithms, the comparison to an optimal off-line algorithm, OPT, is very natural, since the approximation algorithm is compared to another algorithm of the same general type, just with more computing power, while for on-line algorithms, the comparison to OPT is to a different type of algorithm.

Although the competitive ratio has been an extremely useful notion, in many cases, and particularly for the paging problem, it has appeared inadequate at differentiating between on-line algorithms. When this is the goal, performing a direct comparison between the algorithms, instead of involving an intermediate comparison to OPT, seems the obvious choice. A direct comparison on exactly the same sequences will produce the result that many algorithms are not comparable, because one algorithm does well on one type of ordering, while the other does well on another type. With the relative worst order ratio, on-line algorithms are compared directly to each other on their respective worst orderings of multi-sets. In this way, the relative worst order ratio [7] combines some of the desirable properties of the Max/Max ratio [4] and the random order ratio [20].

The Max/Max Ratio The Max/Max ratio [4] allows direct comparison of two on-line algorithms for an optimization problem, without the intermediate comparison to OPT. Rather than comparing two algorithms on the same sequence, they are compared on their respective worst-case sequences of the same length. The Max/Max Ratio applies only when the length of an input sequence yields a bound on the profit/cost of an optimal solution.

* Supported in part by the Danish Natural Science Research Council (SNF) and in part by the Future and Emerging Technologies program of the EU under contract number IST-1999-14186 (ALCOM-FT).
** Part of this work was done while working at the Department of Computer Science, University of Copenhagen, Denmark. Supported in part by the Villum Kann Rasmussen Fund.

The Random Order Ratio The random order ratio [20] gives the possibility of considering some randomness of the request sequences without specifying a complete probability distribution. For an on-line algorithm $\mathbb{A}$, the random order ratio is the worst-case ratio, over all multi-sets of requests, of the expected performance of $\mathbb{A}$ on a random permutation of the multi-set, compared with an optimal solution. If, for all possible multi-sets of requests, any ordering of these requests is equally likely, this ratio gives a meaningful worst-case measure of how well an algorithm can do. Unfortunately, the random order ratio seems to be difficult to compute.

The Relative Worst Order Ratio With the relative worst order ratio, one considers the worst-case performance over all permutations instead of the average-case performance as with the random order ratio. Thus, when comparing two on-line algorithms, one considers a worst-case multi-set and takes the ratio of how the two algorithms perform on their worst orderings of that multi-set. Note that the two algorithms may have different "worst orderings" for the same multi-set.

The Paging Problem We consider the well studied paging problem. The input sequence consists of requests for pages in a slow memory, which contains N pages. There is a fast memory, called the cache, which has space for $k < N$ pages. A request for a page currently in cache is a *hit*, while a request for a page not in cache is a *page fault*. When a page fault occurs, the requested page must be brought into the cache. If the cache already contains k pages when this happens, at least one of them must be evicted. A paging algorithm decides which page to evict on a fault. This decision must usually be made on-line, i.e., without any knowledge about future requests. The goal is to minimize the number of faults.

Paging Algorithms Two major classes of deterministic algorithms for the paging problem are conservative algorithms [30] and marking algorithms [6].

A paging algorithm $\mathbb{A}$ is called *conservative*, if no request sequence has a consecutive subsequence with requests to at most k distinct pages causing $\mathbb{A}$ to fault more than k times. The algorithms, Least-Recently-Used (LRU) and First-In/First-Out (FIFO) are examples of conservative algorithms. On a page fault, LRU evicts the least recently used page in cache and FIFO evicts the page which has been in cache longest.

Marking algorithms work in phases. Each time a page is requested, this page is marked. When a page must be evicted, one of the unmarked pages is chosen, if one exists. Otherwise all marks are erased, and the requested page is marked. This request starts a new phase. Note that LRU is a marking algorithm, whereas FIFO is not. Another example of a marking algorithm is Flush-When-Full (FWF), the algorithm which evicts all pages in cache at the end of each phase. The randomized marking algorithm MARK chooses the unmarked page to be evicted uniformly at random.

We also study $\text{MARK}^{\text{LIFO}}$, LIFO, and $\textsc{Perm}_\pi$ [5] defined in Sections 6 and 7.

Previous Results All conservative and marking algorithms have competitive ratio k [29, 27] and this is optimal among deterministic algorithms [26]. However, in practice, these algorithms do not all have the same performance: LRU is better than FIFO and much better than FWF [30]. Moreover, results from [13] suggest there may be algorithms that perform even better than LRU. In contrast to deterministic algorithms, MARK [12] has been shown to have a competitive ratio of $2H_k - 1$[1], where H_k is the kth harmonic number, i.e., $H_k = \sum_{i=1}^{k} \frac{1}{i} \approx \ln k$. Other randomized algorithms have been shown to have the optimal competitive ratio for randomized algorithms of H_k [24, 1].

Look-Ahead. Look-ahead, where the algorithm deciding which page to evict is allowed to see the next ℓ page requests before making that decision, is a model which intuitively lies between on-line and off-line. It is well known that look-ahead cannot reduce the competitive ratio of any algorithm, but clearly it can be useful when it can be implemented.

Previously, alternative definitions of look-ahead have led to results showing that look-ahead helps. In each case, the algorithm is allowed to see a sequence of future requests satisfying some property. Young [29] proposed *resource-bounded look-ahead*, where the sequence is a maximal sequence of future requests for which it would incur ℓ page faults, Albers [2] proposed *strong look-ahead*, where the sequence of future requests contains ℓ distinct pages different from the current request, and Breslauer [9] proposed *natural look-ahead*, where the sequence of future requests contains ℓ pages not currently in cache. Here we retain the original definition, so the algorithm is only allowed to see the next ℓ pages, regardless of what they are.

The Max/Max Ratio [4] has been somewhat successfully applied to the standard definition of look-ahead, showing that a greedy strategy achieves a Max/Max ratio of $\frac{N-1}{\ell}$ for $N-k<\ell\leq n-1$ (recall that N is the size of the slow memory). *Comparative analysis* [22] is more successful, showing that look-ahead gives a result which is a factor $\min\{k,\ell+1\}$ better than without look-ahead. This is the same result we obtain with the relative worst-order ratio.

Other Measures. In addition to the random order ratio, the Max/Max Ratio, and comparative analysis, there have been several previous proposals for alternatives to or refinements of competitive analysis. Among these are access graphs [6], Markov paging [19], the Max-model and Average-Model [3], diffuse adversaries [22], extra resource analysis [17, 26], the accommodating function [8], and statistical adversaries [25]. Most of these techniques have been applied to only a few closely related problems. So far, the techniques which have been applied to a broader range of problems, the accommodating function, for instance, have given new separation results for only a few closely related problems.

In contrast, the relative worst order ratio has already been applied quite successfully to two very different problem types: bin-packing [7] and now paging, here. For Classical Bin-Packing, Worst-Fit is better than Next-Fit according to the relative worst order ratio, even though they have the same competitive ratio [16]. Thus, the advantage of keeping all bins open, instead of just one, is reflected by the relative worst order ratio. For Dual Bin Packing, the relative worst order ratio shows that First-Fit is better than Worst-Fit, while the competitive ratio indicates the opposite [8].

Other New Results on the Relative Worst Order Ratio The wide applicability of the relative worst order ratio has been confirmed by other new results. Recently, various researchers have applied the relative worst order ratio to other problems and obtained separations not given by the competitive ratio, but consistent with intuition and/or practice.

One simple example is given in [11]. For the problem of minimizing makespan on two related machines with speed ratio s, the optimal competitive ratio of $\frac{s+1}{s}$ for $s\geq\Phi\approx 1.618$ is obtained both by the post-greedy algorithm, which schedules each job on the machine where it will finish earliest, and by the algorithm which simply schedules all jobs on the fast machine. In contrast, the relative worst order ratio shows that the post-greedy algorithm is better. A similar result is obtained for the problem of minimizing makespan on $m\geq 2$ identical machines.

The relative worst order ratio was also found by [21] to give the intuitively correct result for the bin coloring problem, where the competitive ratio gives the opposite result [23].

New Results on Paging First, we propose a new algorithm, Retrospective-LRU (RLRU), which is a variation on LRU that takes into account which pages would be in the cache of the optimal off-line algorithm, LFD, if it were given the subsequence of page requests seen so far. We show that, according to the relative worst order ratio, RLRU is better than LRU. This is interesting, since it contrasts with results on the competitive ratio and with results in [3] where a new model of locality of reference is studied.

It is easily shown that RLRU does not belong to either of the common classes of algorithms, conservative and marking algorithms which all have the optimal competitive ratio k. In fact, the competitive ratio of RLRU is $k+1$ and thus slightly worse than that of LRU. Initial testing of RLRU indicates that it may perform better than LRU in practice.

Analyzing paging algorithms with the relative worst order ratio, we obtain far more detailed information than with competitive analysis. With the relative worst order ratio, LRU is better than FWF, so not all marking algorithms are equivalent, but no marking algorithm is significantly better than LRU. All conservative algorithms are equivalent, so LRU and FIFO have the same performance, but LRU is better than the k-competitive algorithm PERM_π. MARK is better than LRU, which is consistent with competitive analysis.

Look-ahead is shown to help significantly with respect to the relative worst order ratio. Compared to the competitive ratio which does not reflect that look-ahead can be of any use, this is a very nice property of the relative worst order ratio.

A new phenomenon with respect to the relative worst order ratio is observed: in [7], the pairs of algorithms investigated were either *comparable* or *incomparable*, but here some are found to be *weakly comparable*, i.e., while one algorithm performs marginally better than the other on some multi-sets, the other algorithm performs significantly better on other multi-sets. Furthermore, algorithms can be *asymptotically comparable*, which for the paging problem means that, for arbitrarily large cache sizes, the pair of algorithms are "arbitrarily close to being comparable". This is defined more formally in Section 2.

2 The Relative Worst Order Ratio

In this section, we define the relative worst order ratio and the notion of two algorithms being comparable (Definition 2) as in [7]. This is the most important definition, but the new notions of being weakly comparable and asymptotically comparable (defined in Definitions 4 and 5) give the possibility of adding more detail to the description of the relation between two algorithms.

The definition of the relative worst order ratio uses $\mathbb{A}_W(I)$, the performance of an on-line algorithm $\mathbb{A}$ on the "worst ordering" of the multi-set I of requests, formally defined in the following way.

Definition 1. *Consider an on-line optimization problem P and let I be any request sequence of length n. If σ is a permutation on n elements, then $\sigma(I)$ denotes I permuted by σ. Let $\mathbb{A}$ be any algorithm for P.*

If P is a maximization *problem, $\mathbb{A}(I)$ is the value of running $\mathbb{A}$ on I, and $\mathbb{A}_W(I) = \min_\sigma \mathbb{A}(\sigma(I))$.*

If P is a minimization *problem, $\mathbb{A}(I)$ is a cost, and $\mathbb{A}_W(I) = \max_\sigma \mathbb{A}(\sigma(I))$.*

Definition 2. *Let $S_1(c)$ and $S_2(c)$ be statements about algorithms $\mathbb{A}$ and $\mathbb{B}$ defined in the following way.*

$S_1(c)$: There exists a constant b such that $\mathbb{A}_W(I) \leq c \cdot \mathbb{B}_W(I) + b$ for all I.

$S_2(c)$: There exists a constant b such that $\mathbb{A}_W(I) \geq c \cdot \mathbb{B}_W(I) - b$ for all I.

The relative worst order ratio *$WR_{\mathbb{A},\mathbb{B}}$ of on-line algorithm $\mathbb{A}$ to algorithm $\mathbb{B}$ is defined if $S_1(1)$ or $S_2(1)$ holds. In this case, $\mathbb{A}$ and $\mathbb{B}$ are said to be* comparable.

If $S_1(1)$ holds, then $WR_{\mathbb{A},\mathbb{B}} = \sup\{r \mid S_2(r)\}$.

If $S_2(1)$ holds, then $WR_{\mathbb{A},\mathbb{B}} = \inf\{r \mid S_1(r)\}$.

Note that if $S_1(1)$ holds, the supremum involves S_2 rather than S_1, and vice versa. A ratio of 1 means that the two algorithms perform identically with respect to this quality measure; the further away from 1, the greater the difference in performance. The ratio may be greater than or less than one, depending on whether the problem is a minimization or a maximization problem and on which of the two algorithms is better. These possibilities are illustrated in Table 1.

	minimization	maximization
$\mathbb{A}$ better than $\mathbb{B}$	< 1	> 1
$\mathbb{B}$ better than $\mathbb{A}$	> 1	< 1

Table 1. Ratio values for minimization and maximization problems

Although one of the goals in defining the relative worst order ratio was to avoid the intermediate comparison of any on-line algorithm to the optimal off-line algorithm OPT, it is still possible to compare on-line algorithms to OPT. In this case, the measure is called the *worst order ratio*.

It is easily shown [7] that the relative worst order ratio is a transitive measure, i.e., for any three algorithms $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$, $\mathrm{WR}_{\mathbb{A},\mathbb{B}} \leq 1$ and $\mathrm{WR}_{\mathbb{B},\mathbb{C}} \leq 1$ implies $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \leq 1$.

Even if a pair of algorithms are not comparable, there may be something interesting to say about their relative performance. Therefore, we introduce the notion of relatedness that applies to most pairs of algorithms.

Definition 3. *Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for an on-line optimization problem P, and let S_1 and S_2 be defined as in Definition 2.*

Assume first that P is a minimization *problem. If there exists a positive constant c such that $S_1(c)$ is true, let $c_{\mathbb{A},\mathbb{B}} = \inf \{r \mid S_1(r)\}$. Otherwise, $c_{\mathbb{A},\mathbb{B}}$ is undefined.*

- *If $c_{\mathbb{A},\mathbb{B}}$ and $c_{\mathbb{B},\mathbb{A}}$ are both defined, $\mathbb{A}$ and $\mathbb{B}$ are* $(c_{\mathbb{A},\mathbb{B}}, c_{\mathbb{B},\mathbb{A}})$-related.
- *If $c_{\mathbb{A},\mathbb{B}}$ is defined and $c_{\mathbb{B},\mathbb{A}}$ is undefined, $\mathbb{A}$ and $\mathbb{B}$ are* $(c_{\mathbb{A},\mathbb{B}}, \infty)$-related.
- *If $c_{\mathbb{A},\mathbb{B}}$ is undefined and $c_{\mathbb{B},\mathbb{A}}$ is defined, $\mathbb{A}$ and $\mathbb{B}$ are* $(\infty, c_{\mathbb{B},\mathbb{A}})$-related.

If P is a maximization *problem, relatedness is defined similarly: If there exists a positive constant c such that $S_2(c)$ is true, let $c_{\mathbb{A},\mathbb{B}} = \sup \{r \mid S_2(r)\}$. Otherwise, $c_{\mathbb{A},\mathbb{B}}$ is undefined.*

- *If $c_{\mathbb{A},\mathbb{B}}$ and $c_{\mathbb{B},\mathbb{A}}$ are both defined, $\mathbb{A}$ and $\mathbb{B}$ are* $(c_{\mathbb{A},\mathbb{B}}, c_{\mathbb{B},\mathbb{A}})$-related.
- *If $c_{\mathbb{A},\mathbb{B}}$ is defined and $c_{\mathbb{B},\mathbb{A}}$ is undefined, $\mathbb{A}$ and $\mathbb{B}$ are* $(c_{\mathbb{A},\mathbb{B}}, 0)$-related.
- *If $c_{\mathbb{A},\mathbb{B}}$ is undefined and $c_{\mathbb{B},\mathbb{A}}$ is defined, $\mathbb{A}$ and $\mathbb{B}$ are* $(0, c_{\mathbb{B},\mathbb{A}})$-related.

This notation can also be used for algorithms which are comparable. In this case, one of the values is the relative worst order ratio and the other is typically 1 (unless one algorithm is strictly better than the other on all multi-sets).

In Section 7, it is shown that LRU and Last-In/First-Out (LIFO) are $(\frac{k+1}{2}, \infty)$-related. With this result it seems reasonable to prefer LRU to LIFO, even though they are not comparable by Definition 2. We say, therefore, that the pair of algorithms are *weakly* comparable.

Definition 4. *Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for an on-line optimization problem P and let $c_{\mathbb{A},\mathbb{B}}$ be defined as in Definition 3. $\mathbb{A}$ and $\mathbb{B}$ are* weakly comparable *if $\mathbb{A}$ and $\mathbb{B}$ are comparable, if exactly one of $c_{\mathbb{A},\mathbb{B}}$ and $c_{\mathbb{B},\mathbb{A}}$ is defined, or if both are defined and $c_{\mathbb{A},\mathbb{B}} \notin \Theta(c_{\mathbb{B},\mathbb{A}})$. More specifically, if P is a minimization problem and $c_{\mathbb{A},\mathbb{B}} \in o(c_{\mathbb{B},\mathbb{A}})$, or if P is a maximization problem and $c_{\mathbb{A},\mathbb{B}} \in \omega(c_{\mathbb{B},\mathbb{A}})$, $\mathbb{A}$ and $\mathbb{B}$ are* weakly comparable in $\mathbb{A}$'s favor.

We conclude with a definition which is relevant for optimization problems with some limited resource, such as the size of the cache in the paging problem, the capacity of the knapsack in a knapsack problem, or the number of machines in a machine scheduling problem.

Definition 5. *A resource dependent problem is an on-line optimization problem, where each problem instance, in addition to the input data given on-line, also has a parameter k, referred to as the amount of resources, such that for each input, the optimal solution depends monotonically on k.*

Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for a resource dependent problem P and let $c_{\mathbb{A},\mathbb{B}}$ be defined as in Definition 3. For a maximization problem, $\mathbb{A}$ and $\mathbb{B}$ are asymptotically comparable *if*

$$\lim_{k\to\infty}\{c_{\mathbb{A},\mathbb{B}}\} \geq 1 \textit{ and } \lim_{k\to\infty}\{c_{\mathbb{B},\mathbb{A}}\} \leq 1,$$

and for a minimization problem, $\mathbb{A}$ and $\mathbb{B}$ are asymptotically comparable *if*

$$\lim_{k\to\infty}\{c_{\mathbb{A},\mathbb{B}}\} \leq 1 \textit{ and } \lim_{k\to\infty}\{c_{\mathbb{B},\mathbb{A}}\} \geq 1.$$

In both cases, $\mathbb{A}$ and $\mathbb{B}$ are asymptotically comparable in $\mathbb{A}$'s favor *if the first of the two inequalities is strict.*

In Section 8.1 it is shown that MARK and LRU are (a, b)-related with $a \leq 1+\frac{2}{k+1}$ and $b \geq \frac{k}{H_k}$, and hence asymptotically comparable in MARK's favor. Clearly, if a pair of algorithms are comparable, they are also asymptotically comparable.

If one algorithm is not clearly better than the other, as captured by the notions comparable, weakly comparable, and asymptotically comparable, we say the pair of algorithms are *incomparable.*

Definition 6. *Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for an on-line optimization problem. If $\mathbb{A}$ and $\mathbb{B}$ are neither weakly nor asymptotically comparable, we say that they are* incomparable.

3 A Better Algorithm than LRU

In this section, we introduce an algorithm which turns out to be better than LRU according to the relative worst order ratio. This is in contrast to the competitive ratio which says that LRU is best possible among deterministic algorithms. The algorithm, called RLRU, is defined in Figure 1. The name Retrospective-LRU comes from the algorithm's marking policy, which marks pages which OPT would have already had in cache. Since RLRU avoids evicting marked pages if it can, these pages may stay in RLRU's cache longer than they would in LRU's. Among the pages which RLRU may evict at a given time, it always chooses the least recently used. A phase of the execution starts with the removal of all marks and this occurs whenever there would otherwise be a second fault on the same page within the current phase.

The proofs which establish that RLRU is better than LRU according to the relative worst order ratio rely on certain properties of the algorithm. This means that modifications to the algorithm which do not change the basic properties on which the proofs rely will result in other algorithms which, according to the relative worst order ratio, are also better than LRU. One example of this is the test as to whether or not the current page is the same as the previous. This test could be removed and the page marked unconditionally and the proofs still hold. Another example is the decision when to end a phase. The important property is that each phase consists of requests to at least $k+1$ distinct pages and there is at most one fault on each of them. This leaves room for experimentally testing a number of variations, and it could lead to algorithms which are even better in practice than the one we have found here.

Note that RLRU is neither a conservative nor a marking algorithm. This can be seen from the sequence $\langle p_1, p_2, p_3, p_4, p_1, p_2, p_3, p_4, p_3\rangle$ for $k = 3$, where RLRU faults on every request.

Lemma 1. *For any request sequence, each complete phase defined by RLRU contains requests to at least $k+1$ distinct pages.*

Proof. Consider any phase P and the page p which starts the next phase. Page p was requested in phase P, and it must also be evicted in phase P. At that point, all other pages in the cache must have either been marked or requested since the last request to p, so every page in cache at that point has been requested in phase P. The page requested when p is evicted must be different from the k pages in cache at that point. Thus, there must be at least $k+1$ different pages requested in phase P. □

```
The first phase begins with the first request.
On request r to page p:

Update p's timestamp
if p is not in cache then
    if there is no unmarked page then
        evict the least recently used page in cache
    else
        evict the least recently used unmarked page
    if this is the second fault on p
      since the start of the current phase then
        unmark all pages
        start a new phase with r
    if p was in OPT's cache just before this request then
        mark it
else
    if p is different from the previous page then
        mark p
```

Fig. 1. Retrospective-LRU (RLRU)

Lemma 2. *For any multi-set I of page requests, $RLRU_W(I) \leq LRU_W(I)$.*

Proof. Consider a worst ordering I_{RLRU} of I with respect to RLRU. By definition, RLRU never faults twice on the same page within any single phase of I_{RLRU}.

Move the last, possibly incomplete, phase of I_{RLRU} to the beginning and call the resulting sequence I_{LRU}. Process the requests in this sequence phase by phase (the phases are the original RLRU phases), starting at the beginning. LRU faults on each distinct page in the first phase. Since there are at least $k+1$ distinct pages in each of the later phases, all of the distinct pages in a phase can be ordered so that there will be a fault by LRU on each of them. Hence, in each phase, LRU faults at least as many times on as RLRU. Hence, LRU has at least as many faults on I_{LRU} as RLRU on I_{RLRU}. □

Theorem 1. $WR_{LRU,RLRU} \geq \frac{k+1}{2}$.

Proof. Since the previous lemma shows that $\mathrm{WR}_{\mathrm{LRU,RLRU}} \geq 1$, it is sufficient to find a family of sequences I_n with $\lim_{n\to\infty} \mathrm{LRU}(I_n) = \infty$, where there exists a constant b such that for all I_n, $\mathrm{LRU}_W(I_n) \geq \frac{k+1}{2}\mathrm{RLRU}_W(I_n) - b$. Let I_n consist of n phases, where, in each phase, the first $k-1$ requests are to the $k-1$ pages $\langle p_1, p_2, ..., p_{k-1}\rangle$, always in that order, and the last two requests are to completely new pages. LRU will fault on every page, so it will fault $n(k+1)$ times. Regardless of the order this multi-set is given in, OPT will never evict the pages $\langle p_1, p_2, ..., p_{k-1}\rangle$ from cache, so RLRU will mark them the first time they are requested in each phase, if they have ever been requested before. Thus, for each of these pages, there is at most one phase where a new phase is started by faulting on it. This gives at most $k-1$ phases, each containing at most one fault on each of these $k-1$ pages. This limits the number of faults RLRU has on these $k-1$ pages to a constant (dependent on k, but not n), so RLRU faults at most $2n+c$ times for some constant c. Asymptotically, the ratio is $\frac{k+1}{2}$. □

In contrast to Theorem 1, the competitive ratio of RLRU is slightly worse than that of LRU:

Theorem 2. *The competitive ratio of RLRU is $k+1$.*

Proof. The upper bound of $k+1$ follows since each phase of the algorithm contains requests to at least $k+1$ different pages, and RLRU faults at most once on each page within a phase. If there are $s > k$ different pages in a phase, OPT must fault at least $s-k$ times in that phase. The worst ratio is obtained when there are exactly $k+1$ different pages in a phase, giving a ratio of $k+1$.

The lower bound follows from a sequence with $k+1$ distinct pages $p_1, p_2, \ldots, p_{k+1}$, where each request is to the page not in RLRU's cache. This sequence is $\langle p_1, p_2, \ldots, p_{k+1}\rangle^2$ $\langle p_k, p_1, p_2, \ldots, p_{k-1}, p_{k+1} \mid p_{k-1}, p_k, p_1, p_2, \ldots, p_{k-2}, p_{k+1} \mid p_{k-2}, p_{k-1}, p_k, p_1, p_2, \ldots, p_{k-3}, p_{k+1} \mid \cdots \mid p_1, p_2, \ldots, p_{k+1}\rangle^n$, where $\mid$ marks the beginning of a new phase. The part of the sequence which is repeated n times is called a superphase and consists of k phases, the ith phase consisting of the sequence $\langle p_{k+1-i}, \ldots, p_k, p_1, \ldots, p_{k-i}, p_{k+1}\rangle$, for $1 \le i \le k-1$, and $\langle p_1, p_2, \ldots, p_{k+1}\rangle$, for $i = k$. The optimal strategy is to evict page p_{k-1-i} in the ith phase of a superphase for $1 \le i \le k-2$, p_k for $i = k-1$, and p_{k-1} for $i = k$. Hence, an optimal off-line algorithm faults $k+1$ times on the initial $2k+1$ requests and then exactly once per phase, while RLRU faults on all $k+1$ requests of each phase. □

3.1 Implementation

RLRU can be implemented to run in time $O(\log N)$ and space $O(N)$, where N is the number of different pages which have requested. The question is whether or not these time and space bounds are good enough in practice? We believe there are at least two interesting scenarios to consider. One is the interaction between two high speed storage media, the speed of which differ by only a small multiplicative constant, such as primary versus secondary cache. Here, a paging algorithm must be very efficient, which also implies that it cannot be allowed much working space. In such a scenario, even LRU is most often too time and space consuming. Another scenario is the interaction of storage media, the speed of which differ by orders of magnitude. This could be the buffer pool versus the disk in database systems or local file caching of Internet files. In those situations, we can use substantial space, and time logarithmic in either the number of different pages or just in the cache size would both be insignificant compared with almost any small improvement in cache behavior. A similar point is made in [13]. If, in some special application, space is a problem, then it could possibly be reduced to a function of k using the techniques of [1]. In summary, a comparison between LRU and RLRU is interesting because the circumstances under which they can reasonably be applied are close to identical.

3.2 Empirical Analysis

To see if the positive theoretical results are also reflected in practice, we have investigated the behavior of LRU and RLRU on traces[1] collected from very different applications, including key words searches in text files, selections and joins in the Postgres database system, external sorting, and various kernel operations. We have used all ten data files from the site.

In Table 2, we list the results for each data file, and for cache sizes of 8, 16, 32, and 64. Each entry in the table shows first the number of page faults of LRU, then the number of faults of RLRU, and finally the percent-wise improvement of RLRU over LRU. Thus, if ℓ and r denote the number of faults by LRU and RLRU, respectively, then the improvement is computed as $100\frac{\ell - r}{\ell}$. This number is negative if LRU performs best.

Out of the 40 tests, 11 are negative. The largest negative result of -0.74% is from a short sequence and is due to a difference of only one page fault. The remaining negative results lie between zero and approximately half a per cent. There are 28 positive results. RLRU beats LRU with more than half a per cent in 18 cases, more than 1% in 10 cases, and more than 5% in 7 cases. This is illustrated in Figure 2.

3.3 The Worst Order Ratios for LRU and RLRU

When LRU and RLRU are compared to OPT using the worst order ratio, instead of the competitive ratio, one finds that they have the same ratio k, confirming the intuition that information is lost in an indirect comparison to OPT.

[1] `www.cs.wisc.edu/~cao/traces/`

Cache	File names and lengths									
	bigsort	j1	j2	j3	j4	j5	j6	pjoin	pq7	xds
Size	40167	18533	25881	38112	59744	95723	20709	41558	32989	88558
	14632	494	8233	4262	7278	25412	5100	8014	9371	10768
8	13204	491	8197	4276	7278	25385	4545	7200	9419	10724
	9.76	**0.61**	**0.44**	**-0.33**	**0.00**	**0.11**	**10.88**	**10.16**	**-0.51**	**0.41**
	12619	470	8177	4243	7201	25332	4596	7718	9277	10762
16	10736	468	8134	4255	7221	25326	4525	7003	9259	10709
	14.92	**0.43**	**0.53**	**-0.28**	**-0.28**	**0.02**	**1.54**	**9.26**	**0.19**	**0.49**
	10744	463	8138	4239	7180	25307	4516	7401	9216	10756
32	10561	425	8078	4248	7186	25303	4513	6888	9170	10697
	1.70	**8.21**	**0.74**	**-0.21**	**-0.08**	**0.02**	**0.07**	**6.93**	**0.50**	**0.55**
	10587	136	8120	4230	7135	25276	4505	6879	9185	10754
64	10402	137	8057	4239	7140	25278	4506	6838	9103	10695
	1.75	**-0.74**	**0.78**	**-0.21**	**-0.07**	**-0.01**	**-0.02**	**0.60**	**0.89**	**0.55**

Table 2. Empirical comparison of LRU and RLRU

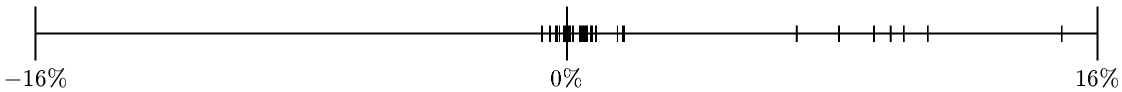

Fig. 2. Percentages with which RLRU is better than LRU

Theorem 3. $WR_{LRU} = k$.

Proof. Consider any multi-set I. Since no algorithm is better than OPT, on OPT's worst ordering of I, LRU will fault at least as many times as OPT, so it also will on its own worst ordering. The sequence consisting of n copies of $k+1$ pages repeated cyclicly is a worst ordering of the underlying multi-set for both LRU and OPT. LRU faults k times for every time that OPT faults. Since the worst order ratio cannot be larger than the competitive ratio, and LRU's competitive ratio is k, $\mathrm{WR}_{\mathrm{LRU}} = k$. □

Theorem 4. $WR_{RLRU} = k$.

Proof. Consider any multi-set I. As in the above proof, on OPT's worst ordering of I, RLRU will fault at least as many times as OPT, so it also will on its own worst ordering. By Lemma 2, for any multi-set I, $\mathrm{RLRU}_{\mathrm{W}}(I) \leq \mathrm{LRU}_{\mathrm{W}}(I)$. Thus, by the above theorem $\mathrm{WR}_{\mathrm{RLRU}} \leq k$. The sequence $\langle p_1, p_2, \ldots, p_{k+1}\rangle^2 \langle p_k, p_1, p_2, \ldots, p_{k-1}, p_{k+1} \mid p_{k-1}, p_k, p_1, p_2, \ldots, p_{k-2}, p_{k+1} \mid p_{k-2}, p_{k-1}, p_k, p_1, p_2, \ldots, p_{k-3}, p_{k+1} \mid \ldots \mid p_1, p_2, \ldots, p_{k+1}\rangle^n$, where $\mid$ marks the beginning of a new phase, will cause RLRU to fault every time. A worst ordering for OPT will repeat the $k+1$ pages in a cyclic manner and OPT will fault once on every k pages, giving the ratio k. □

4 Conservative Algorithms

Recall that a paging algorithm $\mathbb{A}$ is called conservative, if no request sequence has a consecutive subsequence with requests to at most k distinct pages causing $\mathbb{A}$ to fault more than k times. It is easy to see that both LRU and FIFO are conservative algorithms [30]: between any two faults on the same page there must be requests to at least k other pages.

We first prove a technical lemma for LRU.

Lemma 3. *For any multi-set I of page requests, there exists a worst ordering of I for LRU with all faults appearing before all hits.*

Proof. We describe how any permutation I' of I can be transformed to a permutation I_{LRU} with all hits appearing at the end of the sequence, without decreasing the number of faults LRU will have on the sequence. Let I' consist of the requests $r_1, r_2, \ldots, r_n$, in that order.

Consider the first hit r_i in I' with respect to LRU. We construct a new sequence I'' by moving r_i later in I'. Let p denote the page requested by r_i.

First, we remove r_i from the sequence. If p is evicted at some point after r_{i-1} in this shorter sequence, r_i is placed just after the first request r_j, $j > i$, causing p to be evicted (see Figure 3). Otherwise, r_i is inserted after r_n. In this case, let $j = n$.

$$\begin{aligned} I' &: r_1, \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_j, r_{j+1}, \ldots, r_n \\ I'' &: r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_j, r_i, r_{j+1}, \ldots, r_n \end{aligned}$$

Fig. 3. The two sequences I' and I'' in the case where p is evicted at r_j.

LRU maintains a queue of the pages in cache, and, on a fault, evicts the first page in the queue. Moving r_i within the sequence affects the position of p in the queue, but the mutual order of the other pages stays the same. Just before r_{i+1}, the cache contents are the same for both sequences. Therefore, for I'', the behavior of LRU is the same as for I' until p is evicted. Just after this eviction, p is requested by r_i in I''. Thus, just before r_{j+1}, the cache contents are again the same for both sequences, but for I'', p is at the end of queue. This means that all pages that are in cache just before r_{j+1}, except p, are evicted no later for I'' than for I'. The first request to p after the jth request may be a fault in I' and a hit in I''. On the other hand, r_i is a hit in I' and a fault in I''.

Let r_ℓ be the first request after r_i in I'', where p is either requested or evicted. After r_ℓ, the state of LRU is the same for both sequences.

By moving r_i, the number of faults among the first j requests is increased by at least one, and the total number of faults is not decreased. Thus, continuing in this way, we obtain I_{LRU} in a finite number of steps. □

LRU is best possible among conservative algorithms.

Lemma 4. *$WR_{LRU,\mathbb{C}} \leq 1$, for any conservative paging algorithm $\mathbb{C}$.*

Proof. By Lemma 3, we can consider a sequence I where all faults occur before all hits. Let I_1 denote the subsequence consisting of the faults. We prove by induction on the lengths of prefixes of I_1 that, on any request in I_1, any conservative algorithm $\mathbb{C}$ evicts the same page as LRU, and hence has as many faults on I as LRU.

For the base case, consider the first $k+1$ requests in the sequence. Since LRU faults on each request, these $k+1$ requests are all to different pages (ignoring the trivial case with at most k pages in I). Hence, on the $(k+1)$st request, any algorithm must evict a page. Since $\mathbb{C}$ is conservative it evicts p_1 (if it evicted some page $p_i \neq p_1$, requesting p_i after p_{k+1} would yield a sequence with a subsequence $\langle p_2, \ldots, p_{k+1}, p_i \rangle$ with requests to only k distinct pages, but with $k+1$ faults).

The induction step is similar to the base case. By the induction hypothesis, $\mathbb{C}$ has the same pages in cache as LRU. For each request r to some page p, the previous k requests were all to different pages different from p. Hence, $\mathbb{C}$ must evict the first of these k pages, as LRU does. □

In addition, LRU is a worst possible conservative algorithm.

Lemma 5. *$WR_{LRU,\mathbb{C}} \geq 1$, for any conservative paging algorithm $\mathbb{C}$.*

Proof. Consider any conservative algorithm $\mathbb{C}$ and any request sequence I. Divide I into phases, so that $\mathbb{C}$ faults exactly $k+1$ times per phase, starting the next phase with a fault (the last phase may have fewer than $k+1$ faults). Since $\mathbb{C}$ is conservative, each phase, except possibly the last, contains at least $k+1$ distinct pages. These can be reordered, phase by phase, so that LRU faults once on each distinct page within a phase. Thus, with this new ordering LRU faults at least as many times as $\mathbb{C}$ on I, except possibly in the last phase. Since $\mathbb{C}$ faults at least once in the last phase, LRU faults as many times on the new ordering of I as $\mathbb{C}$ on I, except for at most k faults. ☐

Thus, all conservative algorithms are equivalent under the relative worst order ratio:

Theorem 5. *For any pair of conservative algorithms $\mathbb{C}_1$ and $\mathbb{C}_2$, $WR_{\mathbb{C}_1,\mathbb{C}_2} = 1$.*

Proof. By Lemmas 4 and 5, $\mathrm{WR}_{\mathbb{C}_1,\mathrm{LRU}} \geq 1$ and $\mathrm{WR}_{\mathrm{LRU},\mathbb{C}_2} \geq 1$. Since the relative worst order ratio is easily seen to be a transitive measure [7], this means that $\mathrm{WR}_{\mathbb{C}_1,\mathbb{C}_2} \geq 1$. Similarly, $\mathrm{WR}_{\mathbb{C}_1,\mathrm{LRU}} \leq 1$ and $\mathrm{WR}_{\mathrm{LRU},\mathbb{C}_2} \leq 1$, which implies $\mathrm{WR}_{\mathbb{C}_1,\mathbb{C}_2} \leq 1$. ☐

In particular, since LRU and FIFO are both conservative, they are equivalent.

5 Look-Ahead

In the standard on-line model, requests arrive one by one. A model in which the algorithm is informed of the next $\ell \geq 1$ page requests before servicing the current one, is a look-ahead model. This model is in-between the standard on-line model and the off-line model.

It is well known that using standard competitive analysis one cannot show that knowing the next ℓ requests is any advantage for any fixed ℓ; for any input sequence, an adversary can "fill up" the look-ahead by repeating each request ℓ times, adding no cost to the optimal off-line solution. In contrast, results on the relative worst order ratio, indicate that look-ahead helps significantly. Here we only look at a modification of LRU, using look-ahead, though the technique can be applied to other algorithms as well.

Define LRU(ℓ) to be the algorithm which on a fault evicts the least recently used page in cache which is not among the next ℓ requests. If $\ell \geq k$, all pages in cache may be among the next ℓ requests. In this case, the page whose next request is farthest in the future is evicted. One can see that LRU(ℓ) is at least as good as LRU on any multi-set by noting that LRU(ℓ) is conservative.

Lemma 6. *LRU(ℓ) is a conservative algorithm.*

Proof. Let I be a request sequence, and assume that there is an interval, I', in I, containing only k distinct pages, on which LRU(ℓ) faults at least $k+1$ times. Then it must fault on some page, p, twice in I'. Between these two faults, say at request r, page p must be evicted. At this point, it is the least recently used page which is not among the next ℓ. Clearly the second request causing a fault on p must be beyond these next ℓ. So between these two faults on p, the other $k-1$ pages in cache, when request r occurs, must all have been requested. In addition, the request r cannot be for p or any of the other pages in cache at that time. Thus, there must be at least $k+1$ distinct pages in I', giving a contradiction, so LRU(ℓ) is conservative. ☐

Observe that Lemma 5 holds for algorithms using look-ahead, though Lemma 4 does not.

Theorem 6. $WR_{LRU,LRU(\ell)} \geq \min\{k, \ell+1\}$

Proof. Since the previous lemma combined with Lemma 5 show that $\mathrm{WR}_{\mathrm{LRU},\mathrm{LRU}(\ell)} \geq 1$, it is sufficient to find a family of sequences I_n with $\lim_{n\to\infty} \mathrm{LRU}(I_n) = \infty$, where there exists a constant b such that for all I_n, $\mathrm{LRU}_W(I_n) \geq \min\{k, \ell+1\}\mathrm{LRU}(\ell)_W(I_n) - b$. Let I_n consist of n phases, each containing the pages $\langle p_1, p_2, ..., p_k, p_{k+1}\rangle$, in that order. LRU will fault $n(k+1)$ times. However, if $\ell \leq k-1$, after the first k faults, $\mathrm{LRU}(\ell)$ never faults on any of the next ℓ pages after a fault. Thus, regardless of the order, $\mathrm{LRU}(\ell)$ faults on at most $k + \lfloor \frac{n(k+1)-k}{\ell+1} \rfloor$ pages. If $\ell \geq k$, then $\mathrm{LRU}(\ell)$ faults on at most one out of every k pages. □

6 Marking Algorithms

Recall that marking algorithms work in phases. In each phase (except possibly the last), exactly k different pages are requested, and the first page requested within a phase was not requested in the previous phase. A subsequence processed within one marking phase is called a *k-phase*. Note that the partitioning of a sequence into k-phases is independent of the particular marking algorithm.

Although all pairs of conservative algorithms are indistinguishable using the relative worst order ratio, not all pairs of marking algorithms are. In particular, LRU is better than FWF, as it is in practice, despite the fact that they have the same competitive ratio.

Lemma 7. $WR_{FWF,LRU} \geq \frac{2k}{k+1}$.

Proof. Consider any sequence I. If LRU faults on request r in I to page p, then p was not among the last k different pages that were requested. Thus, p could not be in FWF's cache when request r occurs and FWF will also fault. Hence, on any multi-set, FWF will fault at least as many times on its worst ordering as LRU will on its.

This shows that $\mathrm{WR}_{\mathrm{FWF},\mathrm{LRU}} \geq 1$, so it is now sufficient to find a family of sequences I_n with $\lim_{n\to\infty} \mathrm{LRU}(I_n) = \infty$, where there exists a constant b such that for all I_n, $\mathrm{FWF}_W(I_n) \geq \frac{2k}{k+1}\mathrm{LRU}_W(I_n) - b$. Let $I_n = \langle p_1, p_2, \ldots, p_k, p_{k+1}, p_k, \ldots, p_2\rangle^n$. FWF will fault on every page, so it will fault $n(2k)$ times. By Lemma 3, a worst ordering for LRU consists of some permutation of the $k+1$ pages, with p_1 and p_{k+1} at the end, repeated n times, followed by the first $k-1$ pages in the permutation repeated n times. LRU will fault $n(k+1)+k-1$ times. Asymptotically, the ratio is $\frac{2k}{k+1}$. □

FWF is worst possible among marking algorithms.

Lemma 8. *For any marking algorithm* $\mathbb{M}$, $WR_{\mathbb{M},LRU} \leq \frac{2k}{k+1}$.

Proof. For any sequence with n phases, $\mathbb{M}$ faults at most kn times. Since for any request sequence, every pair of consecutive phases contains requests to at least $k+1$ distinct pages, there is a permutation such that LRU faults at least $(k+1)\lfloor \frac{n}{2} \rfloor$ times. □

Corollary 1. $WR_{FWF,LRU} = \frac{2k}{k+1}$.

Not only is LRU better than FWF according to the relative worst order; LRU is close to being a best possible marking algorithm:

Lemma 9. *For any marking algorithm* $\mathbb{M}$ *and any multi-set* I *of page requests,*

$$LRU_W(I) \leq \frac{k+1}{k}\, \mathbb{M}_W(I).$$

Proof. Consider any multi-set I of requests. By Lemma 3, there is a worst ordering I_{LRU} such that LRU faults on each request of a prefix I_1 of I_{LRU} and on no request after I_1. Partition I_1 into consecutive subsequences, each consisting of exactly $k+1$ requests (the last subsequence may contain fewer). Since

LRU faults on all requests in I_1, each subsequence must contain $k+1$ distinct pages. Hence, for each subsequence with pages $p_1, p_2, \ldots, p_{k+1}$, an adversary can create a marking phase, by choosing k of the pages $p_1, p_2, \ldots, p_{k+1}$, such that the marking algorithm faults on all k pages. This is easily seen in the following way. Pages requested within a phase stay in cache throughout the phase. Therefore, when x of the pages $p_1, p_2, \ldots, p_{k+1}$ have been requested, the remaining $k+1-x$ pages cannot all be in the cache. $\square$

Corollary 2. *For any marking algorithm* $\mathbb{M}$ *with* $WR_{LRU,\mathbb{M}}$ *defined,*

$$WR_{LRU,\mathbb{M}} \leq \frac{k+1}{k}.$$

Let $\text{MARK}^{\mathbb{A}}$ denote the marking algorithm using the eviction strategy $\mathbb{A}$ on the unmarked pages in cache, and let LIFO be the algorithm which evicts the page most recently brought into cache.

The following lemma shows that Lemma 9 is tight. It does not settle, however, whether Corollary 2 is tight.

Lemma 10. *There exists a family of multi-sets* I_n *of page requests such that*

$$LRU_W(I_n) = \frac{k+1}{k}\, MARK_W^{LIFO}(I_n),$$

and $\lim_{n\to\infty} LRU_W(I_n) = \infty$*.*

Proof. First note that, for any multi-set I of page requests, there is a worst ordering of I with respect to $\text{MARK}^{\text{LIFO}}$ such that all faults precede all hits. This is because the eviction strategy considers only the order in which the pages where brought into the cache. Therefore, moving a hit to the end of the sequence does not affect which of the other requests will be faults, and hence can only increase the number of faults.

Consider the sequence $\langle p_1, p_2, \ldots, p_{k+1}\rangle^n$ and a permutation for which $\text{MARK}^{\text{LIFO}}$ faults on a longest possible prefix and then has no more faults. Since there are only $k+1$ pages, once the first k requests are given, the remaining part of the prefix is fixed. Hence, there is essentially only one such permutation, namely $\langle p_1, p_2, \ldots, p_k, p_{k+1}, p_k, \ldots, p_2\rangle^{n/2}\langle p_1, p_{k+1}\rangle^{n/2}$. $\text{MARK}^{\text{LIFO}}$ does not fault on the last $n-1$ requests, whereas LRU will fault on all requests in the permutation $\langle p_1, p_2, \ldots, p_{k+1}\rangle^n$. $\square$

Despite Lemma 10, $\text{MARK}^{\text{LIFO}}$ is not better than LRU:

Lemma 11. *There exists a family of multi-sets* I_n *of page requests such that*

$$MARK_W^{LIFO}(I) \geq \frac{2k}{k+1} \cdot LRU_W(I),$$

and $\lim_{n\to\infty} MARK_W^{LIFO}(I_n) = \infty$*.*

Proof. On the sequence $I_n = \langle p_1, p_2, \ldots, p_k, p_{k+1}, p_k, \ldots, p_2\rangle^n$, which was also used in the proof of Theorem 7, $\text{MARK}^{\text{LIFO}}$ faults on every request. Note that each of the pages $p_2, \ldots, p_k$ is requested twice as often as each of p_1 and p_{k+1}. Thus, on any permutation of the sequence, LRU faults on at most half of the requests to each of the pages $p_2, \ldots, p_k$. This is because, between any two faults on a page p_i, $2 \leq i \leq k$, both p_1 and p_{k+1} must be requested. Hence, LRU faults at most $(k+1)n$ times, giving a ratio of

$$\frac{\text{MARK}_{\text{W}}^{\text{LIFO}}(I)}{\text{LRU}_{\text{W}}(I)} \geq \frac{2k}{k+1}.$$

$\square$

Corollary 3. *LRU and* $MARK^{LIFO}$ *are* $(1+\frac{1}{k}, c)$*-related with* $c \geq 2 - \frac{1}{k}$*, i.e., they are asymptotically comparable in LRU's favor.*

7 Other Algorithms

The algorithm LIFO, which evicts the page most recently brought into cache, is clearly much worse than any of the conservative or marking algorithms. This behavior is also reflected in the relative worst order ratio, since there exists a family of multi-sets where LIFO does unboundedly worse than LRU. However, there also exists a family of multi-sets, where LIFO does better than LRU by a factor of $\frac{k+1}{2}$. This factor is tight, though, so LRU can be unboundedly better than LIFO, while LIFO can be at most a factor $\frac{k+1}{2}$ better than LRU.

Theorem 7. *LRU and LIFO are $(\frac{k+1}{2}, \infty)$-related, i.e., they are weakly comparable in LRU's favor.*

Proof. Let I_n be the sequence $\langle p_1, p_2, \ldots, p_{k-1}\rangle\langle p_k, p_{k+1}\rangle^n$. LIFO faults on every request of this sequence. LRU's worst ordering is with $\langle p_k, p_{k+1}\rangle$ first and then the pages $\langle p_1, p_2, \ldots, p_k, p_{k+1}\rangle$, since then it faults $k+3$ times. Hence, for this multi-set, the ratio is unbounded in LRU's favor.

On the other hand, with the sequence $J_n = \langle p_1, p_2, \ldots, p_k, p_{k+1}\rangle^n$, LRU faults $n(k+1)$ times, so this is clearly its worst order of J_n. Regardless of the order of J_n, LIFO faults $2n+k-1$ times, since it holds $k-1$ pages in memory, never changing them. Thus, there is a ratio of $\frac{k+1}{2}$ in LIFO's favor.

Since LRU has a competitive ratio of k it cannot be more than a factor of k worse than LIFO. We can show, however, that the lower bound of $\frac{k+1}{2}$ is tight. Let I be any request sequence. By Lemma 3, there is a worst ordering I_{LRU} such that LRU faults on each request of a prefix I_1 of I_{LRU} and on no request after I_1. Consider the prefix I_1, divided into phases with $k+1$ pages per phase, except possibly fewer in the last phase. We reorder the requests within each complete phase, starting at the beginning. The contents of LIFO's cache at the beginning of a phase will be with respect to the sequence I_1 with all modifications up to that point. Since LRU faults on every page in a phase in the original I_1, the $k+1$ pages must all be different. LIFO has at most k of the $k+1$ pages from the current phase in its cache at the start of the phase. If there were only $k-1$, instead of k, we would be done, since then it must fault at least twice in this phase. Suppose p_i is the page requested in the current phase, which was not in LIFO's cache. Move the request to p_i to the beginning of the phase. This will evict a page requested later in the phase, so LIFO will fault at least twice in this phase. □

The algorithm MixPerm [5] is a mixed, randomized, memory-bounded algorithm which obtains a competitive ratio of $\frac{k+1}{2}$ when $N = k+1$. It is the uniform mixture of all the permutation algorithms Perm_π [10], which have competitive ratio k. The parameter π in the definition of Perm_π is a cyclic permutation of the N pages in slow memory. Let $\pi^{(m)}$ denote the m-fold composition of the cyclic permutation π, and let $m(i)$ be the minimum m with $\pi^{(m(i))}(i)$ currently in cache. The algorithm Perm_π is defined so that on a page fault on page i, the page $\pi^{(m(i))}(i)$ is evicted. Perm_π and LRU are equivalent according to the competitive ratio, and when $N = k+1$, they are equivalent according to the relative worst order ratio. However, for $N \geq k+2$, Perm_π performs worse than LRU according to the relative worst order ratio.[2]

Theorem 8.

$$\textit{If } N = k+1,\ WR_{LRU,\text{Perm}_\pi} = 1.$$
$$\textit{If } N \geq k+2,\ WR_{LRU,\text{Perm}_\pi} < \frac{1}{2} + \frac{k}{2N} + \frac{1}{k}.$$

Proof. We first show that $\text{WR}_{\text{LRU},\text{Perm}_\pi} \leq 1$. Consider any request sequence I. By Lemma 3, there is a worst ordering I_{LRU} such that LRU faults on each request of a prefix I_1 of I_{LRU} and on no request after I_1. Consider the prefix I_1, divided into phases with $k+1$ requests per phase, except possibly fewer

[2] This follows from the proof of Theorem 8, though not from the statement of the theorem. For simplicity, the statement of the theorem is slightly weaker than what is actually proven.

in the last phase. We reorder the requests within each complete phase, starting at the beginning, such that PERM_π will also fault on all of these requests. The contents of PERM_π's cache at the beginning of a phase will be with respect to the sequence I_1 with all modifications up to that point. Since LRU faults on every page in a phase in the original I_1, the $k+1$ pages must all be different.

For each complete phase, divide the $k+1$ pages into two sets, P containing the pages that are in cache at the beginning of the phase and Q containing those that are not. We construct the new order of the $k+1$ pages in $P \cup Q$ in $k+1$ steps. Within each of the two sets, the pages are requested in order, according to π. In each step, if one of the pages in P is not in cache anymore, this page is requested. Otherwise, the next page from Q is requested. In this way, we go through π only once, so no page is requested more than once, and since $|P \cup Q| = k+1$, in each step there will be at least one page in $P \cup Q$ which is not in cache.

We now prove that for $N = k+1$, $\text{WR}_{\text{LRU},\text{PERM}_\pi} \geq 1$. This completes the proof that, for $N = k+1$, $\text{WR}_{\text{LRU},\text{PERM}_\pi} = 1$. Consider any multi-set I with requests to at most $k+1$ pages and let I_{PERM_π} be a worst ordering of I with respect to PERM_π. Whenever PERM_π faults on a page p, it goes through π starting at p until it finds a page which is in cache and evicts this page. Thus, if I_{PERM_π} is reordered such that the pages that PERM_π faults on occur at the beginning of the sequence, in the same mutual order as in I_{PERM_π}, then LRU will also fault on each of these requests.

Now, consider the case $N \geq k+2$. To see that $\text{WR}_{\text{LRU},\text{PERM}_\pi} < \frac{1}{2} + \frac{k}{2N} + \frac{1}{k}$, assume without loss of generality that $\pi = (1, 2, ..., N)$ and consider the family of request sequences I_n consisting of n phases, each consisting of the sequence $\langle p_1, p_2, ..., p_{k+1}, p_1, p_{k+2}, p_1, p_{k+3}, ..., p_1, p_N \rangle$. We show that PERM_π faults on all requests in I_n.

Consider the first phase of I_n. We prove by induction that just before the ith request to p_1, $2 \leq i \leq N-k$, $p_i, \ldots, p_{i+k-1}$ are in the cache. For $i = 2$, this is clearly the case. Therefore, consider the ith request r_i to p_1 and assume that the cache contains the pages $p_i, \ldots, p_{i+k-1}$. Since p_1 is not in the cache, r_i will cause p_i to be evicted. After r_i, p_{k+i} is requested, and this causes p_1 to be evicted. Thus, the induction hypothesis is maintained. At the end of I, the pages $p_{N-k+1}, \ldots, p_N$ are in cache. Therefore, for the following k requests, the request to p_i will cause p_{N-k+i} to be evicted. Thus, after the first k requests of the second phase of I_n, the cache contents are the same as after the first k requests of the first phase. Since the algorithm is memoryless, this is sufficient to prove that it will behave the same during all n phases of I_n.

On the other hand, regardless of the ordering, LRU faults on at most $\lceil n\,\frac{N-1}{k} \rceil$ requests to p_1, thus faulting at most $n(N-1) + \lceil n\,\frac{N-1}{k} \rceil$ times in total. This gives a ratio of at most

$$\begin{aligned}
\frac{n\left(N-1+\frac{N-1}{k}\right)+1}{n(k+1+2(N-(k+1)))} &= \frac{N-1+\frac{N-1}{k}+\frac{1}{n}}{2N-(k+1)} \\
&\leq \frac{N-1+\frac{N}{k}}{2N-(k+1)}, \quad \text{for } n \text{ sufficiently large.} \\
&= \frac{N-\frac{k+1}{2}+\frac{k+1}{2}-1+\frac{N}{k}}{2N-(k+1)} \\
&= \frac{1}{2}+\frac{\frac{k-1}{2}+\frac{N}{k}}{2N-(k+1)} \\
&< \frac{1}{2}+\frac{k}{2(2N-(k+1))}+\frac{N}{k(2N-(k+1))} \\
&< \frac{1}{2}+\frac{k}{2N}+\frac{1}{k}.
\end{aligned}$$

□

8 Randomized Algorithms

The relative worst order ratio can also be applied to randomized algorithms. The only change to the definitions is that an algorithm's expected value/cost on a worst ordering of a sequence is used in place of the value/cost obtained by a deterministic algorithm.

Definition 7. *Consider an on-line optimization problem P and let I be any request sequence of length n. If σ is a permutation on n elements, then $\sigma(I)$ denotes I permuted by σ. Let $\mathbb{A}$ be any randomized algorithm for P.*

If P is a maximization problem, *$E[\mathbb{A}(I)]$ is the expected value of running $\mathbb{A}$ on I, and*

$$\mathbb{A}_W(I) = \min_{\sigma} E[\mathbb{A}(\sigma(I))].$$

If P is a minimization *problem, $\mathbb{A}(I)$ is a cost, and*

$$\mathbb{A}_W(I) = \max_{\sigma} E[\mathbb{A}(\sigma(I))].$$

Using the above definition, the relative worst order ratio, related, weakly comparable, asymptotically comparable, and incomparable are now defined as in the deterministic case.

8.1 MARK vs. LRU

Consider the randomized paging algorithm MARK [12]. On a fault, MARK chooses the unmarked page to be evicted uniformly at random. In this section, we show that LRU and MARK are asymptotically comparable in MARK's favor. This is consistent with the results one obtains with the competitive ratio where MARK has ratio $2H_k - 1$ [1], while LRU has ratio k.

Lemma 12. *There exists a sequence I, which is a worst ordering for both MARK and LRU, where MARK's expected number of faults is H_k per k-phase, while LRU faults k times per k-phase.*

Proof. Consider the sequence with $k+1$ pages repeated cyclicly.

Lemma 13. *For any multi-set I of page requests, there exists a worst ordering I_{MARK} of I, such that all k-phases, except possibly the last, have the following properties:*

1. *The first page requested in a k-phase did not appear in the previous k-phase.*
2. *There are exactly k requests, all different.*

Proof. Consider a worst ordering I_{MARK} of I for MARK and consider its k-phase partition. The first property follows by the definition of a k-phase partition. Within a phase, the first occurrence of each page is the only one MARK has any chance of faulting on. Thus, moving extra occurrences of a page within a phase to the end of the sequence will never decrease the probability of MARK faulting on any page. After this has been completed, each phase (except possibly the last) consists of exactly k requests, all to distinct pages. □

Theorem 9. *MARK and LRU are (a, b)-related with $a \leq 1 + \frac{2}{k+1}$ and $b \geq \frac{k}{H_k}$, i.e., they are asymptotically comparable in MARK's favor.*

Proof. For any request sequence I, consider the worst permutation I_{MARK} of I with respect to MARK, satisfying the conditions of the previous lemma. Partition the sequence I_{MARK} in subsequences each consisting of $k+1$ k-phases. Now consider one of these subsequences I' and let m_i, $2 \leq i \leq k+1$, be the number of "new" pages in the ith k-phase of I'. That is, m_i is the number of pages requested in the ith phase and not in the $i-1$st phase. Let $m = \max_{2 \leq i \leq k+1}\{m_i\}$. Since the expected number of faults in phase i is $m_i(H_k - H_{m_i} + 1)$ [12], the expected number of faults for MARK on the last k phases of I' is no more than $mk(H_k - H_m + 1)$, and clearly, the number of faults in the first phase of I' is no more than k.

By Lemma 13, we can assume that each k-phase in I' contains exactly k requests to k distinct pages. Call pages requested in every phase of I' *frequent* pages, and pages requested in at most k phases *rare* pages. There are at most $k - m$ frequent pages, since a frequent page is never among the m_i new pages in a phase. Hence, there are at least $(k+1)m$ requests to rare pages. Let $f \leq k - m$ be the number of frequent pages. No matter what was requested in the previous subsequence, we can construct an initial subsequence, on which LRU faults on every request, by using f frequent pages and $k - f + 1$ rare pages in each block of $k + 1$ pages. This gives a subsequence of length

$$\left\lfloor \frac{(k+1)m}{k-f+1} \right\rfloor (k+1) \geq \left\lfloor \frac{(k+1)m}{m+1} \right\rfloor (k+1) \geq \frac{km}{m+1}(k+1)$$

on which LRU faults on each request.

Hence, if we prove that the limit of the ratio of $mk(H_k - H_m + 1) + k$ to $\frac{mk}{m+1}(k+1)$ is no more than $1 + \frac{2}{k+1}$, the theorem will follow from Lemma 12, since the limit of $1 + \frac{2}{k+1}$ as k goes to infinity is 1.

$$\begin{aligned}
\frac{mk(H_k - H_m + 1) + k}{\frac{mk}{m+1}(k+1)} &= \frac{(m+1)(H_k - H_m + 1 + \frac{1}{m})}{k+1} = \frac{(m+1)(\sum_{i=m+1}^{k} \frac{1}{i} + 1 + \frac{1}{m})}{k+1} \\
&\leq \frac{(m+1)((k-m)\frac{1}{m+1} + 1 + \frac{1}{m})}{k+1} = \frac{k+2+\frac{1}{m}}{k+1} \leq 1 + \frac{2}{k+1}
\end{aligned}$$

□

9 Conclusion and Open Problems

This second problem area, paging, studied using the relative worst order ratio, gives even more convincing evidence than the first, bin packing, that this new performance measure could become an important tool for analyzing on-line algorithms. Comparing algorithms directly to each other, rather than doing it indirectly through a comparison to OPT, appears to give more meaningful results, both for paging and for bin packing. The results mentioned in the introduction for the bin coloring problem and two scheduling problems show more types of problems where the relative worst order ratio gives "better" results than the competitive ratio. Previous measures and models, proposed as replacements or supplements to the competitive ratio, have been more limited as to applicability, usually to very few problems. Further study is needed to determine how widely applicable the relative worst order ratio is, but paging and bin packing are very different problems. Together with the results on the bin coloring and scheduling problems, this gives a convincing basis for further investigation.

For paging, the competitive ratio of many algorithms with widely varying performance in practice is k. The relative worst order ratio is able to distinguish between some of these algorithms. Most notably, LRU is found to be better than FWF, and look-ahead is shown to help. It is also promising that this new performance measure is leading to the discovery of new algorithms. Further testing is needed to see if RLRU, or some variant of it, is better than LRU in practice.

We expect that for many other problems, the relative worst order ratio will distinguish between known algorithms which have the same competitive ratio, give the correct relationship between algorithms where the competitive ratio fails to, and lead to new algorithms which are better than algorithms which have an optimal competitive ratio.

Corollary 2 shows that no marking algorithm can be much better than LRU. It would be interesting to know if LRU is in fact the best marking algorithm according to the relative worst order ratio.

Theorem 9 says that MARK and LRU are asymptotically comparable. We conjecture that they are in fact comparable.

References

1. D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1–2):203–218, 2000.
2. S. Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18:283–305, 1997.
3. S. Albers, L. M. Favrholdt, and O. Giel. On paging with locality of reference. In *34th Annual ACM Symposium on the Theory of Computing*, pages 258–267, 2002.
4. S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994.
5. A. Borodin and R. El-Yaniv. On randomization in online computation. In *Proceedings of the IEEE Conference on Computational Complexity*, pages 226–238, 1997.
6. A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995.
7. J. Boyar and L. M. Favrholdt. The relative worst order ratio for on-line algorithms. In *5th Italian Conference on Algorithms and Complexity*, volume 2653 of *LNCS*, pages 58–69, 2003.
8. J. Boyar, K. S. Larsen, and M. N. Nielsen. The accommodating function — a generalization of the competitive ratio. *SIAM Journal of Computation*, 31(1):233–258, 2001.
9. D. Breslauer. On competitive on-line paging with lookahead. *Theoretical Computer Science*, 209(1–2):365–375, 1998.
10. M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.
11. L. Epstein, L. M. Favrholdt, and J. S. Kohrt. The relative worst order ratio applied to scheduling problems. Work in progress, 2003.
12. 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.
13. A. Fiat and Z. Rosen. Experimental studies of access graph based heuristics: Beating the LRU standard? In *8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 63 – 72, 1997.
14. J. S. Frederiksen and K. S. Larsen. On-line seat reservations via off-line seating arrangements. In *Eighth International Workshop on Algorithms and Data Structures*, volume 2748 of *LNCS*, pages 174–185. Springer-Verlag, 2003.

15. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.
16. D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.
17. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47:617–643, 2000.
18. A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.
19. A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov paging. *SIAM Journal of Computation*, 30:906–922, 2000.
20. C. Kenyon. Best-fit bin-packing with random order. In *7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 359–364, 1996.
21. J. S. Kohrt. The relative worst order ratio applied to bin coloring. Work in progress, 2003.
22. E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. In *35th Annual Symposium on Foundations of Computer Science*, pages 394–400, 1994.
23. S. O. Krumke, W. E. de Paepe, J. Rambau, and L. Stougie. Online bin coloring. In *9th Annual European Symposium on Algorithms*, volume 2161 of *LNCS*, pages 74–85, 2001.
24. L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6:816–825, 1991.
25. P. Raghavan. A statistical adversary for on-line algorithms. In D. D. Sleator L. A. McGeoch, editor, *On-Line Algorithms*, volume 7 of *Series in Discrete Mathematics and Theoretical Computer Science*, pages 79–83. American Mathematical Society, 1992.
26. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
27. E. Torng. A unified analysis of paging and caching. *Algorithmica*, 20:175–200, 1998.
28. M. J. van Kreveld and M. H. Overmars. Union-copy structures and dynamic segment trees. *Journal of the ACM*, 40(3):635–652, 1993.
29. N. Young. Competitive paging and dual-guided algorithms for weighted caching and matching (thesis). Technical Report CS-TR-348-91, Computer Science Department, Princeton University, 1991.
30. N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11:525–541, 1994.

Additional Comments on an Implementation

In addition to the administration required to evict least recently used pages, which is similar to the administration necessary for LRU, RLRU needs to be able to perform the following operations:

1. Check if it faults on a page for the second time in a phase.
2. Mark a page, and unmark all pages.
3. Check for a page in OPT's cache.

These can all be implemented in time $(\log N)$ and space $O(N)$, where N is the number of different pages which have been requested.

1. We use a balanced binary search tree over all the different pages on which RLRU has faulted during the phase.
2. Using a balanced binary search tree over all the different pages which have been requested, we mark a page by associating the current phase number with the page. Thus, by incrementing the phase number, we can unmark all pages in constant time.
3. At any given point in time, it is of course not possible to compute the entire content of OPT's cache, since this depends on future requests. It is, however, possible, given the next request, to compute whether or not OPT would have that particular page in cache or not. Using techniques [14] inspired by geometric algorithms [28], this can be done by registering the known time intervals of pages in OPT's cache in a balanced binary search tree. Also here, time $O(\log N)$ and space $O(N)$ can be obtained.