Available online at www.sciencedirect.com
ScienceDirect
Journal of Computer and System Sciences 73 (2007) 818–843
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
www.elsevier.com/locate/jcss

The relative worst-order ratio applied to paging ☆,☆☆

Joan Boyar *, Lene M. Favrholdt [1], Kim S. Larsen

Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense, Denmark

Received 30 June 2006; received in revised form 1 March 2007

Available online 12 March 2007

Abstract

The relative worst-order ratio, a relatively new measure for the quality of on-line algorithms, is extended and applied to the paging problem. We obtain results significantly different from those obtained with the competitive ratio. First, we devise a new deterministic paging algorithm, Retrospective-LRU, and show that, according to the relative worst-order ratio and in contrast with the competitive ratio, it performs better than LRU. Our experimental results, though not conclusive, are slightly positive and leave it possible that Retrospective-LRU or similar algorithms may be worth considering in practice. Furthermore, the relative worst-order ratio (and practice) indicates that LRU is better than the marking algorithm FWF, though all deterministic marking algorithms have the same competitive ratio. Look-ahead is also shown to be a significant advantage with this new measure, whereas the competitive ratio does not reflect that look-ahead can be helpful. Finally, with the relative worst-order ratio, as with the competitive ratio, no deterministic marking algorithm can be significantly better than LRU, but the randomized algorithm MARK is better than LRU.

Keywords: On-line algorithms; Relative worst-order ratio; Paging; LRU; RLRU; Look-ahead

1. Introduction

The standard measure for the quality of on-line algorithms is the competitive ratio [19,22,33], 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 is essentially identical to 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 quite natural, since the approximation algorithm is compared to another algorithm of the same general type, just with more computing power. Even though one does not want to execute what is typically an exponential-time algorithm, it is natural to

☆ A preliminary version of this paper appeared in the Sixteenth Annual ACM–SIAM Symposium on Discrete Algorithms, ACM Press, 2005, pp. 718–727.

☆☆ 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).

* Corresponding author. Fax: +45 6593 2691.
E-mail address: joan@imada.sdu.dk (J. Boyar).

[1] 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.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2007.03.001

Table 1
Comparison of measures

Measure	Value
Competitive Ratio	$\text{CR}_{\mathbb{A}} = \max_I \frac{\mathbb{A}(I)}{\text{OPT}(I)}$
Max/Max Ratio	$\text{MR}_{\mathbb{A}} = \frac{\max_{\lvert I\rvert=n} \mathbb{A}(I)}{\max_{\lvert I'\rvert=n} \text{OPT}(I')}$
Random Order Ratio	$\text{RR}_{\mathbb{A}} = \max_I \frac{E_\sigma[\mathbb{A}(\sigma(I))]}{\text{OPT}(I)}$
Relative Worst-Order Ratio	$\text{WR}_{\mathbb{A},\mathbb{B}} = \max_I \frac{\max_\sigma\{\mathbb{A}(\sigma(I))\}}{\max_{\sigma'}\{\mathbb{B}(\sigma'(I))\}}$

compare the result obtained by an approximation algorithm up against the solution which could be computed by the optimal algorithm, given enough time.

For on-line algorithms, the comparison to OPT is to a different type of algorithm. On-line algorithms are designed under the restriction that irrevocable decisions must be made for each input item one at a time without knowing the entire input sequence. In contrast, OPT is an off-line algorithm, knowing the entire input sequence in advance. Thus, the difference is deeper than merely an advantage in computing power.

Although the competitive ratio has been an extremely useful notion and there are numerous analyses of problem scenarios where the competitive ratio provides us with the intuitively correct answer, there are also situations where this is not the case. For the paging problem in particular, where a very large number of algorithms are found to be equally good, as judged by competitive analysis, many researchers have found the need for different analytical tools.

In a few cases (bin coloring [28] and dual bin packing [11]) one algorithm $\mathbb{A}$ even has a better competitive ratio than another algorithm $\mathbb{B}$, though intuitively, $\mathbb{B}$ is clearly better than $\mathbb{A}$. This is discussed further in Sections 1.6.3 and 1.7.

Often, when the competitive ratio fails to distinguish algorithms that are very different in practice, it seems that information is lost in the intermediate comparison to OPT. Thus, when differentiating between on-line algorithms is the goal, performing a direct comparison between the algorithms is a possibility worth exploring. 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 request sequence, 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 permutations of sequences. In this way, the relative worst-order ratio [9] combines some of the desirable properties of the Max/Max ratio [6] and the random order ratio [24]. These measures are compared in Table 1 and explained in more detail below. Note that the ratios given in the table are not the exact definitions of the measures; they are all asymptotic measures, but for simplicity, this is not reflected in the table.

We now describe the two measures that were the inspiration for the relative worst-order ratio.

1.1. The Max/Max ratio

The Max/Max ratio [6] 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. Technically, it can be applied to the paging problem, but the Max/Max ratio of any paging algorithm (deterministic or randomized) approaches 1 as the size of the slow memory approaches infinity.

The paper [6] considers the k-server problem. A memoryless algorithm with a Max/Max ratio of $2k$ is given. In contrast, the competitive ratio of any memoryless k-server algorithm can be arbitrarily large, depending on the underlying metric space. Furthermore, it is shown that the best possible Max/Max ratio can vary from 1 to k depending on the metric space, whereas the optimal competitive ratio lies between k and $2k-1$ and it has been conjectured that it is exactly k [30]. Finally, in contrast to the competitive ratio, the Max/Max ratio shows a slight advantage of look-ahead.

1.2. The random order ratio

The random order ratio [24] 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 input sequences, of the expected performance of $\mathbb{A}$ on a random permutation of the sequence, compared with an optimal solution. If, for all possible input sequences, any permutation of the sequence is equally likely, this ratio gives a meaningful worst-case measure of how well an algorithm can do.

Kenyon [24] shows that the random order ratio of the bin packing algorithm Best-Fit lies between 1.08 and 1.5. Thus, the random order ratio is lower than the competitive ratio of 1.7.

1.3. 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 sequence and takes the ratio of how the two algorithms perform on their respective worst permutations of that sequence. Note that the two algorithms may have different worst permutations for the same sequence. The relative worst-order ratio is formally defined in Section 2.

The relative worst-order ratio can be viewed as a worst-case version of Kenyon's random order ratio, with the modification that on-line algorithms are compared directly, rather than indirectly through OPT. This seems to make it much easier to compute than the random order ratio.

The relative worst-order ratio can also be viewed as a modification of the Max/Max ratio, where a finer partition of the request sequences is used; instead of finding the worst sequence among those having the same length, one finds the worst sequence among those that are permutations of each other. This particular finer partition was inspired by the random order ratio.

1.4. 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, 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 cache. If the cache already contains k pages when this happens, at least one of these 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.

1.5. Paging algorithms

Two major classes of deterministic algorithms for the paging problem are conservative algorithms [37] and marking algorithms [8].

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, where k is the size of the cache. 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 that has been in cache longest.

Marking algorithms work in phases. Each time a page is requested, this page is marked (implicitly in the analysis or explicitly by the algorithm). 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 that 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 PERM_{π} [7] defined in Sections 4 and 5.

1.6. Previous results

All deterministic conservative and marking algorithms have competitive ratio k [34,36] and this is optimal among deterministic algorithms [33]. However, in practice, these algorithms do not all have the same performance: LRU is better than FIFO and much better than FWF [37]. Moreover, results from [18] suggest there may be algorithms that perform even better than LRU.

In [3], an alternative model, the Max-/Average-Model, for the paging problem capturing locality of reference was suggested. It was proven that, in this model, LRU is slightly better than FIFO, but LRU is still best possible among deterministic algorithms. Related to this type of study, in [4], locality of reference for paging algorithms is modeled using diffuse adversaries [27], considering different families of probability distributions for generating sequences. In [4], LRU is the focus of the study, which also compares LRU to FWF, obtaining a large separation. In the access-graph model, LRU is known to be better than FIFO [15] and algorithms have been designed that are better than LRU [8]. Hence, these alternative ways of measuring the quality of paging algorithms give more satisfactory results. However, they are only defined for paging and paging-like problems.

In contrast to deterministic algorithms, MARK [17] has 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 [1,31].

1.6.1. 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 that 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 [36] proposed *resource-bounded look-ahead*, where the sequence is a maximal sequence of future requests for which it would incur ℓ page faults, and Breslauer [13] proposed *natural look-ahead*, where the sequence of future requests contains ℓ pages not currently in cache. Albers [2] proposed *strong look-ahead*, where the sequence of future requests contains ℓ distinct pages different from the current request. In this paper, 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 [6] 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 $N - k < \ell \leqslant N - 1$ (recall that N is the size of the slow memory). *Comparative analysis* [27] 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.

1.6.2. Other measures

Many alternatives to or variations on the competitive ratio have been proposed. We have already mentioned the Max/Max ratio, the random order ratio, access graphs, the Max-/Average-Model, diffuse adversaries, and comparative analysis. Other alternatives are Markov paging [23], extra resource analysis [21,33], the accommodating function [11], the loose competitive ratio (introduced in [37] and refined in [38]), and statistical adversaries [32]. Most of these techniques have been applied to only a few closely related problems. So far, the techniques that have been applied to a broader range of problems, extra resource analysis and the accommodating function, for instance, have given new separation results for only a limited number of different types of problems.

1.6.3. The relative worst-order ratio

One advantage of the relative worst-order ratio compared to other alternatives to the competitive ratio is that the relative worst-order ratio can be applied to quite different problems such as bin packing [9] and paging. For Classical Bin Packing, Worst-Fit is better than Next-Fit according to the relative worst-order ratio, even though they both have competitive ratio 2 [20]. Thus, the advantage of keeping more bins open[2] is reflected by the relative worst-order ratio. For Dual Bin Packing, First-Fit is better than Worst-Fit according to the relative worst-order ratio, while the

[2] It can easily be shown that Worst-Fit will always do at least as well as Next-Fit on any request sequence.

competitive ratio indicates the opposite [11]. In both cases, testing (see [29]), using uniformly distributed data, very clearly agrees with the results obtained using the relative worst-order ratio.

1.7. 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.

Some scheduling examples are given in [16]. For instance, for the problem of minimizing makespan on two related machines with speed ratio s, the optimal competitive ratio of $\frac{s+1}{s}$ for $s \geqslant \Phi \approx 1.618$ is obtained both by the post-greedy algorithm, which schedules each job on the machine where it will finish earliest, and by the algorithm that 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 $m \geqslant 2$ identical machines with preemption.

The relative worst-order ratio was also found by [25] to give the intuitively correct result for the bin coloring problem, where the competitive ratio gives the opposite result [28]: a trivial algorithm using only one open bin has a better competitive ratio than a natural greedy-type algorithm.

The proportional price version of the seat reservation problem has largely been ignored due to very negative impossibility results using competitive analysis [10]. However, algorithms for the problem were compared and separated with the relative worst-order ratio in [12].

1.8. Our results

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 [5], 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 was studied.

It is easily shown that RLRU does not belong to either of the common classes of algorithms, conservative and marking algorithms, that 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 more separations 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. The randomized algorithm, 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 [9], 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 sequences and their permutations, the other algorithm performs significantly better on other sequences and their permutations. 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.

Finally, in Appendix A, we discuss implementation issues for RLRU and initial experimental results. Using a balanced binary search tree, we explain how to implement the algorithm such that it runs in $O(\log N)$ time per request and uses $O(N)$ space, where N is the number of different pages requested. We argue that the scenarios where this is acceptable are probably the same scenarios where it would be acceptable to use LRU.

2. The new measure

In this section, we define the relative worst-order ratio and the notion of two algorithms being comparable (Definition 2) as in [9]. This is the most important definition, but the new notions of being weakly comparable and asymptotically comparable (defined in Definitions 5 and 6) give the possibility of adding more detail to the description of the relationship between two algorithms. Thus, Section 2.1.3 can be skipped until the definitions are used in Theorems 6 and 7 in Sections 4 and 5.

2.1. Minimization problems

For simplicity, we first give the definitions for minimization problems only. This is the natural starting point, since the paging problem is a minimization problem. In Section 2.2, we explain how to adapt the definitions to maximization problems.

2.1.1. The relative worst-order ratio

The definition of the relative worst-order ratio uses $\mathbb{A}_W(I)$, the cost of an algorithm $\mathbb{A}$ on the worst permutation of the input sequence I, formally defined in the following way.

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

$$\mathbb{A}_W(I) = \max_{\sigma} \mathbb{A}\big(\sigma(I)\big).$$

For many on-line problems, some algorithms perform well on particular types of request sequences, while other algorithms perform well on other types. The purpose of comparing on the worst permutation of sequences, rather than on each sequence independently, is to be able to differentiate between such pairs of algorithms, rather than just concluding that they are incomparable. Sequences with the same "content" are considered together, but the measure is worst-case, so the algorithms are compared on their respective worst permutations. This was originally motivated by problems where all permutations are equally likely, but appears to be applicable to other problems as well.

The following definition differs slightly from the definition given in previous papers. The difference is explained in Appendix B.

Definition 2. For any pair of algorithms $\mathbb{A}$ and $\mathbb{B}$, we define

$$c_l(\mathbb{A},\mathbb{B}) = \sup\big\{c \mid \exists b\colon \forall I\colon \mathbb{A}_W(I) \geqslant c\mathbb{B}_W(I) - b\big\} \quad \text{and}$$
$$c_u(\mathbb{A}\mathbb{B}) = \inf\big\{c \mid \exists b\colon \forall I\colon \mathbb{A}_W(I) \leqslant c\mathbb{B}_W(I) + b\big\}.$$

If $c_l(\mathbb{A},\mathbb{B}) \geqslant 1$ or $c_u(\mathbb{A},\mathbb{B}) \leqslant 1$, the algorithms are said to be *comparable* and the *relative worst-order ratio* $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ of algorithm $\mathbb{A}$ to algorithm $\mathbb{B}$ is defined. Otherwise, $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ is undefined.

$$\text{If } c_l(\mathbb{A},\mathbb{B}) \geqslant 1, \quad \text{then } \mathrm{WR}_{\mathbb{A},\mathbb{B}} = c_u(\mathbb{A},\mathbb{B}) \quad \text{and}$$
$$\text{if } c_u(\mathbb{A},\mathbb{B}) \leqslant 1, \quad \text{then } \mathrm{WR}_{\mathbb{A},\mathbb{B}} = c_l(\mathbb{A},\mathbb{B}).$$

If $\mathrm{WR}_{\mathbb{A},\mathbb{B}} < 1$, algorithms $\mathbb{A}$ and $\mathbb{B}$ are said to be *comparable in* $\mathbb{A}$*'s favor*. Similarly, if $\mathrm{WR}_{\mathbb{A},\mathbb{B}} > 1$, the algorithms are said to be *comparable in* $\mathbb{B}$*'s favor*.

Intuitively, $c_l(\mathbb{A},\mathbb{B})$ and $c_u(\mathbb{A},\mathbb{B})$ can be thought of as tight lower and upper bounds, respectively, on the cost of $\mathbb{A}$ relative to $\mathbb{B}$. More precisely, if we do not allow the additive constant b and define $c'_u(\mathbb{A},\mathbb{B})$ and $c'_l(\mathbb{A},\mathbb{B})$ as

$$c'_l(\mathbb{A},\mathbb{B}) = \sup\big\{c \mid \forall I\colon \mathbb{A}_W(I) \geqslant c\mathbb{B}_W(I)\big\} \quad \text{and}$$
$$c'_u(\mathbb{A},\mathbb{B}) = \inf\big\{c \mid \forall I\colon \mathbb{A}_W(I) \leqslant c\mathbb{B}_W(I)\big\},$$

then

$$c'_{\mathrm{l}}(\mathbb{A},\mathbb{B}) \leqslant \frac{\mathbb{A}_{\mathrm{W}}(I)}{\mathbb{B}_{\mathrm{W}}(I)} \leqslant c'_{\mathrm{u}}(\mathbb{A},\mathbb{B}) \quad \text{for all } I.$$

These bounds are tight in the sense that, for any $\varepsilon > 0$, there are sequences I_1 and I_2 such that $\mathbb{A}_{\mathrm{W}}(I_1)/\mathbb{B}_{\mathrm{W}}(I_1) \geqslant (1-\varepsilon)c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ and $\mathbb{A}_{\mathrm{W}}(I_2)/\mathbb{B}_{\mathrm{W}}(I_2) \leqslant (1+\varepsilon)c_{\mathrm{l}}(\mathbb{A},\mathbb{B})$. Note that, for $c_{\mathrm{l}}(\mathbb{B},\mathbb{A}) \neq 0$, $c_{\mathrm{u}}(\mathbb{A},\mathbb{B}) = 1/c_{\mathrm{l}}(\mathbb{B},\mathbb{A})$.

In [9] it was proven that the relative worst-order ratio is a *transitive measure*, i.e., for any three algorithms $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$, $\mathrm{WR}_{\mathbb{A},\mathbb{B}} \geqslant 1$ and $\mathrm{WR}_{\mathbb{B},\mathbb{C}} \geqslant 1$ implies $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \geqslant 1$. Furthermore, when $\mathrm{WR}_{\mathbb{A},\mathbb{B}} \geqslant 1$, $\mathrm{WR}_{\mathbb{B},\mathbb{C}} \geqslant 1$, and both are bounded above by some constant, then $\max\{\mathrm{WR}_{\mathbb{A},\mathbb{B}}, \mathrm{WR}_{\mathbb{B},\mathbb{C}}\} \leqslant \mathrm{WR}_{\mathbb{A},\mathbb{C}} \leqslant \mathrm{WR}_{\mathbb{A},\mathbb{B}} \cdot \mathrm{WR}_{\mathbb{B},\mathbb{C}}$. Thus, when a new algorithm is analyzed, it need not be compared to all other algorithms. In Appendix B we give the proof from [9], adapted to the new definition of the relative worst-order ratio.

2.1.2. The worst-order ratio

Although one of the goals in defining the relative worst-order ratio was to avoid the intermediate comparison of an on-line algorithm, $\mathbb{A}$, 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* of $\mathbb{A}$ and denoted $\mathrm{WR}_{\mathbb{A}}$:

Definition 3. The *worst-order ratio* of $\mathbb{A}$ is defined as

$$\mathrm{WR}_{\mathbb{A}} = \mathrm{WR}_{\mathbb{A},\mathrm{OPT}}.$$

2.1.3. Relaxations of the measure

Even if a pair of algorithms is 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 4. Let c_{u} be defined as in Definition 2. If at least one of the ratios $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ and $c_{\mathrm{u}}(\mathbb{B},\mathbb{A})$ is finite, the algorithms $\mathbb{A}$ and $\mathbb{B}$ are $(c_{\mathrm{u}}(\mathbb{A},\mathbb{B}), c_{\mathrm{u}}(\mathbb{B},\mathbb{A}))$-*related*.

This notation can also be used for algorithms that 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 in all cases).

In Section 5, it is shown that LRU and Last-In/First-Out (LIFO) are $(\frac{k+1}{2}, \infty)$-related. While this is not sufficient for a relative worst-order ratio to exist (the algorithms are not comparable by Definition 2), it is a notion that it seems reasonable to capture. We say, therefore, that the pair of algorithms are *weakly* comparable.

Definition 5. Let $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ be defined as in Definition 4. Algorithms $\mathbb{A}$ and $\mathbb{B}$ are *weakly comparable*

- if $\mathbb{A}$ and $\mathbb{B}$ are comparable,
- if exactly one of $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ and $c_{\mathrm{u}}(\mathbb{B},\mathbb{A})$ is finite, or
- if both are finite and $c_{\mathrm{u}}(\mathbb{A},\mathbb{B}) \notin \Theta(c_{\mathrm{u}}(\mathbb{B},\mathbb{A}))$, where $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ is considered a function of the problem parameters.

More specifically, $\mathbb{A}$ and $\mathbb{B}$ are *weakly comparable in $\mathbb{A}$'s favor*,

- if $\mathbb{A}$ and $\mathbb{B}$ are comparable in $\mathbb{A}$'s favor,
- if $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ is finite and $c_{\mathrm{u}}(\mathbb{B},\mathbb{A})$ is infinite, or
- if $c_{\mathrm{u}}(\mathbb{A},\mathbb{B}) \in o(c_{\mathrm{u}}(\mathbb{B},\mathbb{A}))$.

We conclude with a definition that 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 6. A *resource-dependent problem* is an optimization problem, where each problem instance, in addition to the input data, 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_{u} and c_{l} be defined as in Definition 4. We define

$$c_{\mathrm{l}}^{\infty}(\mathbb{A},\mathbb{B}) = \lim_{k\to\infty}\{c_{\mathrm{l}}(\mathbb{A},\mathbb{B})\} \quad \text{and} \quad c_{\mathrm{u}}^{\infty}(\mathbb{A},\mathbb{B}) = \lim_{k\to\infty}\{c_{\mathrm{u}}(\mathbb{A},\mathbb{B})\}.$$

If $c_{\mathrm{l}}^{\infty}(\mathbb{A},\mathbb{B}) \leqslant 1$ or $c_{\mathrm{u}}^{\infty}(\mathbb{A},\mathbb{B}) \geqslant 1$, the algorithms are *resource-asymptotically comparable* and the *resource-asymptotic relative worst-order ratio* $\mathrm{WR}_{\mathbb{A},\mathbb{B}}^{\infty}$ of $\mathbb{A}$ to $\mathbb{B}$ is defined. Otherwise, $\mathrm{WR}_{\mathbb{A},\mathbb{B}}^{\infty}$ is undefined.

$$\begin{aligned} &\text{If } c_{\mathrm{u}}^{\infty}(\mathbb{A},\mathbb{B}) \leqslant 1, \quad \text{then } \mathrm{WR}_{\mathbb{A},\mathbb{B}}^{\infty} = c_{\mathrm{l}}^{\infty}(\mathbb{A},\mathbb{B}) \quad \text{and} \\ &\text{if } c_{\mathrm{l}}^{\infty}(\mathbb{A},\mathbb{B}) \geqslant 1, \quad \text{then } \mathrm{WR}_{\mathbb{A},\mathbb{B}}^{\infty} = c_{\mathrm{u}}^{\infty}(\mathbb{A},\mathbb{B}). \end{aligned}$$

If $\mathrm{WR}_{\mathbb{A},\mathbb{B}}^{\infty} < 1$, algorithms $\mathbb{A}$ and $\mathbb{B}$ are said to be *resource-asymptotically comparable in $\mathbb{A}$'s favor*. Similarly, if $\mathrm{WR}_{\mathbb{A},\mathbb{B}}^{\infty} > 1$, the algorithms are said to be *resource-asymptotically comparable in $\mathbb{B}$'s favor*.

Finally, $\mathbb{A}$ and $\mathbb{B}$ are said to be *resource-asymptotically* $(c_{\mathrm{u}}^{\infty}(\mathbb{A},\mathbb{B}), c_{\mathrm{u}}^{\infty}(\mathbb{B},\mathbb{A}))$*-related*.

Definition 7. Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for an optimization problem. If $\mathbb{A}$ and $\mathbb{B}$ are neither weakly nor resource-asymptotically comparable, we say that they are *incomparable*.

2.2. Maximization problems

In this section, we adapt the definitions of Section 2.1 to maximization problems. Definitions 3, 4, and 7 are valid for maximization problems without modification.

Definition 1 is adapted to maximization problems changing "cost" to "profit" and max to min.

Definition 2 is valid for maximization problems too, except that $\mathbb{A}$ and $\mathbb{B}$ are comparable in $\mathbb{A}$'s favor if $\mathrm{WR}_{A,B} > 1$ and in $\mathbb{B}$'s favor if $\mathrm{WR}_{A,B} < 1$.

Definition 5 is valid for maximization problems, with "finite" and "infinite" swapped in the second to last line and o replaced by ω in the last line.

Definition 6 is valid for maximization problems, except that the algorithms are resource-asymptotically comparable in $\mathbb{B}$'s favor, if $\mathrm{WR}^{\infty}(\mathbb{A},\mathbb{B}) < 1$, and in $\mathbb{A}$'s favor, if $\mathrm{WR}^{\infty}(\mathbb{A},\mathbb{B}) > 1$.

3. A better algorithm than LRU

In this section, we introduce an algorithm that turns out to be better than LRU according to the relative worst-order ratio. Initial experimental results (see Appendix A) indicates that this could also be the case in practice, but more experimental work is required to investigate this conclusively. This is in contrast to the competitive ratio, which says that LRU is best possible among deterministic algorithms.

The algorithm, called Retrospective-LRU (RLRU), uses the behavior of the optimal off-line algorithm, LFD [5], as part of its basis for deciding which page to evict. More precisely, having already processed requests $r_1, \ldots, r_{i-1}$, when RLRU considers the next request, r_i, the question as to whether LFD would or would not have r_i in cache at this point is part of the decision basis for RLRU.

LFD is the algorithm that, on a fault, evicts the page that will be requested farthest in the future. Though it is generally not possible to know what the entire cache content is for LFD at a given time, since this depends on future requests, it is always possible to decide whether the page p_i requested by the next request r_i is in its cache. If p_i has not been requested before, then it is not in LFD's cache. Otherwise, there is a most recent request r_j to p_i. At any request r_h, $j < h < i$, p_i is evicted if and only if all other pages in cache are requested between r_h and r_i. Clearly, this can be decided upon without knowing requests beyond request r_i.

We now proceed to describe algorithm Retrospective-LRU (RLRU), as defined in Fig. 1. The name comes from the algorithm's marking policy. When evicting pages, RLRU uses the LRU policy, but it chooses only from the unmarked pages in cache, unless they are all marked. Marks are set according to what the optimal off-line algorithm, LFD [5], would have in cache, if given the part of the sequence seen so far.

If RLRU has a fault and LFD does not, RLRU marks the page requested. If RLRU has a hit, the page p requested is marked if it is different from the page of the previous request. Requiring the page to be different from the previous page ensures that at least one other page has been requested since p was brought into the cache. A phase of the

```
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
    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 LFD's cache just before this request then
      mark p
  else
    if p is different from the previous page then
      mark p
```

Fig. 1. Retrospective-LRU (RLRU).

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.

Intuitively, RLRU tries to keep pages in cache that OPT would have had there. For example, consider a very large B-tree in a database application. With LRU, paths from the root down to some fixed number of leaves would be in cache, but which paths were there would keep changing. RLRU would tend to keep more of the frequently accessed nodes close to the root in cache and would thus have better performance.

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 that starts the next phase. Page p was requested in phase P, and was later evicted, also within phase P. At that point, all other pages in the cache must either be marked or have been 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. □

Lemma 2. *For any sequence* I *of page requests,* $\mathrm{RLRU}_{\mathrm{W}}(I) \leqslant \mathrm{LRU}_{\mathrm{W}}(I)$.

Proof. Consider a worst permutation 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, by Lemma 1, 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 as RLRU, i.e., LRU has at least as many faults on I_{LRU} as RLRU on I_{RLRU}. □

Lemma 2 establishes that $\mathrm{WR}_{\mathrm{LRU,RLRU}} \geqslant 1$. To find the exact relative worst-order ratio for the two algorithms, the following technical lemma for LRU is proven. This lemma is also used extensively in the section on conservative and marking algorithms.

Lemma 3. *For any sequence* I *of page requests, there exists a worst permutation of* I *for* LRU *with all faults appearing before all hits.*

$$I: \quad r_1, \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_j, r_{j+1}, \ldots, r_n$$
$$I'': \quad r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_j, r_i, r_{j+1}, \ldots, r_n$$

Fig. 2. The two sequences I' and I'' in the case where p is evicted at r_j.

Proof. We describe how any permutation I' of I can be transformed, step by step, 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, and is not evicted at the same point in I', r_i is placed just after the first request r_j, $j > i$, causing p to be evicted (see Fig. 2). Otherwise, r_i is inserted after r_n. In this case, let $j = n$.

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. □

Theorem 1. $\mathrm{WR}_{\mathrm{LRU,RLRU}} = \frac{k+1}{2}$.

Proof. Since Lemma 2 shows that $\mathrm{WR}_{\mathrm{LRU,RLRU}} \geqslant 1$, for the lower bound, 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,

$$\mathrm{LRU}_W(I_n) \geqslant \frac{k+1}{2}\mathrm{RLRU}_W(I_n) - b.$$

Let I_n consist of n consecutive subsequences, where, in each subsequence, the first $k-1$ requests are to the $k-1$ pages $p_1, p_2, \ldots, p_{k-1}$, 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 sequence is given in, LFD will never evict the pages $p_1, p_2, \ldots, p_{k-1}$ from cache, so RLRU will mark them the first time they are requested in each phase, if they have ever been requested before. Since there are never more than $k-1$ marked pages in cache, none of these pages is ever evicted in a phase in which it is marked. Thus, for each of these pages p', at most one phase is ended because of a fault on p', and the requests to the pages that only occur once cannot end phases. This gives at most $k-1$ phases, each containing at most one fault on each of the pages $p_1, p_2, \ldots, p_{k-1}$, which 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 $\frac{k+1}{2}$.

For the upper bound, note that if $\mathrm{WR}_{\mathrm{LRU,RLRU}} > \frac{k+1}{2}$, there exists a sequence I with the property that LRU faults s times on its worst permutation, I_{LRU}, RLRU faults s' times on its worst permutation, I_{RLRU}, and $s > \frac{k+1}{2} \cdot s' + k$. Then, $s > \frac{k+1}{2} \cdot s'' + k$, where s'' is the number of times RLRU faults on I_{LRU}. Assume, by Lemma 3, that I_{LRU} is such that LRU faults on each request of a prefix I_1 of I_{LRU} and on no request after I_1, i.e., $|I_1| = s$. There must exist a subsequence, $J = \langle r_1, r_2, \ldots, r_{k+1}\rangle$, of consecutive requests in I_1, where RLRU faults at most once. If not, I_1 could be partitioned in consecutive subsequences of $k+1$ requests and possibly one subsequence at the end with less than $k+1$ requests. On each of these subsequences of length $k+1$, RLRU would fault at least twice, implying $s \leqslant \frac{k+1}{2} \cdot s'' + k$.

Since LRU faults on all $k+1$ requests in J, they must be to $k+1$ different pages. One may assume that r_1 is not the first request, since then RLRU would fault on all the requests in J. Let p be the page requested immediately before J. Clearly, p must be in RLRU's cache when it begins processing J.

If r_{k+1} is not a request to p, then the fact that LRU faulted on every request in J means that J contains $k+1$ pages different from p, but at most $k-1$ of them could be in RLRU's cache when it begins processing J. Thus, RLRU must fault at least twice on the requests in J.

On the other hand, if r_{k+1} is a request to p, there are exactly k requests, $r_1, r_2, \ldots, r_k$, to pages different from p in J. At least one of them, r_i, must cause a fault, since at most $k-1$ of them could have been in cache when RLRU began processing J. Assume that r_i is the only fault in J. Then, all pages requested in J, except r_i, must be in cache at the beginning of J. Thus, the $k-1$ requests in $\{r_1, r_2, \ldots, r_k\} \setminus \{r_i\}$ must be to the pages $p_1, p_2, \ldots, p_{k-1}$, because any other page is requested only once in the entire sequence. We consider two cases:

- *All requests $r_1, r_2, \ldots, r_i$ belong to the same phase.* Since each of the pages $p_1, p_2, \ldots, p_{k-1}$ has been requested before J, each of them is marked no later than when it is requested in J. The page evicted on r_i is either an unmarked page or a least-recently-used marked page. In either case, it is not a page that has been requested in J before r_i. Thus, the page evicted is a page that is requested in J after r_i. This page must give rise to an additional fault, contradicting that r_i is the only fault in J.
- *A request r_j, $2 \leqslant j \leqslant i$, starts a new phase.* When evicting a page because of r_i, the pages requested by $r_j, r_{j+1}, \ldots, r_{i-1}$ have been marked. The pages requested by $r_1, r_2, \ldots, r_{j-1}$ are unmarked, but there are $k-(i-1) \geqslant 1$ other pages that are unmarked, and these are less-recently-used. Thus, by the same arguments as in the previous case, we arrive at a contradiction to r_i being the only fault in J. □

The proof of Theorem 1 relies on a few basic properties of RLRU. Modifications to the algorithm that do not change these basic properties 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 or never marked, and the proofs still hold. Another example is the decision when to end a phase. The most 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 that are better in practice than the one we present 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.

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 $\langle p_1, p_2, \ldots, p_{k+1}\rangle^2 \langle p_k, p_1, p_2, \ldots, p_{k-1}, p_{k+1} \mid p_{k-1}, p_k, p_1, p_2, \ldots, p_{k-2}, p_{k+1} \mid p_{k-2}, p_{k-1}, p_k, p_1, p_2, \ldots, p_{k-3}, p_{k+1} \mid \ldots \mid p_1, p_2, \ldots, p_{k+1}\rangle^n$, where $\mid$ marks the beginning of a new phase. The part of the sequence that 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 $1 \leqslant i \leqslant k-1$, 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 $1 \leqslant i \leqslant k-2$, 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. □

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.

Theorem 3. $\mathrm{WR}_{\mathrm{LRU}} = \mathrm{WR}_{\mathrm{RLRU}} = k$.

Proof. Consider any sequence I. Since no algorithm is better than OPT, on OPT's worst permutation of I, LRU will fault at least as many times as OPT, so it also will on its own worst permutation. Since the worst-order ratio cannot be larger than the competitive ratio, and LRU's competitive ratio is k, $\mathrm{WR}_{\mathrm{LRU}} \leqslant k$. For the lower bound, consider a sequence consisting of n copies of $k+1$ pages repeated cyclicly. On this sequence, LRU will fault on each request. On a fault, OPT will evict the page requested farthest in the future. Since there are only $k+1$ different page, this strategy gives a fault only on every kth page on the worst permutation. Thus, $\mathrm{WR}_{\mathrm{LRU}} = k$.

Consider any sequence I. As above, on OPT's worst permutation of I, RLRU will fault at least as many times as OPT, so it also will on its own worst permutation. By Lemma 2, for any sequence I, $\mathrm{RLRU}_{\mathrm{W}}(I) \leqslant \mathrm{LRU}_{\mathrm{W}}(I)$. Thus, since $\mathrm{WR}_{\mathrm{LRU}} = k$, $\mathrm{WR}_{\mathrm{RLRU}} \leqslant k$. 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. Again, since there are only $k+1$ different pages in the sequence, OPT will fault only on every kth page, giving the ratio k. □

4. Conservative and marking algorithms

It is easy to see that both LRU and FIFO are conservative algorithms [37]: between any two faults on the same page, there must be requests to at least k other pages. Using Lemma 3, we can prove that for any sequence I, there exists a permutation $I_{\mathbb{C}}$ of I which is worst possible for any conservative algorithm and that all conservative algorithms behave exactly the same when given $I_{\mathbb{C}}$.

We first prove that LRU is best possible among conservative algorithms.

Lemma 4. $\mathrm{WR}_{\mathbb{C},\mathrm{LRU}} \geqslant 1$, *for any conservative paging algorithm* $\mathbb{C}$.

Proof. By Lemma 3, we can consider a sequence I where all faults by LRU 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. $\mathrm{WR}_{\mathrm{LRU},\mathbb{C}} \geqslant 1$, *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 permutation 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 permutation 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 4. *For any pair of conservative algorithms* $\mathbb{C}_1$ *and* $\mathbb{C}_2$, $\mathrm{WR}_{\mathbb{C}_1,\mathbb{C}_2} = 1$.

Proof. By Lemmas 4 and 5, $\mathrm{WR}_{\mathbb{C}_1,\mathrm{LRU}} \geqslant 1$ and $\mathrm{WR}_{\mathrm{LRU},\mathbb{C}_2} \geqslant 1$. Since the relative worst-order ratio is easily seen to be a transitive measure [9], this means that $\mathrm{WR}_{\mathbb{C}_1,\mathbb{C}_2} \geqslant 1$. Similarly, $\mathrm{WR}_{\mathbb{C}_1,\mathrm{LRU}} \leqslant 1$ and $\mathrm{WR}_{\mathrm{LRU},\mathbb{C}_2} \leqslant 1$, which implies $\mathrm{WR}_{\mathbb{C}_1,\mathbb{C}_2} \leqslant 1$. □

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

Corollary 1. $\mathrm{WR}_{\mathrm{LRU,FIFO}} = 1$.

By Theorems 1 and 4 and the transitivity of the relative worst-order ratio, we have the following:

Corollary 2. *For any conservative algorithm* $\mathbb{C}$, $\mathrm{WR}_{\mathbb{C},\mathrm{RLRU}} = \frac{k+1}{2}$.

In contrast to the competitive ratio, the relative worst-order ratio distinguishes between different marking algorithms. In particular, LRU is better than FWF, as it is in practice. We first show that the marking algorithm FWF is strictly worse than any conservative algorithm:

Lemma 6. *For any conservative algorithm* $\mathbb{C}$, $\mathrm{WR}_{\mathrm{FWF},\mathbb{C}} \geqslant \frac{2k}{k+1}$.

Proof. By Theorem 4 and transitivity, it is sufficient to show that $\mathrm{WR}_{\mathrm{FWF,LRU}} \geqslant \frac{2k}{k+1}$. 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 sequence, FWF will fault at least as many times on its worst permutation as LRU will on its. This shows that $\mathrm{WR}_{\mathrm{FWF,LRU}} \geqslant 1$.

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,

$$\mathrm{FWF}_W(I_n) \geqslant \frac{2k}{k+1}\mathrm{LRU}_W(I_n) - b.$$

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, there is a worst permutation for LRU where all faults occur before all hits. LRU faults on a request to a page p, only if k other pages have been requested since p was last requested, so the only way to make LRU fault on every request is to request the $k+1$ pages cyclicly. Thus, $\langle p_2, p_3, \ldots, p_{k+1}, p_1\rangle^n \langle p_2, p_3, \ldots, p_k\rangle^n$ is a worst permutation of I_n with respect to LRU. LRU will fault $n(k+1) + k - 1$ times. Asymptotically, the ratio is $\frac{2k}{k+1}$. □

Lemma 7. *For any marking algorithm* $\mathbb{M}$ *and any request sequence* I,

$$\mathbb{M}_{\mathrm{W}}(I) \leqslant \frac{2k}{k+1}\mathrm{LRU}_{\mathrm{W}}(I) + k.$$

Proof. For any sequence with n complete 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, for the first $2 \cdot \lfloor \frac{n}{2} \rfloor$ complete phases, there is a permutation such that LRU faults at least $(k+1)\lfloor \frac{n}{2} \rfloor$ times. If the remaining requests consist of a complete phase, plus a partial phase, then LRU will also fault on $k+1$ of those requests if given in the correct order. Thus, the additive constant is bounded by the number of faults $\mathbb{M}$ makes on the last, partial or complete, phase and is thus at most k. □

Combining the above two lemmas, Theorem 4, and the transitivity of the relative worst-order ratio, we find that FWF is worst possible among marking algorithms.

Theorem 5. *For any conservative algorithm* $\mathbb{C}$, $\mathrm{WR}_{\mathrm{FWF},\mathbb{C}} = \frac{2k}{k+1}$.

Furthermore, LRU is close to being a best possible deterministic marking algorithm:

Lemma 8. *For any deterministic marking algorithm* $\mathbb{M}$ *and any sequence* I *of page requests,*

$$\mathrm{LRU}_{\mathrm{W}}(I) \leqslant \frac{k+1}{k}\mathbb{M}_{\mathrm{W}}(I).$$

Proof. Consider any sequence I of requests. By Lemma 3, there is a worst permutation 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, except possibly the last, 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. □

This immediately gives the following:

Corollary 3. *For any deterministic marking algorithm* $\mathbb{M}$ *with* $\mathrm{WR}_{\mathrm{LRU},\mathbb{M}}$ *defined,* $\mathrm{WR}_{\mathrm{LRU},\mathbb{M}} \leqslant \frac{k+1}{k}$.

Let $\mathrm{MARK}^{\mathrm{LIFO}}$ denote the marking algorithm that, on a fault, evicts the unmarked page which was most recently brought into cache. On some sequences, $\mathrm{MARK}^{\mathrm{LIFO}}$ is slightly better than LRU, but on others, LRU is about twice as good as $\mathrm{MARK}^{\mathrm{LIFO}}$:

The following lemma shows that Lemma 8 is tight. It does not settle, however, whether Corollary 3 is tight, since the relative worst-order ratio of LRU to $\mathrm{MARK}^{\mathrm{LIFO}}$ is undefined.

Lemma 9. *For $k \geqslant 2$, there exists a family of sequences I_n of page requests such that*

$$\mathrm{LRU}_{\mathrm{W}}(I_n) = \frac{k+1}{k} \mathrm{MARK}^{\mathrm{LIFO}}_{\mathrm{W}}(I_n),$$

and $\lim_{n\to\infty} \mathrm{LRU}_{\mathrm{W}}(I_n) = \infty$.

Proof. First note that, for any sequence I of page requests, there is a worst ordering of I with respect to $\mathrm{MARK}^{\mathrm{LIFO}}$ 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 $\mathrm{MARK}^{\mathrm{LIFO}}$ 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 $\langle p_1, p_2, \ldots, p_k, p_{k+1}, p_k, \ldots, p_2\rangle^{n/2}\langle p_1, p_{k+1}\rangle^{n/2}$. $\mathrm{MARK}^{\mathrm{LIFO}}$ does not fault on the last $n-1$ requests, whereas LRU will fault on all requests in the permutation $\langle p_1, p_2, \ldots, p_{k+1}\rangle^n$. □

Despite Lemma 9, $\mathrm{MARK}^{\mathrm{LIFO}}$ is not better than LRU:

Lemma 10. *There exists a family of sequences I_n of page requests and a constant b such that*

$$\mathrm{MARK}^{\mathrm{LIFO}}_{\mathrm{W}}(I_n) = \frac{2k}{k+1} \cdot \mathrm{LRU}_{\mathrm{W}}(I_n) - b,$$

and $\lim_{n\to\infty} \mathrm{MARK}^{\mathrm{LIFO}}_{\mathrm{W}}(I_n) = \infty$.

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 Lemma 6, $\mathrm{MARK}^{\mathrm{LIFO}}$ faults on every request. As explained in the proof of Lemma 6, the sequence $\langle p_2, p_3, \ldots, p_{k+1}, p_1\rangle^n\langle p_2, p_3, \ldots, p_k\rangle^n$ is a worst-ordering of I_n with respect to LRU. Hence, LRU faults at most $(k+1)n+k-1$ times, i.e.,

$$\mathrm{MARK}^{\mathrm{LIFO}}_{\mathrm{W}}(I_n) = 2kn = \frac{2k}{k+1}(k+1)n = \frac{2k}{k+1}\mathrm{LRU}_{\mathrm{W}}(I_n) - \frac{2k(k-1)}{k+1}. \qquad \square$$

Combining Lemmas 7–10 gives the following:

Theorem 6. *For $k \geqslant 2$,* LRU *and* $\mathrm{MARK}^{\mathrm{LIFO}}$ *are $(1+\frac{1}{k}, 2-\frac{2}{k+1})$-related. Thus, the resource-asymptotic relative worst-order ratio of* LRU *to* $\mathrm{MARK}^{\mathrm{LIFO}}$ *is*

$$\mathrm{WR}^{\infty}_{\mathrm{LRU},\mathrm{MARK}^{\mathrm{LIFO}}} = 2.$$

5. 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 sequences where LIFO does unboundedly worse than LRU. However, there also exists a family of sequences, 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 sequence, 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 exactly $2n+k-1$ 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 that 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 [7] is a mixed, randomized, memory-bounded algorithm that 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 [14][3] that 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 $\pi^{(m)}(i)$ 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 $N \geqslant k+2$, PERM_π performs worse than LRU according to the relative worst-order ratio.

Theorem 8.

$$\textit{If } N = k+1, \quad \mathrm{WR}_{\text{PERM}_\pi,\text{LRU}} = 1.$$
$$\textit{If } N > k+2, \quad \mathrm{WR}_{\text{PERM}_\pi,\text{LRU}} \geqslant 2 - \frac{k-1}{N} - \frac{2}{k}.$$

Proof. We first show that $\mathrm{WR}_{\text{PERM}_\pi,\text{LRU}} \geqslant 1$. 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 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.

[3] Chrobak et al. called this algorithm ROTATE.

Consider the current phase being processed, phase P_i. Arrange the pages in P_i in a cycle C, in the same order in which they occur in π. We will find a starting point in C so that requesting the pages in the order given by C, starting at this point, will cause PERM_π to fault on every page.

Arrange the pages in PERM_π's cache at the beginning of this phase in a cycle, C', also in the order in which they occur in π. If the cycles C and C' have no pages in common, none of the pages in phase P_i are currently in cache, so PERM_π will fault on every request. Thus, we assume that they have some pages in common. Since both cycles are ordered according to π the pages they have in common occur in the same order in both cycles. Match up the identical pages on the two cycles, and count on each cycle the number of pages strictly between consecutive matched pages. Create a cycle C'' consisting of integers representing the difference between the number of pages in C' and the number of pages in C between consecutive matched pages. Since C has one more page than C', the sum of the numbers in C'' is -1.

Claim. *There exists a starting point on C'' such that, in the string S defined by starting at this point on C'' and including the remainder of C'' in order, the sums of all prefixes are negative.*

Proof. Suppose there is no such starting point. Take any starting point and follow a prefix until a non-negative sum occurs. Continue like this, using the first value not in the previous prefix as the next starting point and stopping when a non-negative sum has been found. This process can be repeated indefinitely. Since C'' is finite, eventually two starting points will be the same. Between these two starting points, one has been around C'' an integer number of times with the sum of all values adding to some non-negative integer value. This is a contradiction since the sum of the numbers in C'' is -1. This establishes the claim. □

Choose a starting point on C'' where all prefix sums are negative. This corresponds to a subsequence of at least one page request in C that is not in C' and thus not in cache. This will be the starting point in C. Each request to a page in C, but not in C', will cause the eviction of a page in C' and these evictions will occur in the order given by C'. When a page p in both C and C' is requested, the number of pages before p in C is greater than the number before p in C', so p will have been evicted and will also cause a fault. Thus PERM_π will also fault $k+1$ times on this set of $k+1$ pages.

This occurs for every complete phase. Thus, LRU faults at most an additive constant k times more on its worst-ordering than what PERM_π does on its worst ordering, on the pages in the incomplete phase, if it exists.

We now prove that for $N = k+1$, $\text{WR}_{\text{PERM}_\pi,\text{LRU}} \leqslant 1$. This completes the proof that, for $N = k+1$, $\text{WR}_{\text{PERM}_\pi,\text{LRU}} = 1$. Consider any sequence 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 that is in cache and evicts this page. Since $N = k+1$, this will be the page p' immediately succeeding p in π, and the next fault will be on p'. 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 $N \geqslant k+2$. To see that $\text{WR}_{\text{PERM}_\pi,\text{LRU}} > 2 - \frac{k-1}{N} - \frac{2}{k}$, assume without loss of generality that $\pi = (1, 2, \ldots, N)$ and consider the family of request sequences

$$I_n = \langle p_1, p_2, \ldots, p_{k+1}, p_1, p_{k+2}, p_1, p_{k+3}, \ldots, p_1, p_N \rangle^n,$$

where each subsequence $\langle p_1, p_2, \ldots, p_{k+1}, p_1, p_{k+2}, p_1, p_{k+3}, \ldots, p_1, p_N \rangle$ is called a phase. We show that PERM_π faults on all requests in I_n. Since each phase has one request to each of the N pages and one extra request to p_1 for each of the $N-(k+1)$ pages $p_{k+2}, p_{k+3}, \ldots, p_N$, this implies

$$(\text{PERM}_\pi)_W(I_n) = \big(N + N - (k+1)\big)n = \big(2N - (k+1)\big)n.$$

Consider the first phase of I_n. We prove by induction that just before the ith request to p_1, $2 \leqslant i \leqslant N-k$, the cache contains $p_i, \ldots, p_{i+k-1}$. 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 the first phase, 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.

Now consider LRU. Between any pair of faults on p_1, at least k of the $N-1$ other pages are requested. Thus, regardless of the ordering, LRU faults on at most $\lfloor \frac{N-1}{k} n \rfloor + 1$ requests to p_1, i.e.

$$\mathrm{LRU_W}(I_n) \leqslant \left\lfloor \frac{N-1}{k} n \right\rfloor + 1 + (N-1)n \leqslant \left(\frac{N-1}{k} + \frac{1}{n} + N - 1 \right) n \leqslant \left(\frac{N}{k} + N - 1 \right) n \quad \text{for } n \geqslant k.$$

This gives a ratio of

$$\frac{(\mathrm{PERM}_\pi)_\mathrm{W}(I_n)}{\mathrm{LRU_W}(I_n)} \geqslant \frac{2N-(k+1)}{N-1+\frac{N}{k}} = \frac{2N-2+\frac{2N}{k}+2-\frac{2N}{k}-(k+1)}{N-1+\frac{N}{k}} \geqslant 2 - \frac{k-1+\frac{2N}{k}}{N-1+\frac{N}{k}}$$
$$> 2 - \frac{k-1}{N} - \frac{2}{k}, \quad \text{since } \frac{N}{k} > 1. \qquad \square$$

6. Look-ahead

In the standard on-line model, requests arrive one by one. A model in which the algorithm is informed of the next $\ell \geqslant 1$ 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 using $\ell + 1$ consecutive copies of each request, 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 that on a fault evicts the least-recently-used page in cache which is not among the next ℓ requests. If $\ell \geqslant k$, 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 sequence by noting that LRU(ℓ) is conservative.

Lemma 11. *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. First assume that $\ell < k$. At this point, p is the least-recently-used page that is not among the next ℓ. Clearly the second request causing a fault on p must be beyond these next ℓ. So the other $k-1$ pages in cache, when request r occurs, must all have been requested between the two faults on p. 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. Now assume that $\ell \geqslant k$. If p is not among the next ℓ requests when r occurs, the previous argument holds, so assume that it is. In this case p must have been the page in cache that was requested furthest in the future, so the other $k-1$ pages are requested between request r and the second fault on p. Again, counting the request r and p, there must be at least $k+1$ distinct pages in I', which is a contradiction. Thus, LRU(ℓ) is conservative. $\square$

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

Theorem 9. $\mathrm{WR}_{\mathrm{LRU},\mathrm{LRU}(\ell)} = \min\{k, \ell+1\}$.

Proof. Since the previous lemma combined with Lemma 5 show that $\mathrm{WR}_{\mathrm{LRU},\mathrm{LRU}(\ell)} \geqslant 1$, to prove the lower bound, 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,

$$\mathrm{LRU}_W(I_n) \geqslant \min\{k, \ell+1\}\, \mathrm{LRU}(\ell)_W(I_n) - b.$$

Let I_n consist of n phases, each containing the pages $p_1, p_2, \dots, p_k, p_{k+1}$, in that order. LRU will fault $n(k+1)$ times. However, if $\ell \leqslant k-1$, after the first k faults, LRU(ℓ) never faults on any of the next ℓ pages after a fault. Thus, regardless of the order, LRU(ℓ) faults on at most $k + \lfloor \frac{n(k+1)-k}{\ell+1} \rfloor$ pages. Asymptotically, this gives a ratio of $\ell + 1$. If $\ell \geqslant k$, then LRU(ℓ) faults on at most one out of every k pages.

For the upper bound, suppose there exists a sequence I, where LRU faults s times on its worst permutation, I_{LRU}, LRU(ℓ) faults s' times on its worst permutation, $I_{\mathrm{LRU}(\ell)}$, and $s > \min(k, \ell+1) \cdot s'$. Then, $s > \min(k, \ell+1) \cdot s''$, where s'' is the number of times LRU(ℓ) faults on I_{LRU}. One cannot have $\ell \geqslant k$, since then LRU(ℓ) faults fewer times than OPT. So suppose $\ell < k$, and assume by Lemma 3, that I_{LRU} is such that LRU faults on each request of a prefix I_1 of I_{LRU} and on no request after I_1. Then there must exist a request r in I_{LRU} where LRU(ℓ) faults, but it does not fault on any of the next $\ell + 1$ requests, all of which are in I_1. The last of these $\ell + 1$ requests caused LRU to fault, so it was not among the last k distinct requests at that point. Since $l < k$, it was not in any of the requests in the look-ahead when LRU(ℓ) processed request r, and all of the pages in the look-ahead were in cache then since LRU(ℓ) did not fault on any of them. Hence, this $(\ell+1)$st page was evicted by LRU(ℓ) when r was requested, and there must have been a fault the next time it was requested after that, giving a contradiction. □

Note that by transitivity, Theorem 9 shows that for any conservative algorithm, $\mathbb{C}$, $\mathrm{WR}_{\mathbb{C},\mathrm{LRU}(\ell)} = \min\{k, \ell+1\}$.

7. Randomized algorithms

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

Definition 8. Consider an optimization problem P and let I be any input sequence of length n. Let $\mathbb{A}$ be any randomized algorithm for P.

If P is a *minimization problem*, $E[\mathbb{A}(I)]$ is the expected cost of running $\mathbb{A}$ on I, and $\mathbb{A}_{\mathrm{W}}(I) = \max_\sigma E[\mathbb{A}(\sigma(I))]$.

If P is a *maximization problem*, $E[\mathbb{A}(I)]$ is the expected profit of running $\mathbb{A}$ on I, and $\mathbb{A}_{\mathrm{W}}(I) = \min_\sigma E[\mathbb{A}(\sigma(I))]$.

Using the above definition, the relative worst-order ratio is now defined as in the deterministic case.

Consider the randomized paging algorithm MARK [17]. On a fault, MARK chooses the unmarked page to be evicted uniformly at random. We show that $\mathrm{WR}_{\mathrm{LRU},\mathrm{MARK}} = k/H_k$, which is consistent with the results one obtains with the competitive ratio where MARK has ratio $2H_k - 1$ [1], while LRU has ratio k.

Recall that marking algorithms, such as MARK, work in phases. In each phase (except possibly the last), exactly k distinct pages are requested, and the first page requested within a phase was not requested in the previous phase. Thus, the subsequence processed within a phase (except possibly the last) is a maximal subsequence containing requests to exactly k distinct pages. 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.

For Lemma 12 and Theorem 10 below, we need the fact that MARK's expected number of faults in the ith k-phase is $m_i(H_k - H_{m_i} + 1)$ [17], where m_i is the number of *new* pages in the ith phase, i.e., the number of pages that are requested in the ith phase and not in the $(i-1)$st phase.

Lemma 12. *There exists a sequence I that is a worst permutation 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 a cyclic repetition of $k+1$ pages. □

Lemma 13. *For any sequence I of page requests, there exists a worst permutation I_{MARK} of I with respect to* MARK*, 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 to distinct pages.*

Proof. Consider a worst permutation 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. □

Lemma 14 below uses the following definition of rare and frequent pages and blocks.

Definition 9. Consider a sequence S consisting of $s \geqslant 2$ consecutive k-phases. Call pages requested in every phase of S *frequent* pages, and the others *rare* pages. The sequence S is called a *block*, if it has the following properties.

(1) Each k-phase in S contains exactly $s-1$ rare pages.
(2) There is no $r < s$ such that the first $r \geqslant 2$ k-phases of S contain exactly $r-1$ rare pages.

Note that any sequence with m k-phases contains at least $\lfloor \frac{m}{k+1} \rfloor$ consecutive blocks.

Lemma 14. *There exists a constant b such that, for any sequence I,* $\text{LRU}_{\text{W}}(I) \geqslant \text{MARK}_{\text{W}}(I) - b$.

Proof. For any request sequence I, consider a worst permutation I_{MARK} of I with respect to MARK, satisfying the conditions of Lemma 13. Partition the sequence I_{MARK} in blocks. Each block in the partition will be analyzed separately, and it will be shown that the sequence can be permuted so that LRU faults at least as many times as the expected number of faults by MARK on the requests of that block.

Consider a block, S, containing $s+1$ k-phases and thus s rare pages and $k-s$ frequent pages in each k-phase. Clearly, no frequent page is a new page in any of the last s k-phases of S. Therefore, if the first k-phase, P_1, in the block has at most s new pages, then MARK's expected number of faults is at most $s(s+1)(1+H_k-H_s)$.

Since each rare page occurs at most s times in S, one can permute the block into s groups of $k+1$ distinct pages, plus $(s+1)k - s(k+1) = k-s$ extra pages. Thus, LRU can be forced to fault $s(k+1)$ times. If P_1 has at most s new pages, MARK's expected number of faults on this block is at most $s(s+1)(1+H_k-H_s) \leqslant s(s+1)(1+\frac{k-s}{s+1}) = s(k+1)$, so in this case the result holds.

Now assume that the first k-phase, P_1, in the block, S, has $s+i$ new pages, where $0 < i \leqslant k-s$. Then, some frequent page in P_1 is also a new page. MARK's expected number of faults is at most $s^2(1+H_k-H_s) + (s+i)(1+H_k-H_{s+i})$.

Let S' be the block immediately preceding S. Assume that it contains $s'+1$ k-phases and thus s' rare pages in each k-phase. Consider any frequent, new page, p, in P_1. It is clearly not a frequent page in S'. Assume for a moment that p occurs in all but the last k-phase of S'. In this case, the first s' k-phases of S' have at least one more frequent page than all of S' does. Generally, removing k-phases from the end of a block cannot decrease the number of frequent pages, and the first two k-phases have at most $k-1$ frequent pages. Thus, removing k-phases from the end of S', we would eventually end up with $2 \leqslant r < s+1$ consecutive k-phases with $r-1$ rare pages. This contradicts the fact that S' is a block, so p occurs at most $s'-2$ times in S'. Hence, one can choose i requests to frequent, new pages in P_1 that can be moved back into the previous block, S', permuting S' such that LRU faults on these i pages, in addition to the $s'(k+1)$ pages originally in S' that it faults on. After removing these i requests from S, there are still s requests to rare pages in each k-phase, and a total of at least $s(k+1)$ requests in S, so the remaining requests can still be permuted to give LRU $s(k+1)$ faults. Thus, one can count $s(k+1)+i$ requests from block S that LRU will fault on. The lemma now follows by the following proposition. □

Proposition 1. *For all integers s, i, k, such that $1 \leqslant s \leqslant k$, $i \geqslant 0$, and $s+i \leqslant k$,*

$$s^2(1+H_k-H_s) + (s+i)(1+H_k-H_{s+i}) \leqslant s(k+1)+i.$$

Proof. For $s > \frac{k}{2}$, the claim holds by the following calculations,

$$s^2(1+H_k-H_s) + (s+i)(1+H_k-H_{s+i}) = s(s+1)\left(1+\sum_{j=s+1}^{k}\frac{1}{j}\right) + i\left(1+\sum_{j=s+i+1}^{k}\frac{1}{j}\right) - s\left(\sum_{j=s+1}^{s+i}\frac{1}{j}\right)$$

$$\leqslant s(s+1)\left(1+\frac{k-s}{s+1}\right)+i\left(1+\frac{k-s-i}{s+i+1}\right)-s\frac{i}{s+1}=s(k+1)+i+\left(\frac{i(k-s-i)}{s+i+1}-\frac{si}{s+1}\right)$$
$$< s(k+1)+i+\left(\frac{i(\frac{k}{2}-i)}{s+i+1}-\frac{\frac{k}{2}i}{s+1}\right),\quad \text{for } s>\frac{k}{2}$$
$$\leqslant s(k+1)+i,\quad \text{since } i\geqslant 0.$$

Note that approximating by integrals, one gets that $H_x - H_y \leqslant \ln(x) - \ln(y)$ for $x > y$. Thus, $s^2(1 + H_k - H_s) + (s+i)(1 + H_k - H_{s+i}) \leqslant s^2(1 + \ln(k) - \ln(s)) + (s+i)(1 + \ln(k) - \ln(s+i))$, so it is sufficient to prove that

$$f(s,i,k) = s^2\bigl(1+\ln(k)-\ln(s)\bigr) + (s+i)\bigl(1+\ln(k)-\ln(s+i)\bigr) - s(k+1) - i \leqslant 0$$

for $k \geqslant 6$ and $s \leqslant k/2$.

Taking the derivative of $f(s,i,k)$ with respect to i gives $f_i'(s,i,k) = \ln(k) - \ln(s+i) - 1$, which is zero at $i = \frac{k-se}{e}$, positive for smaller i and negative for larger. Thus, for fixed s and k, $f(s,i,k)$ has its maximum at $i = \frac{k-se}{e}$.

Assume now that $\frac{k}{e} \leqslant s \leqslant \frac{k}{2}$. In this case, $i = \frac{k-se}{e}$ is negative, and hence outside the specified range for i. Since $f(s,i,k)$ is decreasing for $i > \frac{k-se}{e}$, $f(s,i,k)$ is maximum at its smallest allowable value, $i = 0$. Hence,

$$f(s,i,k) \leqslant f(s,0,k) = s(s+1)\bigl(1+\ln(k)-\ln(s)\bigr) - s(k+1).$$

The derivative of this with respect to s is $-s - k - 2 + (1 + \ln(k) - \ln(s))(2s+1)$ which is zero where $(s+k+2)/(2s+1) = (1 + \ln(k) - \ln(s))$. At this point, $f(s,0,k) = s(s+1)(s+k+2)/(2s+1) - sk - s = \frac{s^3+s^2+s-s^2k}{2s+1}$. This is equal to zero, where $s^2 + s + 1 - sk = 0$, which has no solution in the range $\frac{k}{e} \leqslant s \leqslant \frac{k}{2}$ for $k \geqslant 4$. Looking at the endpoints of this range, we see that $f(\frac{k}{e}, 0, k)$ is negative for $k \geqslant 4$, and $f(\frac{k}{2}, 0, k)$ is negative for $k \geqslant 6$. Thus, $f(s,i,k)$ is negative for $\frac{k}{e} \leqslant s \leqslant \frac{k}{2}$ and $k \geqslant 6$.

Finally, consider $s \leqslant \frac{k}{e}$. In this range, the maximum value of f is

$$f\left(s, \frac{k-se}{e}, k\right) = s^2\bigl(1+\ln(k)-\ln(s)\bigr) + 2s + \frac{k-se}{e} - s(k+1).$$

Taking the derivative with respect to s gives $s - k + 2s(\ln(k) - \ln(s))$, which is negative for s small enough and then positive, so the function has a local minimum where the derivative is zero. Thus, the maximum values are at the endpoints. For $s = 1$, one gets that $f(1, \frac{k}{e} - 1, k) = 1 + \ln(k) + \frac{k}{e} - k$, which is negative for $k \geqslant 4$. For $s = \frac{k}{e}$, one gets that $f(\frac{k}{e}, 0, k) = (\frac{2}{e} - 1)sk + s$, which is negative for $k \geqslant 4$.

Thus, for $1 \leqslant s \leqslant k$, $i \geqslant 0$, and $s + i \leqslant k$, $s^2(1 + H_k - H_s) + (s+i)(1 + H_k - H_{s+i}) \leqslant s(k+1) + i$. □

Theorem 10. $\mathrm{WR}_{\mathrm{LRU,MARK}} = k/H_k$.

Proof. The lower bound follows from Lemmas 12 and 14. To see that the ratio cannot be higher than k/H_k, consider any k-phase in LRU's worst permutation. LRU never faults more than k times on any k-phase, and MARK never has an expected number of faults less than H_k on any complete k-phase [17]. MARK would fault at least as many times on its own worst-ordering. Thus, the result is tight. □

8. Conclusion and open problems

With this paper, the relative worst-order ratio has now been applied to a new problem scenario, namely paging. Previous measures and models, proposed as alternatives or supplements to the competitive ratio, have generally 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, scheduling, and seat reservation problems mentioned in the introduction, this gives a convincing basis for further investigation.

For paging, many algorithms with widely varying performance in practice all have competitive ratio k. The relative worst-order ratio distinguishes between some of these. Most notably, LRU is better than FWF and look-ahead is shown to help, according to the relative worst-order ratio. It is also promising that this new performance measure is leading to the discovery of new algorithms. Further testing is needed to determine which variant of RLRU is best in practice and how these compare with LRU over a broad range of test examples.

Theorem 3 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.

Acknowledgment

We thank an anonymous referee who helped improving the presentation of the new measure.

Appendix A. Experimental results on RLRU

The focus of attention in this paper is the theoretical results comparing various algorithms and algorithm classes using the relative worst-order ratio. Using this measure, we have concluded that RLRU is better than LRU. We believe it would be very interesting to determine if this result is reflected in the behavior of RLRU exhibited in practical applications. In our opinion, a full paper should be devoted to this issue alone. However, initial discussion of implementation considerations and experiments have been included here to demonstrate that a careful investigation could lead to new findings of interest in practice.

A.1. Implementation of RLRU

RLRU decides whether or not to mark pages based on whether or not they are in LFD's cache. At any given point in time, it is of course impossible to compute the entire contents of LFD's cache, since this depends on future requests. It is, however, possible, given a request and the request sequence up to that point, to compute whether or not LFD would have that particular page in cache.

RLRU can be implemented to run in time $O(\log N)$ and space $O(N)$, where N is the number of different pages requested, which we argue below.

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) Find the least-recently-used page, possibly just among unmarked pages.
(4) Check for a page in LFD's cache.

The following implementation strategies will ensure the stated complexities:

(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 that 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) Using a balanced binary search tree ordered on timestamp, the least-recently-used page can be found in logarithmic time. If the timestamp is also associated with pages in cache, then old timestamp entries can be found and updated when a page is requested. By adding information to the nodes in the tree regarding the last phase in which the page stored in the node was marked and information regarding the least recent phase of any node in the subtree, it is also possible in logarithmic time to find the least-recently-used page among those that are unmarked, i.e., not marked in the current phase. In an actual implementation, points 1, 2, and 3 can be combined.
(4) At any given point in time, it is of course impossible to compute the entire contents of LFD's cache, since this depends on future requests. It is, however, possible, given a request and the request sequence up to that point, to compute whether or not LFD would have that particular page in cache. Using techniques [26] inspired by geometric algorithms [35], this can be done by registering the known time intervals of pages in LFD's cache in a balanced binary search tree. Also here, time $O(\log N)$ and space $O(N)$ can be obtained.

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 [18]. 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 quite similar.

A.2. Empirical analysis

To get an indication as to whether or not the positive theoretical results are reflected in practice, we have investigated the behavior of LRU and RLRU on traces[4] collected from very different applications, including key words searches in text files, selections and joins in the Postgres database system, external sorting, and 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 $8, 16, \ldots, 1024$. Each entry shows the percent-wise improvement of RLRU over LRU. 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. In addition to the percentages, each entry shows the number of page faults of each of the three algorithms LFD, LRU, and RLRU, in that order.

Out of the 80 tests, 16 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. RLRU beats LRU with more than half a per cent in 32 cases, more than 1% in 17 cases, and more than 5% in 9 cases. This is illustrated in Fig. 3.

We also consider another variant, RLRU′, of RLRU. The only difference is that RLRU′ never marks pages that are already in cache. Thus, RLRU′ is defined as in Fig. 1 with the else-statement deleted. For this variant, we obtain the results displayed in Table 3.

For this algorithm, only 8 out of the 80 tests are negative. Except for the result of -1.01%, all results are larger than $-\frac{1}{3}\%$. RLRU’ beats LRU with more than $\frac{1}{3}\%$ in 39 cases, more than 1% in 13 cases, and more than 5% in 6 cases. The distribution of the percentages can be seen in Fig. 4.

A.3. Test conclusions

The test performed here is limited. We believe a thorough testing of the value of RLRU (and variants) in practice should be carried out in a full paper devoted to that. However, the preliminary tests reported here seem to indicate that LRU and RLRU (and variants) most often behave very similarly, but in the (relatively few) cases where LRU performs poorly compared with the optimal algorithm, RLRU's behavior is significantly better. Furthermore, we have not seen any scenarios where LRU performs significantly better than RLRU.

Appendix B. On Definition 2

In previous papers [9,12,16,25], the definition of the notion of comparable was slightly more restrictive, requiring that ($\exists b$: $\forall I$: $\mathbb{A}_W(I) \leqslant \mathbb{B}_W(I) + b$) or ($\exists b$: $\forall I$: $\mathbb{A}_W(I) \geqslant \mathbb{B}_W(I) - b$). We illustrate with an example how the new definition differs from the previous. Consider two algorithms $\mathbb{A}$ and $\mathbb{B}$ for some minimization problem. Assume the following:

[4] http://www.cs.wisc.edu/~cao/traces/.

Table 2
Empirical comparison of LRU and RLRU

Cache size	File names and lengths									
	bigsort 40167	j1 18533	j2 25881	j3 38112	j4 59744	j5 95723	j6 20709	pjoin 41558	pq7 32989	xds 88558
	11080	*418*	*8110*	*4194*	*7064*	*25169*	*4490*	*6906*	*9046*	*10665*
	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**
	10346	*331*	*8016*	*4143*	*6945*	*25014*	*4461*	*6760*	*8887*	*10630*
	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**
	10054	*205*	*7882*	*4076*	*6795*	*24773*	*4425*	*6594*	*8718*	*10566*
	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**
	9757	*126*	*7658*	*3974*	*6586*	*24325*	*4386*	*6363*	*8514*	*10438*
	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**
	9440	*126*	*7210*	*3782*	*6370*	*23477*	*4322*	*6026*	*8141*	*10182*
	10466	126	8120	4223	7087	25256	4503	6815	9075	10749
128	10311	126	8057	4234	7093	25211	4505	6808	9069	10694
	1.48	**0.00**	**0.78**	**−0.26**	**−0.08**	**0.18**	**−0.04**	**0.10**	**0.07**	**0.51**
	8928	*126*	*6314*	*3398*	*5986*	*21813*	*4194*	*5474*	*7501*	*9768*
	10238	126	8118	4213	7039	25209	4499	6793	8989	10564
256	10166	126	8057	4221	7038	24913	4492	6780	8984	10534
	0.70	**0.00**	**0.75**	**−0.19**	**0.01**	**1.17**	**0.16**	**0.19**	**0.06**	**0.28**
	8139	*126*	*4522*	*2630*	*5218*	*18771*	*3938*	*4796*	*6221*	*9236*
	10016	126	8115	4171	6933	24470	4491	6782	8870	10272
512	9881	126	8057	4173	6908	24021	4486	6753	8835	10232
	1.35	**0.00**	**0.71**	**−0.05**	**0.36**	**1.83**	**0.11**	**0.43**	**0.39**	**0.39**
	6744	*126*	*1288*	*1180*	*3682*	*14032*	*3426*	*4098*	*4571*	*8571*
	9618	126	5060	1921	6709	24024	4476	6042	8674	10190
1024	9532	126	4157	1799	6674	23693	4470	6040	8607	10183
	0.89	**0.00**	**17.85**	**6.35**	**0.52**	**1.38**	**0.13**	**0.03**	**0.77**	**0.07**

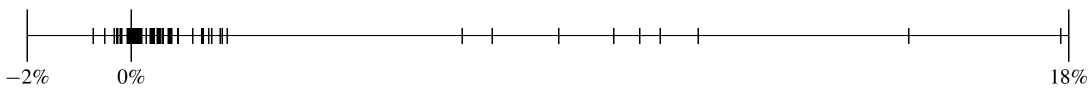

Fig. 3. Percentages with which RLRU is better than LRU.

- For all I,

$$\mathbb{A}_{\mathrm{W}}(I) \leqslant \mathbb{B}_{\mathrm{W}}(I) + \log_2\big(\mathbb{B}_{\mathrm{W}}(I)\big).$$

- There exists an infinite family of sequences I_n, $n = 1, 2, \ldots,$ such that

$$\mathbb{A}_{\mathrm{W}}(I_n) = n + \log_2 n \quad \text{and} \quad \mathbb{B}_{\mathrm{W}}(I_n) = n.$$

- There exists a family of sequences I_n, $n = 1, 2, \ldots,$ such that

$$\mathbb{A}_{\mathrm{W}}(I_n) = \frac{n}{2} \quad \text{and} \quad \mathbb{B}_{\mathrm{W}}(I_n) = n.$$

With the new definition, these two algorithms are comparable, with a relative worst order ratio of $\mathrm{WR}_{\mathbb{B},\mathbb{A}} = 2$, but with the previous definition, they are not comparable.

Table 3
Empirical comparison of LRU and RLRU′

Cache size	File names and lengths									
	bigsort 40167	j1 18533	j2 25881	j3 38112	j4 59744	j5 95723	j6 20709	pjoin 41558	pq7 32989	xds 88558
	11080	*418*	*8110*	*4194*	*7064*	*25169*	*4490*	*6906*	*9046*	*10665*
	14632	494	8233	4262	7278	25412	5100	8014	9371	10768
8	13451	499	8208	4265	7254	25368	5033	7205	9357	10724
	8.07	**−1.01**	**0.30**	**−0.07**	**0.33**	**0.17**	**1.31**	**10.09**	**0.15**	**0.41**
	10346	*331*	*8016*	*4143*	*6945*	*25014*	*4461*	*6760*	*8887*	*10630*
	12619	470	8177	4243	7201	25332	4596	7718	9277	10762
16	10862	470	8149	4220	7181	25315	4570	6986	9220	10714
	13.92	**0.00**	**0.34**	**0.54**	**0.28**	**0.07**	**0.57**	**9.48**	**0.61**	**0.45**
	10054	*205*	*7882*	*4076*	*6795*	*24773*	*4425*	*6594*	*8718*	*10566*
	10744	463	8138	4239	7180	25307	4516	7401	9216	10756
32	10620	426	8097	4217	7131	25299	4516	6927	9152	10703
	1.15	**7.99**	**0.50**	**0.52**	**0.68**	**0.03**	**0.00**	**6.40**	**0.69**	**0.49**
	9757	*126*	*7658*	*3974*	*6586*	*24325*	*4386*	*6363*	*8514*	*10438*
	10587	136	8120	4230	7135	25276	4505	6879	9185	10754
64	10521	136	8079	4199	7051	25250	4507	6895	9122	10703
	0.62	**0.00**	**0.50**	**0.73**	**1.18**	**0.10**	**−0.04**	**−0.23**	**0.69**	**0.47**
	9440	*126*	*7210*	*3782*	*6370*	*23477*	*4322*	*6026*	*8141*	*10182*
	10466	126	8120	4223	7087	25256	4503	6815	9075	10749
128	10422	126	8079	4199	6958	25149	4505	6836	9075	10703
	0.42	**0.00**	**0.50**	**0.57**	**1.82**	**0.42**	**−0.04**	**−0.31**	**0.00**	**0.43**
	8928	*126*	*6314*	*3398*	*5986*	*21813*	*4194*	*5474*	*7501*	*9768*
	10238	126	8118	4213	7039	25209	4499	6793	8989	10564
256	10226	126	8079	4189	6920	25127	4498	6783	8984	10541
	0.12	**0.00**	**0.48**	**0.57**	**1.69**	**0.33**	**0.02**	**0.15**	**0.06**	**0.22**
	8139	*126*	*4522*	*2630*	*5218*	*18771*	*3938*	*4796*	*6221*	*9236*
	10016	126	8115	4171	6933	24470	4491	6782	8870	10272
512	9934	126	8077	4045	6866	24409	4487	6772	8842	10262
	0.82	**0.00**	**0.47**	**3.02**	**0.97**	**0.25**	**0.09**	**0.15**	**0.32**	**0.10**
	6744	*126*	*1288*	*1180*	*3682*	*14032*	*3426*	*4098*	*4571*	*8571*
	9618	126	5060	1921	6709	24024	4476	6042	8674	10190
1024	9617	126	5074	1921	6723	23564	4471	6041	8658	10188
	0.01	**0.00**	**−0.28**	**0.00**	**−0.21**	**1.91**	**0.11**	**0.02**	**0.18**	**0.02**

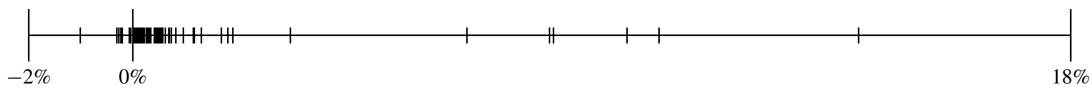

Fig. 4. Percentages with which RLRU′ is better than LRU.

The reason that the algorithms are not comparable with the previous definition is that, for any constant b, there are sequences such that $\mathbb{A}_W(I) \geqslant (1+\varepsilon)\mathbb{B}_W(I) + b$, for some $\varepsilon > 0$, i.e., the set $S_u(\mathbb{A},\mathbb{B}) = \{c \mid \exists b\colon\ \forall I\colon\ \mathbb{A}_W(I) \leqslant c\mathbb{B}_W(I) + b\}$ does not contain 1. The reason that, with the new definition, the algorithms are comparable is that, for any $\varepsilon > 0$, there exists a constant b such that, for all I, $\mathbb{A}_W(I) \leqslant (1+\varepsilon)\mathbb{B}_W(I) + b$, and hence, $\inf S_u(\mathbb{A},\mathbb{B}) = 1$.

Let S_l be defined analogously to S_u, i.e., $S_l(\mathbb{A},\mathbb{B}) = \{c \mid \exists b\colon\ \forall I\colon\ \mathbb{A}_W(I) \geqslant c\mathbb{B}_W(I) - b\}$. Then in general, two algorithms $\mathbb{A}$ and $\mathbb{B}$ are comparable with the new definition and not comparable with the previous definition, if and only if

$$(1 \notin S_l \wedge 1 \notin S_u) \wedge (\sup S_l = 1 \vee \inf S_u = 1).$$

Thus, with the new definition, $c_u(\mathbb{A},\mathbb{B}) \leqslant 1$ is interpreted as $\mathbb{A}$ being at least at good as $\mathbb{B}$, and $c_l(\mathbb{A},\mathbb{B})$ is then a tight bound on how much better $\mathbb{A}$ can be. Similarly, $c_l(\mathbb{A},\mathbb{B}) \geqslant 1$ means that $\mathbb{B}$ is at least as good as $\mathbb{A}$, and in this case, $c_u(\mathbb{A},\mathbb{B})$ gives a tight bound on how much better $\mathbb{B}$ can be.

Note that any pair of algorithms that are comparable according to the previous definition are also comparable with the new definition. Furthermore, all pairs of algorithms shown not to be comparable in [9,12,16,25] are not comparable with the new definition either, and all algorithms shown to be comparable in this paper are also comparable with the previous definition. We are not aware of any other published papers analyzing the relative worst-order ratio.

The proof of the following theorem, saying that the relative worst-order ratio is a transitive measure, is an adaptation of a proof in [9].

Theorem 11. *The ordering of algorithms for a given problem is transitive. More specifically, if* $\mathrm{WR}_{\mathbb{A},\mathbb{B}} \geqslant 1$ *and* $\mathrm{WR}_{\mathbb{B},\mathbb{C}} \geqslant 1$*, then* $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \geqslant \mathrm{WR}_{\mathbb{B},\mathbb{C}} \geqslant 1$*. If furthermore* $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ *is bounded above by some constant, then* $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \geqslant \mathrm{WR}_{\mathbb{A},\mathbb{B}}$.

Similarly, if $\mathrm{WR}_{\mathbb{A},\mathbb{B}} \leqslant 1$ *and* $\mathrm{WR}_{\mathbb{B},\mathbb{C}} \leqslant 1$*, then* $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \leqslant \min\{\mathrm{WR}_{\mathbb{A},\mathbb{B}}, \mathrm{WR}_{\mathbb{B},\mathbb{C}}\}$.

Proof. Suppose that three algorithms $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ for some problem are such that $\mathrm{WR}_{\mathbb{A},\mathbb{B}} \geqslant 1$ and $\mathrm{WR}_{\mathbb{B},\mathbb{C}} \geqslant 1$. Then, for any $\varepsilon > 0$, there exist constants a and b such that for all I, $\mathbb{A}_W(I) \geqslant (1-\varepsilon)\mathbb{B}_W(I) - a$ and $\mathbb{B}_W(I) \geqslant (1-\varepsilon)\mathbb{C}_W(I) - b$, i.e., $\mathbb{A}_W(I) > (1-\varepsilon)^2\mathbb{C}_W(I) - (a+b)$. Thus, $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \geqslant 1$. This proves that the measure is transitive.

Moreover, for any I, any r, and any $\varepsilon > 0$ such that $\mathbb{B}_W(I) \geqslant (1-\varepsilon)r\mathbb{C}_W(I) - d$, there exists a constant a such that

$$\mathbb{A}_W(I) \geqslant (1-\varepsilon)\mathbb{B}_W(I) - a > (1-\varepsilon)^2 r\mathbb{C}_W(I) - (d+a).$$

Therefore, since $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \geqslant 1$, an adversary argument constructing sequences to prove a lower bound of r on $\mathrm{WR}_{\mathbb{B},\mathbb{C}}$ would also give a lower bound of r on $\mathrm{WR}_{\mathbb{A},\mathbb{C}}$. Thus, $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \geqslant \mathrm{WR}_{\mathbb{B},\mathbb{C}}$.

Similarly, if $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ is bounded above by a constant, $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \geqslant \mathrm{WR}_{\mathbb{A},\mathbb{B}}$, since $\mathbb{A}_W(I) \geqslant (1-\varepsilon)r\mathbb{B}_W(I) - d$ implies that there exists a constant b such that

$$\mathbb{A}_W(I) \geqslant (1-\varepsilon)r\mathbb{B}_W(I) - d > (1-\varepsilon)^2 r\mathbb{C}_W(I) - (rb+d).$$

The arguments for the case where $\mathrm{WR}_{\mathbb{A},\mathbb{B}} \leqslant 1$ and $\mathrm{WR}_{\mathbb{B},\mathbb{C}} \leqslant 1$ are essentially the same, but there is no problem if $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ is not bounded away from zero; the value r simply makes the additive constant smaller. □

References

[1] D. Achlioptas, M. Chrobak, J. Noga, Competitive analysis of randomized paging algorithms, Theoret. Comput. Sci. 234 (1–2) (2000) 203–218.
[2] S. Albers, On the influence of lookahead in competitive paging algorithms, Algorithmica 18 (1997) 283–305.
[3] S. Albers, L.M. Favrholdt, O. Giel, On paging with locality of reference, J. Comput. System Sci. 70 (2005) 145–175.
[4] L. Becchetti, Modeling locality: A probabilistic analysis of LRU and FWF, in: Twelfth European Symposium on Algorithms, in: Lecture Notes in Comput. Sci., vol. 3221, Springer-Verlag, 2004, pp. 98–109.
[5] L.A. Belady, A study of replacement algorithms for virtual storage computers, IBM Systems J. 5 (1966) 78–101.
[6] S. Ben-David, A. Borodin, A new measure for the study of on-line algorithms, Algorithmica 11 (1) (1994) 73–91.
[7] A. Borodin, R. El-Yaniv, On randomization in online computation, in: IEEE Conference on Computational Complexity, 1997, pp. 226–238.
[8] A. Borodin, S. Irani, P. Raghavan, B. Schieber, Competitive paging with locality of reference, J. Comput. System Sci. 50 (2) (1995) 244–258.
[9] J. Boyar, L.M. Favrholdt, The relative worst-order ratio for on-line algorithms, in: Fifth Italian Conference on Algorithms and Complexity, in: Lecture Notes in Comput. Sci., vol. 2653, 2003, pp. 58–69. ACM Trans. Algorithms, in press.
[10] J. Boyar, K.S. Larsen, The seat reservation problem, Algorithmica 25 (1999) 403–417.
[11] J. Boyar, K.S. Larsen, M.N. Nielsen, The accommodating function—A generalization of the competitive ratio, SIAM J. Comput. 31 (1) (2001) 233–258.
[12] J. Boyar, P. Medvedev, The relative worst-order ratio applied to seat reservation, in: Ninth Scandinavian Workshop on Algorithm Theory, in: Lecture Notes in Comput. Sci., vol. 3111, Springer-Verlag, 2004, pp. 90–101.
[13] D. Breslauer, On competitive on-line paging with lookahead, Theoret. Comput. Sci. 209 (1–2) (1998) 365–375.
[14] M. Chrobak, H. Karloff, T. Payne, S. Vishwanathan, New results on server problems, SIAM J. Discrete Math. 4 (2) (1991) 172–181.
[15] M. Chrobak, J. Noga, LRU is better than FIFO, Algorithmica 23 (1999) 180–185.
[16] L. Epstein, L.M. Favrholdt, J.S. Kohrt, Separating online scheduling algorithms with the relative worst order ratio, J. Combin. Optim. 12 (4) (2006) 362–385.
[17] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, N.E. Young, Competitive paging algorithms, J. Algorithms 12 (1991) 685–699.

[18] A. Fiat, Z. Rosen, Experimental studies of access graph based heuristics: Beating the LRU standard?, in: Eighth Annual ACM–SIAM Symposium on Discrete Algorithms, 1997, pp. 63–72.
[19] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell Systems Technical J. 45 (1966) 1563–1581.
[20] D.S. Johnson, Fast algorithms for bin packing, J. Comput. System Sci. 8 (1974) 272–314.
[21] B. Kalyanasundaram, K. Pruhs, Speed is as powerful as clairvoyance, J. ACM 47 (2000) 617–643.
[22] A.R. Karlin, M.S. Manasse, L. Rudolph, D.D. Sleator, Competitive snoopy caching, Algorithmica 3 (1) (1988) 79–119.
[23] A.R. Karlin, S.J. Phillips, P. Raghavan, Markov paging, SIAM J. Comput. 30 (2000) 906–922.
[24] C. Kenyon, Best-fit bin-packing with random order, in: Seventh Annual ACM–SIAM Symposium on Discrete Algorithms, 1996, pp. 359–364.
[25] J.S. Kohrt, Online algorithms under new assumptions, PhD thesis, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark, 2004.
[26] J.S. Kohrt, K.S. Larsen, On-line seat reservations via off-line seating arrangements, Internat. J. Found. Comput. Sci. 16 (2) (2005) 381–397.
[27] E. Koutsoupias, C.H. Papadimitriou, Beyond competitive analysis, in: Thirty-Fifth Annual Symposium on Foundations of Computer Science, 1994, pp. 394–400.
[28] S.O. Krumke, W.E. de Paepe, J. Rambau, L. Stougie, Online bin coloring, in: Ninth Annual European Symposium on Algorithms, in: Lecture Notes in Comput. Sci., vol. 2161, 2001, pp. 74–85.
[29] K.S. Larsen, Testing bin packing algorithms on uniformly distributed data, http://www.imada.sdu.dk/~kslarsen/Archive/.
[30] M.S. Manasse, L.A. McGeoch, D.D. Sleator, Competitive algorithms for on-line problems, in: Twentieth Annual ACM Symposium on the Theory of Computing, 1988, pp. 322–333.
[31] L.A. McGeoch, D.D. Sleator, A strongly competitive randomized paging algorithm, Algorithmica 6 (1991) 816–825.
[32] P. Raghavan, A statistical adversary for on-line algorithms, in: D.D. Sleator, L.A. McGeoch (Eds.), On-Line Algorithms, in: Discrete Math. Theor. Comput. Sci., vol. 7, American Mathematical Society, 1992, pp. 79–83.
[33] D.D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, Commun. ACM 28 (2) (1985) 202–208.
[34] E. Torng, A unified analysis of paging and caching, Algorithmica 20 (1998) 175–200.
[35] M.J. van Kreveld, M.H. Overmars, Union-copy structures and dynamic segment trees, J. ACM 40 (3) (1993) 635–652.
[36] N. Young, Competitive paging and dual-guided algorithms for weighted caching and matching, PhD thesis, Technical Report CS-TR-348-91, Computer Science Department, Princeton University, 1991.
[37] N. Young, The k-server dual and loose competitiveness for paging, Algorithmica 11 (1994) 525–541.
[38] N. Young, On-line file caching, Algorithmica 33 (3) (2002) 371–383.