Acta Informatica manuscript No.
(will be inserted by the editor)

A Theoretical Comparison of LRU and LRU-K

Joan Boyar · Martin R. Ehmsen · Jens S. Kohrt · Kim S. Larsen

Received: date / Accepted: date

Abstract The paging algorithm Least Recently Used Second Last Request (LRU-2) was proposed for use in database disk buffering and shown experimentally to perform better than Least Recently Used (LRU). We compare LRU-2 and LRU theoretically, using both the standard competitive analysis and the newer relative worst order analysis. The competitive ratio for LRU-2 is shown to be $2k$ for cache size k, which is worse than LRU's competitive ratio of k. However, using relative worst order analysis, we show that LRU-2 and LRU are comparable in LRU-2's favor, giving a theoretical justification for the experimental results. Many of our results for LRU-2 also apply to its generalization, Least Recently Used Kth Last Request.

Keywords On-line algorithms · relative worst order analysis · paging · competitive ratio · LRU · LRU-2 · LRU-K.

1 Introduction

On many layers in a computer system, one is faced with maintaining a subset of memory units from a relatively slow memory in a significantly smaller fast memory. For ease of terminology, we refer to the fast memory as the *cache* and to the memory units as *pages*. The cache will have size k, meaning that it can hold at most k pages at any time. Pages are requested by the user (possibly indirectly by an operating or a database system) and the requests must be treated one at a time without knowledge of future requests. This makes the problem an *on-line* problem. If a requested page is already in cache, this is referred to as a *hit*. Otherwise, it is a *page fault*. When a page fault occurs, the requested page must be brought into cache. Thus, the only freedom is the choice of a page to evict from cache in order to make room for the requested page in the case of a page fault. An algorithm for this problem is referred to as a *paging algorithm*. Other

A preliminary version of this paper appeared in the Proceedings of the 4th International Workshop on Approximation and Online Algorithms, volume 4368 of Lecture Notes in Computer Science, pages 95–107, Springer, 2006. This work was supported in part by the Danish Natural Science Research Council.

Joan Boyar
Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark
Tel.: +45 6550 2338
Fax: +45 6550 2325
E-mail: joan@imada.sdu.dk

Martin R. Ehmsen
Department of Mathematics and Computer Science, University of Southern Denmark
E-mail: ehmsen@imada.sdu.dk

Jens S. Kohrt
Department of Mathematics and Computer Science, University of Southern Denmark
E-mail: svalle@imada.sdu.dk

Kim S. Larsen
Department of Mathematics and Computer Science, University of Southern Denmark
E-mail: kslarsen@imada.sdu.dk

names for this in the literature are “eviction strategy” or “replacement policy”. Various cost models for this problem have been studied. We focus on the classic model of minimizing the number of page faults. The problem is of great importance in database systems where it is often referred to as the *database disk buffering problem*. See [2] for an overview of the paging problem, cost models, and paging algorithms in general.

Probably the most well-known paging algorithm is LRU (Least-Recently-Used), which on a page fault evicts the least recently used page from cache. Experience from real-life request sequences, according to [18], is that overall LRU performs better than all other paging algorithms which have been proposed up until the introduction of LRU-K. On a page fault, LRU-K evicts the page with the least recent Kth last request. In other words, for each page, a list of times is maintained for the K most recent requests to that page. The first element of the list corresponds to the most recent access, the second corresponds to the access before that, etc. So to choose which page to evict, LRU-K compares the Kth (last) elements of these lists, and chooses the page whose Kth element is oldest. (If there are pages in cache which have been requested fewer than K times, the least recently used of these is evicted.) Compelling empirical evidence is given in [18] in support of the superiority of LRU-2 over LRU in database systems. We return to this issue below. From a statistical point of view, LRU-K has been shown to be optimal under the independent page reference model [19]. Since the introduction of LRU-K, there have been other proposals for better paging algorithms; for example [13].

In the on-line community, there were, to our knowledge, no published results on LRU-K before the preliminary conference version of this paper [4]. We assume that this was because it had not been possible to explain the experimental results using older on-line algorithmic techniques. In this paper, we provide a possible theoretical justification of LRU-2’s superiority over LRU. More specifically, we show using *relative worst order analysis* [5] that LRU-2 and LRU are comparable in LRU-2’s favor. In establishing this result, we prove a general result giving an upper bound on how well any algorithm can perform relative to LRU.

After the earlier version of this paper [4], other researchers have continued the investigation of the LRU-2 algorithm, but have not obtained the same positive results for LRU-2.

In [10], a technique called relative interval analysis, which reflects the range of the difference between the fault rate of two algorithms, is used to investigate paging algorithms. The results regarding LRU-2 are inconclusive, but the partial result in [10] states that for exactly $k+1$ pages in slow memory, LRU is better than LRU-2. The authors conclude from this that further assumptions need to be made in the model they use to reflect the behavior of these algorithms which is seen in practice.

In [8], parameterized analysis is applied to a series of paging algorithms. The parameter used is the amount of locality of reference in a given sequence, using a natural measure for the amount of locality of reference. The results obtained suggest that the performance of LRU is approximately twice as good as that of LRU-2 on sequences with the same amount of locality of reference.

In the operating systems community, further developments of paging algorithms have been studied and evaluated experimentally. A prominent example of this is the Adaptive Replacement Policy [17]. This algorithm tries to balance the key concepts of recency and frequency in a dynamic fashion, but allowing a varying amount of space for pages seen only once and pages seen at least twice recently. For an eviction strategy, this algorithm is quite complicated, which is probably one reason that an online analysis of the algorithm has not appeared yet, though experimental evidence suggests that it could be superior to LRU-2.

It is well-known that analysis of the paging problem is particularly problematic for the most standard quality measure for on-line algorithms, the competitive ratio [15, 20]. This has led researchers to investigate alternative methods. See a long list of these in [1, 9]. A few of the measures described there have been applied to more than one online problem, but most of these methods are only applicable to the paging problem.

In contrast, it has been demonstrated that relative worst order analysis is applicable to many on-line problems. In most cases, relative worst order analysis makes the same distinction between algorithms as competitive analysis does. However, the following is a list of results, where relative worst order analysis has distinguished between algorithms in a situation where competitive analysis cannot distinguish or even in some cases favors the ”wrong” algorithm. This is not an exclusive list; we merely highlight one result from each of these on-line problems. Each of these results holds according to relative worst order analysis:

- For classical bin packing, Worst-Fit is better than Next-Fit [5].
- For dual bin packing, First-Fit is better than Worst-Fit [5].
- For paging, LRU is better than FWF (Flush-When-Full) and look-ahead helps [6].
- For scheduling, minimizing makespan on two related machines, a post-greedy algorithm is better than scheduling all jobs on the fast machine [12].
- For bin coloring [16], a natural greedy-type algorithm is better than just using one open bin at a time [11].
- For proportional price seat reservation, First-Fit is better than Worst-Fit [7].

We refer the reader to the referenced papers for details and more results. Here, we merely want to point out that relative worst order analysis is an appropriate tool to apply to on-line problems in general. After the definitions for relative worst order analysis, we present some intuitive justification for why it also gives interesting results in comparing LRU and LRU-K.

For completeness, we mention that there does exist a result with relative worst order analysis which ranks algorithms in the reverse order from what one would guess. Best-Fit is recognized as being better than First-Fit in practice for the classical bin packing problem. Neither the competitive ratio nor relative worst order analysis separate them, but relative worst order analysis says that First-Fit is better than Best-Fit for the closely related dual bin packing problem [5].

Our paper is structured as follows: LRU-K, along with previous results and testing of the algorithm, is described in Section 2. Its competitive ratio is proven to be Kk in Section 3, showing that LRU-K has a suboptimal competitive ratio in comparison to LRU's competitive ratio of k. However, in Section 4, relative worst order analysis is applied showing that LRU-2 is comparable to LRU in LRU-2's favor, providing the theoretical justification for LRU-2's superiority. For LRU-K in general, we show the slightly weaker result that LRU-K and LRU are resource-asymptotically comparable in LRU-K's favor. However, most of our results are shown to hold for LRU-K. Two results which may be of independent interest are, first, a bound on $c_u(\text{LRU}, \mathbb{A})$, the factor by which any algorithm $\mathbb{A}$ can be better than LRU using relative worst order analysis, and secondly, a proof that the resource-asymptotically comparable relation is transitive.

In this paper, we use k as well as K, which is a little unusual. In the online algorithms community, k is always used for the size of the cache, and we believe it would be confusing to use something else. Previous authors have named their algorithm LRU-K, and the algorithm is in the title of this paper, so it would also be odd to change that. We hope the reader can bear with this slight inconvenience.

2 LRU-2 and Experimental Results

In [18], a new family of paging algorithms, LRU-K, is defined. Here, K is a constant which defines the specific algorithm in the family. On a page fault, LRU-K evicts the page with the least recent Kth last request (LRU-1 is LRU). If there are pages in cache which have been requested fewer than K times, then some subsidiary policy must be employed for those pages. In [18], LRU is suggested as a possible subsidiary policy. However, it would also be natural to recursively use LRU-(K-1). For the case of K = 2, this is the same. The results in this paper are independent of the choice of subsidiary policy.

The authors' motivation for considering LRU-2 (or LRU-K in general for various K) is that LRU does not discriminate between pages with very frequent versus very infrequent references. Both types can be held in cache for a long time once they are brought in. This can be at the expense of pages with very frequent references.

The algorithm LFU (Least-Frequently-Used) which evicts the page which is least frequently used is the ultimate algorithm in the direction of focusing on frequency, but this algorithm appears to adjust too slowly to changing patterns in the request sequence, according to competitive analysis [20]. The family of algorithms, LRU = LRU-1, LRU-2, LRU-3, ...with recursive subsidiary policies can be viewed as approaching the behavior of LFU.

A conscientious testing in [18] of particularly LRU-2 and LRU-3 up against LRU and LFU led the authors to conclude that LRU-2 is the algorithm of choice in database systems. The algorithms are tested in a real database system environment using random references from a Zipfian distribution, using real-life data from a CODASYL database system, and finally using data generated to simulate request sequences which would arise from selected applications where LRU-2 is expected to improve performance. LRU-2 and LRU-3 perform very similarly and in all cases significantly better than the other algorithms. Many test results are reported which can be viewed in different ways. If one should summarize the results in one sentence, we would say that LRU and LFU need 50–100% extra cache space in order to approach the performance of LRU-2.

Further testing of modifications of LRU-2 which require less extra memory [21, 14], have found that these variants are also superior to LRU.

It is not a coincidence that most of the testing of LRU-K algorithms have taken place in a database setting. In the fastest end of the memory hierarchy, between two levels of cache for instance, algorithms must be fast. Additionally, since data units are relatively small, not much space should be used for administration. The LRU-K algorithms are relatively complicated and need additional space for recording the last K accesses to a data unit, so LRU-K algorithms are not well suited for those levels. Using a few words for administration (when using LRU-2 or LRU-3) is not a problem in a database application with units of 8 KB of data, for instance. Also, page faults are very costly since they give rise to additional disk accesses, so running a more complicated algorithm to make better eviction decisions could be well worth the effort.

3 Competitive Analysis

Let $\mathbb{A}(I)$ denote the number of page faults a paging algorithm $\mathbb{A}$ has on request sequence I. The standard measure for the quality of on-line algorithms is the *competitive ratio*. The competitive ratio of $\mathbb{A}$ is

$$\mathrm{CR}(\mathbb{A}) = \inf\{c \mid \exists b\colon \forall I\colon \mathbb{A}(I) \leq c \cdot \mathrm{OPT}(I) + b\},$$

where OPT denotes an optimal off-line algorithm [15, 20].

Conservative algorithms are those which incur at most k faults on any consecutive input subsequence containing at most k distinct pages. A *k-phase partitioning* of a sequence is the recursive partitioning of the sequence into maximal subsequences (each referred to as a k-phase) containing exactly k distinct pages [3]. Suppose that all pages are unmarked at the beginning of every k-phase and then marked the first time they are requested within a k-phase. A *marking algorithm* is one which never evicts marked pages. Neither of the properties “conservative” or “marking” imply the other. For instance, Flush-When-Full is a marking algorithm, but is not conservative, while First-In-First-Out is conservative, but not a marking algorithm. LRU is known to be both a conservative algorithm [22] and a marking algorithm [3]. Both types of algorithms have competitive ratio k.

To see that LRU-K belongs to neither of these classes, consider the request sequence,

$$I = \langle p_1^K, p_2^K, \ldots, p_{k+1}^K, p_1, p_2, p_1, p_2 \rangle.$$

The first $K(k+1)$ requests ensure that the Kth last request to each of the pages is well defined, so p_1 is evicted when the first request to p_{k+1} occurs. This leads to a fault on p_1 and the eviction of p_2, since at this point its Kth last request is the $(K+1)$st request in I. Next p_2 is requested, giving a page fault, and evicting p_1, because its Kth last request is currently the second request in I. The next to last request is a fault on p_1, causing p_2 to be evicted, since at this point its Kth last request is the $(K+2)$nd request in I. Finally, there is a fault on p_2. The subsequence starting with the first request to p_4 and continuing to the end of the sequence contains only k distinct requests, but has $k+2$ faults, so LRU-K is not conservative. The second k-phase begins with the first request to p_{k+1}, marking p_{k+1}. The following requests to p_1 and p_2 mark them. The last two requests are part of this second k-phase, so no marking algorithm would fault on them. Thus, LRU-K is not a marking algorithm.

Since LRU-K is neither a conservative nor a marking algorithm, it is not obvious that its competitive ratio is k. In fact, the lemma below shows that it is larger than k.

Lemma 1 *The competitive ratio of* LRU-K *is at least Kk, for $k \geq 2$.*

Proof Assume that there are $k+1$ distinct pages, $p_1, p_2, \ldots, p_{k+1}$, in slow memory, and assume without loss of generality that the pages $p_1, p_2, \ldots, p_k$ are both in OPT and LRU-K’s caches.

Let

$$P_0 = \langle (p_1, p_2, \ldots, p_{k-1})^K, p_k^K \rangle$$

and let P_1 be K repetitions of $p_1, p_2, \ldots, p_{k-1}$, interleaved with K requests for p_{k+1} such that the first, third, fifth, etc. request is for p_{k+1}, until K requests for p_{k+1} have been made:

$$P_1 = \langle \underbrace{p_{k+1}, p_1, p_{k+1}, p_2, \ldots, p_{k+1}, p_{k-1}, p_{k+1}, p_1, p_{k+1}, p_2, p_{k+1}, p_3, p_4, p_5, \ldots, p_{k-1}}_{\text{exactly } K \text{ requests for } p_{k+1}} \rangle$$

It is clear that neither LRU-K nor OPT faults on any request in P_0, since all these requests are to the pages they initially have in cache. On the first request for p_{k+1} in P_1, both LRU-K and OPT fault. OPT will evict p_k and consequently not fault on any further requests in P_1. For LRU-K, on the other hand, since p_k has K requests just before P_1, making its Kth last request newer than any others until the Kth request to each page in P_1. The first time there are $k-1$ other pages with newer Kth last pages than p_k is at the last request, so LRU-K will not evict p_k until the last request in P_1. Hence, due to the cyclic repetition of $p_1, p_2, \ldots, p_{k-1}$ at the beginning of P_0, at a request for p_i $(1 \leq i \leq k-1)$, LRU-K will evict p_{k+1} until it has had K requests and p_{i+1} (or p_1 if $i = k-1$) after that. Each request for p_{k+1} will evict the following page in P_1 (page p_{i+1}). It follows that LRU-K faults on all Kk requests in P_1.

After P_1, neither OPT nor LRU-K have p_k in cache. Hence, we can now repeat the above request sequence (possibly with renaming of pages) arbitrarily often, and the result follows.

Lemma 2 LRU-K *is Kk-competitive.*

Proof Consider any sequence, $I = \langle r_1, r_2, \ldots, r_n \rangle$, and its k-phase partition. We prove that in each k-phase, LRU-K faults at most K times on each of the k different pages requested in that phase.

Suppose, for the sake of contradiction, that LRU-K faults more than K times on some page in a phase P. Let p be the first page in P with more than K faults in P. At some point between the Kth and $(K+1)$st faults on p, p must have been evicted by a request r_i to some other page q. The page q is one of the k pages in P. Since there are only k different pages in P and p is not in cache just after r_i, there must be some page w in cache just after r_i which is not in P. The Kth last request to w must be before the start of P and thus before the Kth last request to p. Hence, p could not have been evicted at r_i. This gives a contradiction, so there are at most Kk faults in any k-phase.

The following theorem follows immediately from the previous two results:

Theorem 1 $\mathrm{CR}(\text{LRU-K}) = Kk$.

4 Relative Worst Order Analysis

Now we give the theoretical justification for the empirical result that LRU-2 performs better than LRU. In order to do this, we use a different measure for the quality of on-line algorithms, the relative worst order ratio [5,6], which has previously [5,6,12,7] proven capable of differentiating between algorithms in other cases where competitive analysis failed to give the "correct" result. Instead of comparing on-line algorithms to an optimal off-line algorithm (and then comparing their competitive ratios), two on-line algorithms are compared directly. However, instead of comparing their performance on the exact same sequence, they are compared on their respective worst permutations of the same sequence:

Definition 1 Let $\sigma(I)$ denote a permutation of the sequence I, let $\mathbb{A}$ and $\mathbb{B}$ be paging algorithms, and let $\mathbb{A}_W(I) = \max_\sigma\{\mathbb{A}(\sigma(I))\}$. Define

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

The *relative worst order ratio* $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ of algorithm $\mathbb{A}$ to algorithm $\mathbb{B}$ is defined as:

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

Otherwise, $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ is undefined.

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 $\mathbb{B}$. When either $c_l(\mathbb{A},\mathbb{B}) \geq 1$ or $c_u(\mathbb{A},\mathbb{B}) \leq 1$ holds, the relative worst order ratio is a bound on how much better the one algorithm can be, i.e., 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.

In some cases, however, the first algorithm can do significantly better than the second, while the second can sometimes do marginally better than the first. In such cases, we use the following definitions (from [6], but restricted to the paging problem here) and show that the two algorithms are resource-asymptotically comparable in favor of the first algorithm.

Definition 2 Let $\mathbb{A}$ and $\mathbb{B}$ be paging algorithms, and let c_u and c_l be defined as above. When the limits exist, define

$$c_l^\infty(\mathbb{A},\mathbb{B}) = \lim_{k\to\infty}\{c_l(\mathbb{A},\mathbb{B})\} \qquad \text{and} \qquad c_u^\infty(\mathbb{A},\mathbb{B}) = \lim_{k\to\infty}\{c_u(\mathbb{A},\mathbb{B})\}.$$

If $c_u^\infty(\mathbb{A},\mathbb{B}) \leq 1$ or $c_l^\infty(\mathbb{A},\mathbb{B}) \geq 1$, the algorithms are *resource-asymptotically comparable* and the resource-asymptotic relative worst-order ratio $\mathrm{WR}^\infty_{\mathbb{A},\mathbb{B}}$ of $\mathbb{A}$ to $\mathbb{B}$ is defined. Otherwise, $\mathrm{WR}^\infty_{\mathbb{A},\mathbb{B}}$ is undefined.

$$\begin{aligned}
&\text{If } c_u^\infty(\mathbb{A},\mathbb{B}) \leq 1, \quad \text{then } \mathrm{WR}^\infty_{\mathbb{A},\mathbb{B}} = c_l^\infty(\mathbb{A},\mathbb{B}) \quad \text{and}\\
&\text{if } c_l^\infty(\mathbb{A},\mathbb{B}) \geq 1, \quad \text{then } \mathrm{WR}^\infty_{\mathbb{A},\mathbb{B}} = c_u^\infty(\mathbb{A},\mathbb{B}).
\end{aligned}$$

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

The relation, being resource-asymptotically comparable in the first algorithm's favor, is transitive, so it gives a well-defined means of comparing on-line algorithms.

Lemma 3 *Resource-asymptotically comparable in an on-line algorithm's favor is a transitive relation.*

Proof Assume that three algorithms $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$, for the same minimization problem (the proof is easily adapted to maximization problems), are related such that $\mathbb{A}$ is resource-asymptotically comparable to $\mathbb{B}$ in $\mathbb{A}$'s favor and $\mathbb{B}$ is resource-asymptotically comparable to $\mathbb{C}$ in $\mathbb{B}$'s favor.

We need to show that $\mathbb{A}$ is resource-asymptotically comparable to $\mathbb{C}$ in $\mathbb{A}$'s favor, i.e.,

$$c_u^\infty(\mathbb{A},\mathbb{C}) \leq 1 \qquad \text{and} \qquad c_l^\infty(\mathbb{A},\mathbb{C}) < 1.$$

Since $\mathbb{A}$ is resource-asymptotically comparable to $\mathbb{B}$ in $\mathbb{A}$'s favor and $\mathbb{B}$ is resource-asymptotically comparable to $\mathbb{C}$ in $\mathbb{B}$'s favor, we know that

$$c_u^\infty(\mathbb{A},\mathbb{B}) \leq 1 \qquad \text{and} \qquad c_l^\infty(\mathbb{A},\mathbb{B}) < 1$$

and

$$c_u^\infty(\mathbb{B},\mathbb{C}) \leq 1 \qquad \text{and} \qquad c_l^\infty(\mathbb{B},\mathbb{C}) < 1.$$

It follows that

$$\begin{aligned}
1 &\geq c_u^\infty(\mathbb{A},\mathbb{B}) \cdot c_u^\infty(\mathbb{B},\mathbb{C})\\
&= \lim_{k\to\infty}\{c_u(\mathbb{A},\mathbb{B})\} \cdot \lim_{k\to\infty}\{c_u(\mathbb{B},\mathbb{C})\}\\
&= \lim_{k\to\infty}\{\inf\{c_1 \mid \exists b_1\colon \forall I\colon \mathbb{A}_W(I) \leq c_1\mathbb{B}_W(I) + b_1\}\}\cdot\\
&\quad\ \lim_{k\to\infty}\{\inf\{c_2 \mid \exists b_2\colon \forall I\colon \mathbb{B}_W(I) \leq c_2\mathbb{C}_W(I) + b_2\}\}\\
&= \lim_{k\to\infty}\{\inf\{c_1 \mid \exists b_1\colon \forall I\colon \mathbb{A}_W(I) \leq c_1\mathbb{B}_W(I) + b_1\}\cdot\inf\{c_2 \mid \exists b_2\colon \forall I\colon \mathbb{B}_W(I) \leq c_2\mathbb{C}_W(I) + b_2\}\}\\
&= \lim_{k\to\infty}\{\inf\{c_1c_2 \mid \exists b_1, b_2\colon \forall I\colon \mathbb{A}_W(I) \leq c_1\mathbb{B}_W(I) + b_1 \wedge \mathbb{B}_W(I) \leq c_2\mathbb{C}_W(I) + b_2\}\}\\
&= \lim_{k\to\infty}\{\inf\{c \mid \exists b\colon \forall I\colon \mathbb{A}_W(I) \leq c\,\mathbb{C}_W(I) + b\}\}\\
&= \lim_{k\to\infty}\{c_u(\mathbb{A},\mathbb{C})\}\\
&= c_u^\infty(\mathbb{A},\mathbb{C}).
\end{aligned}$$

In the above, we use the fact that the c_1's and c_2's can be assumed to be non-negative.

A similar argument shows that $c_l^\infty(\mathbb{A},\mathbb{C}) < 1$.

In the following, we first show an upper bound on how much better any on-line algorithm can be when compared to LRU on their respective worst permutations of any request sequence. This is basically an observation that the proof in [6], showing there is a limit to how much LIFO can be better than LRU (whereas LRU can be unboundedly better than LIFO), holds for any algorithm $\mathbb{A}$. Next, we show that $\mathrm{WR}_{\mathrm{LRU,LRU\text{-}2}}$ is defined, with LRU-2 being at least as good as LRU, and that LRU-2 achieves this largest possible upper bound with respect to LRU. Finally, we show that LRU-K ($K \geq 3$) and LRU are resource-asymptotically comparable in LRU-K's favor.

Notice that LRU and LRU-K perform very similarly on many sequences, especially the cyclic sequences with very long periods and no repeated requests within each period. These sequences are disastrous for both algorithms, causing them to fault on every request. Such sequences have essentially the same number of requests to every page. In their intuitive justification for why LRU-2 may perform better than LRU, [18] mentions, for example, database applications where, when searching in a tree, the root node is accessed much more than other nodes. This leads to long, cyclic-like request sequences, but the nodes near the root of the tree occur in many "cycles", and other nodes do not. On the worst case ordering with this content, any algorithm would have to fault on the new pages in every cycle. LRU, however, would fault on nearly all pages, while LRU-K can keep the pages which occur most often in the entire sequence, the pages corresponding to the nodes near the root, in cache. Thus, the sequences which actually occur in this application are worst orderings of the sequences (for both algorithms). Intuitively, relative worst order analysis, which compares different algorithms on their worst orderings of sequences with the same content, could give interesting results in this case.

Next, we extend the theorem in [6] stating that for any input sequence I there exists a worst permutation of I with respect to LRU where all faults appear before all hits to also hold for LRU-K in general. Recall that LRU-1 $=$ LRU.

Lemma 4 *For any request sequence I, there exists a worst ordering of I with respect to* LRU-K *with all faults appearing before all hits.*

Proof The proof is done by contradiction. For some K and any input sequence I, assume that there is no worst ordering of I with respect to LRU-K where all faults appear before all hits.

Among the worst orderings of I with respect to LRU-K, consider the orderings where the first group of hits occurs as late as possible. Among these, let I' be one with as small a first group of hits as possible. Let I' consist of the requests $r_1, r_2, \ldots, r_n$, in that order. Let $r_i, r_{i+1}, \ldots, r_j$ be the first group of hits in I, i.e., $r_1, r_2, \ldots, r_{i-1}$, and r_{j+1} are all faults. Let p denote the page requested by r_i.

In the following, we permute I' to I'' by moving r_i, and possibly other later requests to p, after r_{j+1}. Note that moving requests for p within the sequence only affects p's position in the queue that LRU-K evicts from. The relative order of the other pages stays the same. While moving the requests, we maintain the following two invariants.

1. Immediately before any request r_u to a page different from p, if p is in the cache of I', then it is also in the cache of I''.
   Hence, if a request for a page different from p is a fault in I', then it is also a fault in I'', since the pages not equal to p in I'''s cache are a subset of the pages in I''s cache.
2. There are at least the same number of faults at requests for p in I'' as in I'.

Combined, the two invariants ensure that the cost on I'' is at least as high as on I'. Now, start by setting u equal to i and then permute I' to I'' by repeating these steps:

1. Remove the request for p at r_u in I' (currently a hit).
   For any request r_v, $v > u$, this will make the Kth last request to p less recent in I'' than in I', i.e., Invariant 2 must be true. However, this may also make LRU-K evict p at some point after r_u in I'', where it was not evicted in I', violating Invariant 1. If this is not the case, simply insert p after the last request r_n in I'', and stop the procedure.
2. Let r_v be the first point at which LRU-K evicts p in I'', but not in I', and insert p immediately after r_v in I'' as a fault.
   For any request r_w, $w > v$, the Kth last request to p is at least as recent in I'' as in I', i.e., Invariant 1 must now hold, but Invariant 2 may be violated. If this is not the case, we stop the procedure. Otherwise, let r_w, $w > v$ be the first request to p that is a hit in I'', but a fault in I'. Set u equal to w and go to Step 1.

Note that when following Step 1 for the first time, $u = i$. Since there are only hits in $r_i, \ldots, r_j$, p is evicted at the earliest at r_{j+1}, i.e., $v > j$. Consequently, either the first group of hits in I'' is one smaller than in I' or (when $i = j$) the first hit occurs later in I'' than in I'. Furthermore, since both invariants hold at the end of the procedure, I'' has at least as many faults as I'. Combining this, we would initially have chosen I'' over I', thereby proving that I' cannot exist.

We now want to prove an upper bound on LRU's performance compared to LRU-2. However, this bound can be established more generally as a bound that holds in comparison with any deterministic paging algorithm.

Theorem 2 *For any deterministic paging algorithm $\mathbb{A}$ and any input sequence I,*

$$\mathrm{LRU}_W(I) \leq \frac{k+1}{2}\mathbb{A}_W(I) + k$$

Proof Suppose there exists a sequence I, where LRU faults s times on its worst permutation, I_{LRU}. As shown in Lemma 4, there exists a worst permutation I_w of I_{LRU} with respect to LRU where all faults appear before all hits. Let I_f be the prefix of I_w consisting of the s faults. Partition the sequence I_f into subsequences of length $k+1$ (except possibly the last which may be shorter). We process these subsequences one at time, possibly reordering some of them, so that $\mathbb{A}$ faults at least twice on all, except possibly the last, thereby proving the result.

The first subsequence is treated specially below. Suppose the first i subsequences have been considered and consider the $(i+1)$st subsequence, $I' = \langle r_1, r_2, \ldots, r_{k+1}\rangle$, of consecutive requests in I_f where $\mathbb{A}$ faults at most once. Since LRU faults on every request, they must be to $k+1$ different pages, $p_1, p_2, ..., p_{k+1}$. Let p be the page requested immediately before I'. Note that $p \neq p_i$ for $1 \leq i \leq k$, since otherwise LRU would not fault on p_i. Clearly, p must be in $\mathbb{A}$'s cache when it starts processing I'.

If r_{k+1} is not a request to p, then I' contains $k+1$ pages different from p, but at most $k-1$ of them are in $\mathbb{A}$'s cache when it starts processing I' (p is in its cache). Hence, $\mathbb{A}$ must fault at least twice on the requests in I'.

On the other hand, if r_{k+1} is a request to p, then there are exactly k requests in I' which are different from p. At least one of them, say p_i, must cause a fault, since at most $k-1$ of them could have been in $\mathbb{A}$'s cache just before it began processing I'. If $\mathbb{A}$ faults on no other page than p_i in I', then all the pages $p, p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k$ must be in $\mathbb{A}$'s cache just before it starts to process I'. Now, move the request to p_i to the beginning of I'. This causes $\mathbb{A}$ to fault and evict one of the pages $p, p_1, p_2, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k$. Hence, it must fault at least one additional time while processing the rest of this reordering of I'.

For the first subsequence I, define p as follows. If p_{k+1} is in the cache from the very beginning, let p denote p_{k+1}. Otherwise, let p denote any page in the initial cache different from p_1 through p_k. Note that such a page must exist, since otherwise LRU would not fault on these pages. Now, proceed as above.

In preparation for showing that LRU is always at least as bad as LRU-2, up to an additive constant, we need the following knowledge of worst orderings for LRU.

Lemma 5 *For any input sequence I of length at least $k+1$, if each page requested in I is requested at most $\left\lfloor \frac{|I|}{k+1} \right\rfloor$ times, then $\mathrm{LRU}_W(I) = |I|$.*

Proof Let I be any request sequence in which each page is requested at most $\left\lfloor \frac{|I|}{k+1} \right\rfloor$ times. Below, I is reordered to I', thereby making LRU fault on all requests.

If $k+1 \leq |I| < 2(k+1)$, then each page is requested once, and LRU can easily be made to fault on all requests, independent of the initial cache.

Otherwise, consider the k pages in the initial cache of LRU in the order that LRU evicts them if they are not requested. Denote by p_k be the first page that is evicted, p_{k-1} the second, etc. Choosing any arbitrary order, denote the pages requested in I that are not initially in the cache by p_{k+1}, p_{k+2}, etc.

Denote the requests in I' by $r_1, r_2, \ldots, r_{|I|}$. In order to assign pages from I to the requests in I', consider the requests in the following order:

$$\begin{array}{l} r_{(k+1)}, r_{2(k+1)}, r_{3(k+1)}, \cdots \\ r_{(k+1)-1}, r_{2(k+1)-1}, r_{3(k+1)-1}, \cdots \\ \cdots \\ r_{(k+1)-k}, r_{2(k+1)-k}, r_{3(k+1)-k}, \cdots \end{array}$$

In the above, all $|I|$ positions in I' are listed. Considering the pages from I in the order p_1, p_2, etc., requests to each of these pages are placed in the above order, i.e., the requests to p_1 are at position r_{k+1}, $r_{2(k+1)}$, etc. Next, the first request to p_2 is at position $j(k+1)$ for some j or possibly at position $r_{(k+1)-1}$, etc. Thus, the assignment continues from left to right and top to bottom.

Note that by using this ordering:

1. Each row of requests contains either $\left\lfloor \frac{|I|}{k+1} \right\rfloor$ or $\left\lceil \frac{|I|}{k+1} \right\rceil$ positions.
2. For any request $r_i = p$, for some p, the next request to p in the same row of the above order is $k+1$ positions away.
3. For any page p which occurs in more than one row in this construction, if the first placed request to p is at position $j(k+1)-m$ for some j and m, then the last placed request is either at position $(j-1)(k+1)-(m+1)$ (that is, $k+2$ positions away from the first request) or even further away from position $j(k+1)-m$ (depending on the number of requests to p in I).
4. If p_1 is requested in I', the first request to this page is at position $k+1$, i.e., after p_1 is evicted. Similarly, for p_i, $i \leq k$, the earliest request in I' occurs no earlier than at position $(k+1)+1-i = k+2-i$ (in row i of the requests), i.e., after p_i has been evicted for the first time.

Hence, LRU faults on all requests in I'.

The following lemma immediately implies that $\mathrm{WR}_{\mathrm{LRU,LRU\text{-}2}}$ is defined, with LRU-2 being at least as good as LRU.

Lemma 6 *For any input sequence I,* $\mathrm{LRU}_W(I) \geq \mathrm{LRU\text{-}2}_W(I) - k$.

Proof Consider any request sequence I. By Lemma 4, there is a worst ordering, I_w, of I with respect to LRU-2 where all faults appear before all hits. Let I' be the maximal prefix of I_w consisting only of faults.

Below, we show that for any page p, the number of requests, s, for this page in I' is at most $M = \left\lceil \frac{|I'|}{k+1} \right\rceil$. I' contains at most $k+1$ pages that are requested M times. If I' contains exactly $k+1$ pages that are requested M times, then by cyclically requesting the $k+1$ pages, LRU faults on all request. Otherwise, by Lemma 5, I' can be reordered such that LRU faults on all requests, except for one request for each of the at most k pages that are requested exactly M times.

Now, for any page p, denote the requests for p by $p_1, p_2, \ldots, p_s$, where p_1 is the first request for p in I', p_2 is the second request for p in I', etc. For any i, denote by I'_i the requests between p_{i-1} and p_i in I'. I'_1 is defined to be the requests in front of the first request for p.

$$\langle \ldots, p_{i-2}, \underbrace{\ldots\ldots}_{I'_{i-1}}, p_{i-1}, \underbrace{\ldots\ldots}_{I'_i}, p_i, \underbrace{\ldots\ldots}_{I'_{i+1}}, p_{i+1}, \ldots \rangle$$

Define $m_j(I'_i)$ to be the number of pages requested at least j times in I'_i. Thus, $|I'_i| = \sum_{j=1}^{\infty} m_j(I'_i)$.

Next, we show that in total we are at most k requests short of associating at least k requests for pages different from p to each request for p. The proof is carried out by induction on prefixes of I'. Let e_i be the number of requests we are short of having associated k requests for pages different from p to each request for p after having processed the prefix of I' up until and including p_i. In the following, we prove that for any i, $e_i \leq k - m_2(I'_i)$. Note that e_i can be negative.

For $i = 1$, we are $k - |I'_1|$ requests short, and

$$e_1 = k - |I'_1| \leq k - m_2\left(I'_1\right) .$$

For the inductive step $i \geq 2$, first note that since p_i is a fault, we must have

$$k - m_2\left(I'_{i-1}\right) \leq m_1\left(I'_i\right) ,$$

since immediately after p_{i-1}, $m_2\left(I'_{i-1}\right)$ pages in cache have been requested at least twice more recently than p, hence at least $k - m_2\left(I'_{i-1}\right)$ different pages have to be requested in I'_i to evict p and cause p_i to be a fault.

Next,

$$\begin{aligned}
e_i &= e_{i-1} + k - |I'_i| \\
&\leq k - m_2\left(I'_{i-1}\right) + k - \sum_{j=1}^{\infty} m_j\left(I'_i\right) \\
&\leq m_1\left(I'_i\right) + k - \sum_{j=1}^{\infty} m_j\left(I'_i\right) \\
&\leq m_1\left(I'_i\right) + k - \sum_{j=1}^{2} m_j\left(I'_i\right) \\
&= k - m_2\left(I'_i\right) .
\end{aligned}$$

By the above, for any i, $e_i \leq k - m_2\left(I'_i\right) \leq k$, i.e., we are in total at most k requests short. When we also count the requests to p, we have $s(k+1) - k \leq |I'|$. Since s is an integer, by isolating s we get

$$s \leq \left\lfloor \frac{|I'| + k}{k+1} \right\rfloor \leq \left\lceil \frac{|I'|}{k+1} \right\rceil ,$$

thereby proving the result.

Next, we show that for any $K' < K$, LRU-K can perform significantly better than LRU-K$'$ on some sets of input. Since LRU = LRU-1, this also implies that LRU-2 can perform better than LRU.

Lemma 7 *For any* $1 \leq K' < K$*, there exists a family of sequences* I_n *of page requests and a constant* b *such that*

$$\text{LRU-K}'_W(I_n) \geq \frac{k+1}{2}\text{LRU-K}_W(I_n) - b,$$

and $\lim_{n\to\infty} \text{LRU-K}'_W(I_n) = \infty$.

Proof Let I_n consist of $k-1$ different pages, $p_1, p_2, \ldots, p_{k-1}$, each requested $nK' + 1$ times and $2n$ other different pages, $q_1, q_2, \ldots, q_{2n}$, each requested K' times. For large n, $nK' + 1 > K$.

Note that there are only $k-1$ pages requested K or more times. It follows that for any page p_i, LRU-K cannot fault on any request for p_i after it has had K requests to p_i. Hence,

$$\text{LRU-K}_W(I_n) \leq 2nK' + K(k-1).$$

For LRU-K$'$, we arrange the requests in the following way:

$$\begin{aligned}
&\langle (p_1, p_2, \ldots, p_{k-1}, q_1, q_2)^{K'}, \\
&p_1, p_2, \ldots, p_{k-1}, \\
&(q_3, q_4)^{K'}, (p_1, p_2, \ldots, p_{k-1})^{K'}, \\
&(q_5, q_6)^{K'}, (p_1, p_2, \ldots, p_{k-1})^{K'}, \\
&\ldots, \\
&(q_{2n-1}, q_{2n})^{K'}, (p_1, p_2, \ldots, p_{k-1})^{K'} \rangle
\end{aligned}$$

Note, for example, that each time q_3 occurs, except for the first, q_4 is in cache, and is the only page in cache which has not been requested K' times, so it is evicted. The first time, q_1 has the least recent K'th oldest request, so it is evicted. In all cases, one concludes that the next page requested is not in cache and LRU-K' faults on all the requests. It follows that

$$\begin{aligned}\text{LRU-K}'_W(I_n) &= (k-1)(nK'+1)+2nK' \\ &= (k+1)nK'+k-1.\end{aligned}$$

Asymptotically, the ratio is $\frac{k+1}{2}$.

Combining results from above, we obtain the following theorem:

Theorem 3 $\text{WR}_{\text{LRU,LRU-2}} = \frac{k+1}{2}$.

Proof By Lemma 6, for all any input sequence I, $\text{LRU}_W(I) \geq \text{LRU-2}_W(I) - k$. Thus, by Definition 1, $c_l(\text{LRU},\text{LRU-2}) \geq 1$. Again by Definition 1, this means that $\text{WR}_{\text{LRU,LRU-2}} = c_u(\text{LRU},\text{LRU-2})$.

Using LRU-2 with Theorem 2, $c_u(\text{LRU},\text{LRU-2}) \leq \frac{k+1}{2}$. Lemma 7 with $K' = 1$ and $K = 2$ shows that the infimum cannot be any smaller, so $\text{WR}_{\text{LRU,LRU-2}} = \frac{k+1}{2}$.

By improving the result in Lemma 6 to hold when comparing any K' and K for $K' < K$, we conjecture that our result can be improved:

Conjecture 1 For $1 \leq K' < K$, $\text{WR}_{\text{LRU-K}',\text{LRU-K}} = \frac{k+1}{2}$.

However, we can prove that LRU-K and LRU are resource-asymptotically comparable in LRU-K's favor. Let I be a sequence where LRU-K faults on every request. We define the LRU-K-phase partition of I as the recursive division of I into LRU-K-phases, where a LRU-K-phase is a maximal subsequence under the restriction that no page is requested more than K times in the subsequence. It is easy to see that a complete LRU-K-phase (a LRU-K-phase not ending on the last request in the input sequence) contains K requests for each of at least k distinct pages, and as a consequence it contains at least $kK+1$ requests for at least $k+1$ distinct pages.

Lemma 8 *For $K \geq 2$, and for any sequence I of page requests,*

$$\text{LRU-K}_W(I) \leq \left(1+\frac{K-1}{K(k+1)}\right)\text{LRU}_W(I).$$

Proof Consider any request sequence I that is processed by LRU-K for $K \geq 2$. By Lemma 4, there exists a worst permutation, $I_{\text{LRU-K}}$, of I such that LRU-K faults on each request of a prefix I_f of $I_{\text{LRU-K}}$ and on no requests after I_f. Partition I_f into LRU-K-phases. We now inductively reorder I_f to gradually obtain a sequence I'_f such that

$$\text{LRU-K}(I_f) \leq \left(1+\frac{K-1}{K(k+1)}\right)\text{LRU}(I'_f).$$

Start at the beginning of I_f and consider the LRU-K-phase starting with the first request not already placed in a processed phase. Each LRU-K-phase contains at least $kK+1$ requests for at least $k+1$ distinct pages. Since each page requested in a LRU-K-phase is requested at most K times, a LRU-K-phase containing at least $K(k+1)$ requests can be partitioned into K sets, each containing requests for at least $k+1$ pages, none of which are repeated. Each of these sets of requests can then be ordered so that LRU faults on every request.

Hence, assume that the current LRU-K-phase is short, i.e., contains fewer than $K(k+1)$ requests. Let $P_K = \{p_1, p_2, \ldots, p_k\}$ denote the k pages requested K times in the phase. Now, observe that after a complete LRU-K-phase, LRU-K has at least $k-1$ of the pages in P_K in cache, since, by the definition of LRU-K-phases, the first request just after the LRU-K-phase is for the one page, say p_j, from P_K that is not currently in cache. Hence, after that request, all the pages in P_K are in cache. It follows that the next request is for a page, say q, not requested K times in the current phase (not in P_K). By moving the request for p_j to the end of the request sequence, it follows from the above that the modified phase in question now contains at least one more request, the request for q, and no page is requested more than K times in the modified phase (q was not in P_K). By the above argument, it follows that we can add one request to the current short phase by moving one request to the end of the sequence. Finally, observe that a short phase is at most $K-1$ requests short of $K(k+1)$. Hence, for each short phase, we only need to move at most $K-1$ requests to the end of the sequence, where they might become hits, to obtain phases of length $K(k+1)$.

Let l denote the total number of modified phases. For each modified phase i, there are $s_i \geq K(k+1)$ requests, plus possibly at most $K-1$ additional requests that LRU-K faulted on and that have been moved to the end. Thus, LRU faults at least $\sum_{i=1}^{l} s_i$ times and LRU-K faults at most $\sum_{i=1}^{l}(s_i+K-1)$ times. It follows that

$$\begin{aligned}
\text{LRU-K}_W(I) &\leq \frac{\sum_{i=1}^{l}(s_i+K-1)}{\sum_{i=1}^{l} s_i}\text{LRU}_W(I)\\
&\leq \frac{l(K(k+1)+K-1)}{l(K(k+1))}\text{LRU}_W(I)\\
&= \left(1+\frac{K-1}{K(k+1)}\right)\text{LRU}_W(I).
\end{aligned}$$

Combining Lemma 7 and the lemma above we arrive at the following:

Theorem 4 LRU-K *and* LRU *are resource-asymptotically comparable in* LRU-K*'s favor.*

Proof By Lemma 8, for any I,

$$\text{LRU}_W(I) \geq \frac{1}{1+\frac{K-1}{K(k+1)}}\text{LRU-K}_W(I).$$

For $k \to \infty$, the constant in front of LRU-K$_W(I)$ approaches one from below. Thus, $c_l^\infty(\text{LRU}, \text{LRU-K}) \geq 1$. By Definition 2, then $\text{WR}^\infty_{\text{LRU,LRU-K}} = c_u^\infty(\text{LRU}, \text{LRU-K})$.

By Lemma 7, $c_u(\text{LRU}, \text{LRU-K})$ must be at least $\frac{k+1}{2}$, so $c_u^\infty(\text{LRU}, \text{LRU-K}) = \infty$. By definition, LRU and LRU-K are then resource-asymptotically comparable in LRU-K's favor.

5 Concluding Remarks

The relative worst order ratio is a worst case measure, considering sequences with the same content together. Since these worst case orderings have a tendency to occur in certain database applications, this measure is quite appropriate in these cases. In contrast to the results using competitive analysis, relative worst order analysis yields a theoretical justification for the superiority of LRU-2 over LRU, confirming previous empirical evidence.

An interesting open problem is whether the result that LRU-K is resource-asymptotically comparable to LRU in LRU-K's favor can be strengthened to say that LRU-K is comparable to LRU in LRU-K's favor with respect to relative worst order analysis.

Note that, as is the case with resource-asymptotically comparable, the relative worst order ratio is also transitive, i.e, for any three on-line algorithms A, B, and C, if A and B are comparable in favor of A, and B and C are comparable in favor of B, then A and C will be comparable in favor of A. Thus, the results from [6], showing that LRU is at least as good as any conservative algorithm and better than Flush-When-Full (FWF), combined with the results proven here, show that LRU-2 is comparable to any conservative algorithm and FWF, in LRU-2's favor in each case. Similarly, by Lemma 3, LRU-K ($K \geq 3$) is resource-asymptotically comparable to any conservative algorithm and FWF, in LRU-K's favor.

An algorithm called RLRU was proposed in [6] and shown to be better than LRU using relative worst order analysis. We conjecture that LRU-2 is also comparable to RLRU in LRU-2's favor. We have found a family of sequences showing that LRU-2 can be better than RLRU, but would also like to show that the algorithms are comparable.

Acknowledgements The authors would like to thank Peter Sanders for bringing LRU-K to their attention and thank the anonymous referees for constructive comments.

References

1. Susanne Albers. Online Algorithms: A Survey. *Mathematical Programming*, 97(1–2):3–26, 2003.
2. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
3. Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive Paging with Locality of Reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995.
4. Joan Boyar, Martin R. Ehmsen, and Kim S. Larsen. Theoretical Evidence for the Superiority of LRU-2 over LRU for the Paging Problem. In *Proceedings of the 4th International Workshop on Approximation and Online Algorithms*, volume 4368 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2006.
5. Joan Boyar and Lene M. Favrholdt. The Relative Worst Order Ratio for On-Line Algorithms. *ACM Transactions on Algorithms*, 3(2), 2007.
6. Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. The Relative Worst Order Ratio Applied to Paging. *Journal of Computer and System Sciences*, 73(5):818–843, 2007.
7. Joan Boyar and Paul Medvedev. The Relative Worst Order Ratio Applied to Seat Reservation. *ACM Transactions on Algorithms*, 4(4), 2008.
8. Reza Dorrigiv, Martin R. Ehmsen, and Alejandro López-Ortiz. Parameterized analysis of paging and list update algorithms. In *Proceedings of the 7th International Workshop on Approximation and Online Algorithms*, volume 5893 of *Lecture Notes in Computer Science*. Springer, 2010. Accepted for publication.
9. Reza Dorrigiv and Alejandro López-Ortiz. A Survey of Performance Measures for On-line Algorithms. *SIGACT News*, 36(3):67–81, 2005.
10. Reza Dorrigiv, Alejandro López-Ortiz, and J. Ian Munro. On the Relative Dominance of Paging Algorithms. *Theoretical Computer Science*, 410:3694–3701, 2009.
11. Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. Comparing Online Algorithms for Bin Packing Problems. *Journal of Scheduling*. Accepted for publication.
12. Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. Separating Scheduling Algorithms with the Relative Worst Order Ratio. *Journal of Combinatorial Optimization*, 12(4):362–385, 2006.
13. Amos Fiat and Ziv Rosen. Experimental Studies of Access Graph Based Heuristics: Beating the LRU Standard? In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 63–72, 1997.
14. Theodore Johnson and Dennis Shasha. 2q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 439–450. Morgan Kaufmann Publishers Inc., 1994.
15. Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive Snoopy Caching. *Algorithmica*, 3(1):79–119, 1988.
16. Sven Oliver Krumke, Willem de Paepe, Jörg Rambau, and Leen Stougie. Bincoloring. *Theoretical Computer Science*, 407(1–3):231–241, 2008.
17. Nimrod Megiddo and Dharmendra S. Modha. Arc: A self-tuning, low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 115–130. USENIX, 2003.
18. Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 297–306, 1993.
19. Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. An optimality proof of the LRU-K page replacement algorithm. *Journal of the ACM*, 46(1):92–112, 1999.
20. Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202–208, 1985.
21. Gerhard Weikum, Christof Hasse, Axel Mönkeberg, and Peter Zabback. The comfort automatic tuning project. *Information Systems*, 19(5):381–432, 1994.
22. Neal Young. The k-Server Dual and Loose Competitiveness for Paging. *Algorithmica*, 11(6):525–541, 1994.