Relative Worst-Order Analysis: A Survey

Joan Boyar(✉), Lene M. Favrholdt, and Kim S. Larsen

University of Southern Denmark, Odense, Denmark
{joan,lenem,kslarsen}@imada.sdu.dk

Abstract. Relative worst-order analysis is a technique for assessing the relative quality of online algorithms. We survey the most important results obtained with this technique and compare it with other quality measures.

1 Introduction

Online problems are optimization problems where the input arrives one request at a time, and each request must be processed without knowledge of future requests. The investigation of online algorithms was largely initiated by the introduction of competitive analysis by Sleator and Tarjan [61]. They introduced the method as a general analysis technique, inspired by approximation algorithms. The term "competitive" is from Karlin et al. [51] who named the worst-case ratio of the performance of the online to the offline algorithm the "competitive ratio". Many years earlier, Graham carried out what is now viewed as an example of a competitive analysis [44].

The over-all goal of a theoretical quality measure is to predict behavior of algorithms in practice. In that respect, competitive analysis works well in some cases, but, as pointed out by the inventors [61] and others, fails to discriminate between good and bad algorithms in other cases. Ever since its introduction, researchers have worked on improving the measure, defining variants, or defining measures based on other concepts to improve on the situation. Relative worst-order analysis (RWOA), a technique for assessing the relative quality of online algorithms, is one of the most thoroughly tested such proposals.

RWOA was originally defined by Boyar and Favrholdt [17], and the definitions were extended together with Larsen [20]. As for all quality measures, an important issue is to be able to separate algorithms, i.e., determine which of two algorithms is the best. RWOA has been shown to be applicable to a wide variety of problems and provide separations, not obtainable using competitive analysis, corresponding better to experimental results or intuition in many cases.

In this survey, we motivate and define RWOA, outline the background for its introduction, survey the most important results, and compare it to other measures.

Supported in part by the Independent Research Fund Denmark, Natural Sciences, grants DFF-7014-00041.

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 216–230, 2018.
https://doi.org/10.1007/978-3-319-98355-4_13

2 Relative Worst-Order Analysis

As a motivation for RWOA, consider the following desirable property of a quality measure for online algorithms: For a given problem P and two algorithms $\mathbb{A}$ and $\mathbb{B}$ for P, if $\mathbb{A}$ performs at least as well as $\mathbb{B}$ on every possible request sequence and better on many, then the quality measure indicates that $\mathbb{A}$ is better than $\mathbb{B}$. We consider an example of such a situation for the paging problem.

2.1 A Motivating Example

In the paging problem, there is a cache with k pages and a larger, slow memory with $N > k$ pages. The request sequence consists of page numbers in $\{1, \ldots, N\}$. When a page is requested, if it is not among the at most k pages in cache, there is a fault, and the missing page must be brought into cache. If the cache is full, this means that some page must be evicted from the cache. The goal is to minimize the number of faults. Clearly, the only thing we can control algorithmically is the eviction strategy.

We consider two paging algorithms, LRU (Least-Recently-Used) and FWF (Flush-When-Full). On a fault with a full cache, LRU always evicts its least recently used page from cache. FWF, on the other hand, evicts everything from cache in this situation. It is easy to see that, if run on the same sequence, whenever LRU faults, FWF also faults, so LRU performs at least as well as FWF on every sequence. LRU usually faults less than FWF [64]. It is well known that LRU and FWF both have competitive ratio k, so competitive analysis does not distinguish between them, and there are relatively few measures which do. RWOA, however, is one such measure [20]. In Subsect. 3.1, we consider LRU and FWF in greater detail to give a concrete example of RWOA.

2.2 Background and Informal Description

Table 1 gives informal "definitions" of the relative worst-order ratio and related measures. The ratios shown in the table capture the general ideas, although they do not reflect that the measures are asymptotic measures. We discuss the measures below, ending with a formal definition of the relative worst-order ratio.

RWOA compares two online algorithms directly, rather than indirectly by first comparing both to an optimal offline algorithm. When differentiating between online algorithms is the goal, performing a direct comparison between the algorithms can be an advantage; first comparing both to an optimal offline algorithm and then comparing the results, as many performance measures including competitive analysis do, can lead to a loss of information. This appears to be at least part of the problem when comparing LRU to FWF with competitive analysis, which finds them equally bad. Measures comparing directly, such as RWOA, bijective and average analysis [5], and relative interval analysis [37], would generally indicate correctly that LRU is the better algorithm.

Table 1. Simplified "definitions" of measures

Measure	Value
Competitive ratio	$\mathrm{CR}_{\mathbb{A}} = \sup_I \frac{\mathbb{A}(I)}{\textsc{Opt}(I)}$
Max/max ratio	$\mathrm{MR}_{\mathbb{A}} = \frac{\max_{\lvert I\rvert=n} \mathbb{A}(I)}{\max_{\lvert I'\rvert=n} \textsc{Opt}(I')}$
Random-order ratio	$\mathrm{RR}_{\mathbb{A}} = \sup_I \frac{E_\pi\left[\mathbb{A}(\pi(I))\right]}{\textsc{Opt}(I)}$
Relative worst-order ratio	$\mathrm{WR}_{\mathbb{A},\mathbb{B}} = \sup_I \frac{\sup_\pi \left\{\mathbb{A}(\pi(I))\right\}}{\sup_{\pi'} \left\{\mathbb{B}(\pi'(I))\right\}}$

Up to permutations of the request sequences, if an algorithm is always at least as good and sometimes better than another, RWOA separates them. RWOA compares two algorithms on their respective worst orderings of sequences having the same content. This is different from competitive analysis where an algorithm and Opt are compared on the same sequence. When comparing algorithms directly, using exactly the same sequences will tend to produce the result that many algorithms are not comparable, because one algorithm does well on one type of sequence, while the other does well on another type. In addition, comparing on possibly different sequences can make it harder for the adversary to produce unwanted, pathological sequences which may occur seldom in practice, but skew the theoretical results. Instead, with RWOA, online algorithms are compared directly to each other on their respective worst permutations of the request sequences. This comparison in RWOA combines some of the desirable properties of the max/max ratio [9] and the random-order ratio [52].

The Max/Max Ratio. With the max/max ratio defined by Ben-David and Borodin, an algorithm is compared to Opt on its and Opt's respective worst-case sequences of the same length. Since Opt's worst sequence of any given length is the same, regardless of which algorithm it is being compared to, comparing two online algorithms directly gives the same result as dividing their max/max ratios. Thus, the max/max ratio allows direct comparison of two online algorithms, to some extent, without the intermediate comparison to Opt. The max/max ratio can only provide interesting results when the length of an input sequence yields a bound on the cost/profit of an optimal solution.

In the paper [9] introducing the max/max ratio, the k-server problem is analyzed. This is the problem where k servers are placed in a metric space, and the input is a sequence of requests to points in that space. At each request, a server must be moved to the requested point if there is not already a server at the point. The objective is to minimize the total distance the servers are moved. It is demonstrated that, for k-server on a bounded metric space, the max/max ratio can provide more optimistic and detailed results than competitive analysis.

Unfortunately, there is still the loss of information as generally occurs with the indirect comparison to Opt, and the max/max ratio does not distinguish between LRU and FWF, or actually between any two deterministic online paging algorithms.

However, the possibility of directly comparing online algorithms and comparing them on their respective worst-case sequences from some partition of the space of request sequences was inspirational. RWOA uses a more fine-grained partition than partitioning with respect to the sequence length. The idea for the specific partition used stems from the random-order ratio.

The Random-Order Ratio. The random-order ratio was introduced in [52] by Kenyon (now Mathieu). The appeal of this quality measure is that it allows considering some randomness of the input sequences without specifying a complete probability distribution. It was introduced in connection with bin packing, i.e., the problem of packing items of sizes between 0 and 1 into as few bins of size 1 as possible. For an algorithm $\mathbb{A}$ for this minimization problem, the random-order ratio is the maximum ratio, over all multi-sets of items, of the expected performance, over all permutations of the multi-set, of $\mathbb{A}$ compared with an optimal solution; see also Table 1. If, for all possible multi-sets of items, any permutation of these items is equally likely, this ratio gives a meaningful worst-case measure of how well an algorithm can perform.

In the paper introducing the random-order ratio, it was shown that for bin packing, the random-order ratio of Best-Fit lies between 1.08 and 1.5. In contrast, the competitive ratio of Best-Fit is 1.7 [49].

Random-order analysis has also been applied to other problems, e.g., knapsack [7], bipartite matching [35,43], scheduling [42,59], bin covering [29,41], and facility location [56]. However, the analysis is often rather challenging, and in [32], a simplified version of the random-order ratio is used for bin packing.

2.3 Definitions

Let I be a request sequence of length n for an online problem P. If π is a permutation on n elements, then $\pi(I)$ denotes I permuted by π.

If P is a minimization problem, $\mathbb{A}(I)$ denotes the cost of the algorithm $\mathbb{A}$ on the sequence I, and

$$\mathbb{A}_W(I) = \max_{\pi} \mathbb{A}(\pi(I)),$$

where π ranges over the set of all permutations of n elements.

If P is a maximization problem, $\mathbb{A}(I)$ denotes the profit of the algorithm $\mathbb{A}$ on the sequence I, and

$$\mathbb{A}_W(I) = \min_{\pi} \mathbb{A}(\pi(I)).$$

Informally, RWOA compares two algorithms, $\mathbb{A}$ and $\mathbb{B}$, by partitioning the set of request sequences as follows: Sequences are in the same part of the partition

if and only if they are permutations of each other. The relative worst-order ratio is defined for algorithms $\mathbb{A}$ and $\mathbb{B}$, whenever one algorithm performs at least as well as the other on every part of the partition, i.e., whenever $\mathbb{A}_W(I) \leq \mathbb{B}_W(I)$, for all request sequences I, or $\mathbb{A}_W(I) \geq \mathbb{B}_W(I)$, for all request sequences I (in the definition below, this corresponds to $c_u(\mathbb{A}, \mathbb{B}) \leq 1$ or $c_l(\mathbb{A}, \mathbb{B}) \geq 1$). In this case, to compute the relative worst-order ratio of $\mathbb{A}$ to $\mathbb{B}$, we compute a bound ($c_l(\mathbb{A}, \mathbb{B})$ or $c_u(\mathbb{A}, \mathbb{B})$) on the ratio of how the two algorithms perform on their respective worst permutations of some sequence. Note that the two algorithms may have different worst permutations for the same sequence.

We now state the formal definition:

Definition 1. *For any pair of algorithms* $\mathbb{A}$ *and* $\mathbb{B}$*, we define*

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

If $c_l(\mathbb{A}, \mathbb{B}) \geq 1$ *or* $c_u(\mathbb{A}, \mathbb{B}) \leq 1$*, 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 as*

$$\mathrm{WR}_{\mathbb{A},\mathbb{B}} = \begin{cases} c_u(\mathbb{A}, \mathbb{B}), & \text{if } c_l(\mathbb{A}, \mathbb{B}) \geq 1, \text{ and} \\ c_l(\mathbb{A}, \mathbb{B}), & \text{if } c_u(\mathbb{A}, \mathbb{B}) \leq 1. \end{cases}$$

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

For a minimization (maximization) problem, the algorithms $\mathbb{A}$ *and* $\mathbb{B}$ *are said to be* comparable in $\mathbb{A}$'s favor *if* $\mathrm{WR}_{\mathbb{A},\mathbb{B}} < 1$ *(*$\mathrm{WR}_{\mathbb{A},\mathbb{B}} > 1$*). Similarly, the algorithms are said to be* comparable in $\mathbb{B}$'s favor*, if* $\mathrm{WR}_{\mathbb{A},\mathbb{B}} > 1$ *(*$\mathrm{WR}_{\mathbb{A},\mathbb{B}} < 1$*).*

Note that the ratio $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ can be larger than or smaller than one depending on whether the problem is a minimization problem or a maximization problem and which of $\mathbb{A}$ and $\mathbb{B}$ is the better algorithm. Table 2 indicates the result in each case. Instead of saying that two algorithms, $\mathbb{A}$ and $\mathbb{B}$, are comparable in $\mathbb{A}$'s favor, one would often just say that $\mathbb{A}$ is better than $\mathbb{B}$ according to RWOA.

Table 2. Relative worst-order ratio interpretation, depending on whether the problem is a minimization or a maximization problem.

Result	Minimization	Maximization
$\mathbb{A}$ better than $\mathbb{B}$	<1	>1
$\mathbb{B}$ better than $\mathbb{A}$	>1	<1

For quality measures evaluating algorithms by comparing them to each other directly, it is particularly important to be transitive: If $\mathbb{A}$ and $\mathbb{B}$ are comparable in $\mathbb{A}$'s favor and $\mathbb{B}$ and $\mathbb{C}$ are comparable in $\mathbb{B}$'s favor, then $\mathbb{A}$ and $\mathbb{C}$ are comparable in $\mathbb{A}$'s favor. When this transitivity holds, to prove that a new algorithm is better than all previously known algorithms, one only has to prove that it is better than the best among them. This holds for RWOA [17].

3 Paging

In this section, we survey the most important RWOA results for paging and explain how they differ from the results obtained with competitive analysis. As a relatively simple, concrete example of RWOA, we first explain how to obtain the separation of LRU and FWF [20] mentioned in Subsect. 2.1.

3.1 LRU vs. FWF

The first step in computing the relative worst-order ratio, $\mathrm{WR}_{\mathrm{FWF,LRU}}$, is to show that LRU and FWF are comparable. Consider any request sequence I for paging with cache size k. For any request r to a page p in I, if LRU faults on r, either p has never been requested before or there have been at least k different requests to distinct pages other than p since the last request to p. In the case where p has never been requested before, any online algorithm faults on r. If there have been at least k requests to distinct pages other than p since the last request to p, FWF has flushed since that last request to p, so p is no longer in its cache and FWF faults, too. Thus, for any request sequence I, $\mathrm{FWF}(I) \geq \mathrm{LRU}(I)$. Consider LRU's worst ordering, I_{LRU}, of a sequence I. Since FWF's performance on its worst ordering of any sequence is at least as bad as its performance on the sequence itself, $\mathrm{FWF}_W(I_{\mathrm{LRU}}) \geq \mathrm{FWF}(I_{\mathrm{LRU}}) \geq \mathrm{LRU}(I_{\mathrm{LRU}}) = \mathrm{LRU}_W(I_{\mathrm{LRU}})$. Thus, $c_l(\mathrm{FWF}, \mathrm{LRU}) \geq 1$.

As a remark, in general, to prove that one algorithm is at least as good as another on their respective worst orderings of all sequences, one usually starts with an arbitrary sequence and its worst ordering for the better algorithm. Then, that sequence is gradually permuted, starting at the beginning, so that the poorer algorithm does at least as badly on the permutation being created.

The second step is to show the separation, giving a lower bound on the term $c_u(\mathrm{FWF}, \mathrm{LRU})$. We assume that the cache is initially empty. Consider the sequence $I_s = \langle 1, 2, \ldots, k, k+1, k, \ldots, 2 \rangle^s$, where FWF faults on all $2ks$ requests. LRU only faults on $2s + k - 1$ requests in all, the first k requests and every request to 1 or $k + 1$ after that, but we need to consider how many times LRU faults on its worst ordering of I_s.

It is proven in [20] that, for any sequence I, there is a worst ordering of I for LRU that has all faults before all hits (requests which are not faults). The idea is to consider any worst order of I for LRU and move requests which are hits, but are followed by a fault towards the end of the sequence without decreasing the number of faults. Since LRU needs k distinct requests between two requests to the same page in order to fault, with only $k+1$ distinct pages in all, the faults at the beginning must be a cyclic repetition of the $k + 1$ pages. Thus, a worst ordering of I_s for LRU is $I'_s = \langle 2, 3, \ldots, k, k+1, 1 \rangle^s, \langle 2, \ldots, k \rangle^s$, and $\mathrm{LRU}(I'_s) = (k+1)s+k-1$. This means that, asymptotically, $c_u(\mathrm{FWF}, \mathrm{LRU}) \geq \frac{2k}{k+1}$. We now know that $\mathrm{WR}_{\mathrm{FWF,LRU}} \geq \frac{2k}{k+1}$, showing that FWF and LRU are comparable in LRU's favor, which is the most interesting piece of information.

However, one can prove that this is the exact result. In the third step, we prove that $c_u(\mathrm{FWF}, \mathrm{LRU})$ cannot be larger than $\frac{2k}{k+1}$, asymptotically. In fact,

this is shown in [20] by proving the more general result that, for any *marking* algorithm [13], $\mathbb{M}$, and for any request sequence I, $\mathbb{M}_W(I) \leq \frac{2k}{k+1}\text{LRU}_W(I) + k$. A marking algorithm is defined with respect to *k-phases*, a partitioning of the request sequence. Starting at the beginning of I, the first phase ends with the request immediately preceding the $(k+1)$st distinct page, and succeeding phases are also longest intervals containing at most k distinct pages. An algorithm is a marking algorithm if, assuming we mark a page each time it is requested and start with no pages marked at the beginning of each phase, the algorithm never evicts a marked page. As an example, FWF is a marking algorithm. Now, consider any sequence, I, with m k-phases. A marking algorithm $\mathbb{M}$ faults at most km times on I. Any two consecutive k-phases in I contain at least $k+1$ pages, so there must be a permutation of the sequence where LRU faults at least $k+1$ times on the requests of each of the $\lfloor \frac{m}{2} \rfloor$ consecutive pairs of k-phases in I. This gives the desired asymptotic upper bound, showing that $\text{WR}_{\text{FWF,LRU}} = \frac{2k}{k+1}$.

3.2 Other Paging Algorithms

Like LRU and FWF, the algorithm FIFO also has competitive ratio k [12]. FIFO simply evicts the first page that entered the cache, regardless of its use while in cache. In experiments, both LRU and FIFO are consistently much better than FWF. LRU and FIFO are both *conservative* algorithms [64], meaning that on any sequence of requests to at most k different pages, each of them faults at most k times. This means that, according to RWOA, they are equally good and both are better than FWF, since for any pair of conservative paging algorithms, $\mathbb{A}$ and $\mathbb{B}$, $\text{WR}_{\mathbb{A},\mathbb{B}} = 1$ and $\text{WR}_{\text{FWF},\mathbb{A}} = \frac{2k}{k+1}$ [20].

With a quality measure that separates FWF and LRU, an obvious question to ask is: Is there a paging algorithm which is better than LRU according to RWOA? The answer to this is "yes". LRU-2 [58], which was proposed for database disk buffering, is the algorithm which evicts the page with the earliest second-to-last request. LRU-2 and LRU are $(1 + \frac{1}{2k+2}, \frac{k+1}{2})$-related. This concept was introduced in [20], expressing that $c_u(\text{LRU-2}, \text{LRU}) = 1 + \frac{1}{2k+2}$ and $c_u(\text{LRU}, \text{LRU-2}) = \frac{k+1}{2}$ (see Definition 1 for a definition of c_u). Thus, the algorithms are *asymptotically comparable* in LRU-2's favor [14].

In addition, a new algorithm, RLRU (Retrospective LRU), was defined in [20] and shown to be better than LRU according to RWOA. Experiments, simply comparing the number of page faults on the same input sequences, have shown that RLRU is consistently slightly better than LRU [57]. RLRU is a phase-based algorithm. When considering a request, it determines whether Opt would have had the page in cache given the sequence seen so far (this is efficiently computable), and uses that information in a marking procedure.

Interestingly, LRU-2 has competitive ratio $2k$ and RLRU has competitive ratio $k+1$, so both are worse than LRU according to competitive analysis.

Also for paging, considering LRU and LRU(ℓ), which is LRU adapted to use look-ahead ℓ (the next ℓ requests after the current one), evicting a least recently used page *not* occurring in the look-ahead, both algorithms have

competitive ratio k, though look-ahead helps significantly in practice. Using RWOA, $\mathrm{WR}_{\mathrm{LRU,LRU}(\ell)} = \min\{k, \ell + 1\}$, so LRU($\ell$) is better [20].

4 Other Online Problems

In this section, we give further examples of problems and algorithms where RWOA gives results that are qualitatively different from those obtained with competitive analysis. We consider various problems, including list accessing, bin packing, bin coloring, and scheduling.

List accessing [4,61] is a classic problem in data structures, focusing on maintaining an optimal ordering in a linked list. In online algorithms, it also has the rôle of a theoretical benchmark problem, together with paging and a few other problems, on which many researchers evaluate new techniques or quality measures.

The problem is defined as follows: A list of items is given and requests are to items in the list. Treating a request requires accessing the item, and the cost of that access is the index of the item, starting with one. After the access, the item can be moved to any location closer to the front of the list at no cost. In addition, any two consecutive items may be transposed at a cost of one. The objective is to minimize the total cost of processing the input sequence.

We consider three list accessing algorithms: On a request to an item x, the algorithm Move-To-Front (MTF) [55] moves x to the front of the list, whereas the algorithm Transpose (Trans) just swaps x with its predecessor. The third algorithm, Frequency-Count (FC), keeps the list sorted by the number of times each item has been requested.

For list accessing [61], letting l denote the length of the list, the algorithm Move-To-Front has strict competitive ratio $2 - \frac{2}{l+1}$ [47] (referring to personal communication, Irani credits Karp and Raghavan with the lower bound). In contrast, Frequency-Count and Transpose both have competitive ratio $\Omega(l)$ [12]. Extensive experiments demonstrate that MTF and FC are approximately equally good, whereas Trans is much worse [8,10]. Using RWOA, MTF and FC are equally good, whereas both $\mathrm{WR}_{\mathrm{Trans,MTF}} \in \Omega(l)$ and $\mathrm{WR}_{\mathrm{Trans,FC}} \in \Omega(l)$, so Trans is much worse [38].

For bin packing, both Worst-Fit (WF), which places an item in a bin with largest available space (but never opens a new bin unless it has to), and Next-Fit (NF), which closes its current bin whenever an item does not fit (and never considers that bin again), have competitive ratio 2 [48]. However, WF is at least as good as NF on every sequence and sometimes much better [16]. Using RWOA, $\mathrm{WR}_{\mathrm{NF,WF}} = 2$, so WF is the better algorithm.

Bin coloring is a variant of bin packing, where items are unit-sized and each have a color. The goal is to minimize the maximum number of colors in any bin, under the restriction that only a certain number, q, of bins are allowed to be open at any time and a bin is not closed until it is full. Consider the algorithms One-Bin, which never has more than one bin open, and Greedy-Fit, which always keeps q open bins, placing an item in a bin already having that color, if

possible, and otherwise in a bin with fewest colors. We claim that Greedy-Fit is obviously the better algorithm, but if the bin size is larger than approximately q^3, One-Bin has a better competitive ratio than Greedy-Fit [54]. However, according to RWOA, Greedy-Fit is better [40].

For Scheduling on two related machines to minimize makespan (the time when all jobs are completed), the algorithm Fast, which only uses the fast machine, is $\frac{s}{s+1}$-competitive, where s is the speed ratio of the two machines. If s is larger than the golden ratio, this is the best possible competitive ratio. However, the algorithm Post-Greedy, which schedules each job on the machine where it would finish first, is never worse than Fast and sometimes better. This is reflected in the relative worst-order ratio, since $\mathrm{WR}_{\textsc{Fast},\textsc{Post-Greedy}} = \frac{s+1}{s}$ [39].

In addition to these examples, it is widely believed and consistent with experiments that for bin packing problems, First-Fit algorithms perform better than Worst-Fit algorithms, and that processing larger items first is better than processing smaller items first. For the problem examples below, competitive analysis cannot distinguish between the algorithms, that is, they have the same competitive ratio, whereas using RWOA, we get the separation in the right direction. The examples are the following: For dual bin packing (the variant of bin packing where there is a fixed number of bins, the aim is to pack as many items as possible, and all bins are considered open from the beginning), First-Fit is better than Worst-Fit [17]. For grid scheduling (a variant of bin packing where the items are given from the beginning and variable-sized bins arrive online), First-Fit-Decreasing is better than First-Fit-Increasing [18]. For seat reservation (the problem where a train with a certain number of seats travels from station 1 to some station $k \in \mathbb{Z}^+$, requests to travel from some station i to a station $j > i$ arrive online, and the aim is to maximize either the number of passengers or the total distance traveled), First-Fit and Best-Fit are better than Worst-Fit [28] with regards to both objective functions.

5 Approaches to Understanding Online Computation

In this section, we discuss other means of analyzing and thereby gaining insight into online computation. This includes other performance measures and advice complexity.

5.1 Other Performance Measures

Other than competitive analysis, many alternative measures have been introduced with the aim of getting a better or more refined picture of the (relative) quality of online algorithms.

In chronological order, the main contributions are the following: online/online ratio [45], statistical adversary [60], loose competitive ratio [64], max/max ratio [9], access graphs (incorporating locality of reference) [13], random-order ratio [52], accommodating ratio [24], extra resource analysis [50], diffuse adversary

[53], accommodating function [27], smoothed analysis [62], working set (incorporating locality of reference) [2], relative worst-order analysis [17,20], bijective and average analysis [5], relative interval analysis [37], bijective ratio [6], and online-bounded analysis [15].

We are not defining all of these measures here, but we give some insight into the strengths and weaknesses of selected measures in the following. We start with a discussion of work directly targeted at performance measure comparison.

Comparisons of Performance Measures. A systematic comparison of performance measures for online algorithms was initiated in [23], comparing some measures which are applicable to many types of problems. To make this feasible, a particularly simple problem was chosen: the 2-server problem on a line with three points, one point farther away from the middle point than the other.

A well known algorithm, Double Coverage (DC), is 2-competitive and best possible for this problem [30] according to competitive analysis. A lazy version of this, LDC, is at least as good as DC on every sequence and often better. Investigating which measures can make this distinction, LDC was found to be better than DC by bijective analysis and RWOA, but equivalent to DC according to competitive analysis, the max/max ratio, and the random-order ratio. The first proof, for any problem, of an algorithm being best possible under RWOA established this for LDC.

Greedy performs unboundedly worse than LDC on certain sequences, so ideally a performance measure would not find Greedy to be superior to LDC. According to the max/max ratio and bijective analysis, Greedy is the better algorithm, but not according to competitive analysis, random-order analysis, or RWOA.

Further systematic comparisons of performance measures were made in [25,26], again comparing algorithms on relatively simple problems. The paper [25] considered competitive analysis, bijective analysis, average analysis, relative interval analysis, random-order analysis, and RWOA. There were differences between the measures, but the most clear conclusions were that bijective analysis found all algorithms incomparable and average analysis preferred an intuitively poorer algorithm.

Notable omissions from the investigations above are extra resource analysis [50] and the accommodating function [27], both focusing on resources, which play a major rôle in most online problems. Both measures have been applied successfully to a range of problems, giving additional insight; extra resource analysis (also referred to as resource augmentation) has been used extensively. They can both be viewed as extensions of competitive analysis, explaining observed behavior of algorithms by expressing ratios as functions of resource availability.

5.2 Advice Complexity

As a means of analyzing problems, as opposed to algorithms for those problems, *advice complexity* was proposed [11,36,46]. The "no knowledge about the future"

property of online algorithms is relaxed, and it is assumed that some bits of advice are available; such knowledge is available in many situations. One asks how many bits of advice are necessary and sufficient to obtain a given competitive ratio, or indeed optimality. For a survey on advice complexity, see [19].

6 Applicability

Competitive analysis has been used for decades and sophisticated, supplementary analysis techniques have been developed to make proofs more manageable, or with the purpose of capturing more fine-grained properties of algorithms.

We discuss two of the most prominent examples of these supplementary techniques: *list factoring* for analyzing list accessing and *access graphs* for modeling locality of reference for paging. Both techniques have been shown to work with RWOA. As far as we know, list factoring has not been established as applicable to any other alternative to competitive analysis. Access graphs have also been studied for relative interval analysis [22] with less convincing results.

6.1 List Factoring for Analyzing List Accessing

The idea behind *list factoring* is to reduce the analysis to lists of two elements, thereby making the analysis much more manageable. The technique was first introduced by Bentley and McGeoch [10] and later extended and improved [1,3, 47,63]. In order to use this technique, one uses the *partial cost model*, where the cost of each request is one less than in the standard (full) cost model (the access to the item itself is not counted). The list factoring technique is applicable for algorithms where, in treating any request sequence I, one gets the same result by counting only the costs of passing through x or y when searching for y or x (denoted $\mathbb{A}_{xy}(I)$), as one would get if the original list contained only x and y and all requests different from those were deleted from the request sequence, denoted I_{xy}. If this is the case, that is $\mathbb{A}_{xy}(I) = \mathbb{A}(I_{xy})$ for all I, then $\mathbb{A}$ is said to have the *pairwise property*, and it is not hard to prove that then $\mathbb{A}(I) = \sum_{x \neq y} \mathbb{A}(I_{xy})$. Thus, we can reduce the analysis of $\mathbb{A}$ to an analysis of lists of length two. The results obtained also apply in the full cost model if the algorithms are *cost independent*, meaning that their decisions are independent of the costs of the operations.

Since the cost measure is different, some adaption is required to get this to work for RWOA:

We now say that $\mathbb{A}$ has the *worst-order projection property* if and only if, for all sequences I, there exists a worst ordering $\pi_{\mathbb{A}}(I)$ of I with respect to $\mathbb{A}$, such that for all pairs $\{a, b\} \subseteq L$ $(a \neq b)$, $\pi_{\mathbb{A}}(I)_{ab}$ is a worst ordering of I_{ab} with respect to $\mathbb{A}$ on the initial list L_{ab}.

The results on Move-To-Front, Frequency-Count, and Transpose, reported on in Subsect. 3.2, as well as results on Timestamp [1], were obtained [38] using this tool.

6.2 Access Graphs for Modeling Locality of Reference for Paging

Locality of reference refers to the observed behavior of certain sequences from real life, where requests seem to be far from uniformly distributed, but rather exhibit some form of locality; for instance with repetitions of pages appearing in close proximity [33,34]. Performance measures that are worst-case over all possible sequences will usually not reflect this, so algorithms exploiting locality of reference are not deemed better using the theoretical tools, though they may be superior in practice. This has further been underpinned by the following result [5] on bijective analysis: For the class of demand paging algorithms (algorithms that never evict a page unless necessary), for any two positive integers $m, n \in \mathbb{N}$, all algorithms have the same number of input sequences of length n that result in exactly m faults.

One attempt at formalizing locality of reference, making it amenable to theoretical analysis, was made in [13], where *access graphs* were introduced. An access graph is an undirected graph with vertices representing pages and edges indicating that the two pages being connected could be accessed immediately after each other. In the performance analysis of an algorithm, only sequences respecting the graph are considered, i.e., any two distinct, consecutive requests must be to the same page or to neighbors in the graph.

Under this restriction on inputs, [13,31] were able to show that, according to competitive analysis, LRU is strictly better than FIFO on some access graphs and never worse on any graph. Thus, they were the first to obtain a separation, consistent with empirical results.

Using RWOA, [21] proved that on the primary building blocks of access graphs, paths and cycles, LRU is strictly better than FIFO.

7 Open Problems and Future Work

For problems where competitive analysis deems many algorithms best possible or gives counter-intuitive results, comparing algorithms with RWOA can often provide additional information. Such comparisons can be surprisingly easy, since it is often possible to use parts of previous results when applying RWOA.

Often the exploration for new algorithms for a given problem ends when an algorithm is proven to have a best possible competitive ratio. Using RWOA to continue the search for better algorithms after competitive analysis fails to provide satisfactory answers can lead to interesting discoveries. As an example, the paging algorithm RLRU was designed in an effort to find an algorithm that could outperform LRU with respect to RWOA.

Also for the paging problem, RLRU and LRU-2 are both known to be better than LRU according to RWOA. It was conjectured [14] that LRU-2 is comparable to RLRU in LRU-2's favor. This is still unresolved. It would be even more interesting to find a new algorithm better than both. It might also be interesting to apply RWOA to an algorithm from the class of OnOpt algorithms from [57].

For bin packing, it would be interesting to know whether Best-Fit is better than First-Fit, according to RWOA.

Acknowledgment. The authors would like to thank an anonymous referee for many constructive suggestions.

References

1. Albers, S.: Improved randomized on-line algorithms for the list update problem. SIAM J. Comput. **27**(3), 682–693 (1998)
2. Albers, S., Favrholdt, L.M., Giel, O.: On paging with locality of reference. J. Comput. Syst. Sci. **70**(2), 145–175 (2005)
3. Albers, S., von Stengel, B., Werchner, R.: A combined BIT and TIMESTAMP algorithm for the list update problem. Inf. Process. Lett. **56**, 135–139 (1995)
4. Albers, S., Westbrook, J.: Self-organizing data structures. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms. LNCS, vol. 1442, pp. 13–51. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0029563
5. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 229–237 (2007)
6. Angelopoulos, S., Renault, M.P., Schweitzer, P.: Stochastic dominance and the bijective ratio of online algorithms. arXiv arXiv:1607.06132 [cs.DS] (2016)
7. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: A knapsack secretary problem with applications. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) APPROX/RANDOM-2007. LNCS, vol. 4627, pp. 16–28. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74208-1_2
8. Bachrach, R., El-Yaniv, R.: Online list accessing algorithms and their applications: recent empirical evidence. In: 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 53–62 (1997)
9. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica **11**(1), 73–91 (1994)
10. Bentley, J.L., McGeoch, C.C.: Amortized analyses of self-organizing sequential search heuristics. Commun. ACM **28**, 404–411 (1985)
11. Böckenhauer, H.-J., Komm, D., Královic, R., Královic, R., Mömke, T.: Online algorithms with advice: the tape model. Inf. Comput. **254**, 59–83 (2017)
12. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
13. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. J. Comput. Syst. Sci. **50**(2), 244–258 (1995)
14. Boyar, J., Ehmsen, M.R., Kohrt, J.S., Larsen, K.S.: A theoretical comparison of LRU and LRU-K. Acta Inf. **47**(7–8), 359–374 (2010)
15. Boyar, J., Epstein, L., Favrholdt, L.M., Larsen, K.S., Levin, A.: Online-bounded analysis. J. Sched. **21**(4), 429–441 (2018)
16. Boyar, J., Epstein, L., Levin, A.: Tight results for Next Fit and Worst Fit with resource augmentation. Theor. Comput. Sci. **411**(26–28), 2572–2580 (2010)
17. Boyar, J., Favrholdt, L.M.: The relative worst order ratio for on-line algorithms. ACM Trans. Algorithms **3**(2), 24 (2007). Article 22
18. Boyar, J., Favrholdt, L.M.: Scheduling jobs on grid processors. Algorithmica **57**(4), 819–847 (2010)
19. Boyar, J., Favrholdt, L.M., Kudahl, C., Larsen, K.S., Mikkelsen, J.W.: Online algorithms with advice: a survey. ACM Comput. Surv. **50**(2), 19:1–19:34 (2017)
20. Boyar, J., Favrholdt, L.M., Larsen, K.S.: The relative worst order ratio applied to paging. J. Comput. Syst. Sci. **73**(5), 818–843 (2007)

21. Boyar, J., Gupta, S., Larsen, K.S.: Access graphs results for LRU versus FIFO under relative worst order analysis. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 328–339. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31155-0_29
22. Boyar, J., Gupta, S., Larsen, K.S.: Relative interval analysis of paging algorithms on access graphs. Theor. Comput. Sci. **568**, 28–48 (2015)
23. Boyar, J., Irani, S., Larsen, K.S.: A comparison of performance measures for online algorithms. Algorithmica **72**(4), 969–994 (2015)
24. Boyar, J., Larsen, K.S.: The seat reservation problem. Algorithmica **25**(4), 403–417 (1999)
25. Boyar, J., Larsen, K.S., Maiti, A.: A comparison of performance measures via online search. Theor. Comput. Sci. **532**, 2–13 (2014)
26. Boyar, J., Larsen, K.S., Maiti, A.: The frequent items problem in online streaming under various performance measures. Int. J. Found. Comput. Sci. **26**(4), 413–440 (2015)
27. Boyar, J., Larsen, K.S., Nielsen, M.N.: The accommodating function: a generalization of the competitive ratio. SIAM J. Comput. **31**(1), 233–258 (2001)
28. Boyar, J., Medvedev, P.: The relative worst order ratio applied to seat reservation. ACM Trans. Algorithms **4**(4), 22 (2008). Article 48
29. Christ, M.G., Favrholdt, L.M., Larsen, K.S.: Online bin covering: expectations vs guarantees. Theor. Comput. Sci. **556**, 71–84 (2014)
30. Chrobak, M., Karloff, H.J., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM J. Discret. Math. **4**(2), 172–181 (1991)
31. Chrobak, M., Noga, J.: LRU is better than FIFO. Algorithmica **23**(2), 180–185 (1999)
32. Coffmand Jr., E.G., Csirik, J., Rónyai, L., Zsbán, A.: Random-order bin packing. Discrete Appl. Math. **156**, 2810–2816 (2008)
33. Denning, P.J.: The working set model for program behaviour. Commun. ACM **11**(5), 323–333 (1968)
34. Denning, P.J.: Working sets past and present. IEEE Trans. Softw. Eng. **6**(1), 64–84 (1980)
35. Devanur, N.R., Hayes, T.P.: The adwords problem: online keyword matching with budgeted bidders under random permutations. In: 10th ACM Conference on Electronic Commerce (EC), pp. 71–78 (2009)
36. Dobrev, S., Kralović, R., Pardubskă, D.: Measuring the problem-relevant information in input. RAIRO - Theor. Inf. Appl. **43**(3), 585–613 (2009)
37. Dorrigiv, R., López-Ortiz, A., Munro, J.I.: On the relative dominance of paging algorithms. Theor. Comput. Sci. **410**, 3694–3701 (2009)
38. Ehmsen, M.R., Kohrt, J.S., Larsen, K.S.: List factoring and relative worst order analysis. Algorithmica **66**(2), 287–309 (2013)
39. Epstein, L., Favrholdt, L.M., Kohrt, J.S.: Separating scheduling algorithms with the relative worst order ratio. J. Comb. Optim. **12**(4), 362–385 (2006)
40. Epstein, L., Favrholdt, L.M., Kohrt, J.S.: Comparing online algorithms for bin packing problems. J. Sched. **15**(1), 13–21 (2012)
41. Fischer, C., Röglin, H.: Probabilistic analysis of the dual Next-Fit algorithm for bin covering. In: Kranakis, E., Navarro, G., Chávez, E. (eds.) LATIN 2016. LNCS, vol. 9644, pp. 469–482. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49529-2_35
42. Göbel, O., Kesselheim, T., Tönnis, A.: Online appointment scheduling in the random order model. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 680–692. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48350-3_57

43. Goel, G., Mehta, A.: Online budgeted matching in random input models with applications to adwords. In: 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 982–991 (2008)
44. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**(2), 416–429 (1969)
45. Gyárfás, A., Lehel, J.: First fit and on-line chromatic number of families of graphs. Ars Comb. **29**(C), 168–176 (1990)
46. Hromkovič, J., Královič, R., Královič, R.: Information complexity of online problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15155-2_3
47. Irani, S.: Two results on the list update problem. Inf. Process. Lett. **38**(6), 301–306 (1991)
48. Johnson, D.S.: Fast algorithms for bin packing. J. Comput. Syst. Sci. **8**, 272–314 (1974)
49. Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L.: Worst-case performance bound for simple one-dimensional packing algorithms. SIAM J. Comput. **3**, 299–325 (1974)
50. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM **47**, 617–643 (2000)
51. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica **3**, 79–119 (1988)
52. Kenyon, C.: Best-fit bin-packing with random order. In: 7th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 359–364 (1996)
53. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. SIAM J. Comput. **30**(1), 300–317 (2000)
54. Krumke, S.O., de Paepe, W., Rambau, J., Stougie, L.: Bincoloring. Theor. Comput. Sci. **407**(1–3), 231–241 (2008)
55. McCabe, J.: On serial files with relocatable records. Oper. Res. **13**(4), 609–618 (1965)
56. Meyerson, A.: Online facility location. In: 42nd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 426–433 (2001)
57. Moruz, G., Negoescu, A.: Outperforming LRU via competitive analysis on parametrized inputs for paging. In: 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1669–1680 (2012)
58. O'Neil, E.J., O'Neil, P.E., Weikum, G.: The LRU-K page replacement algorithm for database disk buffering. In: ACM SIGMOD International Conference on Management of Data, pp. 297–306 (1993)
59. Osborn, C.J., Torng, E.: List's worst-average-case or WAC ratio. J. Sched. **11**, 213–215 (2008)
60. Raghavan, P.: A statistical adversary for on-line algorithms. In: On-Line Algorithms, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, vol. 7, pp. 79–83. American Mathematical Society (1992)
61. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)
62. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. J. ACM **51**(3), 385–463 (2004)
63. Teia, B.: A lower bound for randomized list update algorithms. Inf. Process. Lett. **47**, 5–9 (1993)
64. Young, N.E.: The k-server dual and loose competitiveness for paging. Algorithmica **11**, 525–541 (1994)