Relative Worst-Order Analysis: A Survey

JOAN BOYAR, University of Southern Denmark
LENE M. FAVRHOLDT, University of Southern Denmark
KIM S. LARSEN, University of Southern Denmark

The standard measure for the quality of online algorithms is the competitive ratio. This measure is generally applicable, and for some problems it works well, but for others it fails to distinguish between algorithms that have very different performance. Thus, 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) is one of the most thoroughly tested such proposals. With RWOA, many separations of algorithms not obtainable with competitive analysis have been found.

In RWOA, two algorithms are compared directly, rather than indirectly as is done in competitive analysis, where both algorithms are compared separately to an optimal offline algorithm. If, up to permutations of the request sequences, one algorithm is always at least as good and sometimes better than another, the first algorithm is deemed the better algorithm by RWOA.

We survey the most important results obtained with this technique and compare it with other quality measures. The survey includes a quite complete set of references.

CCS Concepts:•**Theory of computation** → **Online algorithms;**

Additional Key Words and Phrases: online algorithms, relative worst-order analysis, competitive analysis

ACM Reference Format:
Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen, 2018. Relative Worst-Order Analysis: A Survey *ACM Comput. Surv.* 0, 0, Article 0 (0), 22 pages.
DOI: 0000001.0000001

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 [Sleator and Tarjan 1985]. They introduced the method as a general analysis technique, inspired by approximation algorithms, where the result of the algorithm is compared to that of an optimal algorithm. Thus, the competitive ratio is the worst-case ratio of the algorithm's performance to the performance of an optimal offline algorithm, OPT. The term "competitive" is from Karlin et al. [Karlin et al. 1988] who named the worst-case ratio of the performance of the online to the

This work is supported in part by the Independent Research Fund Denmark, Natural Sciences, grants DFF-7014-00041.
Authors' address: Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark. Email: {joan,lenem,kslarsen}@imada.sdu.dk.
A preliminary version of this work appeared in Adventures Between Lower Bounds and Higher Altitudes – Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday, volume 11011 of Lecture Notes in Computer Science, pages 216–230. Springer, 2018. This substantial extension includes several refinements of the introduced performance measure and its properties, more details on other performance measures and alternative models of computation, an account of results on edge coloring, more open problems, and further references.

Table I. Simplified “definitions” of measures. For the last two measures, $\pi(I)$ denotes a permutation of the sequence I.

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	$\mathrm{WR}_{\mathbb{A},\mathbb{B}} = \sup_I \frac{\sup_\pi \left\{\mathbb{A}(\pi(I))\right\}}{\sup_{\pi'} \left\{\mathbb{B}(\pi'(I))\right\}}$

offline algorithm the “competitive ratio”. Many years earlier, Graham carried out what is now viewed as an example of a competitive analysis [Graham 1969].

The over-all goal of a theoretical quality measure is to predict performance of algorithms in practice. In that respect, competitive analysis works well in some cases, but, as pointed out by the inventors [Sleator and Tarjan 1985] 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 [Boyar and Favrholdt 2007], and the definitions were extended together with Larsen [Boyar et al. 2007]. 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 many separations, not obtainable using competitive analysis, corresponding to experimental results and/or intuition.

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.

2. RELATIVE WORST-ORDER ANALYSIS

With RWOA, two algorithms are compared directly (rather than indirectly by first comparing both to an optimal offline algorithm). The algorithms are compared on their respective worst orderings of sequences having the same content. Thus, up to permutations of the request sequences, if an algorithm is always at least as good and sometimes better than another, RWOA separates them. Table I 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 in Section 2.2, ending with a formal definition of the relative worst-order ratio in Section 2.3.

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 room for k memory pages and a larger, slow memory that can hold $N > k$ 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 [Young 1994]. 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 [Boyar et al. 2007]. In Section 4.1, we consider LRU and FWF in greater detail to give a concrete example of RWOA.

There are many less clear-cut examples. Consider, the paging algorithm LRU-2, for example. Instead of evicting the page with the earliest last request, it evicts the page with the earliest second-to-last request. LRU-2 is known to perform better than LRU in, for example, database disk buffering [O'Neil et al. 1993]. However, there are also sequences where LRU performs better. For two concrete examples, consider the sequences $\langle p_1, p_2, \ldots, p_{k-1}, q_1, q_2, p_1, p_2, \ldots, p_{k-1}, q_3, q_4\rangle^*$, where LRU will have $(k+1)/2$ times as many faults as LRU-2, asymptotically, and $\langle p_1, p_2, \ldots, p_{k+1}, p_{k+1}, \ldots, p_2, p_1\rangle^*$, where LRU-2 will fault k times as often as LRU, asymptotically. Under RWOA, LRU-2 is the better algorithm; see Section 4.4. Similar examples are FIRST-FIT versus WORST-FIT for bin packing and seat reservation; see Section 5.2.

2.2. Background

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, OPT, 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 algorithms directly, such as RWOA, bijective and average analysis [Angelopoulos et al. 2019], and relative interval analysis [Dorrigiv et al. 2009], would generally indicate correctly that LRU is the better algorithm.

When comparing algorithms directly, using exactly the same sequences will tend to produce the result that many algorithms are not comparable, as illustrated for LRU and LRU-2 in Section 2.1. 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. 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 [Ben-David and Borodin 1994] and the random-order ratio [Kenyon 1996].

2.2.1. 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. However, 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 [Ben-David and Borodin 1994] 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 any other pair of 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.

2.2.2. The Random-Order Ratio. The random-order ratio was introduced in [Kenyon 1996] 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 I. 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 [Johnson et al. 1974].

Random-order analysis has also been applied to other problems, e.g., knapsack [Babaioff et al. 2007], bipartite matching [Goel and Mehta 2008; Devanur and Hayes 2009], scheduling [Osborn and Torng 2008; Göbel et al. 2015], bin covering [Christ et al. 2014; Fischer and Röglin 2016], and facility location [Meyerson 2001]. However, the analysis is often rather challenging, and in [Coffman Jr. et al. 2008], 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

$$\mathbb{A}_{\mathrm{W}}(I) = \max_{\pi} \mathbb{A}(\pi(I)),$$

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

$$\mathbb{A}_{\mathrm{w}}(I) = \min_{\pi} \mathbb{A}(\pi(I)).$$

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 $\mathbb{A}_{\mathrm{w}}(I) \leq \mathbb{B}_{\mathrm{w}}(I)$, for all request sequences I, or $\mathbb{A}_{\mathrm{w}}(I) \geq \mathbb{B}_{\mathrm{w}}(I)$, for all request sequences I (in the definition below, this corresponds to $c_{\mathrm{u}}(\mathbb{A},\mathbb{B}) \leq 1$ or $c_{\mathrm{l}}(\mathbb{A},\mathbb{B}) \geq 1$). In this case, to compute the relative worst-order ratio of $\mathbb{A}$ to $\mathbb{B}$, we compute a bound ($c_{\mathrm{l}}(\mathbb{A},\mathbb{B})$ or $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$) 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.

In the formal definition of the relative worst-order ratio, we use $c_{\mathrm{l}}(\mathbb{A},\mathbb{B})$ and $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$, which are tight lower and upper bounds, respectively, on the ratio $\mathbb{A}_{\mathrm{w}}(I)/\mathbb{B}_{\mathrm{w}}(I)$ (up to an additive constant). Thus, $c_{\mathrm{l}}(\mathbb{A},\mathbb{B}) = 1/c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$.

Definition 2.1. For any pair of algorithms $\mathbb{A}$ and $\mathbb{B}$, we define

$$\begin{aligned} c_{\mathrm{l}}(\mathbb{A},\mathbb{B}) &= \sup \ \{c \mid \exists b \,\forall I\colon \mathbb{A}_{\mathrm{w}}(I) \geq c\,\mathbb{B}_{\mathrm{w}}(I) - b\} \text{ and} \\ c_{\mathrm{u}}(\mathbb{A},\mathbb{B}) &= \inf \ \{c \mid \exists b \,\forall I\colon \mathbb{A}_{\mathrm{w}}(I) \leq c\,\mathbb{B}_{\mathrm{w}}(I) + b\}\,. \end{aligned}$$

If $c_{\mathrm{l}}(\mathbb{A},\mathbb{B}) \geq 1$ or $c_{\mathrm{u}}(\mathbb{A},\mathbb{B}) \leq 1$, 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

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

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 II indicates the result in each case.

Table II. 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

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.

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.

THEOREM 2.2 (BOYAR, FAVRHOLDT, 2007). *With RWOA, the ordering of algorithms for a specific problem is transitive. Moreover:*
If $\mathrm{WR}_{\mathbb{A},\mathbb{B}} \geq 1$ *and* $\mathrm{WR}_{\mathbb{B},\mathbb{C}} \geq 1$*, then* $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \geq \mathrm{WR}_{\mathbb{B},\mathbb{C}}$*. If, furthermore,* $\mathrm{WR}_{\mathbb{A},\mathbb{B}}$ *is bounded above by some constant, then* $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \geq \mathrm{WR}_{\mathbb{A},\mathbb{B}}$*.*
If $\mathrm{WR}_{\mathbb{A},\mathbb{B}} \leq 1$ *and* $\mathrm{WR}_{\mathbb{B},\mathbb{C}} \leq 1$*, then* $\mathrm{WR}_{\mathbb{A},\mathbb{C}} \leq \min(\mathrm{WR}_{\mathbb{A},\mathbb{B}}, \mathrm{WR}_{\mathbb{B},\mathbb{C}})$*.*

The *strict* relative worst-order ratio, $\mathrm{sWR}_{,}$, is defined similarly, simply removing the additive constant, b, from the definitions of c_{l} and c_{u}. Transitivity also holds in this case:

THEOREM 2.3 (BOYAR, FAVRHOLDT, 2007). *With strict RWOA, the ordering of algorithms for a specific problem is transitive. Moreover:*
If $\mathrm{sWR}_{\mathbb{A},\mathbb{B}} \geq 1$ *and* $\mathrm{sWR}_{\mathbb{B},\mathbb{C}} \geq 1$*, then* $\mathrm{sWR}_{\mathbb{A},\mathbb{C}} \geq \max\{\mathrm{sWR}_{\mathbb{A},\mathbb{B}}, \mathrm{sWR}_{\mathbb{B},\mathbb{C}}\}$*.*
If $\mathrm{sWR}_{\mathbb{A},\mathbb{B}} \leq 1$ *and* $\mathrm{sWR}_{\mathbb{B},\mathbb{C}} \leq 1$*, then* $\mathrm{sWR}_{\mathbb{A},\mathbb{C}} \leq \min(\mathrm{sWR}_{\mathbb{A},\mathbb{B}}, \mathrm{sWR}_{\mathbb{B},\mathbb{C}})$*.*

2.4. Extensions of Relative Worst-Order Analysis

In this subsection, we give definitions that are used only for a minority of the results described in this survey. Thus, most of the survey is accessible without reading this subsection.

Although one of the purposes of introducing RWOA was to compare online algorithms directly, an online algorithm can, of course, also be compared to OPT. In this case, we call the ratio the *worst-order ratio*. Along with other measures, including the relative worst-order ratio, this was used in [Boyar and Favrholdt 2012] to analyze a new variable-sized bin packing problem motivated by the problem of partitioning the subproblems defined by BLASTing [Altschul et al. 1990] a genome against a DNA sequence database.

There are some cases where the relative worst-order ratio for two algorithms is undefined because the algorithms are not comparable, but they are in some sense close to being comparable. This has led to two extensions of RWOA [Boyar et al. 2007], but first we need the notion of relatedness:

Definition 2.4. Let c_{u} be defined as in Definition 2.1. If at least one of the ratios $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ and $c_{\mathrm{u}}(\mathbb{B},\mathbb{A})$ is finite, the algorithms $\mathbb{A}$ and $\mathbb{B}$ are $(c_u(\mathbb{A},\mathbb{B}), c_u(\mathbb{B},\mathbb{A}))$*-related*.

If the two algorithms are comparable, one of the values is the relative worst-order ratio and the other is typically 1 (unless one algorithm is strictly better than the other in all cases).

An example of the use of this measure is from [Boyar et al. 2015b], where a simple k-server problem is investigated; see Section 6 for details.

Definition 2.5. Let $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ be defined as in Definition 2.4. Algorithms $\mathbb{A}$ and $\mathbb{B}$ are *weakly comparable*

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

Definition 2.6. A *resource-dependent problem* is an optimization problem, where each problem instance, in addition to the input data, also has a parameter k, referred to as the amount of resources, such that for each input, the optimal solution depends monotonically on k.

Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for a resource-dependent problem P and let c_{u} and c_{l} be defined as in Definition 2.4. We define

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

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

$$\mathrm{WR}_{\mathbb{A},\mathbb{B}}^{\infty} = \begin{cases} c_{\mathrm{u}}^{\infty}(\mathbb{A},\mathbb{B}), & \text{if } c_{\mathrm{l}}^{\infty}(\mathbb{A},\mathbb{B}) \geq 1 \\ c_{\mathrm{l}}^{\infty}(\mathbb{A},\mathbb{B}), & \text{if } c_{\mathrm{u}}^{\infty}(\mathbb{A},\mathbb{B}) \leq 1 \end{cases}$$

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

Definition 2.7. Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms for an optimization problem. If $\mathbb{A}$ and $\mathbb{B}$ are neither weakly nor resource-asymptotically comparable, we say that they are *incomparable*.

2.5. Applicability

As we will discuss in the next section, many performance measures have been suggested over the years. However, most of these measures have been used on only one online problem. In contrast, the permutation-based RWOA technique has been applied to a multitude of different online problems with quite different characteristics. However, the very design idea of focusing on permutations imply that there are some online problems where the measure does not apply well, including some problems where the requests refer explicitly to time (release times, for instance) or contain precedence constraints, and where this makes permutations meaningless.

Further, it has been established that RWOA can be used in combination with other analysis techniques such as access graphs (defined in Section 3.1) and list factoring.

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 [Boyar et al. 2015a] with less convincing results.

To a large extent, the above text describes applicability by what has been done. Other issues, comparison of randomized algorithms, for instance, could also be considered. We know it is possible, since the randomized paging algorithm MARK was compared to LRU, indicating that (as with competitive analysis) MARK is significantly better than LRU according to RWOA [Boyar et al. 2007]: $\mathrm{WR}_{\mathrm{LRU},\mathrm{MARK}} = \frac{k}{H_k}$. However, our impression is that the comparison of randomized and deterministic algorithms gives very little useful information in many cases. This seems to also be true of comparisons using the competitive ratio for paging [Young 1994] and for list access [Bachrach et al. 2002]. From those papers, it is clear that comparisons between an expected ratio for a randomized algorithm and the usual worst-case ratio for a deterministic algorithm is not a good predictor for the performance in practice. This is not surprising, but worth recalling, since randomized ratios are often presented as “improvements” compared with some worst-case ratio.

Finally, whereas RWOA fairly often reveals details concerning online problems and the algorithms solving them that escapes competitive analysis, it places an extra burden on the researchers. Using competitive analysis, the developers of each algorithm can compute its competitive ratio and present their algorithm with that quality stamp for comparison with other algorithms. Using RWOA, it is necessary to know the details of two algorithms to compute their relative performance. Usually, for a collection of algorithms to be analyzed, this does not lead to a quadratic number of analyses since the measure is transitive, so many comparisons can be deduced with no additional effort.

3. 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. This survey is about RWOA, and treating other performance measures as thoroughly would make the survey prohibitively long. However, to sketch the bigger picture, we also briefly describe other measures and compare RWOA results with related results on these other measures.

In this section, we first list many performance measures that have been introduced over the years, with a short informal description. The concept of access graphs is discussed in a separate section since it is used in connection with RWOA. Then we discuss other ways of shedding light on the properties of online problems: relaxing or investigating the significance of the "no knowledge of the future" constraint via advice complexity and relaxing or investigating the "irrevocability" constraint via various forms of recourse. In Section 6, we give some insight into the strengths and weaknesses of selected measures, discussing work directly targeted at performance measure comparison.

We present these alternative performance measures in chronological order:

Online/online ratio [Gyárfás and Lehel 1990]
Instead of comparing to an optimal offline algorithm, as in competitive analysis, the online algorithms are compared to an optimal online algorithm. For any input, the optimal online algorithm may be designed specifically for that input, but it has to work for any permutation of the input sequence, without knowing the permutation from the beginning.

Statistical adversary [Raghavan 1992]
This measure considers worst-case input sequences among sequences with some given statistical properties and measures the absolute performance, i.e., without comparing to an optimal algorithm.

Loose competitive ratio [Young 1994]
This measure was introduced in connection with the paging problem. It gives the possibility of ignoring a small fraction of the possible cache sizes and also ignores sequences with a relatively low fault rate.

Max/max ratio [Ben-David and Borodin 1994]
This measure was discussed in Section 2.2.1.

Access graphs [Borodin et al. 1995]
Access graphs are used to model locality of reference; see Section 3.1.

Random-order ratio [Kenyon 1996]
This measure was discussed in Section 2.2.2.

Accommodating ratio [Boyar and Larsen 1999]
This measure works for problems with some limited resource, such as the cache in the paging problem or the number or speed of machines in scheduling problems. It is the same as the competitive ratio, but considering only input sequences, where the optimal offline solution would not improve by having more resources available.

Extra resource analysis [Kalyanasundaram and Pruhs 2000]
This is also called resource augmentation. As the accommodating ratio, this form of analysis addresses problems with some kind of limited resource. Competitive analysis is performed, but with a parameter x and letting the online algorithm use x times as much of the resource as the optimal offline algorithm.

Diffuse adversary [Koutsoupias and Papadimitriou 2000]
A class of input distributions is considered. The adversary chooses a distribution from the given class, and the expected performance of the online algorithm is com-

pared to the expected performance of an optimal offline algorithm. Note that this is a generalization of the statistical adversary and the notion of access graphs.

Accommodating function [Boyar et al. 2001]
This is a generalization of the accommodating ratio, giving the competitive ratio as a function of a parameter α. An input sequence is called α-accommodating, if having access to α times as many resources as those available does not change the value of an optimal offline solution. The parameter α can be smaller [Boyar et al. 2003] as well as larger than 1. Negative results on the accommodating function translate to negative results with extra resource analysis, and positive results with extra resource analysis translate to positive results on the accommodating function.

Smoothed analysis [Spielman and Teng 2004]
Instead of considering single worst-case sequences, neighborhoods of sequences are considered, where a neighborhood consists of a sequence I together with a set of sequences that can be obtained by slightly perturbing I.

Working set [Albers et al. 2005]
Inspired by the concept of working sets [Denning 1968] modeling locality of reference, only sequences with a limited average or maximum number of different pages within subsequences of certain lengths are considered. Instead of using a relative measure such as the competitive ratio, the fault rate is considered.

Relative worst-order analysis [Boyar and Favrholdt 2007; Boyar et al. 2007]
The topic of this survey.

Characteristic vector [Panagiotou and Souza 2006]
This models locality of reference for the paging problem. For a request r to a page p that has already been requested earlier, let ℓ_r be the number of different pages that have been requested since the previous request to p. The ith entry of the characteristic vector of a request sequence gives the number of requests r with $\ell_r = i$.

Bijective and average analysis [Angelopoulos et al. 2019]
If, for two algorithms $\mathbb{A}$ and $\mathbb{B}$ and for each possible input length n, there is a bijection f from the set of input sequences of length n to that same set, such that $\mathbb{A}(I)$ is always at least as good as $\mathbb{B}(f(I))$ and sometimes better, then according to bijective analysis, $\mathbb{A}$ is better than $\mathbb{B}$. If the weaker result that the average of $\mathbb{A}(I)$ is better than the average of $\mathbb{B}(f(I))$ can be obtained, then $\mathbb{A}$ is better than $\mathbb{B}$ according to average analysis.

Relative interval analysis [Dorrigiv et al. 2009]
Pairs of algorithms $\mathbb{A}$ and $\mathbb{B}$ are compared directly: For sequences of a given length n, the minimum and maximum value of $\mathbb{A}(I) - \mathbb{B}(I)$ is calculated for sequences I of length n. The measure is the interval between these two extremes, as n tends to infinity.

Non-locality [Dorrigiv et al. 2015]
This models the non-locality of request sequences for the paging problem. Let ℓ_r be defined as for the characteristic vector. The non-locality of a request sequence is the average value of ℓ_r over the request sequence.

Attack rate [Moruz and Negoescu 2012]
For the paging problem, the attack rate of an input sequence is a number between 1 and k, rating how difficult it is to guess the cache content of an optimal offline algorithm on the fly. Let a new page be a page that has not been requested earlier, and let an unrevealed page be a page for which it cannot be decided at the time of the request whether or not the page could be in the cache of an optimal offline algorithm. The attack rate is the ratio of the number of requests to new or unrevealed pages to the number of requests to new pages.

Bijective ratio [Angelopoulos et al. 2020]
This ratio is a generalization of bijective analysis: If, for two algorithms $\mathbb{A}$ and $\mathbb{B}$ and for each input length n, there is a $\rho > 0$ and a bijection f from the set of input sequences of length n to that same set, such that $\mathbb{A}(I)$ is at most ρ times worse than $\mathbb{B}(f(I))$, the bijective ratio of $\mathbb{A}$ against $\mathbb{B}$ is at most ρ.

Online-bounded analysis [Boyar et al. 2018]
The online-bounded ratio is the same as the competitive ratio, except that the adversary is restricted to using input sequences for which there is an optimal solution fulfilling that, on any prefix of the sequence, the value of the optimal offline solution must be at least as good as the value of the online solution on that prefix.

3.1. Access Graphs

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 for the paging problem, with repetitions of pages appearing in close proximity [Denning 1968; 1980]. 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 [Angelopoulos et al. 2019] 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.

The concept of *access graphs* [Borodin et al. 1995] has been used to model locality of reference for the paging problem. 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 (by any performance measure), 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.

In [Karlin et al. 2000], probabilities are added to the edges of access graphs, thus obtaining Markov models.

3.2. Advice Complexity

As a means of analyzing problems, as opposed to algorithms for those problems, *advice complexity* was proposed [Dobrev et al. 2009; Hromkovič et al. 2010; Böckenhauer et al. 2017]. 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 [Boyar et al. 2017b].

3.3. Recourse

Instead of relaxing the constraint on knowledge about the future, online algorithms with recourse deals with relaxing the constraint on decisions being irrevocable, and this idea has been studied under many different names for different problems.

For the knapsack problem, items are usually referred to as being *removable*; see for example [Iwama and Taketomi 2002; Han et al. 2014; Han et al. 2015; Cygan et al. 2016; Han and Makino 2016]. For online Steiner tree problems, MST, and TSP [Imase and Waxman 1991; Megow et al. 2016; Gu et al. 2016; Gupta and Kumar 2014], one can allow replacing an accepted edge with another, under the objective of minimizing the number of times this occurs (while obtaining a good competitive ratio). This is commonly referred to as *rearrangements* or *recourse*.

The most common term when allowing late rejections is *preemption*, and this has been studied in variants of many online problems, including call control [Bartal et al. 1996; Garay et al. 1997], maximum coverage [Saha and Getoor 2009; Rawitz and Rosén 2016], weighted matching problems [Epstein et al. 2011; Epstein et al. 2013], and submodular maximization [Buchbinder et al. 2015].

Whereas late rejection has had significant focus, late acceptance was studied in [Boyar et al. 2016], followed by a systematic study of all combinations of a number of classic graph problems combined with either late accept, late reject, or both (where late reject is irrevocable) in [Boyar et al. 2017a], detailing the performance implications of each choice. In [Angelopoulos et al. 2018], the work of [Boyar et al. 2017a] was continued for the matching problem, considering more levels of late accept/reject.

Finally, [Komm et al. 2016] discusses preemption together with advice.

4. PAGING

In this section, we survey the most important RWOA results for paging and explain how they compare to results obtained with competitive analysis and other alternative measures. As a relatively simple, concrete example of RWOA, we first explain how to obtain the separation of LRU and FWF [Boyar et al. 2007] mentioned in Section 2.1.

4.1. LRU vs. FWF

Recall that LRU is the algorithm that evicts the least recently used page in cache when there is a fault and the cache is full, and FWF evicts everything from cache (flushes the cache) in this situation.

The first step in computing the relative worst-order ratio, $\mathrm{WR}_{\mathrm{FWF},\mathrm{LRU}}$, 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, $\mathrm{FWF}_{\mathrm{w}}(I_{\mathrm{LRU}}) \geq \mathrm{FWF}(I_{\mathrm{LRU}}) \geq \mathrm{LRU}(I_{\mathrm{LRU}}) = \mathrm{LRU}_{\mathrm{w}}(I_{\mathrm{LRU}})$. Thus, $c_{\mathrm{l}}(\mathrm{FWF}, \mathrm{LRU}) \geq 1$.

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 $c_{\mathrm{u}}(\mathrm{FWF}, \mathrm{LRU})$. 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, after the cache fills up the first time, FWF faults on, and thus flushes on, every occurrence of 1 or $k+1$. Thus, FWF faults on all $2ks$ requests in I. 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 [Boyar et al. 2007] 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, $c_{\mathrm{u}}(\mathrm{FWF}, \mathrm{LRU}) \geq \frac{2k}{k+1}$. We now know that $\mathrm{WR}_{\mathbf{FWF},\mathbf{LRU}} \geq \frac{2k}{k+1}$, 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 $c_{\mathrm{u}}(\mathrm{FWF}, \mathrm{LRU})$ cannot be larger than $\frac{2k}{k+1}$, asymptotically. In fact, this is shown in [Boyar et al. 2007] by proving the more general result that, for any *marking* algorithm [Borodin et al. 1995], $\mathbb{M}$, and for any request sequence I, $\mathbb{M}_{\mathrm{w}}(I) \leq \frac{2k}{k+1}\,\mathrm{LRU}_{\mathrm{w}}(I) + k$. 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 $\mathrm{WR}_{\mathbf{FWF},\mathbf{LRU}} = \frac{2k}{k+1}$.

There are several other results separating LRU and FWF, e.g., using locality of reference [Becchetti 2004; Albers et al. 2005; Boyar et al. 2012; 2015a; Dorrigiv et al. 2015; Albers and Frascaria 2018] or competitive analysis parameterized by the attack rate [Moruz and Negoescu 2012]. Moreover, diffuse adversaries separate LRU from both FWF and FIFO [Young 2000] and average analysis combined with locality of reference as modeled in [Albers et al. 2005] separates LRU from any other deterministic algorithm [Angelopoulos et al. 2019]. In contrast, all marking algorithms have the same competitive ratio, even when applying extra resource analysis.

4.2. LRU vs. FIFO

Like LRU and FWF (and any other marking algorithm), the algorithm FIFO also has competitive ratio k [Borodin and El-Yaniv 1998]. 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 [Young 1994], 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 $\mathbb{B}$, $\mathrm{WR}_{\mathbb{A},\mathbb{B}} = 1$ and $\mathrm{WR}_{\mathbf{FWF},\mathbb{A}} = \frac{2k}{k+1}$ [Boyar et al. 2007]. This is consistent with results on the competitive ratio. Any pair of conservative algorithms have the same competitive ratio, with or without extra resource analysis. Furthermore, parameterizing by the attack rate r, both LRU and FIFO have a competitive ratio of r [Moruz and Negoescu 2012].

On the other hand, there are measures separating LRU and FIFO. One measure, relative interval analysis, gives the result that FIFO is better than LRU [Dorrigiv et al. 2009; Boyar et al. 2015a]. The remaining measures prefer LRU. Two of these, diffuse adversaries and average analysis with locality of reference, were mentioned in

Section 4.1. Two others use locality of reference, working sets in [Albers et al. 2005] and competitive analysis with characteristic vectors in [Albers and Frascaria 2018]. We discuss locality of reference modeled as access graphs in more detail below.

4.2.1. Access Graphs. The papers [Borodin et al. 1995; Chrobak and Noga 1999] 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.

RWOA confirms the competitive analysis result [Borodin et al. 1995] that LRU is better than FIFO for path access graphs [Boyar et al. 2012]. Since these two quality measures are so different, this is a a strong indicator of the robustness of the result.

Using RWOA, [Boyar et al. 2012] proved that with regards to cycle access graphs, LRU is strictly better than FIFO. Note that this separation does not hold under competitive analysis.

Paths and cycles are the two most fundamental building blocks of access graphs, but the question of how LRU and FIFO compare on general graphs is not entirely settled. LRU and FIFO are incomparable under RWOA on general access graphs, but the proof uses a family of graphs where the size is proportional to the length of the request sequence [Boyar et al. 2012]. It is still open as to whether or not this incomparability holds for some family of graphs where the size is not dependent on the length of the request sequence.

4.3. A Marking Algorithm worse than LRU

Although all marking algorithms have the same competitive ratio as LRU, some marking algorithms seem much less "reasonable" than LRU. Consider, for instance, the algorithm $\textsc{Mark}^{\text{LIFO}}$ that, on a fault, evicts the unmarked page which was most recently brought into cache. For $k \geq 2$, LRU and $\textsc{Mark}^{\text{LIFO}}$ are $(1 + \frac{1}{k}, 2 - \frac{2}{k+1})$-related, and hence, the resource-asymptotic relative worst-order ratio of $\textsc{Mark}^{\text{LIFO}}$ to LRU is $\text{WR}^{\infty}_{\textsc{Mark}^{\text{LIFO}},\text{LRU}} = 2$ [Boyar et al. 2007].

4.4. Algorithms better than LRU?

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? With competitive analysis, the answer would be "no", since no deterministic paging algorithm has a competitive ratio better than k. However, according to RWOA, the answer to this is "yes".

LRU-2 [O'Neil et al. 1993], which was proposed for database disk buffering, is the algorithm which evicts the page with the earliest second-to-last request. The paper presents compelling empirical evidence in support of the superiority of LRU-2 over LRU in database systems. It was shown in [Boyar et al. 2010] that $\text{WR}_{\text{LRU},\text{LRU-2}} = \frac{k+1}{2}$, so LRU-2 is better according to RWOA. A generalization of LRU-2, LRU-K, was also considered in [Boyar et al. 2010], and LRU and LRU-K were shown to be *resource-asymptotically comparable* (see Definition 2.6; this is slightly weaker than being comparable) in LRU-K's favor. It is interesting to note that according to competitive analysis, LRU-K is only kK-competitive, for any $K \geq 2$, so these algorithms are deemed worse than LRU and FWF according to competitive analysis.

In addition, a new algorithm, RLRU (Retrospective LRU), was defined in [Boyar et al. 2007] 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 [Moruz and Negoescu 2012]. RLRU is a phase-based algorithm. When considering a request, it determines whether $\textsc{Opt}$

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. The competitive ratio of RLRU was shown to be $k+1$, so, as with LRU-2, RLRU is worse than LRU according to competitive analysis, but better according to relative worst-order analysis.

4.5. Look-ahead

Considering LRU and LRU(ℓ), which is LRU adapted to using 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, $\mathrm{WR}_{\mathrm{LRU},\mathrm{LRU}(\ell)} = \min\{k, \ell+1\}$, so LRU($\ell$) is better [Boyar et al. 2007].

The same kind of separation has been proven with bijective analysis [Angelopoulos et al. 2019] and relative interval analysis [Dorrigiv et al. 2009]. Similarly, for the k-server problem which is a generalization of paging, the max/max-ratio gives results reflecting that look-ahead can be an advantage. However, for the paging problem, the max/max-ratio cannot separate algorithms; as the ratio between the sizes of the slow memory and the cache tend to infinity, the max/max ratio of any paging algorithm tends to 1.

5. 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, scheduling, and edge coloring. Other problems have also been considered, see [Boyar and Favrholdt 2007; 2010; 2012; Boyar and Medvedev 2008; Ehmsen et al. 2010]. [Boyar and Kudahl 2018] presents results which apply to the independent set problem and other problems, for different performance measures, including relative worst-order analysis.

5.1. List Accessing

List accessing [Sleator and Tarjan 1985; Albers and Westbrook 1998] 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. (called *free* transpositions). In addition, any two consecutive items may be transposed at a cost of one. (*paid* transpositions). 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) [McCabe 1965] 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 [Sleator and Tarjan 1985], letting l denote the length of the list, the algorithm MOVE-TO-FRONT has strict competitive ratio $2 - \frac{2}{l+1}$ [Irani 1991] (referring to personal communication, Irani credits Karp and Raghavan with the lower bound). In contrast, FREQUENCY-COUNT and TRANSPOSE both have competitive ratio $\Omega(l)$ [Borodin and El-Yaniv 1998]. Extensive experiments demonstrate that MTF and FC are approximately equally good, whereas TRANS is much worse [Bentley and McGeoch 1985; Bachrach and El-Yaniv 1997]. 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 [Ehmsen et al. 2013].

List accessing with locality of reference has been explored in [Albers and Lauer 2016; Angelopoulos et al. 2019; Dorrigiv et al. 2015], proving MOVE-TO-FRONT optimal under various locality models.

5.1.1. List Factoring. 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 [Bentley and McGeoch 1985] and later extended and improved [Irani 1991; Teia 1993; Albers et al. 1995; Albers 1998]. 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 above, as well as results on TIMESTAMP [Albers 1998], were obtained [Ehmsen et al. 2013] using this tool.

5.2. Bin Packing and Friends

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 [Johnson 1974]. However, WF is at least as good as NF on every sequence and sometimes much better [Boyar et al. 2010]. Using RWOA, $\mathrm{WR}_{\mathrm{NF,WF}} = 2$, so WF is the better algorithm.

It is widely believed and consistent with experiments that for bin packing type problems, FIRST-FIT algorithms perform better than WORST-FIT algorithms [Johnson 1973]. For dual bin packing, grid scheduling, and seat reservation, described below, competitive analysis cannot distinguish between the algorithms, that is, they have the same competitive ratio, whereas RWOA gives a separation in the right direction.

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 [Boyar and Favrholdt 2007].

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 [Boyar and Favrholdt 2010].

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 [Boyar and Medvedev 2008] with regards to both objective functions.

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. If the bin size is larger than approximately q^3, ONE-BIN has a better competitive ratio than GREEDY-FIT [Krumke et al. 2008]. However, according to strict RWOA, GREEDY-FIT is better [Epstein et al. 2012]. The latter result is supported by [Hiller and Vredeveld 2008], proving that GREEDY-FIT is stochastically better than ONE-BIN, i.e., for any given input length, any input distribution, and any number C, the probability that the maximum number of colors used in a bin is at least C is no larger for GREEDY-FIT than for ONE-BIN.

5.3. Scheduling

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 $\frac{s+1}{s}$-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 $\mathrm{WR}_{\text{FAST},\text{POST-GREEDY}} = \frac{s+1}{s}$ [Epstein et al. 2006].

5.4. Edge Coloring

In graph edge coloring, the edges of a graph must be colored with as few colors as possible, under the restriction that no two adjacent edges are given the same color. In the online version, the edges arrive one by one, and the best possible competitive ratio of 2 [Bar-Noy et al. 1992] is attained by the algorithm FIRST-FIT, i.e., the algorithm also called GREEDY which colors each edge with the lowest numbered color possible.

Perhaps a more reasonable way to assign the predicate "greedy" to an algorithm would be to refer to an algorithm as greedy if it never increases the objective function unnecessarily, i.e., it only uses a new color if none of the colors already used on edges given earlier results in a valid coloring. Any such algorithm is best possible according to competitive analysis, including a less natural algorithm, NEXT-FIT. This algorithm behaves as FIRST-FIT, except that it considers the colors used so far in a cyclic order, starting with the color appearing just after the color that was most recently used. With RWOA, FIRST-FIT and NEXT-FIT are comparable in FIRST-FIT's favor [Ehmsen et al. 2010].

6. COMPARISONS OF PERFORMANCE MEASURES

Whereas we have also compared results from different performance measures in the previous sections, it was the online problems that was the focus in those sections. In this section, it is not the online problem, which is extremely simple, but rather the performance measures that are the focus points.

A systematic comparison of performance measures for online algorithms was initiated in [Boyar et al. 2015b], comparing some measures which are applicable to many types of problems. To make this feasible, a particularly simple problem was chosen:

the baby server problem, which is the 2-server problem on a line with three points, one extreme 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 [Chrobak et al. 1991] according to competitive analysis. DC moves the closest server when the request is to the same side of both servers, but moves *both* servers at the same speed when the request is somewhere between the servers. A lazy version of this, LDC, memorizes the moves that should be made by DC, and makes decisions based on the memorized locations, but only moves a server when it will actually reach a request. It is fairly clear that 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.

Another algorithm for the problem, GREEDY, simply always moves the server that is closest to the request. 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. Using one of the extensions to RWOA described in Section 2.4, it was shown in [Boyar et al. 2015b] that LDC and GREEDY are $(2, \infty)$-related and thus weakly comparable in LDC's favor.

Further systematic comparisons of performance measures were made in [Boyar et al. 2014] and [Boyar et al. 2015], again comparing algorithms on relatively simple problems. The paper [Boyar et al. 2014] focuses on *online search*, a maximization problem where bids in the interval $[m, M]$ arrive online and the aim is to choose (sell for) as large a bid as possible. It is a "one-shot" problem, i.e., the processing terminates when a bid is selected or at the end of the sequence with the last bid as the result. [Boyar et al. 2015] investigates the *frequent items* problem, where the goal is to collect the most frequent items over the entire sequence, having a (small) buffer available. The online difficulty is that once an item is evicted from the buffer, it cannot be added again, until a possible later occurrence in the input stream.

The larger collection of performance measures was considered in [Boyar et al. 2014] for online search and we discuss these results. The paper considers an algorithm family, $\mathcal{R}_p$, parameterized by a so-called *reservation price*, p. The algorithm sells at the first bid of at least p. The most clear conclusions were that bijective analysis found all algorithms incomparable and average analysis preferred an intuitively poorer algorithm, $\mathcal{R}_M$. Relative interval analysis focuses on additive differences and points to $\mathcal{R}_{\frac{m+M}{2}}$, whereas competitive analysis focuses on multiplicative comparisons, arriving at the classic result in trading that $\mathcal{R}_{\sqrt{mM}}$ is the best algorithm [El-Yaniv 1998; El-Yaniv et al. 2001]; the situation is the same for random-order analysis. One gets more nuanced answers from RWOA, but the measure also points to $\mathcal{R}_{\sqrt{mM}}$. Technically, $\mathcal{R}_{\sqrt{mM}}$ compares favorably to any other $\mathcal{R}_q$ when using the relatedness comparison.

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 [Boyar et al. 2010] 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 [Moruz and Negoescu 2012].

As mentioned in Section 3.1, the existing proof that LRU and FIFO are incomparable under RWOA on general access graphs uses a family of graphs where the size is proportional to the length of the request sequence [Boyar et al. 2012]. It is not known if this dependency on the length of the request sequence is necessary.

For bin packing, it would be interesting to know whether BEST-FIT is better than FIRST-FIT, according to RWOA.

For the version of graph edge coloring where a limited number, k, of colors are available and the aim is to color as many edges as possible, the results so far are not as conclusive as for the classic version of the problem. The two online algorithms, FIRST-FIT and NEXT-FIT, have very similar competitive ratios [Favrholdt and Nielsen 2003]: NEXT-FIT has a competitive ratio of $2\sqrt{3} - 3 \approx 0.46$, which is worst possible among algorithms that never leave an edge uncolored, unless all k colors have already been used on adjacent edges. The competitive ratio of FIRST-FIT lies between $2\sqrt{3} - 3$ and $2(\sqrt{10} - 1)/9 \approx 0.48$, so the two algorithms have not been separated using competitive analysis, and the competitive ratio of FIRST-FIT can only be slightly better than that of NEXT-FIT. Using RWOA, FIRST-FIT and NEXT-FIT are not comparable [Ehmsen et al. 2010], but it is still open if they are asymptotically comparable.

Finally, as far as we know, no performance measure other than competitive analysis has been used in the context of advice complexity or online problems with recourse. This is of course due to the fact that competitive analysis is the natural starting point, but probably also that with the extra technical complications of advice or recourse, it has been more important to keep the basic measure simple. It would be interesting to explore these directions using RWOA.

Acknowledgment

We thank anonymous referees for their valuable comments.

REFERENCES

Susanne Albers. 1998. Improved Randomized On-Line Algorithms for the List Update Problem. *SIAM J. Comput.* 27, 3 (1998), 682–693.

Susanne Albers, Lene M. Favrholdt, and Oliver Giel. 2005. On Paging with Locality of Reference. *J. Comput. Syst. Sci.* 70, 2 (2005), 145–175.

Susanne Albers and Dario Frascaria. 2018. Quantifying Competitiveness in Paging with Locality of Reference. *Algorithmica* 80 (2018), 3563–3596.

Susanne Albers and Sonja Lauer. 2016. On List Update with Locality of Reference. *J. Comput. Syst. Sci.* 82, 5 (2016), 627–653.

Susanne Albers, Bernhard von Stengel, and Ralph Werchner. 1995. A Combined BIT and TIMESTAMP Algorithm for the List Update Problem. *Inform. Process. Lett.* 56 (1995), 135–139.

Susanne Albers and Jeffrey Westbrook. 1998. Self-Organizing Data Structures. In *Online Algorithms — The State of the Art*, Amos Fiat and Gerhard J. Woeginger (Eds.). Lecture Notes in Computer Science, Vol. 1442. Springer, 13–51.

Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. 1990. Basic Local Alignment Search Tool. *J. Mol. Biol.* 215 (1990), 403–410.

Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. 2019. On the Separation and Equivalence of Paging Strategies and Other Online Algorithms. *Algorithmica* 81, 3 (2019), 1152–1179.

Spyros Angelopoulos, Christoph Dürr, and Shendan Jin. 2018. Online Maximum Matching with Recourse. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 117. Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, 8:1–8:15.

Spyros Angelopoulos, Marc P. Renault, and Pascal Schweitzer. 2020. Stochastic Dominance and the Bijective Ratio of Online Algorithms. *Algorithmica* 82, 5 (2020), 1101–1135.

Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. 2007. A Knapsack Secretary Problem with Applications. In *10th International Workshop on Approximation Algorithms for Combinatorial Optimization and 11th International Workshop on Randomization and Computation (APPROX/RANDOM) (Lecture Notes in Computer Science)*, Vol. 4627. Springer, 16–28.

Ran Bachrach and Ran El-Yaniv. 1997. Online List Accessing Algorithms and Their Applications: Recent Empirical Evidence. In *8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 53–62.

Ran Bachrach, Ran El-Yaniv, and M. Reinstädtler. 2002. On the Competitive Theory and Practice of Online List Accessing Algorithms. *Algorithmica* 32 (2002), 201–246.

Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. 1992. The Greedy Algorithm is Optimal for On-Line Edge Coloring. *Inform. Process. Lett.* 44, 5 (1992), 251–253.

Yair Bartal, Amos Fiat, and Stefano Leonardi. 1996. Lower Bounds for On-line Graph Problems with Application to On-line Circuit and Optical Routing. In *28th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 531–540.

Luca Becchetti. 2004. Modeling Locality: A Probabilistic Analysis of LRU and FWF. In *12th Annual European Symposium on Algorithms (ESA) (Lecture Notes in Computer Science)*, Vol. 3221. Springer, 98–109.

Shai Ben-David and Allan Borodin. 1994. A New Measure for the Study of On-Line Algorithms. *Algorithmica* 11, 1 (1994), 73–91.

Jon Louis Bentley and Catherine C. McGeoch. 1985. Amortized Analyses of Self-Organizing Sequential Search Heuristics. *Commun. ACM* 28 (1985), 404–411.

Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. 2017. Online Algorithms with Advice: The Tape Model. *Inform. Comput.* 254 (2017), 59–83.

Allan Borodin and Ran El-Yaniv. 1998. *Online Computation and Competitive Analysis*. Cambridge University Press.

Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. 1995. Competitive Paging with Locality of Reference. *J. Comput. Syst. Sci.* 50, 2 (1995), 244–258.

Joan Boyar, Martin R. Ehmsen, Jens S. Kohrt, and Kim S. Larsen. 2010. A Theoretical Comparison of LRU and LRU-K. *Acta Inform.* 47, 7–8 (2010), 359–374.

Joan Boyar, Stephan J. Eidenbenz, Lene M. Favrholdt, Michal Kotrbčík, and Kim S. Larsen. 2016. Online Dominating Set. In *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT) (Leibniz International Proceedings in Informatics LIPIcs)*, Vol. 53. Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, 21:1–21:15.

Joan Boyar, Leah Epstein, Lene M. Favrholdt, Kim S. Larsen, and Asaf Levin. 2018. Online-Bounded Analysis. *J. Scheduling* 21, 4 (2018), 429–441.

Joan Boyar, Leah Epstein, and Asaf Levin. 2010. Tight Results for Next Fit and Worst Fit with Resource Augmentation. *Theor. Comput. Sci.* 411, 26-28 (2010), 2572–2580.

Joan Boyar and Lene M. Favrholdt. 2007. The Relative Worst Order Ratio for On-Line Algorithms. *ACM T. Algorithms* 3, 2 (2007), article 22, 24 pages.

Joan Boyar and Lene M. Favrholdt. 2010. Scheduling Jobs on Grid Processors. *Algorithmica* 57, 4 (2010), 819–847.

Joan Boyar and Lene M. Favrholdt. 2012. A New Variable-Sized Bin Packing Problem. *J. Scheduling* 15, 3 (2012), 273–287.

Joan Boyar, Lene M. Favrholdt, Michal Kotrbčík, and Kim S. Larsen. 2017a. Relaxing the Irrevocability Requirement for Online Graph Algorithms. In *15th International Algorithms and Data Structures Symposium (WADS) (Lecture Notes in Computer Science)*, Vol. 10389. Springer, 217–228.

Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. 2017b. Online Algorithms with Advice: A Survey. *ACM Comput. Surv.* 50, 2 (2017), 19:1–19:34.

Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. 2007. The Relative Worst Order Ratio Applied to Paging. *J. Comput. Syst. Sci.* 73, 5 (2007), 818–843.

Joan Boyar, Sushmita Gupta, and Kim S. Larsen. 2012. Access Graphs Results for LRU versus FIFO under Relative Worst Order Analysis. In *13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT) (Lecture Notes in Computer Science)*, Vol. 7357. Springer, 328–339.

Joan Boyar, Sushmita Gupta, and Kim S. Larsen. 2015a. Relative Interval Analysis of Paging Algorithms on Access Graphs. *Theor. Comput. Sci.* 568 (2015), 28–48.

Joan Boyar, Sandy Irani, and Kim S. Larsen. 2015b. A Comparison of Performance Measures for Online Algorithms. *Algorithmica* 72, 4 (2015), 969–994.

Joan Boyar and Christian Kudahl. 2018. Adding Isolated Vertices Makes some Greedy Online Algorithms Optimal. *Discrete Appl. Math.* 246 (2018), 12–21.

Joan Boyar and Kim S. Larsen. 1999. The Seat Reservation Problem. *Algorithmica* 25, 4 (1999), 403–417.

Joan Boyar, Kim S. Larsen, and Abyayananda Maiti. 2014. A Comparison of Performance Measures via Online Search. *Theor. Comput. Sci.* 532 (2014), 2–13.

Joan Boyar, Kim S. Larsen, and Abyayananda Maiti. 2015. The Frequent Items Problem in Online Streaming Under Various Performance Measures. *Int. J. Found. Comput. S.* 26, 4 (2015), 413–440.

Joan Boyar, Kim S. Larsen, and Morten N. Nielsen. 2001. The Accommodating Function: a generalization of the competitive ratio. *SIAM J. Comput.* 31, 1 (2001), 233–258.

Joan Boyar, Kim S. Larsen, and Morten N. Nielsen. 2003. Extending the Accommodating Function. *Acta Inform.* 40, 1 (2003), 3–35.

Joan Boyar and Paul Medvedev. 2008. The Relative Worst Order Ratio Applied to Seat Reservation. *ACM T. Algorithms* 4, 4 (2008), article 48, 22 pages.

Niv Buchbinder, Moran Feldman, and Roy Schwartz. 2015. Online Submodular Maximization with Preemption. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 1202–1216.

Marie G. Christ, Lene M. Favrholdt, and Kim S. Larsen. 2014. Online Bin Covering: Expectations vs. Guarantees. *Theor. Comput. Sci.* 556 (2014), 71–84.

Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. 1991. New Results on Server Problems. *SIAM J. Discrete Math.* 4, 2 (1991), 172–181.

Marek Chrobak and John Noga. 1999. LRU Is Better than FIFO. *Algorithmica* 23, 2 (1999), 180–185.

Edward G. Coffman Jr., János Csirik, Lajos Rónyai, and Ambrus Zsbán. 2008. Random-Order Bin Packing. *Discrete Appl. Math.* 156 (2008), 2810–2816.

Marek Cygan, Łukasz Jeż, and Jiří Sgall. 2016. Online Knapsack Revisited. *Theor. Comput. Syst.* 58, 1 (2016), 153–190.

Peter J. Denning. 1968. The Working Set Model for Program Behaviour. *Commun. ACM* 11, 5 (1968), 323–333.

Peter J. Denning. 1980. Working Sets Past and Present. *IEEE T. Software Eng.* 6, 1 (1980), 64–84.

Nikhil R. Devanur and Thomas P. Hayes. 2009. The Adwords Problem: online keyword matching with budgeted bidders under random permutations. In *10th ACM Conference on Electronic Commerce (EC)*. ACM, 71–78.

Stefan Dobrev, Rastislav Kralovič, and Dana Pardubská. 2009. Measuring the Problem-Relevant Information in Input. *RAIRO - Theor. Inf. Appl.* 43, 3 (2009), 585–613.

Reza Dorrigiv, Martin R. Ehmsen, and Alejandro López-Ortiz. 2015. Parameterized Analysis of Paging and List Update Algorithms. *Algorithmica* 71, 2 (2015), 330–353.

Reza Dorrigiv, Alejandro López-Ortiz, and J. Ian Munro. 2009. On the Relative Dominance of Paging Algorithms. *Theor. Comput. Sci.* 410 (2009), 3694–3701.

Martin R. Ehmsen, Lene M. Favrholdt, Jens S. Kohrt, and Rodica Mihai. 2010. Comparing First-Fit and Next-Fit for Online Edge Coloring. *Theor. Comput. Sci.* 411 (2010), 1734–1741.

Martin R. Ehmsen, Jens S. Kohrt, and Kim S. Larsen. 2013. List Factoring and Relative Worst Order Analysis. *Algorithmica* 66, 2 (2013), 287–309.

Ran El-Yaniv. 1998. Competitive Solutions for Online Financial Problems. *Comput. Surveys* 30, 1 (1998), 28–69.

Ran El-Yaniv, Amos Fiat, Richard M. Karp, and G. Turpin. 2001. Optimal Search and One-Way Trading Online Algorithms. *Algorithmica* 30, 1 (2001), 101–139.

Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. 2006. Separating Scheduling Algorithms with the Relative Worst Order Ratio. *J. Comb. Optim.* 12, 4 (2006), 362–385.

Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. 2012. Comparing Online Algorithms for Bin Packing Problems. *J. Scheduling* 15, 1 (2012), 13–21.

Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. 2011. Improved Approximation Guarantees for Weighted Matching in the Semi-streaming Model. *SIAM J. Discrete Math.* 25, 3 (2011), 1251–1265.

Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. 2013. Improved Bounds for Online Preemptive Matching. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, 389–399.

Lene M. Favrholdt and Morten N. Nielsen. 2003. On-Line Edge Coloring with a Fixed Number of Colors. *Algorithmica* 35, 2 (2003), 176–191.

Carsten Fischer and Heiko Röglin. 2016. Probabilistic Analysis of the Dual Next-Fit Algorithm for Bin Covering. In *16th Latin American Symposium on Theoretical Informatics (LATIN) (Lecture Notes in Computer Science)*, Vol. 9644. Springer, 469–482.

Juan A. Garay, Inder S. Gopal, Shay Kutten, Yishay Mansour, and Moti Yung. 1997. Efficient On-Line Call Control Algorithms. *J. Algorithms* 23, 1 (1997), 180–194.

Oliver Göbel, Thomas Kesselheim, and Andreas Tönnis. 2015. Online Appointment Scheduling in the Random Order Model. In *23rd Annual European Symposium on Algorithms (ESA) (Lecture Notes in Computer Science)*, Vol. 9294. Springer, 680–692.

Gagan Goel and Aranyak Mehta. 2008. Online Budgeted Matching in Random Input Models with Applications to Adwords. In *19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 982–991.

Ronald L. Graham. 1969. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.* 17, 2 (1969), 416–429.

Albert Gu, Anupam Gupta, and Amit Kumar. 2016. The Power of Deferral: Maintaining a Constant-Competitive Steiner Tree Online. *SIAM J. Comput.* 45, 1 (2016), 1–28.

Anupam Gupta and Amit Kumar. 2014. Online Steiner Tree with Deletions. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 455–467.

András Gyárfás and Jenő Lehel. 1990. First Fit and On-Line Chromatic Number of Families of Graphs. *Ars Comb.* 29, C (1990), 168–176.

Xin Han, Yasushi Kawase, and Kazuhisa Makino. 2015. Randomized algorithms for online knapsack problems. *Theor. Comput. Sci.* 562 (2015), 395–405.

Xin Han, Yasushi Kawase, Kazuhisa Makino, and He Guo. 2014. Online removable knapsack problem under convex function. *Theor. Comput. Sci.* 540 (2014), 62–69.

Xin Han and Kazuhisa Makino. 2016. Online minimization knapsack problem. *Theor. Comput. Sci.* 609 (2016), 185–196.

Benjamin Hiller and Tjark Vredeveld. 2008. Probabilistic analysis of Online Bin Coloring algorithms via Stochastic Comparison. In *16th Annual European Symposium on Algorithms (ESA) (Lecture Notes in Computer Science)*, Vol. 5193. Springer, 528–539.

Juraj Hromkovič, Rastislav Královič, and Richard Královič. 2010. Information Complexity of Online Problems. In *35th International Symposium on the Mathematical Foundations of Computer Science (MFCS) (Lecture Notes in Computer Science)*, Vol. 6281. Springer, 24–36.

Makoto Imase and Bernard M. Waxman. 1991. Dynamic Steiner Tree Problem. *SIAM J. Discrete Math.* 4, 3 (1991), 369–384.

Sandy Irani. 1991. Two Results on the List Update Problem. *Inform. Process. Lett.* 38, 6 (1991), 301–306.

Kazuo Iwama and Shiro Taketomi. 2002. Removable Online Knapsack Problems. In *29th International Colloquium on Automata, Languages and Programming (ICALP) (Lecture Notes in Computer Science)*, Vol. 2380. Springer, 293–305.

David S. Johnson. 1973. *Near-optimal bin packing algorithms*. Ph.D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA.

David S. Johnson. 1974. Fast Algorithms for Bin Packing. *J. Comput. Syst. Sci.* 8 (1974), 272–314.

David S. Johnson, Alan Demers, Jeffrey D. Ullman, M. R. Garey, and Ronald L. Graham. 1974. Worst-Case Performance Bound for Simple One-Dimensional Packing Algorithms. *SIAM J. Comput.* 3 (1974), 299–325.

Bala Kalyanasundaram and Kirk Pruhs. 2000. Speed is as Powerful as Clairvoyance. *J. ACM* 47 (2000), 617–643.

Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. 1988. Competitive Snoopy Caching. *Algorithmica* 3 (1988), 79–119.

Anna R. Karlin, Steven J. Phillips, and Prabhakar Raghavan. 2000. Markov Paging. *SIAM J. Comput.* 30, 3 (2000), 906–922.

Claire Kenyon. 1996. Best-Fit Bin-Packing with Random Order. In *7th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 359–364.

Dennis Komm, Rastislav Královič, Richard Královič, and Christian Kudahl. 2016. Advice Complexity of the Online Induced Subgraph Problem. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 58. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 59:1–59:13.

Elias Koutsoupias and Christos H. Papadimitriou. 2000. Beyond Competitive Analysis. *SIAM J. Comput.* 30, 1 (2000), 300–317.

Sven Oliver Krumke, Willem de Paepe, Jörg Rambau, and Leen Stougie. 2008. Bincoloring. *Theor. Comput. Sci.* 407, 1–3 (2008), 231–241.

John McCabe. 1965. On Serial Files with Relocatable Records. *Oper. Res.* 13, 4 (1965), 609–618.

Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. 2016. The Power of Recourse for Online MST and TSP. *SIAM J. Comput.* 45, 3 (2016), 859–880.

Adam Meyerson. 2001. Online Facility Location. In *42nd IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 426–433.

Gabriel Moruz and Andrei Negoescu. 2012. Outperforming LRU via competitive analysis on parametrized inputs for paging. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 1669–1680.

Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. 1993. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *ACM SIGMOD International Conference on Management of Data*. ACM, 297–306.

Christopher J. Osborn and Eric Torng. 2008. List's Worst-Average-Case or WAC ratio. *J. Scheduling* 11 (2008), 213–215.

Konstantinos Panagiotou and Alexander Souza. 2006. On adequate performance measures for paging. In *38th Annual ACM Symposium on Theory of Computing*. ACM, 487–496.

Prabhakar Raghavan. 1992. A Statistical Adversary for On-Line Algorithms. In *On-Line Algorithms (DIMACS: Series in Discrete Mathematics and Theoretical Computer Science)*, Vol. 7. American Mathematical Society, 79–83.

Dror Rawitz and Adi Rosén. 2016. Online Budgeted Maximum Coverage. In *24th Annual European Symposium on Algorithms (ESA) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 57. Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, 73:1–73:17.

Barna Saha and Lise Getoor. 2009. On Maximum Coverage in the Streaming Model & Application to Multi-topic Blog-Watch. In *SIAM International Conference on Data Mining*. SIAM, 697–708.

Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM* 28, 2 (1985), 202–208.

Daniel A. Spielman and Shang-Hua Teng. 2004. Smoothed Analysis of Algorithms: Why the Simplex Algorithm usually Takes Polynomial Time. *J. ACM* 51, 3 (2004), 385–463.

Boris Teia. 1993. A Lower Bound for Randomized List Update Algorithms. *Inform. Process. Lett.* 47 (1993), 5–9.

Neal E. Young. 1994. The k-Server Dual and Loose Competitiveness for Paging. *Algorithmica* 11 (1994), 525–541.

Neal E. Young. 2000. On-Line Paging Against Adversarially Biased Random Inputs. *J. Algorithms* 37 (2000), 218–235.