Algorithmica manuscript No.
(will be inserted by the editor)

A Comparison of Performance Measures for Online Algorithms

Joan Boyar · **Sandy Irani** · **Kim S. Larsen**

Received: date / Accepted: date

Abstract This paper provides a systematic study of several proposed measures for online algorithms in the context of a specific problem, namely, the two server problem on three colinear points. Even though the problem is simple, it encapsulates a core challenge in online algorithms which is to balance greediness and adaptability. We examine Competitive Analysis, the Max/Max Ratio, the Random Order Ratio, Bijective Analysis and Relative Worst Order Analysis, and determine how these measures compare the Greedy Algorithm, Double Coverage, and Lazy Double Coverage, commonly studied algorithms in the context of server problems. We find that by the Max/Max Ratio and Bijective Analysis, Greedy is the best of the three algorithms. Under the other measures, Double Coverage and Lazy Double Coverage are better, though Relative Worst Order Analysis indicates that Greedy is sometimes better. Only Bijective Analysis and Relative Worst Order Analysis indicate that Lazy Double Coverage is better than Double Coverage. Our results also provide the first proof of optimality of an algorithm under Relative Worst Order Analysis.

Keywords Online algorithms · K-server problem · Performance measures

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

S. Irani
Department of Computer Science, University of California, Irvine, CA 92697, USA
Tel.: +1 (949) 824-6346
E-mail: irani@ics.uci.edu

K. Larsen
Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark
Tel.: +45 6550 2328
E-mail: kslarsen@imada.sdu.dk

1 Introduction

Since its introduction by Sleator and Tarjan in 1985 [19], Competitive Analysis has been the most widely used method for evaluating online algorithms. A problem is said to be *online* if the input to the problem is given a piece at a time, and the algorithm must commit to parts of the solution over time before the entire input is revealed to the algorithm. *Competitive Analysis* evaluates an online algorithm in comparison to the optimal offline algorithm which receives the input in its entirety in advance and has unlimited computational power in determining a solution. Informally speaking, one considers the worst-case input which maximizes the ratio of the cost of the online algorithm for that input to the cost of the optimal offline algorithm on that same input. The maximum ratio achieved is called the *Competitive Ratio*. Thus, one factors out the inherent difficulty of a particular input (for which the offline algorithm is penalized along with the online algorithm) and measures what is lost in making decisions with partial information and/or limited power.

Despite the popularity of Competitive Analysis, researchers have been well aware of its deficiencies and have been seeking better alternatives almost since the time that it came into wide use. (See [10] for a fairly recent survey.) Many of the problems with Competitive Analysis stem from the fact that it is a worst case measure and fails to examine the performance of algorithms on instances that would be expected in a particular application. It has also been observed that Competitive Analysis sometimes fails to distinguish between algorithms which have very different performance in practice and intuitively differ in quality.

Over the years, researchers have devised alternatives to Competitive Analysis, each designed to address one or all of its shortcomings. There are exceptions, but it is fair to say that many alternatives are application-specific, and very often, the papers in which they are introduced only present a direct comparison between a new measure and Competitive Analysis.

This paper is a study of several generally-applicable alternative measures for evaluating online algorithms that have been suggested in the literature. We perform this comparison in the context of a particular problem: the 2-server problem on the line with three possible request points, nick-named here the *baby server problem*. Investigating simple *k*-server problems to shed light on new ideas has also been done in [3], for instance.

We focus on three algorithms, GREEDY, DOUBLE COVERAGE (DC) [9], and LAZY DOUBLE COVERAGE (LDC), and four different analysis techniques (performance measures): Bijective Analysis, the Max/Max Ratio, the Random Order Ratio, and Relative Worst Order Analysis.

In investigating the baby server problem, we find that according to some quality measures for online algorithms, GREEDY is better than DC and LDC, whereas for others, DC and LDC are better than GREEDY. In addition, for some measures LDC is better than DC, while for others they are indistinguishable.

The analysis methods that conclude that DC and LDC are better than GREEDY are focused on a worst-case sequence for the ratio of an algorithm's cost compared to OPT. In the case of GREEDY vs. DC and LDC, this conclusion makes use of the fact that there exists a family of sequences for which GREEDY's cost is unboundedly

larger than the cost of OPT, whereas for each of DC and LDC, the cost is always at most a factor of two larger than the cost of OPT.

On the other hand, the measures that conclude that GREEDY is best compare two algorithms based on the multiset of costs stemming from the set of all sequences of a fixed length. In the case of GREEDY and LDC, this makes use of the fact that for any fixed n, both the maximum as well as the average cost of LDC over all sequences of length n are greater than the corresponding values for GREEDY.

Using Relative Worst Order Analysis a more nuanced result is obtained, concluding that LDC can be at most a factor of two worse than GREEDY, while GREEDY can be unboundedly worse than LDC.

The analysis methods that distinguish between DC and LDC (Bijective Analysis and Relative Worst Order Analysis) take advantage of the fact that LDC performs at least as well as DC on every sequence and performs better on some. The others (Competitive Analysis, the Max/Max Ratio, and the Random Order Ratio) cannot distinguish between them, due to the intermediate comparison to OPT, i.e., algorithms are compared to OPT and then the results of this comparison are compared. On some sequences where DC and LDC do worst in comparison with OPT, they perform identically, so these worst case measures conclude that the two algorithms perform identically overall. This phenomenon occurs in other problems also. For example, some analysis methods fail to distinguish between the paging algorithms LRU and FWF, even though the former is clearly better and is at least as good on every sequence.

The simplicity of the baby server problem also enables us to give the first proof of optimality in Relative Worst Order Analysis: LDC is an optimal algorithm for this problem.

Though our main focus is the greediness/adaptability issue that we investigate through the analyses of GREEDY and LDC over a broad collection of quality measures, we also include some results about the balance algorithm [18], BAL. Because of the interest for this server algorithm in the literature, we find it natural to mention the results for BAL that can be obtained relatively easily within our framework.

2 Preliminaries

In this section, we define the server problem used throughout this paper as the basis for our comparison. We also define the server algorithms used, and the quality measures which are the subject of this study.

2.1 The Server Problem

Server problems [5] have been the objects of many studies. In its full generality, one assumes that some number k of servers are available in some metric space. Then a sequence of requests must be treated. A request is simply a point in the metric space, and a k-server algorithm must move servers in response to the request to ensure that at least one server is placed on the request point. A cost is associated with any move of a server (this is usually the distance moved in the given metric space), and the

objective is to minimize total cost. The initial configuration (location of servers) may or may not be a part of the problem formulation.

In investigating the strengths and weaknesses of the various measures for the quality of online algorithms, we define the simplest possible nontrivial server problem:

Definition 1 The *baby server problem* is a 2-server problem on the line with three possible request points *A*, *B*, and *C*, in that order from left to right, with distance one between *A* and *B* and integral distance $d \geq 2$ between *B* and *C*. The cost of moving a server is defined to be the distance it is moved. We assume that initially the two servers are placed on *A* and *C*.

As a side remark, we have considered most proofs in this paper in the context of a non-integral distance *d* between *B* and *C*. The main conclusions remain the same, but many of the proofs become longer and the formulas less readable. In a few places, we consider variants of LDC, where the right-most server moves at a speed *a* times faster than the left-most server. Also in this case we assume that d/a is integral in order to highlight the core findings.

All results in the paper pertain to the baby server problem. Even though the problem is simple, it requires balancing greediness and adaptability which is a central problem in all *k*-server settings and many online problems in general. This simple problem we consider is sufficient to show the non-competitiveness of GREEDY with respect to Competitive Analysis [5].

2.2 Server Algorithms

First, we define some relevant properties of server algorithms:

Definition 2 A server algorithm is called

- *noncrossing* if servers never change their relative position on the line.
- *lazy* [18] if it never moves more than one server in response to a request and it does not move any servers if the requested point is already occupied by a server.

A server algorithm fulfilling both these properties is called *compliant*.

Given an algorithm, $\mathbb{A}$, we define the algorithm *lazy* $\mathbb{A}$, $\mathscr{L}\mathbb{A}$, as follows: $\mathscr{L}\mathbb{A}$ will maintain a *virtual* set of servers and their locations as well as the real set of servers in the metric space. There is a one-to-one correspondence between real servers and virtual servers. The virtual set will simulate the behavior of $\mathbb{A}$. The initial server positions of the virtual and real servers are the same.

When a request arrives, the virtual servers are moved in accordance with algorithm $\mathbb{A}$. After this happens, there will always be at least one virtual server on the requested point. Then the real servers move to satisfy the request: If there is already a real server on the requested point, nothing more happens. Otherwise, the real server corresponding to the virtual server on the requested point moves to the requested point. If there is more than one virtual server on the requested point, tie-braking rules

may be applied. In our case, we will pick the closest server to move to the requested point.

General *k*-server problems that are more complicated than the baby server problem may need more involved tie-breaking rules to be deterministically defined. Note that as a special case of the above, a virtual move can be of distance zero, while still leading to a real move of non-zero distance.

In [9], it was observed that for any 2-server algorithm, there exists a noncrossing algorithm with the same cost on all sequences. In [18], it was observed that for an algorithm $\mathbb{A}$ and its lazy version $\mathscr{L}\mathbb{A}$, for any sequence I of requests, $\mathbb{A}(I) \geq \mathscr{L}\mathbb{A}(I)$ (we refer to this as the *laziness observation*). Note that the laziness observation applies to the general *k*-server problem, so the results that depend only on this observation can also be generalized beyond the baby server problem.

We define a number of algorithms by specifying their behavior on the next request point, p. For all algorithms considered here, no moves are made if a server already occupies the request point (though internal state changes are sometimes made in such a situation).

GREEDY moves the closest server to p. Note that due to the problem formulation, ties cannot occur (and the server on C is never moved).

If p is in between the two servers, Double Coverage (DC), moves both servers at the same speed in the direction of p until at least one server reaches the point. If p is on the same side of both servers, the nearest server moves to p.

We define a-DC to work in the same way as DC, except that the right-most server moves at a speed $a \leq d$ times faster than the left-most server. We refer to the lazy version of DC as LDC and the lazy version of a-DC as a-LDC.

The balance algorithm [18], BAL, makes its decisions based on the total distance travelled by each server. For each server, s, let d_s denote the total distance travelled by s from the initiation of the algorithm up to the current point in time. On a request, BAL moves a server, aiming to obtain the smallest possible $\max_s d_s$ value *after* the move. In case of a tie, BAL moves the server which must move the furthest.

As an example, showing that some care must be taken when defining the lazy algorithms, consider the following server problem which is slightly more complicated than the one we consider in the rest of the paper. We illustrate the example in Figure 1. There are four points $A = 0$, $B = 2$, $C = 6$, and $D = 11$ in use, and three servers, initially on A, B, and D. We consider the request sequence CBC, served by LDC. After the first request to C, we have the configuration A (A), C (C), D (7), where the server positions are listed from left to right with their virtual positions in parentheses. At the request to B, it becomes B (B), C (4), D (7). Now, when requesting C again, note that virtually, the right-most server is closest, but the middle server is actually on C.

2.3 Quality Measures

In analyzing algorithms for the baby server problem, we consider input sequences I of request points. An algorithm $\mathbb{A}$, which treats such a sequence has some cost, which is the total distance moved by the two servers. This cost is denoted by $\mathbb{A}(I)$. Since I

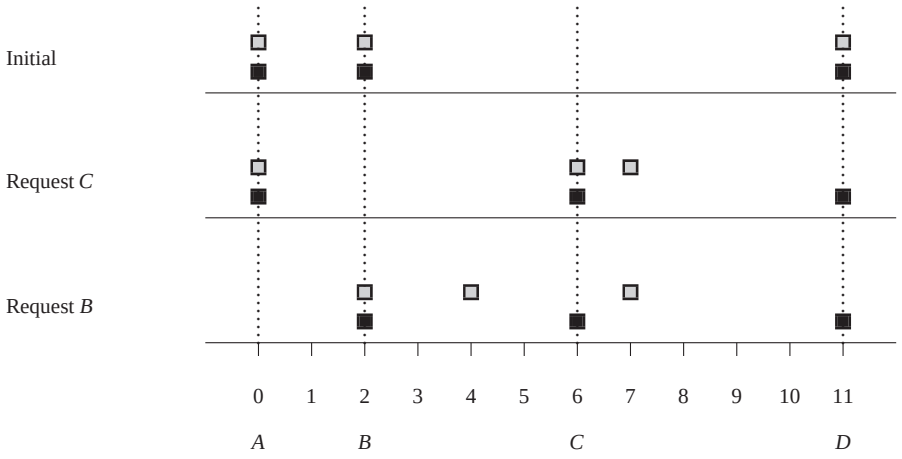

Fig. 1 Illustration of the 3-server example. The server positions are given in black and the virtual positions in gray.

is of finite length, it is clear that there exists an offline algorithm with minimal cost. By OPT, we refer to such an algorithm and $\text{OPT}(I)$ denotes the unique minimal cost of processing I.

Two of the measures below use permutations of input sequences. If I is an input sequence of length n and σ is a permutation on n elements, then we let $\sigma(I)$ denote I permuted by σ.

All of the measures described below can lead to a conclusion as to which one of two algorithms is better. In contrast to the others, Bijective Analysis does not quantify how much better one algorithm is than another.

2.3.1 Competitive Analysis

In Competitive Analysis [13,19,15], we define an algorithm $\mathbb{A}$ to be c-competitive if there exists a constant α such that for all input sequences I, $\mathbb{A}(I) \leq c\,\text{OPT}(I) + \alpha$.

2.3.2 The Max/Max Ratio

The Max/Max Ratio [4] compares an algorithm's worst cost for any sequence of length n to OPT's worst cost for any sequence of length n. The Max/Max Ratio of an algorithm $\mathbb{A}$, $w_M(\mathbb{A})$, is $M(\mathbb{A})/M(\text{OPT})$, where

$$M(\mathbb{A}) = \limsup_{t\to\infty} \max_{|I|=t} \mathbb{A}(I)/t.$$

2.3.3 The Random Order Ratio

Kenyon [16] defines the Random Order Ratio to be the worst ratio obtained over all sequences I, comparing the expected value of an algorithm, $\mathbb{A}$, with respect to a

uniform distribution of all permutations of I, to the value of OPT on I:

$$\limsup_{\text{OPT}(I)\to\infty} \frac{E_\sigma\left[\mathbb{A}(\sigma(I))\right]}{\text{OPT}(I)}$$

The original context for this definition is Bin Packing for which the optimal packing is the same, regardless of the order in which the items are presented. Therefore, it does not make sense to take an average over all permutations for OPT. For server problems, however, the order of requests in the sequence may very well change the cost of OPT, so we compare to OPT's performance, also on a random permutation of the input sequence. In addition, taking the limit as $\text{OPT}(I) \to \infty$, causes a problem with analyzing GREEDY on the baby server problem (and presumably other algorithms for other problems), since there is an infinite family of sequences, I_n, where OPT's cost on I_n is the same constant for all n, but GREEDY's cost grows with n. Thus, we consider the limit as the length of the sequence goes to infinity, as in another alternative definition of the Random Order Ratio in [14]. The obvious possible modifications to the Random Order Ratio to include the expectation over OPT are the following two:

$$\limsup_{|I|\to\infty} \frac{E_\sigma\left[\mathbb{A}(\sigma(I))\right]}{E_\sigma\left[\text{OPT}(\sigma(I))\right]} \qquad \limsup_{|I|\to\infty} E_\sigma\left[\frac{\mathbb{A}(\sigma(I))}{\text{OPT}(\sigma(I))}\right]$$

We prefer the one to the left. In general, these two definitions could give different results. However, due to the concrete nature of our proofs, the results in this paper hold independently of which definition is chosen.

2.3.4 Bijective Analysis and Average Analysis

In [1], Bijective and Average Analysis are defined, as methods of comparing two online algorithms directly. We adapt those definitions to the notation used here. As with the Max/Max Ratio and Relative Worst Order Analysis, the two algorithms are not necessarily compared on the same sequence.

In Bijective Analysis, the sequences of a given length are mapped, using a bijection, onto the same set of sequences. The performance of the first algorithm on a sequence, I, is compared to the performance of the second algorithm on the sequence I is mapped to. If I_n denotes the set of all input sequences of length n, then an online algorithm $\mathbb{A}$ is no worse than an online algorithm $\mathbb{B}$ according to Bijective Analysis if there exists an integer $n_0 \geq 1$ such that for each $n \geq n_0$, there is a bijection $f\colon I_n \to I_n$ satisfying $\mathbb{A}(I) \leq \mathbb{B}(f(I))$ for each $I \in I_n$. $\mathbb{A}$ is strictly better than $\mathbb{B}$ if $\mathbb{A}$ is no worse than $\mathbb{B}$, and there is no bijection showing that $\mathbb{B}$ is no worse than $\mathbb{A}$.

Average Analysis can be viewed as a relaxation of Bijective Analysis. An online algorithm $\mathbb{A}$ is no worse than an online algorithm $\mathbb{B}$ according to Average Analysis if there exists an integer $n_0 \geq 1$ such that for each $n \geq n_0$, $\Sigma_{I\in I_n}\mathbb{A}(I) \leq \Sigma_{I\in I_n}\mathbb{B}(I)$. $\mathbb{A}$ is strictly better than $\mathbb{B}$ if this inequality is strict.

Measure	Value
Competitive Ratio	$CR_{\mathbb{A}} = \max_I \frac{\mathbb{A}(I)}{\textsc{Opt}(I)}$
Max/Max Ratio	$MR_{\mathbb{A}} = \frac{\max_{\lvert I\rvert=n} \mathbb{A}(I)}{\max_{\lvert I'\rvert=n} \textsc{Opt}(I')}$
Random Order Ratio	$RR_{\mathbb{A}} = \max_I \frac{E_\sigma\left[\mathbb{A}(\sigma(I))\right]}{E_\sigma\left[\textsc{Opt}(\sigma(I))\right]}$
Relative Worst Order Ratio	$WR_{\mathbb{A},\mathbb{B}} = \max_I \frac{\max_\sigma \mathbb{A}(\sigma(I))}{\max_{\sigma'} \mathbb{B}(\sigma'(I))}$

Table 1 Comparison of those measures which give a ratio.

2.3.5 Relative Worst Order Analysis

Relative Worst Order Analysis was introduced in [6] and extended in [7]. It compares two online algorithms directly. As with the Max/Max Ratio, it compares two algorithms on their worst sequence in the same part of a partition. The partition is based on the Random Order Ratio, so that the algorithms are compared on sequences having the same content, but possibly in different orders.

Definition 3 For any pair of algorithms $\mathbb{A}$ and $\mathbb{B}$, we define

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

where $\mathbb{A}_{\mathrm{W}}(I) = \max_\sigma \mathbb{A}(\sigma(I))$.

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

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

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

If at least one of the ratios $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ and $c_{\mathrm{u}}(\mathbb{B},\mathbb{A})$ is finite, then the algorithms $\mathbb{A}$ and $\mathbb{B}$ are called $(c_u(\mathbb{A},\mathbb{B}), c_u(\mathbb{B},\mathbb{A}))$*-related*.

Algorithms $\mathbb{A}$ and $\mathbb{B}$ are *weakly comparable in $\mathbb{A}$'s favor*, 1) if $\mathbb{A}$ and $\mathbb{B}$ are comparable in $\mathbb{A}$'s favor, 2) if $c_{\mathrm{u}}(\mathbb{A},\mathbb{B})$ is finite and $c_{\mathrm{u}}(\mathbb{B},\mathbb{A})$ is infinite, or 3) if $c_{\mathrm{u}}(\mathbb{A},\mathbb{B}) \in o(c_{\mathrm{u}}(\mathbb{B},\mathbb{A}))$.

An informal summary, comparing these measures is given in Table 1. Note that some details are missing, including the additive constants for asymptotic analysis.

Measure	Favored Algorithm	DC vs. LDC
Competitive Ratio	LDC	identical
Max/Max Ratio	GREEDY	identical
Random Order Ratio	LDC	identical
Bijective Analysis	GREEDY	LDC best
Average Analysis	GREEDY	LDC best
Relative Worst Order Ratio	LDC weakly favored	LDC best

Table 2 The second column summarizes the results comparing LDC and GREEDY on the baby server problem using each of the measures defined. In addition to the information in the column, GREEDY is uniquely optimal according to Bijective and Average Analysis, and LDC and GREEDY are $(2,\infty)$-related according to Relative Worst Order Analysis. The third column lists which measures distinguish between DC and its lazy variant, LDC.

Table 2 is a summary of the results comparing LDC and GREEDY on the baby server problem using each of the measures defined. Additionally, it lists the effect of laziness applied to DC.

3 Competitive Analysis

The k-server problem has been studied using Competitive Analysis starting in [17]. In [9], it is shown that on the real line, the Competitive Ratios of DC and LDC are k, which is optimal, and that GREEDY is not competitive. The result in [17], showing that the Competitive Ratio of BAL is $n-1$ on a metric space with n points if $k=n-1$, shows that BAL has the same Competitive Ratio of 2 as DC and LDC on the baby server problem.

4 The Max/Max Ratio

In [4], a concrete example is given with two servers and three non-colinear points. It is observed that the Max/Max Ratio favors the greedy algorithm over the balance algorithm, BAL.

BAL behaves similarly to LDC and identically on LDC's worst case sequences. The following theorem shows that the same conclusion is reached when the three points are on the line.

Theorem 1 GREEDY *is better than* DC *and* LDC *on the baby server problem with respect to the Max/Max Ratio, with* $\frac{w_M(\text{DC})}{w_M(\text{GREEDY})} = \frac{w_M(\text{LDC})}{w_M(\text{GREEDY})} = 1+\frac{d-1}{d+1}$.

Proof Given a sequence of length n, GREEDY's maximum cost is n, implying that $M(\text{GREEDY}) = 1$.

Since OPT is at least as good as GREEDY, its cost is at most n. Thus, $M(\text{OPT}) \leq 1$. To obtain a lower bound for $M(\text{OPT})$, we consider request sequences consisting of repetitions of the sequence $((BA)^dC)^k$. In each such repetition, OPT must incur a cost of at least $2d$. Thus, we can bound $M(\text{OPT})$ by $M(\text{OPT}) \geq \frac{2d}{2d+1}$.

We now determine $M(\text{LDC})$, and the same argument holds for $M(\text{DC})$.

For any positive integer n, we define the sequence $I_n = ((BA)^dBC)^pX$ of length n, where the length of the alternating A/B-sequence before the C is $2d+1$, X is a possibly empty alternating sequence of As and Bs starting with a B, $|X| = n \bmod (2d+2)$, and $p = \frac{n-|X|}{2d+2}$.

First, we claim that I_n is a sequence of length n where LDC has the largest average cost per move. Each move that the right-most server, originally on C, makes costs $d > 1$ and the left-most server's moves cost only one. For every move the right-most server makes from C to B, there are d moves by the left-most server from A to B and thus d moves back from B to A. The subsequence $(BA)^d$ does this with cost one for LDC for every move. Since the move after every C has cost one, it is impossible to define another sequence with a larger average cost per move.

If $|X| < 2d+1$, then the server on C does not move again, and $\text{LDC}(I_n) = p(2d+2d) + |X| = n + \frac{(d-1)(n-|X|)}{d+1}$.

Otherwise, $|X| = 2d+1$, the server on C is moved to B, and we obtain $\text{LDC}(I_n) = p(2d+2d) + |X| + d - 1 = n + \frac{(d-1)(n-|X|)}{d+1} + d - 1$.

Since we are taking the supremum, we restrict our attention to sequences where $|X| = 0$. Thus, $M(\text{LDC}) = \frac{n+\frac{(d-1)n}{d+1}}{n} = 1 + \frac{d-1}{d+1}$.

Finally,

$$w_M(\text{GREEDY}) = \frac{M(\text{GREEDY})}{M(\text{OPT})} = \frac{1}{M(\text{OPT})},$$

while

$$w_M(\text{LDC}) = \frac{M(\text{LDC})}{M(\text{OPT})} = \frac{1+\frac{d-1}{d+1}}{M(\text{OPT})}.$$

Since $M(\text{OPT})$ is bounded, $\frac{w_M(\text{LDC})}{w_M(\text{GREEDY})} = 1 + \frac{d-1}{d+1}$, which is greater than one for $d > 1$.

It follows from the proof of this theorem that GREEDY is close to optimal with respect to the Max/Max Ratio, since the cost of GREEDY divided by the cost of OPT tends toward one for large d.

Since LDC and DC perform identically on their worst sequences of any given length, they also have the same Max/Max Ratio.

5 The Random Order Ratio

The Random Order Ratio categorizes DC and LDC as being equally good. The proof is structured into several lemmas below.

In the following, we use the term *run* to mean a sequence of the same item in a longer sequence, and it is *maximal* if it cannot be made longer by including a possible neighboring item. For example, the three maximal runs of *A*s in *AAABAAAABBA* have lengths 3, 4, and 1, respectively.

The Random Order Ratio is the worst ratio obtained over all sequences, comparing the expected value of an algorithm over all permutations of a given sequence to the expected value of OPT over all permutations of the given sequence. The intuition in establishing the following result is that if one chooses a random permutation of a sequence with many more *A*s and *B*s than *C*s, then, with high probability, there will be sufficiently many switches between requests to *A*s and *B*s in between each two successive occurrences of *C*s that both DC and LDC will experience the full penalty compared to OPT, i.e., after each request to *C*, they will use one server to serve requests to both *A* and *B* before eventually moving the server from *C* to *B*, where it will stay until the next request to *C*.

The two main components in the proof are the following: First, even though we choose a sequence with many more *A*s and *B*s than *C*s, we must prove that with high probability, there are enough requests between any two *C*s. If there are just a small constant fraction of pairs of successive *C*s that do not have enough *A*s and *B*s in between them, we will not get the Random Order Ratio of two that we are trying to obtain. Second, even though there are many requests to *A*s and *B*s in between two consecutive *C*s, if the *A*s or *B*s, respectively, appear as runs too frequently (many *A*s in a row, followed by many *B*s in a row), then there will not be sufficiently many switches between requests to *A*s and *B*s to pull a server from *C* to *B*. Again, we cannot afford to have this problem occur a constant fraction of the times if we want a ratio of two.

In the proof, we choose to use n requests to *A*s as well as *B*s and $\lfloor \log n \rfloor$ requests to *C*s. In addition, we limit the successive requests to *A*s and *B*s separately to $\lfloor \sqrt{n} \rfloor$ with high probability. The choice of the functions n, $\log n$, and $\sqrt{n}$ is mostly to work with familiar functions in the lemmas below. Many other choices of functions would work, as long as their rates of growth are similar. It is not quite sufficient that they are different, since we also need to use, for instance, that $\sqrt{n}\log^2 n \in o(n)$.

We use the notation $[n]_r$, where $r \leq n$, for the expression $n(n-1)(n-2)\cdots(n-r+1)$.

The following result is from [8], using the index for the last term of the summation from [2, page 56]. We have substituted in our variable names:

Proposition 1 *In a random permutation of n As and n Bs, the probability that the longest run of As (or Bs) is shorter than r is*

$$P(r) = 1 - \tbinom{n+1}{1}\frac{[n]_r}{[2n]_r} + \tbinom{n+1}{2}\frac{[n]_{2r}}{[2n]_{2r}} - \tbinom{n+1}{3}\frac{[n]_{3r}}{[2n]_{3r}} + \cdots$$
$$+(-1)^{\lfloor \frac{n}{r} \rfloor}\binom{n+1}{\lfloor \frac{n}{r} \rfloor}\frac{[n]_{(\lfloor \frac{n}{r} \rfloor r)}}{[2n]_{(\lfloor \frac{n}{r} \rfloor r)}}$$

We first derive a simple lower bound on this probability.

Lemma 1 *If $r \geq \log n$, then in a random permutation of n As and n Bs, the probability $P(r)$ that the longest run of Bs is shorter than r is at least $1 - \frac{n+1}{2^r}$.*

Proof We first prove that the absolute value of the terms in the expression for $P(r)$ from Proposition 1 are non-increasing. Let $1 \leq i \leq \left\lfloor \frac{n}{r} \right\rfloor - 1$. We consider two successive terms and show that the absolute value of the first is at least as large as the absolute value of the second, provided that $r \geq \log n$.

$$
\begin{array}{l}
\binom{n+1}{i} \frac{[n]_{ir}}{[2n]_{ir}} \geq \binom{n+1}{i+1} \frac{[n]_{(i+1)r}}{[2n]_{(i+1)r}} \\
\Updownarrow \\
\binom{n+1}{i} \frac{n(n-1)\cdots(n-ir+1)}{2n(2n-1)\cdots(2n-ir+1)} \geq \binom{n+1}{i+1} \frac{n(n-1)\cdots(n-(i+1)r+1)}{2n(2n-1)\cdots(2n-(i+1)r+1)} \\
\Updownarrow \\
\binom{n+1}{i} \geq \binom{n+1}{i+1} \frac{(n-ir)(n-ir-1)\cdots(n-(i+1)r+1)}{(2n-ir)(2n-ir-1)\cdots(2n-(i+1)r+1)} \\
\Uparrow \\
\frac{(n+1)!}{i!(n+1-i)!} \geq \frac{(n+1)!}{(i+1)!(n-i)!} \left(\frac{n-ir}{2n-ir}\right)^{ir} \\
\Uparrow \\
1 \geq \frac{n-i+1}{i+1} \left(\frac{1}{2}\right)^{ir} \\
\Updownarrow \\
2^{ir} \geq \frac{n-i+1}{i+1} \\
\Uparrow \\
r \geq \log n
\end{array}
$$

where the first implication follows from considering the fractions of corresponding factors from the numerator and denominator and choosing the largest.

Since we have now shown that the terms are non-increasing, it follows that $P(r) \geq 1 - \binom{n+1}{1} \frac{[n]_r}{[2n]_r}$, i.e., dropping all but the first two terms. Since, for corresponding factors in $[n]_r$ and $[2n]_r$, we have that $\frac{n-j}{2n-j} \leq \frac{1}{2}$, we can conclude that $P(r) \geq 1 - \frac{n+1}{2^r}$.

We can use this lemma to show that switches between As and Bs occur quite often.

Lemma 2 *Let $I_n = A^n B^n$. For any $\varepsilon > 0$, there exists an n_0 such that for all $n \geq n_0$, the probability when selecting a random permutation of I_n that all maximal runs of As (or Bs) have lengths at most $\lfloor \sqrt{n} \rfloor$ is at least $1 - \varepsilon$.*

Proof By Lemma 1, for any given n, the probability is at least $1 - \frac{n+1}{2^{\lfloor \sqrt{n} \rfloor}}$. Since $n+1 \in o(2^{\sqrt{n}})$, this probability approaches one for increasing values of n.

Now we show that when having so few Cs compared to As and Bs, we can be almost certain to find a large number of As and Bs between two successive occurrences of Cs.

Lemma 3 *For any $\varepsilon > 0$, there exists an n_0 such that for all $n \geq n_0$, the probability when selecting a random permutation of $I_n = A^n B^n C^{\lfloor \log n \rfloor}$ that all maximal runs of As and Bs (looking at As and Bs as the same item) have length at least $(2d+2)\lfloor \sqrt{n} \rfloor$ is at least $1 - \varepsilon$.*

Proof We do not distinguish between As and Bs here, so we just use that there are a total of $2n$ of them, and refer to all of them as Xs.

To compute the probability, we consider the number of ways the Cs can be placed as dividers into a sequence of $2n$ Xs, creating $\lfloor \log n \rfloor + 1$ groups. The standard method is to consider $2n + \lfloor \log n \rfloor$ positions and place the Cs in $\lfloor \log n \rfloor$ of these, which can be done in $\binom{2n+\lfloor \log n \rfloor}{\lfloor \log n \rfloor}$ ways. Similarly, if we want $(2d+2)\lfloor \sqrt{n} \rfloor$ Xs in each group, we may reserve these $(2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1)$ Xs and just consider the division of the remaining Xs. Thus, this can be done in

$$\binom{2n - (2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1) + \lfloor \log n \rfloor}{\lfloor \log n \rfloor}$$

ways.

We now find a lower bound on the probability of there being this many As and Bs between Cs using the above counting argument:

$$\begin{aligned}
&\frac{\binom{2n - (2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1) + \lfloor \log n \rfloor}{\lfloor \log n \rfloor}}{\binom{2n + \lfloor \log n \rfloor}{\lfloor \log n \rfloor}} \\
&= \frac{[(2n - (2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1) + \lfloor \log n \rfloor)]_{\lfloor \log n \rfloor}}{[(2n + \lfloor \log n \rfloor)]_{\lfloor \log n \rfloor}} \\
&\geq \left(\frac{2n - (2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1) + 1}{2n+1} \right)^{\lfloor \log n \rfloor} \\
&= \left(1 - \frac{(2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1)}{2n+1} \right)^{\lfloor \log n \rfloor}
\end{aligned}$$

where the inequality follows from considering corresponding factors in the numerator and denominator, and using the smallest fraction of these.

Using the binomial theorem, this last expression can be written

$$\begin{aligned}
&\sum_{i=0}^{\lfloor \log n \rfloor} \binom{\lfloor \log n \rfloor}{i} \left(\frac{-(2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1)}{2n+1} \right)^i \\
&= 1 - \lfloor \log n \rfloor \left(\frac{(2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1)}{2n+1} \right) + T
\end{aligned}$$

where T contains the additional terms of the binomial expansion.

We now argue that the absolute values of successive terms in T decrease for large enough n:

$$\begin{array}{l}
\binom{\lfloor \log n \rfloor}{i} \left(\frac{(2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1)}{2n+1} \right)^i > \binom{\lfloor \log n \rfloor}{i+1} \left(\frac{(2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1)}{2n+1} \right)^{i+1} \\
\Updownarrow \\
\frac{[\lfloor \log n \rfloor]_i}{i!} > \frac{[\lfloor \log n \rfloor]_{i+1} (2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1)}{(i+1)!(2n+1)} \\
\Updownarrow \\
1 > \frac{(\lfloor \log n \rfloor - i)(2d+2)\lfloor \sqrt{n} \rfloor (\lfloor \log n \rfloor + 1)}{(i+1)(2n+1)}
\end{array}$$

Since $\sqrt{n}\log^2 n \in o(n)$, this holds when n is sufficiently large.

For n large enough, this means that $T \geq 0$ and the probability we are computing will be bounded from below by $1 - \lfloor \log n \rfloor \left(\frac{(2d+2)\lfloor\sqrt{n}\rfloor(\lfloor\log n\rfloor+1)}{2n+1} \right)$.

Again, since $\sqrt{n}\log^2 n \in o(n)$, the probability approaches one as n increases.

With the use of the lemmas above, we can establish the theorem.

Theorem 2 DC *and* LDC *both have the Random Order Ratio two.*

Proof The upper bounds follow directly from the fact that their Competitive Ratios are two. Thus, if that is the factor on worst case sequences, clearly the expected ratio cannot be worse, since the averages for these algorithms and OPT is over the same set of sequences.

For the lower bound, let $I_n = A^n B^n C^{\lfloor \log n \rfloor}$. We show below that for any $\varepsilon > 0$, there exists an n_0 so that for $n \geq n_0$, the probability of DC and LDC incurring a cost of a factor two more than OPT is at least $1 - \varepsilon$. Given this, if we let $\mathbb{A}$ denote either DC or LDC, for any ε, we can choose an n such that $E_\sigma[\mathbb{A}(\sigma(I_n))] \geq (1 - \varepsilon) 2 E_\sigma[\text{OPT}(\sigma(I_n))]$, implying that $\frac{E_\sigma[\mathbb{A}(\sigma(I_n))]}{E_\sigma[\text{OPT}(\sigma(I_n))]} \geq 2(1-\varepsilon)$. Thus, any claim of a ratio smaller than two can be disproven by choosing a small enough ε, and this will give us the result.

By Lemma 3, there exists an n' so that for all $n \geq n'$, the probability that all maximal runs of As and Bs have length at least $(2d+2)\lfloor\sqrt{n}\rfloor$ is at least $1 - \frac{\varepsilon}{2}$.

Considering only the As and Bs, by Lemma 2, there exists an n'' so that for all $n \geq n''$, the probability that all maximal runs of As and Bs, respectively, have lengths at most $\lfloor\sqrt{n}\rfloor$ is at least $1 - \frac{\varepsilon}{2}$.

Thus, for all $n \geq \max\{n', n''\}$, the probability of having both properties is at least $1 - \varepsilon$, and we argue that in this case, the cost of DC and LDC are a factor two larger than the cost of OPT.

Since the number of As and Bs between two Cs is at least $(2d+2)\lfloor\sqrt{n}\rfloor$ and the length of maximal runs of As and Bs, respectively, is at most $\lfloor\sqrt{n}\rfloor$, there must at least $2d+2$ runs in between two successive Cs, and at least $2d+1$ runs if we want to count from the first run of Bs.

For both algorithms, this is sufficient for the algorithm to move the server from C to B. DC will have both servers on B after the dth run of Bs has been processed, whereas for LDC, the right-most server will only virtually be at B at that point, but will be moved there at the $(d+1)st$ run of Bs.

For each C, OPT incurs the cost $2d$ of moving a server from C to B and back again, and it incurs cost d after the last C. The online algorithms have the same cost, plus the additional cost of moving a server back and forth between A and B until the server from C is moved to B. This additional cost consists of $2d$ complete moves from A to B and back.

Asymptotically, the requests after the last C can be ignored, so this gives the ratio $4d/2d = 2$.

We return briefly to the discussion from the end of Section 2.3.3, where we presented the definition of Random Order Ratio that we are using and an alternative

version, and argue that the result above holds for the alternative version as well. The upper bound comes from Competitive Analysis, which is a sequence-based comparison, so clearly $\frac{\mathbb{A}(\sigma(I))}{\text{OPT}(\sigma(I))} \leq 2$ for any sequence and any permutation. For the lower bound, the proof above establishes that with probability approaching one for large enough n, on a permutation σ chosen uniformly at random, the costs $\text{DC}(\sigma(I_n))$ and $\text{LDC}(\sigma(I_n))$ are twice that of $\text{OPT}(\sigma(I_n))$. This means that $E_\sigma\left[\frac{\mathbb{A}(\sigma(I_n))}{\text{OPT}(\sigma(I_n))}\right]$ must approach two for $n \to \infty$. Thus, this result holds for the alternative definition of the Random Order Ratio as well. A similar argument can be made for the next and final theorem of this section.

On another point, the theorem above, saying that LDC and DC are equivalent according the Random Order Ratio, is an example of where a counter-intuitive result is partially due to the intermediate comparison to OPT and partially due to the worst-case element of the measure. This is because on some of the sequences (or rather multisets, since it considers the expected value on all permutations) where LDC and DC do worst compared to OPT, their comparison to OPT gives the same ratio. Since the measure is worst-case over all multisets, the algorithms are deemed equivalent. We illustrate the problem with the intermediate comparison to OPT by showing below how avoiding this comparison could give the result that LDC is better than DC.

If the definition was modified in the most straightforward manner to allow direct comparison of algorithms, one would first note that for any sequence I, by the laziness observation, $E_\sigma[\text{DC}(\sigma(I))] \geq E_\sigma[\text{LDC}(\sigma(I))]$. Then, one would consider some families of sequences with relatively large numbers of Cs and show that LDC's cost is some constant fraction better than DC's on random permutations of that sequence.

For example, let $I = (CABC)^n$. Whenever the subsequence *CABC* occurs in $\sigma(I)$, DC moves a server from C towards B and back again, while moving the other server from A to B. In contrast, LDC lets the server on C stay there, and has cost two less than DC.

One can show that the expected number of occurrences of *CABC* in $\sigma(I)$ is at least $\frac{n}{16}$ (any constant fraction of n would illustrate the point) by considering any of the possible starting locations for this pattern, $1 \leq i \leq 4n-3$, and noting that the probability that the pattern *CABC* begins there is $\frac{1}{2} \cdot \frac{n}{4n-1} \cdot \frac{n}{4n-2} \cdot \frac{2n-1}{4n-3}$. By the linearity of expectations, the expected number of occurrences of *CABC* is $(\frac{1}{2} \cdot \frac{n}{4n-1} \cdot \frac{n}{4n-2} \cdot \frac{2n-1}{4n-3}) \cdot (4n-3) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{n^2}{4n-1} \geq \frac{n}{16}$.

The expected costs of both OPT and LDC on $\sigma(I)$ are also bounded above and below by some constants times n. Thus, LDC's "modified random order ratio" will be less than DC's.

It is easier to compare GREEDY and LDC using the (original) Random Order Ratio, getting a result very similar to that of Competitive Analysis: LDC is strictly better than GREEDY.

Theorem 3 DC *and* LDC *are better than* GREEDY *on the baby server problem with regards to the Random Order Ratio.*

Proof As noted in the proof of Theorem 2, since the Competitive Ratios of both DC and LDC are two, their Random Order Ratios are also at most two.

Consider all permutations of the sequence $I_n = (BA)^{\frac{n}{2}}$. We consider positions from 1 through n in these sequences. We again refer to a maximal consecutive subsequence consisting entirely of either As or Bs as a *maximal run*.

Given a sequence containing h As and t Bs, one can see from well known results that the expected number of maximal runs is $1+\frac{2ht}{h+t}$: In [12, Problem 28, Chapter 9, Page 240], it is stated that the expected number of runs of As is $\frac{h(t+1)}{h+t}$, so the expected number of runs of Bs is $\frac{t(h+1)}{h+t}$. One can see that this holds for As by considering the probability that a run of As starts at some index i in the sequence. The probability that it starts at the beginning of the sequence, at index $i=1$, is the probability that the first element is an A, $\frac{h}{h+t}$. The probability that it starts at some index $i>1$ is the probability that there is a B at index $i-1$ and an A at index i, $\frac{t}{h+t}\cdot\frac{h}{h+t-1}$. By the linearity of expectations, the expected number of runs of As is thus $\frac{h}{h+t}+\sum_{i=2}^{h+t}\frac{th}{(h+t)(h+t-1)} = \frac{h(t+1)}{h+t}$. Adding the expectations for As and Bs gives the result $1+\frac{2ht}{h+t}$. Thus, with $h=t=\frac{n}{2}$, we get that $\frac{n}{2}+1$ is the expected number of runs.

The cost of GREEDY is equal to the number of runs if the first run is a run of Bs. Otherwise, the cost is one smaller. Thus, GREEDY's expected cost on a permutation of I_n is $\frac{n}{2}+\frac{1}{2}$.

The cost of OPT for any permutation of I_n is d, since it simply moves the server from C to B on the first request to B and has no other cost after that.

Thus, the Random Order Ratio is $\frac{\frac{n}{2}+\frac{1}{2}}{d}$, which, as n tends to infinity, is unbounded.

The same argument shows that BAL is better than GREEDY with respect to the Random Order Ratio.

6 Bijective Analysis

Bijective analysis correctly distinguishes between DC and LDC, indicating that the latter is the better algorithm. This follows from the following general theorem about lazy algorithms, and the fact that there are some sequences where one of DC's servers repeatedly moves from C towards B, but moves back to C before ever reaching B, while LDC's server stays on C.

Theorem 4 *The lazy version of any algorithm for the baby server problem is at least as good as the original algorithm according to both Bijective Analysis and Average Analysis.*

Proof By the laziness observation, the identity function, id, is a bijection such that $\mathscr{L}\mathbb{A}(I) \leq \mathbb{A}(id(I))$ for all sequences I. If an algorithm is better than another algorithm with regards to Bijective Analysis, then it is also better with regards to Average Analysis [1].

We first show that GREEDY is at least as good as any other lazy algorithm; including LDC and BAL.

Theorem 5 GREEDY *is at least as good as any other lazy algorithm* LAZY *for the baby server problem according to Bijective Analysis.*

Proof Since GREEDY has cost zero for the sequences consisting of only the point A or only the point C and cost one for the point B, it is easy to define a bijection f for sequences of length one, such that $\text{GREEDY}(I) \leq \text{LAZY}(f(I))$. Suppose that for all sequences of length k we have a bijection, f, from GREEDY's sequences to LAZY's sequences, such that for each sequence I of length k, $\text{GREEDY}(I) \leq \text{LAZY}(f(I))$. To extend this to length $k+1$, consider the three sequences formed from a sequence I of length k by adding one of the three requests A, B, or C to the end of I, and the three sequences formed from $f(I)$ by adding each of these points to the end of $f(I)$. At the end of sequence I, GREEDY has its two servers on different points, so two of these new sequences have the same cost for GREEDY as on I and one has cost exactly 1 more. Similarly, LAZY has its two servers on different points at the end of $f(I)$, so two of these new sequences have the same cost for LAZY as on $f(I)$ and one has cost either 1 or d more. This immediately defines a bijection f' for sequences of length $k+1$ where $\text{GREEDY}(I) \leq \text{LAZY}(f'(I))$ for all I of length $k+1$.

Corollary 1 GREEDY *is the unique optimal algorithm with regards to Bijective and Average Analysis.*

Proof Note that the proof of Theorem 5 shows that GREEDY is strictly better than any lazy algorithm which ever moves the server away from C, so it is better than any other lazy algorithm with regards to Bijective Analysis. By Theorem 4, it is better than any algorithm. Again, since separations with respect to Bijective Analysis also hold for Average Analysis, the result also holds for Average Analysis.

According to Bijective Analysis, there is also a unique worst algorithm among compliant server algorithms for the baby server problem: If p is in between the two servers, the algorithm moves the server that is furthest away to the request point. If p is on the same side of both servers, the nearest server moves to p. Again, due to the problem formulation, ties cannot occur (and the server on A is never moved). The proof that this algorithm is unique worst is similar to the proof of Theorem 5, but now with cost d for every actual move.

Lemma 4 *If $a \leq b$, then there exists a bijection*

$$\sigma_n \colon \{A,B,C\}^n \to \{A,B,C\}^n$$

such that a-$\text{LDC}(I) \leq b$-$\text{LDC}(\sigma_n(I))$ for all sequences $I \in \{A,B,C\}^n$.

Proof We use the bijection from the proof of Theorem 5, showing that GREEDY is the unique best algorithm, but specify the bijection completely, as opposed to allowing some freedom in deciding the mapping in the cases where we are extending by a request where the algorithms already have a server. Suppose that the bijection σ_n is already defined. Consider a sequence I_n of length n and the three possible ways, I_nA, I_nB and I_nC, of extending it to length $n+1$. Suppose that a-LDC has servers on points $X_a, Y_a \in \{A,B,C\}$ after handling the sequence I_n, and b-LDC has servers on points $X_b, Y_b \in \{A,B,C\}$ after handling $\sigma_n(I_n)$. Let Z_a be the point where a-LDC does not have a server and Z_b the point where b-LDC does not. Then $\sigma_{n+1}(I_nZ_a)$ is defined to be $\sigma_n(I_n)Z_b$. In addition, since the algorithms are lazy, both algorithms have their

servers on two different points of the three possible, so there must be at least one point P where both algorithms have a server. Let U_a be the point in $\{X_a, Y_a\} \setminus \{P\}$ and U_b be the point in $\{X_b, Y_b\} \setminus \{P\}$. Then, $\sigma_{n+1}(I_n P)$ is defined to be $\sigma_n(I_n)P$ and $\sigma_{n+1}(I_n U_a)$ to be $\sigma_n(I_n)U_b$.

Consider running a-LDC on a sequence I_n and b-LDC on $\sigma_n(I_n)$ simultaneously. The sequences are clearly constructed so that, at any point during this simultaneous execution, both algorithms have servers moving or neither does.

The result follows if we can show that b-LDC moves away from and back to C at least as often as a-LDC does. By construction, the two sequences, I_n and $\sigma_n(I_n)$, will be identical up to the point where b-LDC (and possibly a-LDC) moves away from C for the first time. In the remaining part of the proof, we argue that if a-LDC moves away from and back to C, then b-LDC will also do so before a-LDC can do it again. Thus, the total cost of b-LDC will be at least that of a-LDC.

Consider a request causing the slower algorithm, a-LDC, to move a server away from C.

If b-LDC also moves a server away from C at this point, both algorithms have their servers on A and B, and the two sequences continue identically until the faster algorithm again moves a server away from C (before or at the same time as the slower algorithm does).

If b-LDC does not move a server away from C at this point, since, by construction, it does make a move, it moves a server from A to B. Thus, the next time both algorithms move a server, a-LDC moves from B to C and b-LDC moves from B to A. Then both algorithms have servers on A and C. Since a-LDC has just moved a server to C, whereas b-LDC must have made at least one move from A to B since it placed a server at C, b-LDC must, as the faster algorithm, make its next move away from C strictly before a-LDC does so. In conclusion, the sequences will be identical until the faster algorithm, b-LDC, moves a server away from C.

Theorem 6 *According to Bijective Analysis and Average Analysis, slower variants of* LDC *are better than faster variants for the baby server problem.*

Proof Follows immediately from Lemma 4 and the definition of the measures.

Thus, the closer a variant of LDC is to GREEDY, the better Bijective and Average Analysis predict that it is.

7 Relative Worst Order Analysis

Similarly to the Random Order Ratio and Bijective Analysis, Relative Worst Order Analysis correctly distinguishes between DC and LDC, indicating that the latter is the better algorithm. This follows from the following general theorem about lazy algorithms, and the fact that there are some sequences where one of DC's servers repeatedly moves from C towards B, but moves back to C before ever reaching B, while LDC's server stays on C.

Let $I_{\mathbb{A}}$ denote a worst ordering of the sequence I for the algorithm $\mathbb{A}$.

Theorem 7 *The lazy version of any algorithm for the baby server problem is at least as good as the original algorithm according to Relative Worst Order Analysis.*

Proof This follows from the laziness observation, since for any request sequence I, we have that $\mathscr{L}\mathbb{A}(I_{\mathscr{L}\mathbb{A}}) \leq \mathbb{A}(I_{\mathscr{L}\mathbb{A}}) \leq \mathbb{A}(I_{\mathbb{A}})$.

Theorem 8 DC *(*LDC*) and* GREEDY *are* $(2,\infty)$*-related and are thus weakly comparable in* DC*'s (*LDC*'s) favor for the baby server problem according to Relative Worst Order Analysis.*

Proof We write this proof for DC, but exactly the same holds for LDC. First we show that $c_{\mathrm{u}}(\textsc{Greedy}, \textsc{Dc})$ is unbounded. Consider the sequence $(BA)^{\frac{n}{2}}$. As n tends to infinity, GREEDY's cost is unbounded, whereas DC's cost is at most $3d$ for any permutation.

Next we turn to $c_{\mathrm{u}}(\textsc{Dc}, \textsc{Greedy})$. Since the Competitive Ratio of DC is 2, for any sequence I and some constant b, $\textsc{Dc}(I_{\textsc{Dc}}) \leq 2\textsc{Opt}(I_{\textsc{Dc}}) + b \leq 2\textsc{Greedy}(I_{\textsc{Dc}}) + b \leq 2\textsc{Greedy}(I_{\textsc{Greedy}}) + b$. Thus, $c_{\mathrm{u}}(\textsc{Dc}, \textsc{Greedy}) \leq 2$.

For the lower bound, consider a family of sequences

$$I_p = ((BA)^d BC)^p.$$

$\textsc{Dc}(I_p) = p(4d)$.

A worst ordering for GREEDY alternates As and Bs. Since there is no cost for the Cs and the A/B sequences start and end with Bs, $\textsc{Greedy}(\sigma(I_p)) \leq p(2d) + 1$ for any permutation σ.

Then, $c_{\mathrm{u}}(\textsc{Dc}, \textsc{Greedy}) \geq \frac{p(4d)}{p(2d)+1}$. As p goes to infinity, this approaches 2.

Thus, DC and GREEDY are weakly comparable in DC's favor.

For clarity in the exposition, we assume that $\frac{d}{a}$ is integral. By the definition of a-LDC, a request for B is served by the right-most server if it is within a virtual distance of no more than a from B and the other server is at A. Thus, when the left-most server moves and its virtual move is over a distance of l, then the right-most server virtually moves a distance al. When the right-most server moves and its virtual move is over a distance of al, then the left-most server virtually moves a distance of l.

In the results that follow, we frequently look at the worst ordering of an arbitrary sequence.

Definition 4 The *canonical worst ordering* of a sequence, I, for an algorithm $\mathbb{A}$ is the sequence produced by allowing the cruel adversary (the one which always lets the next request be the unique point where $\mathbb{A}$ does not currently have a server) to choose requests from the multiset defined from I. This process continues until there are no requests remaining in the multiset for the point where $\mathbb{A}$ does not have a server. The remaining points from the multiset are concatenated to the end of this new request sequence in any order.

The canonical worst ordering of a sequence for a-LDC is as follows.

Proposition 2 *Consider an arbitrary sequence I containing n_A As, n_B Bs, and n_C Cs. A canonical worst ordering of I for a-*LDC *is*

$$I_a = ((BA)^{\frac{d}{a}}BC)^{p_a}X,$$

where we assume that $\frac{d}{a}$ is integral. Here, X is a possibly empty sequence. The first part of X is an alternating sequence of As and Bs, starting with a B, until there are not both As and Bs left. Then we continue with all remaining As or Bs, followed by all remaining Cs. Finally,

$$p_a = \min\left\{\left\lfloor\frac{n_A}{\frac{d}{a}}\right\rfloor, \left\lfloor\frac{n_B}{\frac{d}{a}+1}\right\rfloor, n_C\right\}.$$

Lemma 5 *Assume that $\frac{d}{a}$ is integral, and let I_a be the canonical worst ordering of I for a-*LDC*. I_a is a worst permutation of I for a-*LDC*, and the cost for a-*LDC* on I_a is c_a, where $p_a(2\frac{d}{a}+2d) \leq c_a \leq p_a(2\frac{d}{a}+2d)+2\frac{d}{a}+d$.*

Proof Consider a request sequence, I. Between any two moves from B to C, there must have been a move from C to B. Consider one such move. Between the last request to C and this move, the other server must move from A to B exactly $\frac{d}{a}$ times, which requires some first request to B in this subsequence, followed by at least $\frac{d}{a}$ occurrences of requests to A, each followed by a request to B, the last one causing the move from C to B. (Clearly, extra requests to A or B could also occur, either causing moves or not.) Thus, for every move from B to C, there must be at least $\frac{d}{a}+1$ Bs, $\frac{d}{a}$ As and one C. Thus, the number of moves from B to C is bounded from above by p_a. There can be at most one more move from C to B than from B to C. If such a move occurs, there are no more Cs after that in the sequence. Therefore, the sequences defined above give the maximal number of moves of distance d possible. More As or Bs in any alternating A/B-sequence would not cause additional moves (of either distance one or d), since each extra point requested would already have a server. Fewer As or Bs between two Cs would eliminate the move away from C before it was requested again. Thus, the canonical worst ordering is a worst ordering of I.

Within each of the p_a repetitions of $(BABA...BC)$, each of the requests for A and all but the last request for B cause a move of distance one, and the last two requests each cause a move of distance d, giving the lower bound on c_a. Within X, each of the first $2\frac{d}{a}$ requests could possibly cause a move of distance one, and this could be followed by a move of distance d. After that, no more moves occur. Thus, adding costs to the lower bound gives the upper bound on c_a.

Theorem 9 *If $a \leq b$, and $\frac{d}{a}$ and $\frac{d}{b}$ are integral, then a-*LDC *and b-*LDC *are*

$$\left(\frac{1+\frac{1}{a}}{1+\frac{1}{b}}, \frac{b+1}{a+1}\right)\text{-related}$$

for the baby server problem according to Relative Worst Order Analysis.

Proof By Lemma 5, in considering a-LDC's performance in comparison with the performance of b-LDC, the asymptotic ratio depends only on the values p_a and p_b defined for the canonical worst orderings I_a and I_b for a-LDC and b-LDC, respectively. Since $a \leq b$, the largest value of $\frac{p_a}{p_b}$ occurs when $p_a = n_C$, since more Cs would allow more moves of distance d by b-LDC. Since the contribution of X to a-LDC's cost can be considered to be a constant, we may assume that $n_A = n_C \frac{d}{a}$ and $n_B = n_C \left(\frac{d}{a} + 1\right)$.

When considering b-LDC's canonical worst ordering of this sequence, all the Cs will be used in the initial part. By Lemma 5, we obtain the following ratio, for some constant c:

$$\frac{(2\frac{d}{a} + 2d)n_C}{(2\frac{d}{b} + 2d)n_C + c} = \frac{(\frac{1}{a} + 1)n_C}{(\frac{1}{b} + 1)n_C + \frac{c}{2d}}$$

Similarly, a sequence giving the largest value of $\frac{p_b}{p_a}$ will have $p_b = \left\lfloor \frac{n_A}{\frac{d}{b}} \right\rfloor$, since more As would allow a-LDC to have a larger p_a. Since the contribution of X to b-LDC can be considered to be a constant, we may assume that $n_A = n_C \frac{d}{b}$, $n_B = n_C \left(\frac{d}{b} + 1\right)$, and $p_b = n_C$.

Now, when considering a-LDC's worst permutation of this sequence, the number of periods, p_a, is restricted by the number of As. Since each period has $\frac{d}{a}$ As, $p_a = \left\lfloor \frac{n_A}{\frac{d}{a}} \right\rfloor = \left\lfloor \frac{n_C \frac{d}{b}}{\frac{d}{a}} \right\rfloor$. After this, there are a constant number of As remaining, giving rise to a constant additional cost c'.

Thus, the ratio is the following:

$$\frac{(2\frac{d}{b} + 2d)n_C}{(2\frac{d}{a} + 2d)\left\lfloor n_C \frac{a}{b} \right\rfloor + c'} = \frac{(\frac{1}{b} + 1)n_C}{(\frac{1}{a} + 1)\left\lfloor n_C \frac{a}{b} \right\rfloor + \frac{c'}{2d}} = \frac{(1+b)n_C}{(1+a)n_C + c''},$$

for some constant c''. Considering the two ratios relating b-LDC's and a-LDC's worst permutations asymptotically as n_C goes to infinity, we obtain that a-LDC and b-LDC are $\left(\frac{1+\frac{1}{a}}{1+\frac{1}{b}}, \frac{b+1}{a+1}\right)$-related.

Although with the original definition of relatedness in Relative Worst Order Analysis, the values are not interpreted further, one could use the concept of *better performance* (see [11]) from Relative Interval Analysis to compare two algorithms using Relative Worst Order Analysis. Using the previous result, we show that LDC has better performance than b-LDC for $b \neq 1$. Again, for clarity, we consider integral cases in the following result.

Theorem 10 *The following holds for the baby server problem evaluated according to Relative Worst Order Analysis:*

For $b > 1$ such that $\frac{d}{b}$ is integral, LDC *and b-*LDC *are (r, r_b)-related for some r and r_b where $1 < r < r_b$.*

*For $a < 1$ such that $\frac{d}{a}$ is integral, a-*LDC *and* LDC *are (r_a, r)-related for some r_a and r where $1 < r < r_a$.*

Proof By Theorem 9, a-LDC and b-LDC are $\left(\frac{1+\frac{1}{a}}{1+\frac{1}{b}}, \frac{b+1}{a+1}\right)$-related.

To see that $\frac{1+\frac{1}{a}}{1+\frac{1}{b}} < \frac{b+1}{a+1}$ when $1 = a < b$, note that this holds if and only if $(1+\frac{1}{a})(a+1) = 4 < (1+\frac{1}{b})(b+1)$, which clearly holds for $b > 1$. Hence, if LDC and b-LDC are (c_1, c_2)-related, then $c_1 < c_2$.

To see that $\frac{1+\frac{1}{a}}{1+\frac{1}{b}} > \frac{b+1}{a+1}$ when $a < b = 1$, note that this holds if and only if $(1+\frac{1}{a})(a+1) > 4 = (1+\frac{1}{b})(b+1)$. This clearly holds for $a < 1$. Thus, a-LDC and LDC are (c_1, c_2)-related, where $c_1 > c_2$.

The algorithms a-LDC and $\frac{1}{a}$-LDC are in some sense of equal quality:

Corollary 2 *If $\frac{d}{a}$ and $\frac{d}{b}$ are integral and $b = \frac{1}{a}$, then a-LDC and b-LDC are (b, b)-related*

Theorem 10 shows that LDC is in some sense optimal among the a-LDC algorithms. We now set out to prove that LDC is an optimal algorithm in the following sense: there is no other algorithm $\mathbb{A}$ such that LDC and $\mathbb{A}$ are comparable and $\mathbb{A}$ is strictly better or such that LDC and $\mathbb{A}$ are weakly comparable in $\mathbb{A}$'s favor.

We emphasize that comparisons using Relative Worst Order Analysis does not give rise to a total ordering so there could be more than one optimal algorithm, and two different optimal algorithms could be incomparable. If another algorithm should be strictly better than LDC, then it must be strictly better on some infinite family of sequences and at least as good (up to an additive constant) on all other sequences. The proof below is based on showing that no algorithm can fulfill both of these two conditions.

Theorem 11 LDC *is optimal for the baby server problem according to Relative Worst Order Analysis.*

Proof In order for LDC and $\mathbb{A}$ to be comparable in $\mathbb{A}$'s favor, $\mathbb{A}$ has to be comparable to LDC and perform more than an additive constant better on some infinite family of sequences.

Assume that there exists a family of sequences $S_1, S_2, \ldots$ such that for any positive c there exists an i such that $\text{LDC}_W(S_i) \geq \mathbb{A}_W(S_i) + c$. Then we prove that there exists another family of sequences $S'_1, S'_2, \ldots$ such that for any positive c' there exists an i such that $\mathbb{A}_W(S'_i) \geq \text{LDC}_W(S'_i) + c'$.

This establishes that if $\mathbb{A}$ performs more than a constant better on its worst permutations of some family of sequences than LDC does on its worst permutations, then there exists a family where LDC has a similar advantage over $\mathbb{A}$, which implies that the algorithms are not comparable.

Now assume that we are given a constant c. Since we must find a value greater than any constant to establish the result, we may assume without loss of generality that c is large enough that $3dc \geq \frac{3d+1}{d-1}(3d) + 3d$.

Consider a sequence S from the family $S_1, S_2, \ldots$ such that $\text{LDC}_W(S) \geq \mathbb{A}_W(S) + 3dc$. From S we create a member S' of the family $S'_1, S'_2, \ldots$ such that $\mathbb{A}_W(S') \geq \text{LDC}_W(S') + c$.

The idea behind the construction is to have the cruel adversary against $\mathbb{A}$ choose requests from the multiset defined from S as in the definition of canonical worst orderings. This process continues until the cruel adversary has used all of either the As, Bs, or Cs in the multiset, resulting in a sequence S'. If the remaining requests from the multiset are concatenated to S' in any order, this creates a permutation of S. The performance of $\mathbb{A}$ on this permutation must be at least as good as its performance on its worst ordering of S.

We now consider the performance of LDC and $\mathbb{A}$ on S' and show that LDC is strictly better.

Let n'_A, n'_B, and n'_C denote the number of As, Bs, and Cs in S', respectively.

Let $p = \min\left\{\left\lfloor \frac{n'_A}{d} \right\rfloor, \left\lfloor \frac{n'_B}{d+1} \right\rfloor, n'_C\right\}$.

By Lemma 5, the cost of LDC on its canonical worst ordering of S' is at most $p(4d)+3d$.

The cost of $\mathbb{A}$ is $2dn'_C + n'_A + n'_B - n'_C$, since every time there is a request for C, this is because a server in the step before moved away from C. These two moves combined have a cost of $2d$. Every request to an A or a B has cost one, except for the request to B immediately followed by a request to C, which has already been counted in the $2dn'_C$ term. A similar argument shows that LDC's cost is bounded from above by the same term.

Assume first that $\frac{n'_A}{d} = \frac{n'_B}{d+1} = n'_C$. Then S' can be permuted so that it is a prefix of LDC's canonical worst ordering on S (see Lemma 5 with $a = 1$). Since, by construction, we have run out of either As, Bs, or Cs (that is, one type is missing from S minus S' as multisets), LDC's cost on its worst ordering of S is at most its cost on its worst ordering on S' plus $3d$. Thus, $\mathrm{LDC}_W(S) \geq \mathbb{A}_W(S) + c$ does not hold in this case, so we may assume that these values are not all equal.

We compare LDC's canonical worst orderings of S and S'. For both sequences, the form is as in Lemma 5, with $a = 1$. Thus, for S' the form is $((BA)^d BC)^p X$, and for S, it is $((BA)^d BC)^{p+l} Y$ for some nonnegative integer l. The sequence X must contain all of the As, all of the Bs or all of the Cs contained in $((BA)^d BC)^l$, since after this the cruel adversary has run out of something. Thus, it must contain at least ld As, $l(d+1)$ Bs or l Cs. The extra cost that LDC has over $\mathbb{A}$ on S is at most its cost on $((BA)^d BC)^l Y$ minus cost ld for the As, Bs or Cs contained in X, so at most $l(2d+2d)+3d-ld = 3dl+3d$. Thus, $\mathrm{LDC}_W(S) - \mathbb{A}_W(S) \leq 3dl+3d$.

Since we could assume that not all of $\frac{n'_A}{d}$, $\frac{n'_B}{d+1}$, and n'_C were equal, we have the following cases:

Case $n'_A > dp$: LDC's cost on S' is at most the cost of $\mathbb{A}$ minus $(n'_A - dp)$ plus $3d$.

Case $n'_B > (d+1)p$: LDC's cost on S' is at most the cost of $\mathbb{A}$ minus $(n'_B - (d+1)p)$ plus $3d$.

Case $n'_C > p$: LDC's cost on S' is at most the cost of $\mathbb{A}$ minus $(2d-1)(n'_C - p)$ plus 1.

Thus, $\mathbb{A}_W(S') - \mathrm{LDC}_W(S') \geq dl - 3d$.

From the choice of c, the definition of the S_i family, and the bound on the difference between the two algorithms on S, we find that

$$\frac{3d+1}{d-1}(3d) + 3d \leq 3dc \leq \mathrm{LDC}_W(S) - \mathbb{A}_W(S) \leq 3dl + 3d$$

Thus, $l \geq \frac{3d+1}{d-1}$, which implies the following:

$$l \geq \frac{3d+1}{d-1} \Leftrightarrow ld - 3d \geq l+1 \Leftrightarrow ld - 3d \geq \frac{3dl+3d}{3d}$$

Now,

$$\mathbb{A}_W(S') - \text{LDC}_W(S') \geq ld - 3d \geq \frac{3dl+3d}{3d} \geq \frac{3dc}{3d} = c.$$

Finally, to show that LDC and $\mathbb{A}$ are not weakly comparable in $\mathbb{A}$'s favor, we show that $c_u(\text{LDC}, \mathbb{A})$ is bounded. Since the Competitive Ratio of LDC is 2, for any algorithm $\mathbb{A}$ and any sequence I, there is a constant b such that $\text{LDC}(I_{\text{LDC}}) \leq 2\text{OPT}(I_{\text{LDC}}) + b \leq 2\mathbb{A}(I_{\text{LDC}}) + b \leq 2\mathbb{A}(I_{\mathbb{A}}) + b$. Thus, $c_u(\text{LDC}, \mathbb{A}) \leq 2$.

Considering the request sequence as constructed by the cruel adversary against some algorithm $\mathbb{A}$, it consists of a first part, where the cruel adversary keeps requesting unoccupied points, and a second part which are all remaining requests. The proof of optimality depends on LDC performing as well as any algorithm on the first part, and having constant cost on the second part. Since the first part consists of subsequences where $\mathbb{A}$ at some point has a server pulled away from C and then right back again, it is easy to see that if the distribution of As, Bs, and Cs in those subsequences is different from the distribution in a canonical worst ordering for LDC, LDC will simply do better. On the second part, if there are only requests to two points, LDC will have its two servers on those two points permanently after a cost of at most $3d$. Thus, similar proofs will show that a-LDC and BAL are also optimal algorithms, whereas GREEDY is not.

In the definitions of LDC and BAL given in Section 2, different decisions are made as to which server to use in cases of ties. In LDC the server which is really closer is moved in the case of a tie (with regard to virtual distances from the point requested). The rationale behind this is that the server which would have the least cost is moved. In BAL the server which is further away is moved to the point. The rationale behind this is that, since $d > 1$, when there is a tie, the total cost for the closer server is already significantly higher than the total cost for the other, so moving the server which is further away evens out how much total cost they have. With these tie-breaking decisions, the two algorithms behave very similarly.

Theorem 12 LDC *and* BAL *are equivalent for the baby server problem according to Relative Worst Order Analysis.*

Proof Consider any request sequence I. LDC's canonical worst ordering has a prefix of the form $((BA)^d BC)^k$, while BAL's canonical worst ordering has a prefix of the form

$$(BA)^{\lfloor \frac{d}{2} \rfloor} BC((BA)^d BC)^{k'},$$

such that the remaining parts have constant costs. These prefixes of LDC's and BAL's canonical worst orderings of I are identical, except for the constant cost sequence that BAL starts with. This also leads to a small constant cost difference at the end. Thus, their performance on their respective worst orderings will be identical up to an additive constant.

8 Concluding Remarks

The purpose of quality measures is to give information for use in practice, to choose the best algorithm for a particular application. What properties should such quality measures have?

First, it may be desirable that if one algorithm does at least as well as another on every sequence, then the measure decides in favor of the better algorithm. This is especially desirable if the better algorithm does significantly better on important sequences. Bijective Analysis and Relative Worst Order Analysis have this property, but Competitive Analysis, the Max/Max Ratio, and the Random Order Ratio do not. This was seen here in the lazy vs. non-lazy version of Double Coverage for the baby server problem (and the more general metric k-server problem). Similar results have been presented previously for the paging problem—LRU vs. FWF and look-ahead vs. no look-ahead. See [7] for these results under Relative Worst Order Analysis and [1] for Bijective Analysis. It appears that analysis techniques that avoid a comparison to OPT have an advantage in this respect.

Secondly, it may be desirable that, if one algorithm does unboundedly worse than another on some important families of sequences, the quality measure reflects this. For the baby server problem, GREEDY is unboundedly worse than LDC on all families of sequences which consist mainly of alternating requests to the closest two points. This is reflected in Competitive Analysis, the Random Order Ratio, and Relative Worst Order Analysis, but not by the Max/Max Ratio or Bijective Analysis. Similarly, according to Bijective Analysis, LIFO and LRU are equivalent for paging, but LRU is often significantly better than LIFO, which keeps the first $k-1$ pages it sees in cache forever. In both of these cases, Relative Worst Order Analysis says that the algorithms are weakly comparable in favor of the "better" algorithm.

Another desirable property would be ease of computation for many different problems, as with Competitive Analysis and Relative Worst Order Analysis. It is not clear that the Random Order Ratio or Bijective Analysis have this property.

In this paper, we have initiated a systematic comparison of quality measures for online algorithms. We hope this will inspire researchers to similarly investigate a range of online problems to enable the community to draw stronger conclusions on the relative strengths of the different measures.

Acknowledgements The first and third author were supported in part by the Danish Council for Independent Research, Natural Sciences. Part of this work was carried out while these authors were visiting the University of California, Irvine, and the University of Waterloo, Canada. The second author was supported in part by NSF Grants CCR-0514082 and CCF-0916181.

The authors would like to thank Christian Kudahl for calling their attention to two oversights in a previous version of this paper, one in the definition of the lazy version of an algorithm, and another in the modified definition of the Random Order Ratio.

A preliminary version of this paper appeared in *11th International Algorithms and Data Structures Symposium (WADS 2009)*, volume 5664 of *Lecture Notes in Computer Science*, pages 119–130, Springer, 2009.

References

1. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: 18th ACM-SIAM Symposium on Discrete Algorithms, pp. 229–237 (2007)
2. Balakrishnan, N., Koutras, M.V.: Runs and Scans with Applications. John Wiley & Sons, Inc. (2002)
3. Bein, W.W., Iwama, K., Kawahara, J.: Randomized competitive analysis for two-server problems. In: 16th Annual European Symposium on Algorithms, *Lecture Notes in Computer Science*, vol. 5193, pp. 161–172. Springer (2008)
4. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica **11**(1), 73–91 (1994)
5. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
6. Boyar, J., Favrholdt, L.M.: The relative worst order ratio for on-line algorithms. ACM Transactions on Algorithms **3**(2) (2007). Article No. 22
7. Boyar, J., Favrholdt, L.M., Larsen, K.S.: The relative worst order ratio applied to paging. Journal of Computer and System Sciences **73**(5), 818–843 (2007)
8. Burr, E.J., Cane, G.: Longest run of consecutive observatons having a specified attribute. Biometrika **48**(3/4), 461–465 (1961)
9. Chrobak, M., Karloff, H.J., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM Journal on Discrete Mathematics **4**(2), 172–181 (1991)
10. Dorrigiv, R., López-Ortiz, A.: A survey of performance measures for on-line algorithms. SIGACT News **36**(3), 67–81 (2005)
11. Dorrigiv, R., López-Ortiz, A., Munro, J.I.: On the relative dominance of paging algorithms. Theoretical Computer Science **410**(38–40), 3694–3701 (2009)
12. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1, 3rd edn. John Wiley & Sons, Inc., New York (1968)
13. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell Systems Technical Journal **45**, 1563–1581 (1966)
14. Jr., E.G.C., Csirik, J., Rónyai, L., Zsbán, A.: Random-order bin packing. Discrete Applied Mathematics **156**(14), 2810–2816 (2008)
15. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica **3**, 79–119 (1988)
16. Kenyon, C.: Best-fit bin-packing with random order. In: 7th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 359–364 (1996)
17. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for on-line problems. In: 20th Annual ACM Symposium on the Theory of Computing, pp. 322–333 (1988)
18. Manasse, M.S., McGeoch, L.A., Sleator, D.D.: Competitive algorithms for server problems. Journal of Algorithms **11**(2), 208–230 (1990)
19. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM **28**(2), 202–208 (1985)