A Comparison of Performance Measures for Online Algorithms*

Joan Boyar[1], Sandy Irani[2], and Kim S. Larsen[1]

[1] Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark, {joan,kslarsen}@imada.sdu.dk
[2] Department of Computer Science, University of California, Irvine, CA 92697, USA, irani@ics.uci.edu

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 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 better algorithm. Under the other measures, Lazy Double Coverage is better, though Relative Worst Order Analysis indicates that Greedy is sometimes better. Our results also provide the first proof of optimality of an algorithm under Relative Worst Order Analysis.

1 Introduction

Since its introduction by Sleator and Tarjan in 1985 [16], 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, we look at 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, we factor out the inherent difficulty of a particular input (for which the offline algorithm is penalized along

* The work of Boyar and Larsen was supported in part by the Danish Natural Science Research Council. Part of this work was carried out while these authors were visiting the University of California, Irvine. The work of Irani was supported in part by NSF Grant CCR-0514082.

with the online algorithm) and measure what is lost in making decisions with partial information.

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 [9] for a 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, these papers 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-servers problems to shed light on new ideas has also been done in [2], for instance.

We concentrate on two algorithms (Greedy and Lazy Double Coverage (Ldc) [8]) and four different analysis techniques (measures): Bijective Analysis, the Max/Max Ratio, 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 Ldc, whereas for others, Ldc is better than Greedy.

The ones that conclude that Ldc is best are focused on a worst-case sequence for the ratio of an algorithm's cost compared to Opt. In the case of Greedy 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 Ldc's cost is always at most a factor 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 a factor at most two worse than Greedy, while Greedy can be unboundedly worse than Ldc.

All omitted proofs may be found in the full version of the paper [7].

2 Preliminaries

2.1 The Server Problem

Server problems [4] 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 distance $d > 1$ 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.*

All results in the paper pertain to this problem. Even though the problem is simple, it contains a core k-server problem of balancing greediness and adaptability, and this simple set-up is sufficient to show the non-competitiveness of Greedy with respect to Competitive Analysis [4].

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 *[15] 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* $\mathbb{A}$, $\mathcal{L}$A, as follows: $\mathcal{L}$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. Whenever a virtual server reaches a request point, the corresponding real server is also moved to that point (unless both virtual servers reach the point simultaneously, in which case only the physically closest is moved there). Otherwise the real servers do not move.

In [8], it was observed that for any 2-server algorithm, there exists a non-crossing algorithm with the same cost on all sequences. In [15], it was observed that for an algorithm $\mathbb{A}$ and its lazy version $\mathcal{L}$A, for any sequence I of requests, $\mathbb{A}(I) \geq \mathcal{L}\mathrm{A}(I)$ (we refer to this as the *laziness obervation*). Note that the laziness observation applies to the general k-server problem in metric spaces, so the results which depend on it can also be generalized beyond the baby server problem.

We define a number of algorithms by defining their behavior on the next request point, p. For all algorithms, 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 [15], 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.

If p is in between the two servers, DUMMY 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).

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 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 OPT(I) denotes the unique minimal cost of processing I.

All of the measures described below can lead to a conclusion as to which algorithm of two is better. In contrast to the others, Bijective Analysis does not indicate how much better the one algorithm might be; it does not produce a ratio, as the others do.

Competitive Analysis: In Competitive Analysis [11, 16, 12], we define an algorithm $\mathbb{A}$ to be c-competitive if there exists a constant α such that for all input sequences I, $\mathbb{A}(I) \leq c\,\textsc{Opt}(I) + \alpha$.

The Max/Max Ratio: The Max/Max Ratio [3] 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 $M(\mathbb{A})/M(\textsc{Opt})$, where

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

The Random Order Ratio: Kenyon [13] defines the Random Order Ratio to be the worst ratio obtained over all sequences, comparing the expected value of an algorithm, $\mathbb{A}$, with respect to a uniform distribution of all permutations of a given sequence, to the value of Opt of the given sequence:

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

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. We choose to generalize the Random Order Ratio as shown to the left, but for the results presented here, the definition to the right would give the same:

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

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 $f : I_n \to I_n$ satisfying $\mathbb{A}(I) \leq \mathbb{B}(f(I))$ for each $I \in I_n$.

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 $n \geq n_0$, $\Sigma_{I\in I_n}\mathbb{A}(I) \leq \Sigma_{I\in I_n}\mathbb{B}(I)$.

Relative Worst Order Analysis: Relative Worst Order Analysis was introduced in [5] and extended in [6]. 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. *Let I be any input sequence, and let n be the length of I. If σ is a permutation on n elements, then $\sigma(I)$ denotes I permuted by σ. Let $\mathbb{A}$ be any algorithm. Then, $\mathbb{A}(I)$ is the cost of running $\mathbb{A}$ on I, and*

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

Definition 4. *For any pair of algorithms $\mathbb{A}$ and $\mathbb{B}$, we define*

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

If $c_l(\mathbb{A},\mathbb{B}) \geq 1$ or $c_u(\mathbb{A},\mathbb{B}) \leq 1$, the algorithms are said to be comparable *and the* Relative Worst-Order Ratio *$WR_{\mathbb{A},\mathbb{B}}$ of algorithm $\mathbb{A}$ to algorithm $\mathbb{B}$ is defined. Otherwise, $WR_{\mathbb{A},\mathbb{B}}$ is undefined.*

$$\begin{aligned} &\textit{If } c_l(\mathbb{A},\mathbb{B}) \geq 1, \ \textit{then } WR_{\mathbb{A},\mathbb{B}} = c_u(\mathbb{A},\mathbb{B}), \ \textit{and} \\ &\textit{if } c_u(\mathbb{A},\mathbb{B}) \leq 1, \ \textit{then } WR_{\mathbb{A},\mathbb{B}} = c_l(\mathbb{A},\mathbb{B})\,. \end{aligned}$$

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

Definition 5. *Let c_u be defined as in Definition 4. If at least one of the ratios $c_u(\mathbb{A},\mathbb{B})$ and $c_u(\mathbb{B},\mathbb{A})$ is finite, the algorithms $\mathbb{A}$ and $\mathbb{B}$ are $(c_{\mathrm{u}}(\mathbb{A},\mathbb{B}), c_{\mathrm{u}}(\mathbb{B},\mathbb{A}))$-*related.

Definition 6. *Let $c_u(\mathbb{A},\mathbb{B})$ be defined as in Definition 4. 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 $c_u(\mathbb{A},\mathbb{B})$ is finite and $c_u(\mathbb{B},\mathbb{A})$ is infinite, or 3) if $c_u(\mathbb{A},\mathbb{B}) \in o(c_u(\mathbb{B},\mathbb{A}))$.*

3 Competitive Analysis

The k-server problem has been studied using Competitive Analysis starting in [14]. In [8], it is shown that the competitive ratios of Dc and Ldc are k, which is optimal, and that Greedy is not competitive.

4 The Max/Max Ratio

In [3], 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* LDC *on the baby server problem with respect to the Max/Max Ratio.*

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 correctly distinguishes between DC and LDC, indicating that the latter is the better algorithm.

Theorem 2. LDC *is better than* DC *according to the Random Order Ratio.*

Proof. For any sequence I, $E_\sigma[\text{DC}(\sigma(I))] \geq E_\sigma[\text{LDC}(\sigma(I))]$, by the laziness observation. Let $I = (ABC)^n$. 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 2 less than DC. The expected number of occurrences of $CABC$ in $\sigma(I)$ is cn for some constant c. The expected costs of both OPT and LDC on $\sigma(I)$ are bounded above and below by some other constants times n. Thus, LDC's random order ratio will be less than DC's.

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

Proof. 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.

Since the competitive ratio of LDC is two, on any given sequence, LDC's cost is bounded by two times the cost of OPT on that sequence, plus an additive constant. Thus, the Random Order Ratio is also at most two.

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

Given a sequence containing h As and t Bs, the expected number of runs is $1 + \frac{2ht}{h+t}$. (A problem in [10] gives that the expected number of runs of As is $\frac{h(t+1)}{h+t}$, so the expected number of runs of Bs is $\frac{t(h+1)}{h+t}$. Adding these gives the result.) Thus, with $h = t = \frac{n}{2}$, we get $\frac{n}{2} + 1$ 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 s is $\frac{n}{2} + \frac{1}{2}$.

The cost of OPT for any permutation of s 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 $\frac{n+1}{2d}$, which, as n tends to infinity, is unbounded.

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 Bijective Analysis.*

Theorem 5. GREEDY *is at least as good as any other lazy algorithm* LAZY *(including* LDC*) 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 that 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 $k+1$.

If an algorithm is better than another algorithm with regards to Bijective Analysis, then it is also better with regards to Average Analysis [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. By the observation above, it also holds for Average Analysis.

Theorem 6. DUMMY *is the unique worst algorithm among compliant server algorithms for the baby server problem according to Bijective Analysis.*

Lemma 1. *If $a \leq b$, then there exists a bijection $\sigma_n : \{A, B, C\}^n \rightarrow \{A, B, C\}^n$ such that a-*LDC*$(I) \leq b$-*LDC*$(\sigma_n(I))$ for all sequences $I \in \{A, B, C\}^n$.*

Theorem 7. *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 1 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. If d is just marginally larger than some integer, even on LDC's worst ordering of this sequence, it does better than DC.

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

Theorem 8. *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.*

Theorem 9. GREEDY *and* LDC *are $(\infty, 2)$-related and are thus weakly comparable in* LDC*'s favor for the baby server problem according to Relative Worst Order Analysis.*

Proof. First we show that $c_{\mathrm{u}}(\text{GREEDY}, \text{LDC})$ is unbounded. Consider the sequence $(BA)^{\frac{n}{2}}$. As n tends to infinity, GREEDY's cost is unbounded, whereas LDC's cost is at most $3d$ for any permutation.

Next we turn to $c_{\mathrm{u}}(\text{LDC}, \text{GREEDY})$. Since the competitive ratio of LDC is 2, for any sequence I and some constant b, $\text{LDC}(I_{\text{LDC}}) \leq 2\text{GREEDY}(I_{\text{LDC}}) + b \leq 2\text{GREEDY}(I_{\text{GREEDY}}) + b$. Thus, $c_{\mathrm{u}}(\text{LDC}, \text{GREEDY}) \leq 2$.

For the lower bound of 2, consider a family of sequences $I_p = (BABA...BC)^p$, where the length of the alternating A/B-sequence before the C is $2\lfloor d \rfloor + 1$.

$\text{LDC}(I_p) = p(2\lfloor d \rfloor + 2d)$.

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, $\text{GREEDY}(\sigma(I_p)) \leq p(2\lfloor d \rfloor) + 1$ for any permutation σ.

Then, $c_{\mathrm{u}}(\text{LDC}, \text{GREEDY}) \geq \frac{p(2\lfloor d \rfloor + 2d)}{p(2\lfloor d \rfloor)+1} \geq \frac{p(4d)}{p(2d)+1}$. As p goes to infinity, this approaches 2.

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

Recalling 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. 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 7. *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 1. *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 $I_a = (BABA...BC)^{p_a}X$, where the length of the alternating A/B-sequence before the C is $2\left\lfloor \frac{d}{a} \right\rfloor + 1$. 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,*

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

Theorem 10. *If $a \leq b$, then a-*LDC *and b-*LDC *are $\left(\frac{\left\lfloor \frac{d}{a} \right\rfloor + d}{\left\lfloor \frac{d}{b} \right\rfloor + d}, \frac{(\left\lfloor \frac{d}{b} \right\rfloor + d)\left\lfloor \frac{d}{a} \right\rfloor}{(\left\lfloor \frac{d}{a} \right\rfloor + d)\left\lfloor \frac{d}{b} \right\rfloor}\right)$-related for the baby server problem according to Relative Worst Order Analysis.*

We provide strong indication that LDC is better than b-LDC for $b \neq 1$. If $b > 1$, this is always the case, whereas if $b < 1$, it holds in many cases, including all integer values of d.

Theorem 11. *Consider the baby server problem evaluated according to Relative Worst Order Analysis. For $b > 1$, if* LDC *and b-*LDC *behave differently, then they are (r, r_b)-related, where $1 < r < r_b$. If $a < 1$, a-*LDC *and* LDC *behave differently, and d is a positive integer, then they are (r_a, r)-related, where $1 < r_a < r$.*

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

Corollary 2. *When $\frac{d}{a}$ and $\frac{d}{b}$ are integers, then a-*LDC *and b-*LDC *are (b, b)-related when $b = \frac{1}{a}$.*

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.

Theorem 12. LDC *is optimal for the baby server problem according to Relative Worst Order Analysis.*

Similar proofs show that a-LDC and BAL are also optimal algorithms.

In the definitions of LDC and BAL given in Sect. 2, different decisions are made as to which server to use in cases of ties. In LDC the server which is physically closer is moved in the case of a tie (equal 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, at least temporarily. With these tie-breaking decisions, the two algorithms behave very similarly when d is an integer.

Theorem 13. LDC *and* BAL *are not comparable on the baby server problem with respect to Relative Worst Order Analysis, except when d is an integer, in which case they are equivalent.*

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, Relative Worst Order Analysis, and the Random Order Ratio have this property, but Competitive Analysis and the Max/Max Ratio do not. This was seen 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 [6] for these results under Relative Worst Order Analysis and [1] for Bijective Analysis.

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 other measures have this property.

References

1. Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. On the separation and equivalence of paging strategies. In *18th ACM-SIAM Symposium on Discrete Algorithms*, pages 229–237, 2007.
2. Wolfgang W. Bein, Kazuo Iwama, and Jun Kawahara. Randomized competitive analysis for two-server problems. In *16th Annual European Symposium on Algorithms*, volume 5193 of *Lecture Notes in Computer Science*, pages 161–172. Springer, 2008.
3. Shai Ben-David and Allan Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994.
4. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
5. Joan Boyar and Lene M. Favrholdt. The relative worst order ratio for on-line algorithms. *ACM Transactions on Algorithms*, 3(2), 2007. Article No. 22.
6. Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. The relative worst order ratio applied to paging. *Journal of Computer and System Sciences*, 73(5):818–843, 2007.
7. Joan Boyar, Sandy Irani, and Kim S. Larsen. A comparison of performance measures for online algorithms. Technical report, arXiv:0806.0983v1, 2008.
8. Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.
9. Reza Dorrigiv and Alejandro López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News*, 36(3):67–81, 2005.
10. William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley & Sons, Inc., New York, 3rd edition, 1968. Problem 28, Chapter 9, page 240.
11. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.
12. Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.
13. Claire Kenyon. Best-fit bin-packing with random order. In *7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 359–364, 1996.
14. Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *20th Annual ACM Symposium on the Theory of Computing*, pages 322–333, 1988.
15. Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.
16. Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.