5-1-2013

An Online Algorithm for the 2-Server Problem On The Line with Improved Competitiveness

Lucas Adam Bang
University of Nevada, Las Vegas

AN ONLINE ALGORITHM FOR THE 2–SERVER PROBLEM ON THE LINE WITH IMPROVED COMPETITIVENESS

by

Lucas Bang

Bachelor of Science (B.Sc.) Mathematics
University of Nevada, Las Vegas
2010
Bachelor of Science (B.A.) Computer Science
University of Nevada, Las Vegas
2010

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science Degree in Computer Science

School of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2013

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Lucas Bang

entitled

An Online Algorithm for the 2-Server Problem on the Line with Improved Competitiveness

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Lawrence Larmore, Ph.D., Committee Chair

Wolfgang Bein, Ph.D., Committee Member

Jan Pedersen, Ph.D., Committee Member

Ebrahim Salehi, Ph.D., Graduate College Representative

Thomas Piechota, Ph.D., Interim Vice President for Research &
Dean of the Graduate College

May 2013

Abstract

In this thesis we present a randomized online algorithm for the 2-server problem on the line, named R–LINE (for Randomized Line). This algorithm achieves the lowest competitive ratio of any known randomized algorithm for the 2-server problem on the line. The competitiveness of R–LINE is less than 1.901. This result provides a significant improvement over the previous known competitiveness of $\frac{155}{78} \approx 1.987$, by Bartal, Chrobak, and Larmore, which was the first randomized algorithm for the 2-server problem one the line with competitiveness less than 2. Taking inspiration from this algorithm, we improve this result by utilizing ideas from T-theory, game theory, and linear programming.

Acknowledgements

I would first like to thank the members of my committee for their time and effort in reviewing this thesis and their important contributions to my education. Matt Pedersen has taught me a great deal about the art of writing computer programs. I am also grateful for Wolfgang Bein's guidance and many useful recommendations during the writing process. In addition, Ebrahim Salehi's courses helped provide me with a firm mathematical foundation for computer science theory. I would like to sincerely thank my advisor and committee chair, Lawrence Larmore, for his mentorship. Professor Larmore's infectious enthusiasm for theoretical computer science and mathematics has been an source of constant inspiration and motivation over the course of my education.

There are several other people who have played an important role in the completion of my graduate degree. Ajoy Datta in particular taught me what it really means to work tenaciously toward a goal. And finally, the support and encouragement of my close friends, Gabriel Allred, Michael Lwin, Amy Nakatani, and Amanda Rida have been an invaluable asset.

Lucas Bang

University of Nevada, Las Vegas
May 2013

Table of Contents

List of Figures

Chapter 1

Introduction

This first chapter provides an introduction to the important concepts of online computation including optimization problems, online algorithms, competitive analysis, potential functions, and adversary models. We also give an introduction to the k-server problem along with some related applications and a simple example to aid the reader in forming a basic understanding of the problem.

1.1 Optimization and Online Algorithms

The design and analysis of algorithms is often concerned with optimization. Typically one hopes to produce an algorithm that minimizes some measure of algorithm performance. For instance, the goal of an algorithm designer is usually to minimize the amount of time required to solve a given problem. Under different circumstances we are more interested in minimizing the amount of memory used by the algorithm. In addition, there are problems that require the algorithm to output a solution that minimizes some cost function over feasible solutions – job scheduling and network routing are two such classes of problems. The theory of online algorithms focuses on providing methods for solving optimization problems when the input is provided one piece at a time.

As we will soon see, we can measure the performance of an online algorithm by comparing it to a hypothetical optimal algorithm. Before we discuss online algorithms it is useful to have a basic understanding of the notation and vocabulary of optimization problems. (Please note that while some problems are phrased as maximization problems, they can often be transformed into equivalent minimization problems. Therefore in these definitions and in the remainder of this work we will restrict our attention to minimization problems.) We make use of the following definitions and notation conventions [BEY98].

Definition 1 *An* **optimization problem**, *denoted $\mathcal{P}$, for cost minimization begins with a set $\mathcal{I}$*

of possible inputs. For every input $I \in \mathcal{I}$ there is a set of feasible outputs $F(I)$. For every output $O \in F(I)$ we associate a positive real number $C(I, O)$ which defines the cost of the output O on input I.

Definition 2 *An* **algorithm** *ALG for optimization problem $\mathcal{P}$ is such that for any input I, ALG computes a feasible solution $ALG[I] \in F(I)$.*

Definition 3 *The* **cost** *of the output of ALG on input I is denoted $ALG(I)$ and is given by $ALG(I) = C(I, ALG[I])$.*

Definition 4 *An* **optimal algorithm**, *which we refer to as OPT, for an optimization problem $\mathcal{P}$ is such that for all inputs I,*

$$OPT(I) = \min_{O \in F(I)} C(I, O).$$

Traditional algorithm analysis takes place within an **offline** setting where all of the information necessary to compute a feasible solution is available when an algorithm begins to execute. The typical computer science student is familiar with a variety of optimization problems that are posed in the offline setting: the traveling salesman problem, minimum graph cut, and integer linear programming to name just a few.

In contrast, an **online algorithm** for an optimization problem must process input and produce outputs one piece at a time. The study of online algorithms is useful because many commonly encountered problems are inherently online – financial investing and route planning are two such problems. The difficulty in producing an online algorithm is that such an algorithm is required to make immediate decisions that might effect its overall performance, without knowledge of any future information.

One famous online problem is that of cache page eviction, where we must develop a protocol for moving pages between fast cache memory and slower main memory without knowledge of which pages might be needed in the near future. There is no cost when a memory page that is already stored in the cache is requested. On the other hand, a requested page that is not in the cache must be fetched from main memory. In addition, if the cache is full, we must evict a page to make room for the new page, also taking some large amount of time. We would like an algorithm for this problem that minimizes the number of page faults. One can imagine that analyzing the performance of such a paging protocol might be difficult, as it depends on the order of the page requests. Fortunately, the field of competitive analysis provides useful tools for dealing with this type of uncertainty. The following two sections describe this idea in general.

1.2 Competitive Analysis

As described in the previous section, we would like a formal way to analyze the performance of an online algorithm. Given an optimization problem, we will measure the performance of an online algorithm by comparing its cost with that of an optimal algorithm. At this point, one might wonder how we can reason about the cost an optimal algorithm for an online problem, for if we knew the optimal algorithm we could simply use it. For now we will hypothesize an optimal offline algorithm OPT, as in Definition 4, for a given online problem and explain how to model OPT as an adversary in the following section.

The performance of an online algorithm can be measure by its *competitive ratio*. Let $I = I_1, I_2, I_3 \ldots, I_n$ be a sequence of inputs to an online algorithm ALG and let OPT be an optimal offline algorithm that accepts I as input.

Definition 5 *We say that ALG is* c**-competitive** *if for any input sequence I there is a constant b such that*

$$ALG(I) \leq c \cdot OPT(I) + b.$$

If ALG is c-competitive we sometimes say that ALG has a **competitive ratio** of c. Intuitively, this means that the cost to the online algorithm ALG is no more than a constant factor of c times larger than the cost to a hypothetical optimal offline algorithm up to an additive constant, independent of the input sequence.

A common tactic in the design and analysis of algorithms is to make use of randomization in order to improve performance. A well known example of this principle is the randomized choice of the pivot element in the quicksort algorithm. In the deterministic quicksort algorithm, it is possible to generate an input set that causes quicksort to make $\Omega(n^2)$ comparisons. Using randomization, quicksort makes $O(n \log n)$ comparisons on average. We could imagine a malicious adversary whose goal is to cause the quicksort algorithm to behave poorly. Randomized choices prevent such an adversary from constructing an input set that guarantees a worst case running time.

We can extend the use of randomization to the analysis of online algorithms. For a random variable X, let $E[X]$ denote the expected value of X.

Definition 6 *We say that randomized online algorithm ALG is* c**-competitive** *if for any input sequence I there is a constant b such that*

$$E[ALG(I)] \leq c \cdot OPT(I) + b.$$

In both the randomized and deterministic settings we wish to design an algorithm that minimizes the competitive ratio c. This provides us with a guarantee on the performance of our online algorithm in comparison with an optimal algorithm.

1.3 Adversary Models in Competitive Analysis

There are a few subtly different ways to theorize about an optimal algorithm, OPT. One might think of this optimal algorithm as one which is able to perfectly predict future inputs. One might also think of the optimal algorithm as one that is allowed to look at the entire input before computing its outputs – an offline optimal algorithm. In this work we will use the idea of an adversary to talk concretely about the optimal algorithm.

As is common in computer science and the analysis of optimization problems, we will imagine that our input is generated by a malicious adversary who knows the details of the online algorithm. The adversary attempts to generate inputs that cause our algorithm to perform poorly. For example, one possible algorithm for the paging problem is to evict the least recently used (LRU) page. An adversary for this problem might then choose to request every page that was evicted during the previous step, knowing that such a sequence of requests causes the LRU algorithm to spend time fetching pages from slow memory at every step.

In general, we will suppose that the adversary creates the input sequence in such a way that maximizes the competitive ratio. The adversary is therefore attempting to make the cost to the online algorithm high while simultaneously making the cost to a hypothetical optimal algorithm low. In this way we are competing with the adversary, hence the phrase "competitive ratio". We often think of this interaction as a game between the online algorithm and the adversary in which the players alternate making moves; the adversary generates an input I_i and the online algorithm must generate a corresponding output O_i before I_{i+1} is revealed. In fact, this outlook allows us to use ideas from two-person zero-sum game theory when analyzing the behavior of online algorithms.

1.4 Results From Game Theory

Here we present a few useful concepts from two-person zero-sum game theory that will help us in proving the competitiveness of our algorithm R–Line in Chapter 3. We make use of a standard game theory definitions and results taken from [Bar08].

Definition 7 *A* **two-person zero-sum game** *is represented by a matrix A, where rows represent the strategies of the "row player" and columns represent the strategies of the "column player". The payoff when the row player uses strategy i and the column player uses strategy j is given by the entry*

$a_{i,j}$. *The row player seeks to maximize the payoff while the column player seeks to minimize the payoff.*

Definition 8 *A* **saddle point** *of a zero-sum game is defined to be an entry $a_{i,j}$ of the payoff matrix that is both a maximum of its row and a minimum of its column.*

Theorem 1.4.1 *If a game has a saddle point $a_{i,j}$, then the value the game is the value of the saddle point, and it is optimum for the row player to always play the i^{th} row, and for the column player to always play the j^{th} column.*

Often we can rule out a strategy in a game by using the concept of dominance. Intuitively, a row (column) is dominated when there exists a row (column) that is always a better choice. We can then simplify the game by ruling out dominated strategies.

Definition 9 *We say that a row strategy i is* **strictly dominated** *if there exists a row strategy i' such that $a_{i,j} < a_{i',j}$ for all column strategies j. Similarly, a column strategy j is* **strictly dominated** *if there exists a column strategy j' such that $a_{i,j} < a_{i,j'}$ for all row strategies i.*

Theorem 1.4.2 *If $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ is the payoff matrix for a two-person zero-sum game G, and there is no saddle point. Then*

$$v(G) = \frac{\det A}{a_{11} - a_{12} - a_{21} + a_{22}}$$

Furthermore, the optimum strategy for the row player is:

Play row 1 with probability $\frac{a_{22}-a_{21}}{a_{11}-a_{12}-a_{21}+a_{22}}$

Play row 2 with probability $\frac{a_{11}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}}$

While the optimum strategy for the column player is:

Play column 1 with probability $\frac{a_{22}-a_{12}}{a_{11}-a_{12}-a_{21}+a_{22}}$

Play column 2 with probability $\frac{a_{11}-a_{21}}{a_{11}-a_{12}-a_{21}+a_{22}}$

Later we will see that we can use this theorem to select a strategy to compete against our adversary for the server problem, which we can now describe in detail.

1.5 The k-Server Problem

The server problem was first proposed by Manasse, McGeoch and Sleator [MMS90] and the problem has been widely studied since then. We first provide formal definitions and follow with a simple example. We then briefly discuss related problems and previous results regarding the competitiveness of the server problem.

1.5.1 Detailed Definitions

In order to define the k-server problem, we will remind the reader of the definition of a metric space.

Definition 10 *A* **metric space** $\mathcal{M}$ *is a pair* (S, d) *where* S *is a set of points and* $d : S \times S \to \mathbb{R}$ *is a distance function that satisfies the following four conditions for every* $x, y, z \in S$*.*

1. *If* $x \neq y$*, then* $d(x, y) > 0$*.*
2. $d(x, x) = 0$*.*
3. $d(x, y) = d(y, x)$
4. $d(x, y) + d(y, z) \geq d(x, z)$

We now formally define the k-server problem using the traditional notation and language [BEY98]. Let $k > 1$ be an an integer and let $\mathcal{M} = (S, d)$ be a metric space where $|S| > k$ and d the metric over S. An algorithm for the k-server problem controls the motion of k server, $s_1, s_2, \ldots, s_k$, that are able to move about in S. The algorithm receives a sequence of requests $\sigma = r_1, r_2, \ldots, r_n$ where each $r_i \in S$. We say that a request r is *served* when at least one server s is located at r. The algorithm must satisfy each request in a sequential online fashion by moving its servers. For any request sequence σ and k-server algorithm ALG, the cost of service $ALG(\sigma)$ is defined to be the total distance moved by all servers, as measured by the metric d.

Definition 11 *The* **k-server problem** *is to provide an online algorithm for the movement of servers that makes the competitive ratio as small as possible.*

For an example of the 4-server problem, see Figure 1.1.

1.5.2 Adversary Servers

We will conduct our analysis of server problems by supposing that we are in competition with an adversary. Therefore we shall introduce the following notation. Our online algorithm coordinates the motion of k servers which we number $s_1, s_2, \ldots, s_k$. We then suppose that the adversary is in control of k adversary servers, which we denote by $a_1, a_2, \ldots, a_k$.

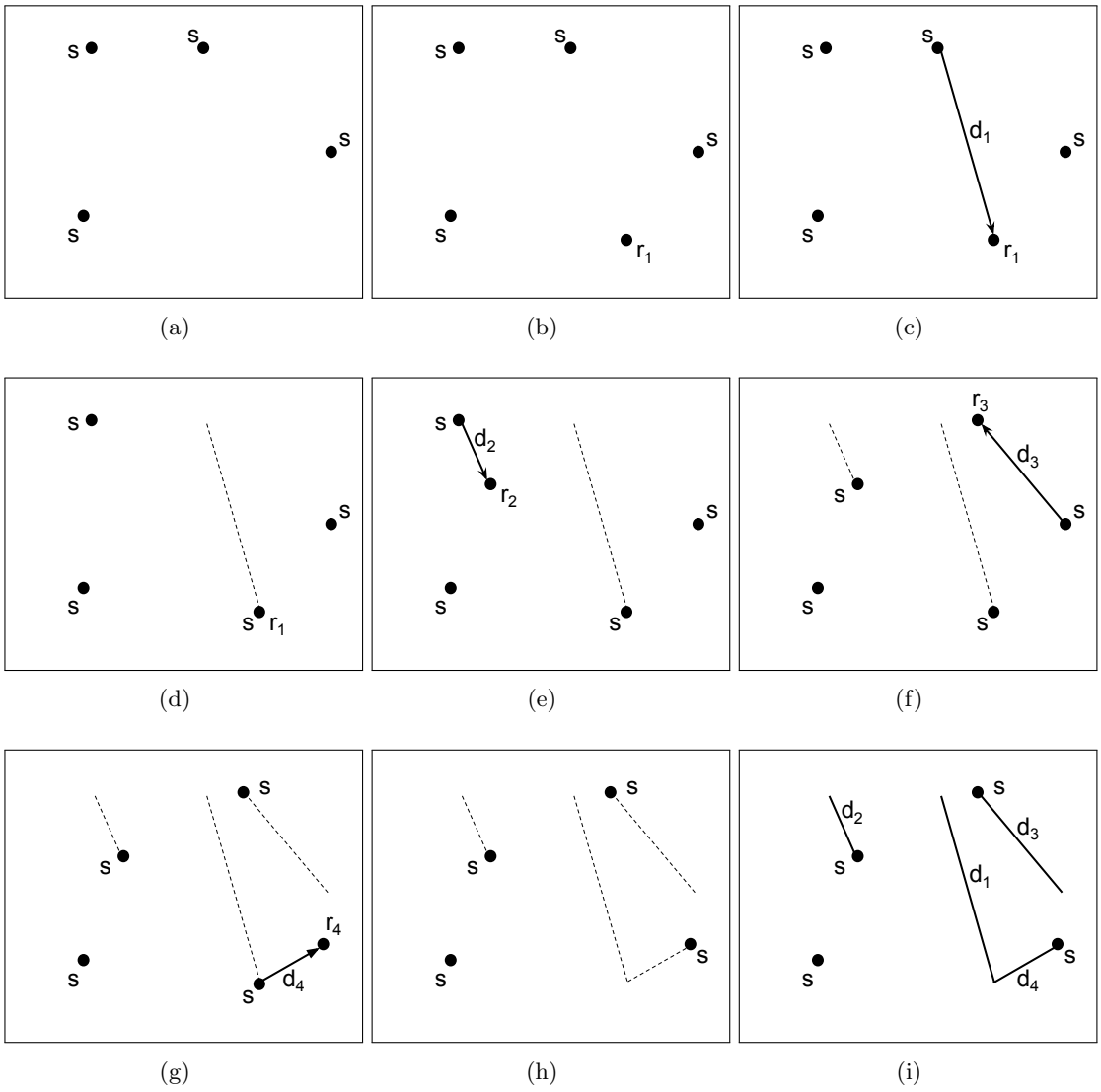


Figure 1.1: Sample execution of a 4-server algorithm. (a) There are 4 servers in initial positions. (b) The first request r_1 is made. (c) The indicated server will movie a distance of d_1 to serve the request. (d) The first request is served. (e) A server moves a distance of d_2 to serve the second request r_2. (f) A server moves a distance of d_3 to serve the third request r_3. (g) A server moves a distance of d_4 to serve the fourth request r_4. (h) The request sequence ends. (i) For the sequence $\sigma = r_1, r_2, r_3, r_4$ the cost to the algorithm is given by $ALG(\sigma) = d_1 + d_2 + d_3 + d_4$.

1.5.3 Related Problems and Applications

The k-server problem is related to several other important optimization problems. Here we will discuss three particular applications of the k-server problem.

Paging

As mentioned previously, the paging problem is a famous online optimization problem. Recall that in the paging problem we are faced with the problem of developing a protocol for evicting pages from fast cache memory whenever a page fault occurs. We can model this problem as an instance of the k-server problem where the size of $\mathcal{M}$ is the number of memory pages and the distance between any pair of distinct points is equal to 1. The k servers represent the k memory locations in the cache. This means that we consider a request to be served with a cost of 0 when the requested page is already in the cache, but the cost is 1 when a page fault occurs, in which case we evict a page from the cache and move the requested page from slow memory to the cache.

The k-Headed Disk Access Problem

Imagine a hard disk with k read/write heads. In order to access data on the disk, one head must move along the radius of the disk to the requested storage location. By coordinating the motion of multiple heads we improve overall access time compared to the time required by a single head. The performance of an algorithm for this problem can be measured as the total distance moved by the k heads. In this case, the metric space is the line segment representing the radius of the disk.

Emergency Vehicle Response

As a very direct application of the k-server problem, let us consider the example of police patrol vehicles that must respond to crimes in the city of Manhattan, New York. However, because crimes occur at unpredictable locations at unpredictable times, deciding which patrol car to dispatch is an inherently online problem. Richard C. Larson investigated this problem in 1972 before the theory of online algorithms had been formalized [Lar72]. In the language of the server problem, the police cars are our servers, the crimes are the request points, the movement cost is the driving distance to the crime, and the metric space is Manhattan Island (which lends its name to the Manhattan metric space).

In his analysis, Larson begins with a simple case in which there are only two patrol cars. He also simplifies his analysis by assuming that the crimes are committed uniformly at random within the patrol sector of the two cars. While Larson's treatment is interesting, we shall take a different,

more pessimistic approach. We assume that the crimes are committed by a coordinated team of two criminals who are dedicated to causing our dispatching algorithm to behave poorly. In the terms of online analysis, this outlook represents our assumption that we are in competition with a malevolent adversary. The criminals represent the adversary servers, and the optimal cost is the total distance moved by the two criminals. Thought of this way, Larson's problem can be modeled as a 2-server problem in the L_1 metric space over the real plane $\mathbb{R}^2$, commonly referred to at the Manhattan metric space. Recall that in the L_1 space, the distance between two points x and y is given by $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$.

1.5.4 Previous Results

The server problem was first proposed by Manasse, McGeoch and Sleator [MMS90] and the problem has been widely studied since then. They also introduced the now well-known *k-server conjecture*, which states that, for each k, there exists an online algorithm for k servers which is k-competitive for any metric space. The conjecture was immediately proved true by the same researchers for $k = 2$, but for larger k, the conjecture remains open, although it has been proved for a number of special classes of metric spaces, such as trees [CL91], spaces with at most $k + 2$ points [KP94], and the Manhattan plane for $k = 3$ [BCL02].

In the randomized case, little is known. Bartal *et al.* [BBM01] have an asymptotic lower bound, namely that the competitiveness of any randomized online algorithm for an arbitrary metric space is $\Omega(\log k / \log^2 \log k)$. It is conjectured that there is an $O(\log k)$ competitive algorithm for general metric spaces. A recent breakthrough is the algorithm by Bartal *et al.* [BBMN11], which gives a poly-logarithmic competitive algorithm for finite metric spaces.

Surprisingly, no randomized competitive algorithm for the 2-server problem for general spaces is known to have competitiveness less than 2, although that barrier has been broken for a number of classes of spaces. The competitiveness is known to be $\frac{3}{2}$ for uniform spaces, and Bein *et al.* [BIK08] have shown that there is a randomized algorithm with competitive ratio of at most 1.5897 for all 3-point spaces. Bein *et al.* [BIK$^+$11] have recently given a "better than 2" competitive algorithm for crosspolytope spaces using knowledge states [BLNR11]. A lower bound of $1 + e^{-1/2} \approx 1.606$ has been shown [CLLR97].

1.6 The Contribution of This Master's Thesis Research.

Define the (m, n)-*server problem*, for $m > n$, to be the variation where there are m mobile servers in the metric space, and each request must be served by at least n of them. We give a detailed

description of this idea in Section 2.4. For the 2-server problem on the line, Bartal *et al.* give a randomized online algorithm for the 2-server problem on the line, with competitive ratio $\frac{155}{78} \approx 1.987$ [BCL98]; their method is to define a deterministic online algorithm for the $(6, 3)$-server problem with that competitiveness, from which three deterministic online algorithms are defined. The randomized algorithm is simply to pick one of those three at random, each with probability $\frac{1}{3}$, and then use the chosen algorithm for the entire request sequence.

In this thesis, we generalize this concept and give a randomized online algorithm for the $(2n, n)$-server problem on the line, for every $n \geq 3$. This then can be thought of as n randomized 2-server algorithms. By Theorems 2.4.1 and 2.4.3, we obtain a randomized algorithm for the 2-server problem on the line. As n increases, the competitiveness of our algorithm decreases, and the limiting value is less than 1.901. This represents a significant improvement over the previous result of $\frac{155}{78} \approx 1.987$ [BCL98]. The competitiveness of this algorithm is proved using tools from T-theory, game theory, linear programming, and numeric solutions to differential equations.

Chapter 2

Important Aspects of the Server Problem

In this chapter we discuss some aspects of the server problem that will be relevant in later discussions. We start by demonstrating that a simple greedy algorithm is not competitive. Next, we discuss the RANDOM–SLACK algorithm and prove that it is 2-competitive. The RANDOM–SLACK algorithm is useful in understanding the operation of a randomized server algorithm. Lastly, we describe the (kn, n)-server problem, a generalization of the k-server problem and we prove three useful theorems.

2.1 The Greedy Algorithm Is Not Competitive

When presented with the server problem for the first time, one typically considers the greedy algorithm as a possible solution: serve the current request by moving the server that is closest to the request point. We can easily demonstrate that this algorithm is not competitive. Consider the 2-server problem for a three point metric space on three points p, q, and r (Figure 2.1) where $d(p, r) = d(p, q) + d(q, r)$ and $d(p, q) < d(q, r)$.

Figure 2.1: The greedy algorithm is not competitive. In the 3-point metric space shown, the greedy algorithm does not have a bounded competitive ratio.

Now suppose the adversary creates a request sequence $\sigma = r, p, q, p, q, p, q \ldots$, continuing to alternate between requesting p and q after a first request of r. A greedy algorithm ALG will first

position one of its two servers at r. Then all future requests will be served by the same server moving back and forth between p and q. Suppose that the adversary generates a total of n requests. Then $ALG(\sigma) \geq d(q,r) + (n-1) \cdot d(p,q)$. On the other hand, an optimal algorithm OPT will satisfy this request sequence by moving the server from r to p or q after the first request is served. Then it is easy to see that $OPT(\sigma) \leq d(p,q) + 2 \cdot d(q,r)$. As n can be made arbitrarily large, $ALG(I)$ is unbounded. Hence, there are no constants c and b such that $ALG(I) \leq c \cdot OPT(I) + b$, and so the greedy algorithm is not competitive.

2.2 The Potential Function Method For Proving Competitiveness

In order to prove the competitiveness of an online algorithm, we often employ the use of a potential function. The potential function method works for a variety of online problems, and we will make heavy use of it when analyzing server problems in this work. The potential function depends on the states of the online algorithm and the optimal algorithm.

Definition 12 *Given an optimization problem $\mathcal{P}$ let S_{ALG} and S_{OPT} be the sets of possible states of an online algorithm ALG and an optimal offline algorithm OPT for $\mathcal{P}$, respectively. A* **potential function** *ϕ is a mapping from all possible algorithm states to the real numbers:*

$$\phi : S_{ALG} \times S_{OPT} \rightarrow \mathbb{R}$$

In general, the state of an algorithm depends on the type of problem at hand and we do not give a general definition here. Intuitively, the state of an optimization algorithm is a representation of its knowledge of past requests, past decisions, and currently available information. In the case of server problems, the state of an algorithm is simply the known location of each server.

Now, recall that the algorithms operate on a sequence of inputs $I = I_1, I_2, \ldots, I_n$. We will define ϕ_i to be the value of the potential function immediately following the processing of I_i. Further, we let ϕ_0 be the initial value of the potential, before any inputs have been processed. Note that ϕ_0 is therefore a constant depends only on the initial state of the algorithms. We also define the incremental costs $\Delta ALG(I_i)$ and $\Delta OPT(I_i)$ at the ith step of computation. We can now prove a simple but powerful theorem that provides a method for proving the competitiveness of an online algorithm.

Theorem 2.2.1 *If the following conditions hold at every step for any input sequence then ALG is c-competitive.*

1. $\Delta\phi_i + \Delta ALG(I_i) \leq c \cdot \Delta OPT(I_i)$

2. There is a constant b such that for all i $\phi_i \geq b$.

Proof: We may take the summation over all inputs to achieve the desired result. To see this, suppose that the first condition holds:

$$\Delta\phi_i + \Delta ALG(I_i) \leq c \cdot \Delta OPT(I_i).$$

Let us add the inequalities over the entire input sequence:

$$\sum_{i=1}^{n} \Delta\phi_i + \sum_{i=1}^{n} \Delta ALG(I_i) \leq \sum_{i=1}^{n} c \cdot \Delta OPT(I_i).$$

Successive terms of the change in potential cancel, except for the initial and final values of the potential. The incremental cost to the algorithms adds up to the total cost of each algorithm. After rearranging:

$$ALG(I) \leq c \cdot OPT(I) + \phi_0 - \phi_n.$$

By the second condition, $\phi_n \geq b$. Thus

$$ALG(I) \leq c \cdot OPT(I) + \phi_0 - b.$$

Since ϕ_0 is a constant that does not depend on I, this shows that ALG is c-competitive by Definition 6. ■

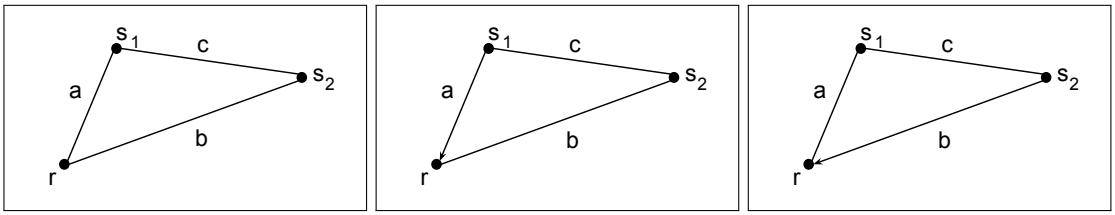
We will later use this method to analyze individual steps of server algorithms. As we are often concerned with only a single step of execution, we may drop the subscripted indices of Properties 1 and 2 in future discussions when no confusion arises.

2.3 The RANDOM–SLACK Algorithm

Here we present an algorithm, RANDOM–SLACK, for the 2-server problem in an arbitrary metric space. This algorithm is useful in the development and understanding of randomized algorithms for the 2-server problem. The algorithm is quite simple to state . Let the two algorithm servers be s_1 and s_2 and the current request be denoted r.

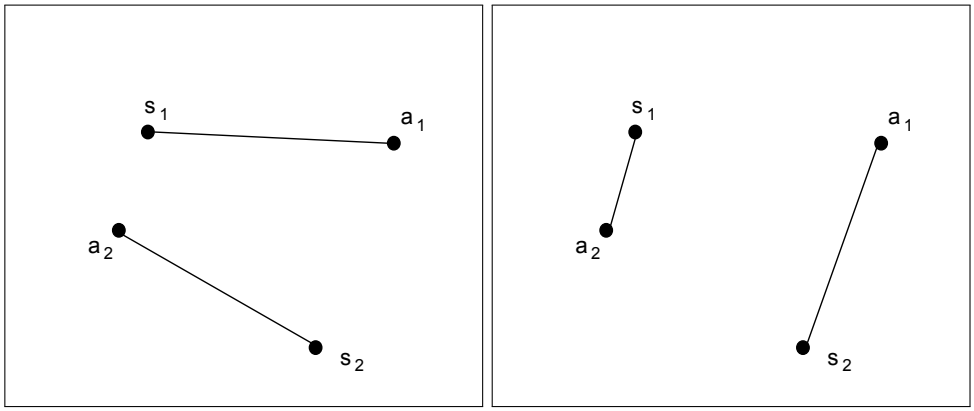
> **Algorithm RANDOM–SLACK:** Move server s_i to r with probability $p_i = \frac{\epsilon_j}{d(s_i,s_j)}$, where $\epsilon_i = \frac{1}{2}[d(s_i,s_j) + d(s_i,r) - d(s_j,r)]$.

A single step of execution is demonstrated in Figure 2.2.



(a) (b) (c)

Figure 2.2: A single move by RANDOM–SLACK. (a) Before serving the current request r, the two servers s_1 and s_2 are in the positions shown. The distances are $d(s_1, r) = a$, $d(s_2, r) = b$, and $d(s_1, s_2) = c$. (b) Server s_1 moves to r with probability $p_1 = \frac{\frac{1}{2}(c+b-a)}{c}$. (c) Alternatively, server s_2 moves to r with probability $p_2 = \frac{\frac{1}{2}(c+a-b)}{c}$.



(a) (b)

Figure 2.3: Minimal matchings. Consider matchings for the 2-server problem in the real plane. We see by visual inspection that the matching in (a) $\{s_1, a_1\}, \{s_2, a_2\}$ is not minimal while the matching in (b) $\{s_1, a_2\}, \{s_2, a_1\}$ is minimal.

2.3.1 Competitiveness of RANDOM–SLACK

It turns out that RANDOM–SLACK is 2-competitive for any metric space. Here we will prove this fact for the simpler case where the metric space is the real line $\mathbb{R}$. In order to do so, we will define a potential function ϕ that depends on the positions of the servers and the minimum matching distance.

Definition 13 *The* **minimum matching** *of algorithm servers $s_1, s_2, \ldots, s_k$ and adversary servers $a_1, a_2, \ldots, a_k$ is a permutation π such that the quantity $M = \sum_{i=1}^{k} d(s_i, a_{\pi(i)})$ is minimal. We call the value of M the* **minimum matching distance**.

The idea of the minimum matching is described in Figure 2.3. It is straightforward to observe that in the case of the real line, the minimum matching is obtained by numbering the servers from left to right as $s_1, s_2, \ldots, s_k$ and $a_1, a_2, \ldots, a_k$, matching each s_i with a_i. Using the minimum

matching distance M, we may now define a potential function that will allow us to prove that RANDOM–SLACK is 2-competitive on the line [CDRS90].

Definition 14 *For the k-server problem, we define the* **Coppersmith–Doyle–Raghavan–Snir (CDRS) potential** *,*

$$\phi = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} d(s_i, s_j) + k \cdot M.$$

Armed with this potential function we may now prove the following theorem.

Theorem 2.3.1 *The algorithm RANDOM–SLACK is 2-competitive on the line.*

Proof: Without loss of generality, we may always label the servers in such a way that $s_1 \leq s_2$ and $a_1 \leq a_2$. Furthermore, we may separate the steps of the computation into stages. Every step of the computation shall have the following form:

1. The adversary generates a request r and immediately serves that request with one of its servers.
2. The algorithm moves one of its servers to the request point.

Analysis of the adversary's move. We refer to the adversary's server algorithm as ADV and first show that for every adversary move, $\Delta\phi \leq 2 \cdot \text{ADV}(r)$. Due to the symmetry of the line, we may assume that the adversary serves request r with server a_1. This analysis is accompanied by Figure 2.4. The cost of the adversary move is given by

$$\text{ADV}(r) = d(a_1, r).$$

Using the definition of the CDRS potential, we can compute ϕ, the value of the potential before the move, and ϕ', the value of the potential after the move.

$$\phi = d(s_1, s_2) + 2d(s_1, a_1) + 2d(s_2, a_2).$$

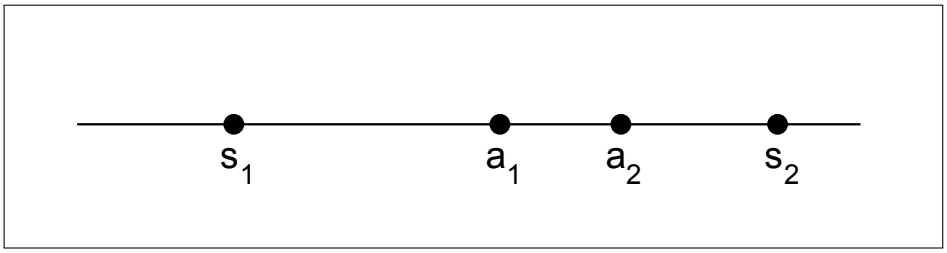
$$\phi' = d(s_1, s_2) + 2d(s_1, r) + 2d(s_2, a_2).$$

Subtracting to calculate the change in potential and using the triangle inequality we have

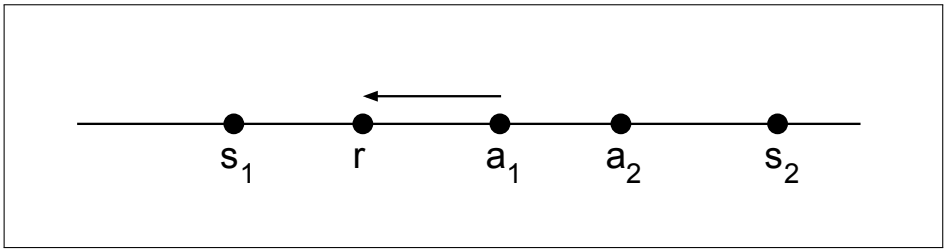
$$\Delta\phi = 2(d(s_1, r) - d(s_1, a_1)) \leq 2d(a_1, r) = 2 \cdot ADV(r).$$

Analysis of RANDOM–SLACK move. We will refer the service cost of RANDOM–SLACK on request r as $\text{RS}(r)$ and show that for every move by RANDOMSLACK, $\Delta\phi + \text{RS}(r) \leq 0$. By definition of the CDRS potential, before our servers move

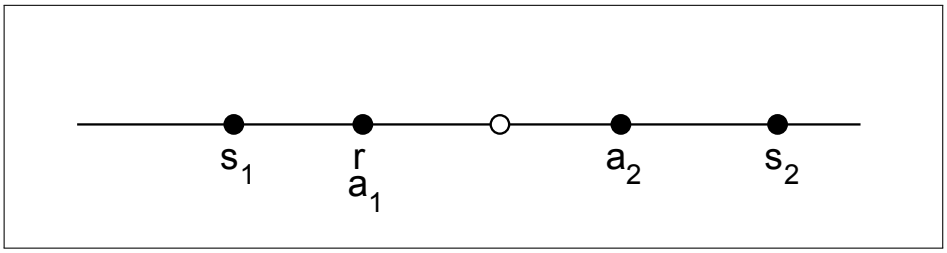
$$\phi = d(s_1, s_2) + 2d(s_1, a_1) + 2d(s_2, a_2).$$



(a)

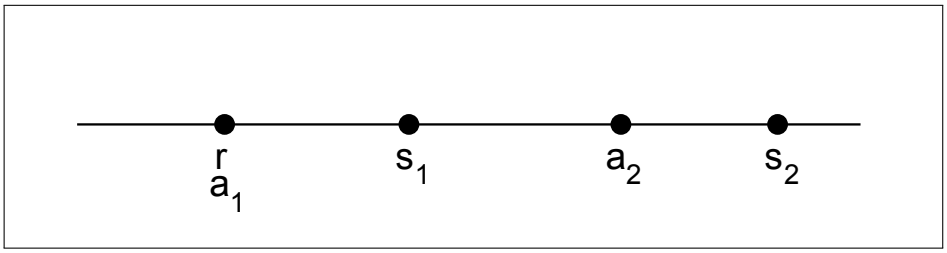


(b)

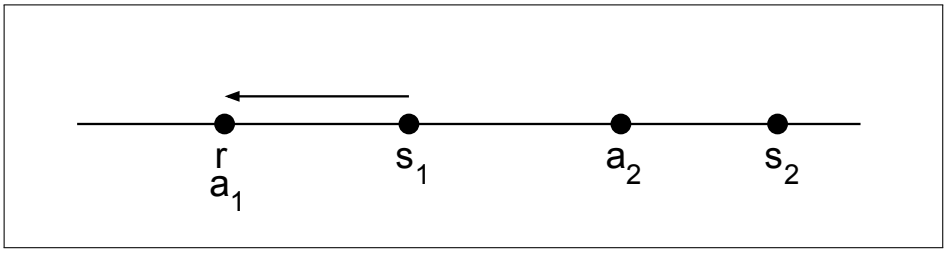


(c)

Figure 2.4: An adversary move in analysis of the RANDOM–SLACK algorithm. An example adversary move in the proof of Theorem 2.3.1. (a) A possible configuration of servers at the beginning of a round. (b) A request is made at r and the adversary serves with a_1 at cost $\mathrm{ADV}(r) = d(a_1, r)$. (c) The adversary move is done.



(a)



(b)

Figure 2.5: An example of Case 1 from the proof of Theorem 2.3.1. (a) A possible configuration of servers before RANDOM–SLACK moves. (b) Server s_1 moves to the request point since $p_1 = 1$ and $p_2 = 0$ using the definition of RANDOM–SLACK. The cost for this move is given by $\mathrm{RS}(r) = d(s_1, r)$.

By symmetry of the line, there are only two cases to consider for moves made by RANDOM–SLACK.

Case 1 The servers in in a configuration where $a_1 \leq s_1 \leq s_2$. Refer to Figure 2.5. In this case we move s_1 deterministically, since $p_1 = 1$ and $p_2 = 0$ using the probabilities in the definition of RANDOM-SLACK. We know that $a_1 = r$ by definition of the adversary move, and so we have

$$RS(r) = d(s_1, a_1).$$

We can compute the value of the potential after s_1 moves to $r = a_1$ as

$$\phi' = d(a_1, s_2) + 2d(s_2, a_2).$$

Thus, $\Delta\phi = -d(s_1, a_1)$. We then have that

$$\Delta\phi + \mathrm{RS}(r) \leq 0.$$

Case 2 The servers are in a configuration where $s_1 \leq a_1 \leq s_2$. Refer to Figure 2.6. In this case, by the definition of RANDOM–SLACK, we move s_1 to a_1 with probability

$$p_1 = \frac{d(s_2, a_1)}{d(s_1, s_2)}$$

with the cost of the move given by

$$\mathrm{RS}_1(r) = d(s_1, a_1).$$

The value of the potential after s_1 moves is

$$\phi_1' = d(s_1, a_1) + 2d(s_2, a_2).$$

Hence,

$$\Delta\phi_1 = -3d(s_1, a_1).$$

On the other hand, we move s_2 with probability

$$p_2 = \frac{d(s_1, a_1)}{d(s_1, s_2)}$$

and the associated cost of the move is

$$\mathrm{RS}_2(r) = d(s_2, a_1).$$

The value of the potential after s_2 moves is

$$\phi_2' = d(s_1, a_1) + 2d(s_1, a_1) + 2d(a_1, a_2).$$

Computing the change in potential for this case and applying the triangle inequality,

$$\Delta\phi_2 = -d(a_1, s_2) + 2(d(a_1, a_2) - d(s_2, a_2)) \leq -d(a_1, s_2) + 2d(s_2, a_2) = d(s_2, a_1).$$

We can now compute the expected sum of the change in potential and incremental cost.

$$E[\Delta\phi] = p_1 \cdot \Delta\phi_1 + p_2 \cdot \Delta\phi_2 = -2\frac{d(s_1, a_1)d(s_2, a_1)}{d(s_1, s_2)}.$$

$$E[\mathrm{RS}(r)] = p_1 \cdot \mathrm{RS}_1(r) + p_2 \cdot \mathrm{RS}_2(r) = 2\frac{d(s_1, a_1)d(s_2, a_1)}{d(s_1, s_2)}.$$

Adding these two inequalities gives us the desired result:

$$E[\Delta\phi] + E[\Delta\mathrm{RS}(r)] = 0.$$

In both Case 1 and Case 2, adding the inequalities derived from both the adversary move and the server move,

$$E[\Delta\phi] + E[\Delta\mathrm{RS}(r)] \leq 2 \cdot \mathrm{ADV}(r),$$

and so algorithm RANDOMSLACK is 2-competitive by Theorem 2.2.1. ■

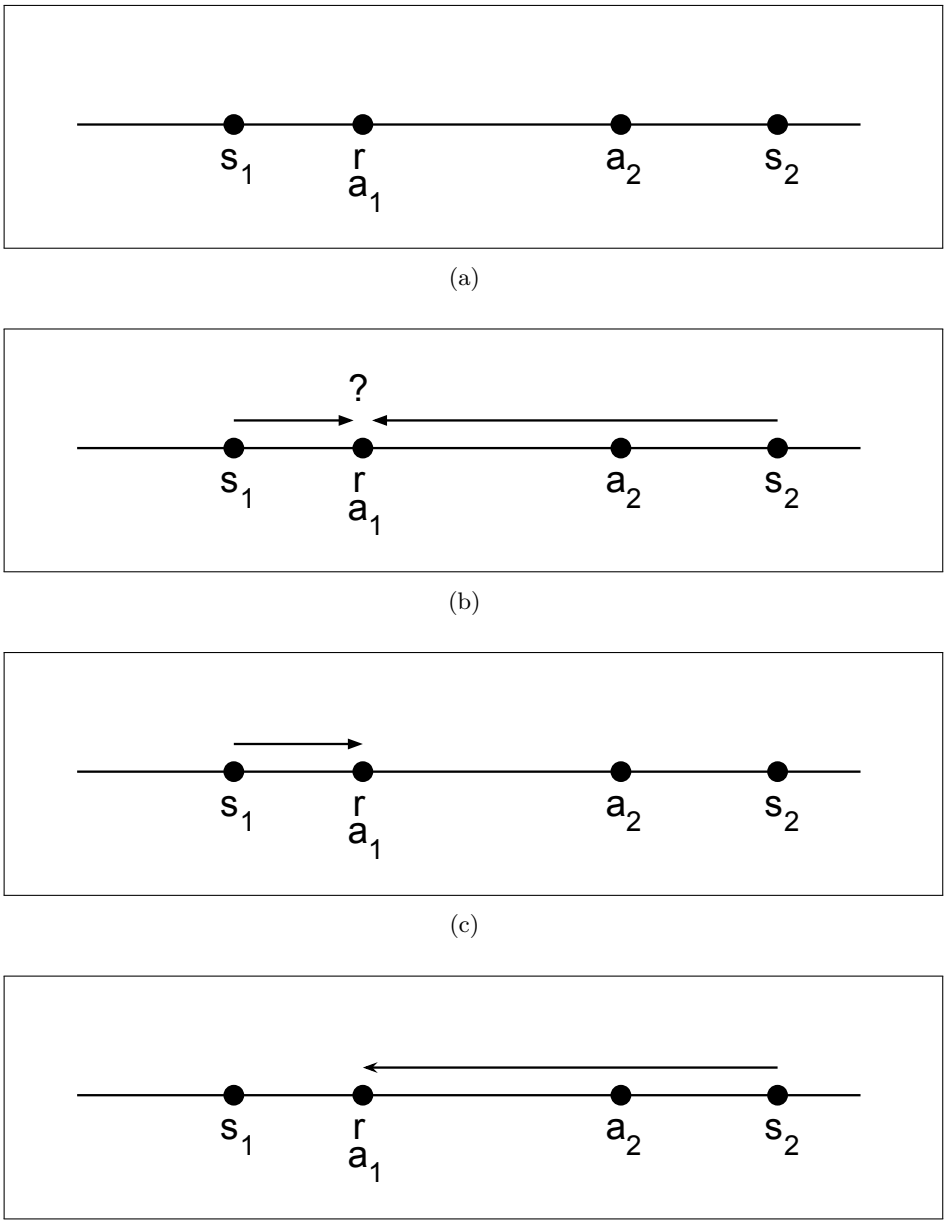


Figure 2.6: An example of Case 2 from the proof of Theorem 2.3.1. (a) A possible configuration of servers before RANDOM–SLACK moves. (b) Either one of the servers might move. (c) Server s_1 moves with probability p_1 and cost $\mathrm{RS}_1(r) = d(s_1, r)$. (d) Server s_2 moves with probability p_2 and cost $\mathrm{RS}_1(r) = d(s_2, r)$.

2.4 The (kn, n)-Server Problem

We now define a generalization of the k-server problem. Recall the emergency vehicle response example of Section 1.5.3 and suppose that each crime requires two vehicles to respond; every car requires a "back up" car. Thus, whenever there is a request we must send two vehicles to scene of the crime. For the general server problem, suppose that there are m servers and each request requires $n < m$ servers to be moved to a request point. We then define the (m, n)-server problem to be the same as a server problem where at each step n servers are moved to the requested point. Then the k-server problem is an instance of the (k, n) server problem, where $n = 1$. In this thesis, we will make use of the variation where m is an integer multiple of n, say $m = kn$, and so we are working with the (kn, n)-server problem. Refer to Figure 2.7 for a detailed example.

In this discussion we will assume that the servers are initially situated in k groups of n servers in the same location. This is a reasonable assumption, for if the servers are not in such a configuration we can move them until they are grouped as described. Their movement into this configuration will incur some cost, but this cost does not depend on the request sequence and will not affect the competitiveness of an algorithm for the (kn, n) server problem. We will now prove three important theorems that will be useful in Chapter 3. These theorems are given in [BCL98] and we give the details of each proof.

Theorem 2.4.1 *Given any strategy for the $(2n, n)$-server problem, we can derive a randomized strategy for the 2-server problem.*

Proof: Suppose A is any algorithm for the $(2n, n)$-server problem and that the servers are numbered $s_1, \ldots s_{2n}$. Define the "partner" of a server s_i by the function

$$\text{partner}(s_i) = \begin{cases} s_{i+n} & : i \leq n \\ s_{i-n} & : i > n \end{cases}$$

That is, s_1 is partnered with s_{n+1}, s_2 is partnered with s_{n+2}, and so on.

We now describe an algorithm A' that mimics the behavior of A. In addition A' behaves in such a way that for any request at step r, A' moves either s_i or partner(s_i) to serve r, but not both. To see how this works, we recall that initially there are 2 groups of n servers located at the same point. Furthermore, directly after any previous request, say r', there must be n servers located at r' for the request to have been satisfied. So, suppose that at some step of the request sequence, $s_1, \ldots s_n$ have served r', and let the next request be r.

Now A will move some number of servers from among $s_{n+1}, \ldots, s_{2n}$ to serve r. Let the number of moved servers be z. Let T be the set of servers from $s_{n+1}, \ldots, s_{2n}$ that move to r. Then A' also moves the same servers to r. Now, A must move $n - z$ servers from r' to r. Then we have A' move

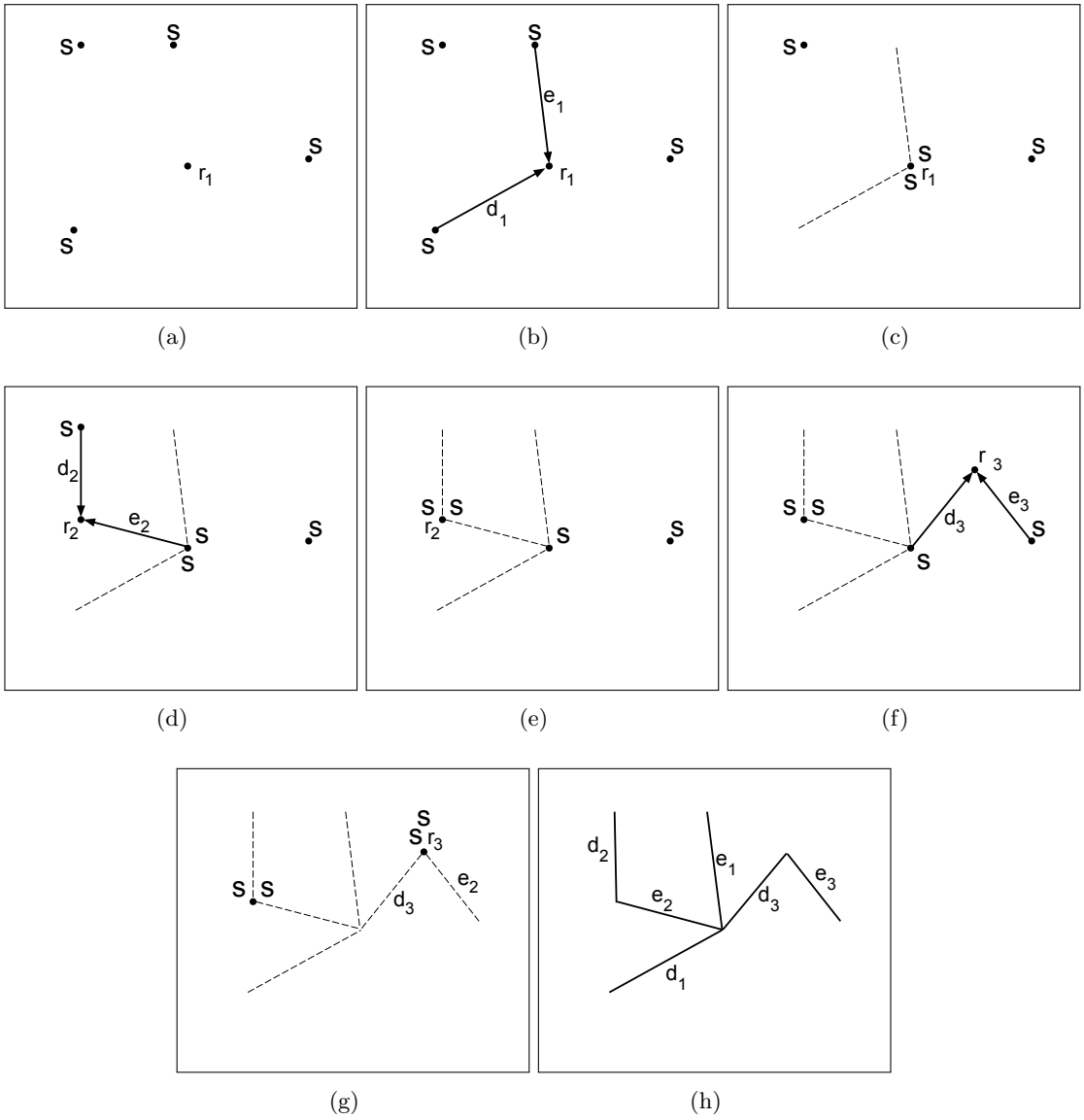


Figure 2.7: A sample execution of a $(4, 2)$-server algorithm. (a) There are 4 servers in initial positions with the first request r_1. (b) The first request r_1 served by two servers moving distances of d_1 and e_1. (c) Request r_1 is satisfied. (d) The first request r_2 served by two servers moving distances of d_2 and e_2. (e) Request r_2 is satisfied. (f) The first request r_3 served by two servers moving distances of d_3 and e_3. (f) Request r_3 is satisfied. (h) The request sequence ends. For the sequence $\sigma = r_1, r_2, r_3$ the cost to the algorithm is given by $ALG(\sigma) = d_1 + e_1 + d_2 + e_2 + d_3 + e_3$.

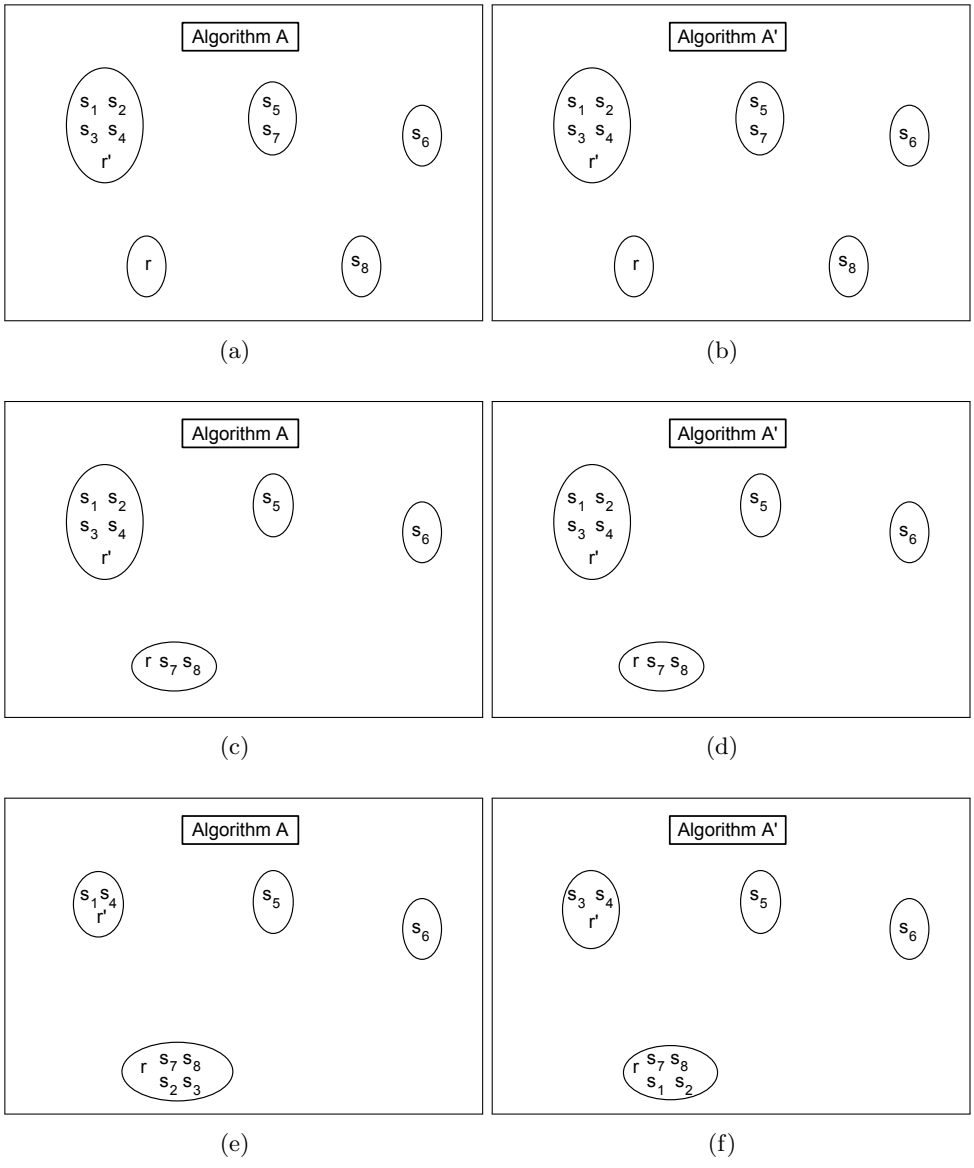


Figure 2.8: An example for the proof of Theorem 2.4.1 for the $(8, 4)$-server problem.. We indicate the server movements of algorithm A and the derived algorithm A' which mimics A. Here we illustrate the $(8, 4)$-server case. We indicate that a group of servers are located together at the same point by enclosing them in an oval. For instance, s_1, s_2, s_3, and s_4 are located together at the previous request r'. According to the definition of partner(s_i), the paired partners are $\{s_1, s_5\}, \{s_2, s_6\}, \{s_3, s_7\}$, and $\{s_4, s_8\}$.(a) A possible configuration of A at the beginning of a round. (b) A' has the same initial configuration. (c) A moves s_7 and s_8 to r. (d) A' also moves s_7 and s_8 to r. (e) A moves s_2 and s_3 to finish the request. (f) According to the derivation of A', we cannot move s_3 or s_4 which are the partners of s_7 and s_8. Thus, we move s_1 and s_2 to finish the request.

the servers $\{s_1, s_2, \ldots, s_n\}/partner(T)$ from r' to r. In this way, A' mimics A with equal cost. After any request round we may relabel the servers and use the same derivation again to mimic A in the following round.

Since A maintains the separation of partner servers at every step, A' can be thought of as a collection of n independent 2-server algorithms, say $A' = \{A_1, A_2, \ldots, A_n\}$ where A_i determines the movement of s_i and s_{i+n}. We may then define a randomized 2-server strategy A'' that uses A_i with probability $p_i = \frac{1}{n}$. ■

Theorem 2.4.2 *Any optimal offline strategy for the $(2n, n)$ server problem keeps the servers in two blocks of n each, assuming that the servers are together in two blocks in the initial configuration.*

Proof: Suppose that A is an optimal offline strategy for the $(2n, n)$-server problem. Consider the derived strategy $A' = \{A_0, A_1, \ldots, A_{n-1}\}$ as described in the previous proof. For any request sequence σ,

$$A'(\sigma) = \sum_{i=0}^{n-1} A_i(\sigma).$$

Now consider the average cost of the algorithms $A_0, A_1, \ldots, A_{n-1}$. We have

$$\frac{1}{n} \sum_{i=0}^{n-1} A_i(\sigma) = \frac{1}{n} A'(\sigma) = \frac{1}{n} A(\sigma).$$

Then there exists some j such that $A_j(\sigma) \leq \frac{1}{n} A(\sigma)$, and so $A'(\sigma) \geq n \cdot A_j(\sigma)$. Now let A'' be an algorithm for the $(2n, n)$-server problem that uses n copies of A_j moving $2n$ servers in two groups of size n. Then

$$A''(\sigma) = n \cdot A_j(\sigma) \leq A(\sigma).$$

However, A is optimal by assumption, so $A''(\sigma) \geq A(\sigma)$ as well. Hence, $A''(\sigma) = A(\sigma)$. Consequently, A'' is an optimal algorithm and moves the servers in two blocks of size n each. ■

Theorem 2.4.3 *Given a C-competitive online strategy for the $(2n, n)$-server problem in a metric space X, the derived randomized online strategy for the 2-server problem is C-competitive.*

Proof: Let A be an online c-competitive algorithm for the $(2n, n)$-server problem, and let $OPT_{2n,n}$ be an optimal algorithm for the $(2n, n)$-server problem. Then for any request sequence σ,

$$A(\sigma) \leq c \cdot OPT_{2n,n}(\sigma).$$

Let A' be the derived online randomized strategy as described in the proof of Theorem 2.4.1. Then

$$E[A'(\sigma)] = \frac{1}{n} \sum_{i=0}^{n-1} A_i(\sigma) = \frac{1}{n} A(\sigma).$$

Since $OPT_{2n,n}$ moves servers in two blocks of size n by the previous theorem, $OPT_{2n,n}$ is equivalent to n copies of an optimal 2-server algorithm operating in tandem. Let that optimal 2-server algorithm be $OPT_{2,1}$. Then

$$OPT_{2,1}(\sigma) = \frac{1}{n} OPT_{2n,n}(\sigma).$$

We have that

$$\frac{1}{n} A(\sigma) \leq \frac{1}{n} \cdot c \cdot OPT_{2n,n}(\sigma),$$

which gives us

$$E[A''(\sigma)] \leq c \cdot OPT_{2,1}(\sigma).$$

Consequently, A'' is c-competitive. ■

These three theorems are useful in the following way. In Chapter 3, Theorem 2.4.1 will allow us to develop a c-competitive algorithm for the $(2n, n)$-server problem which immediately gives us the derived randomized algorithm for the 2-server problem. Then by Theorem 2.4.3, the derived 2-server problem is also c-competitive. Lastly, Theorem 2.4.2 allows us to simplify the analysis of adversary moves by reducing it to the case where the adversary is using an optimal 2-server algorithm.

Chapter 3

The Algorithm R–LINE

This chapter provides the details of the main result of this thesis: a competitive algorithm for the 2-server problem on the line. Given here is an outline of the steps involved in this result.

1. Define a randomized algorithm for the $(2n, n)$-server problem.
2. Provide a set of inequalities $\mathbb{S}$, involving C, such that if $\mathbb{S}$ is satisfied, then R–LINE is C–competitive.
3. Use a series of substitutions to transform $\mathbb{S}$ into a differential equation, D.
4. Approximate the solution to D that minimizes C for a given value of n.
5. The resulting minimum computed value of C is 1.90079728 when $n = 10000$.
6. R–LINE is then defined as a randomized algorithm, derived from the $(2n, n)$-server algorithm, and so is 1.90079728–competitive.

3.1 Preliminaries

Our algorithm, R–LINE, is defined to be a randomized algorithm for the $(2n, n)$-server problem, for $n \geq 3$. By Theorem 2.4.2, without loss of generality we can assume that the adversary is using an optimal 2-server algorithm, but serves with cost equal to n times the distance moved. We will use the notation s_i both to refer to the i^{th} server and its location, when no confusion arises. We assume that $s_1 \leq s_2 \leq ... \leq s_{2n-1} \leq s_{2n}$. We also refer to the adversary's servers as a_1 and a_2, and assume that $a_1 \leq a_2$. The algorithm thus knows the location of one of the adversary's servers, which we call the *visible* server, and which, by a slight abuse of notation, we also call r. We denote the adversary's other server by a, and refer to it as the *hidden* server, since the algorithm does not know where it is.

We define a configuration of servers (R–LINE's as well as the adversary's) to be *satisfying* if at least n of R–LINE's servers are at r. We refer to a satisfying configuration as an S-configuration, and we assume that the initial configuration is an S-configuration.

Every round begins by the adversary choosing a new request point r and moving one of its two servers to r. R–LINE then moves as many of its servers as necessary to r, and the resulting configuration is once again an S-configuration. No R–LINE server will pass another R–LINE server that does not serve. In general, R–LINE deterministically moves zero or more servers to r, and then uses randomization to decide which additional servers to move. R–LINE is *lazy*, meaning that it never moves any server that does not serve the request.

3.2 The Potential

The algorithm R–LINE is given based on a suitable potential, which is used in Section 3.4 to prove competitiveness. For each fixed $n \geq 3$, we define a competitiveness C for R–LINE as well as a potential ϕ on configurations. This potential will satisfy the following property:

Property 3.2.1 *If ϕ is the potential at the configuration before a round and ϕ' the potential after the round, and if R–LINE(r) and ADV(r) are the costs incurred by R–LINE and the adversary on request r, respectively, then*

$$E[\text{R–LINE}(r) + \phi' - \phi] \leq C \cdot \text{ADV}(r)$$

where E denotes expected value.

Isolation indices. For $0 \leq i \leq 2n$ and $0 \leq j \leq 2$, if $1 \leq i + j \leq 2n + 1$, we define $\alpha_{i,j}$, the $(i,j)^{\text{th}}$ *isolation index* of a configuration, to be the length of the longest interval that has exactly i algorithm servers to the left and exactly j adversary servers to the left. If there is no such interval, then the value of the isolation index is 0. More formally,

$$\alpha_{i,j} = \max \begin{cases} \min\{s_{i+1}, a_{j+1}\} - \max\{s_i, a_j\} \\ 0 \end{cases}$$

where we let $s_0 = a_0 = -\infty$ and $s_{2n+1} = a_3 = \infty$ by default. We refer the reader to Figure 3.1 for further understanding. [1]

Isolation index coefficients. For each $0 \leq i \leq 2n$ and $0 \leq j \leq 2$, we define a constant $\eta_{i,j}$, the $(i,j)^{\text{th}}$ *isolation index coefficient*. The isolation index coefficients satisfy a symmetry property,

[1] A more general definition of isolation indices is given in [BD92].

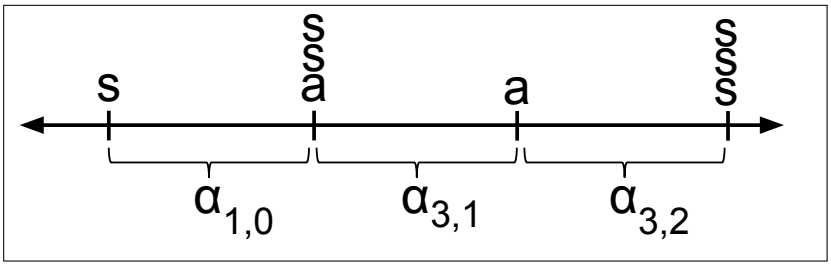


Figure 3.1: Isolation indices on the line for the $(6, 3)$-server problem. here we illustrate the nonzero isolation indices for a particular configuration of the $(6, 3)$ case. Here, $\alpha_{1,0}$ is the length of the interval that has 1 algorithm server and 0 adversary servers to its left. Isolation index $\alpha_{3,1}$ is the length of the interval that has 3 algorithm servers and 1 adversary server to its left. In a similar fashion $\alpha_{3,2}$ is the length of the interval that has 3 algorithm server and 2 adversary servers to its left. All other finite isolation indices are 0. For this configuration the potential can be computed as $\phi = \eta_{1,0} \cdot \alpha_{1,0} + \eta_{3,1} \cdot \alpha_{3,1} + \eta_{3,2} \cdot \alpha_{3,2}$.

namely $\eta_{i,j} = \eta_{2n-i,2-j}$; furthermore, $\eta_{0,0} = \eta_{2n,n} = 0$. We formally define the potential of a configuration to be

$$\phi = \sum \{\eta_{i,j} \cdot \alpha_{i,j} \ : \ (0 \le i \le 2n) \wedge (0 \le j \le 2) \wedge (1 \le i + j \le 2n + 1)\}.$$

Intuitively, $\eta_{i,j}$ is a weight on the isolation index $\alpha_{i,j}$ for any configuration. For each given n, the competitiveness C and the isolation index coefficients $\{\alpha_{i,j}\}$ must satisfy a system of inequalities given in Section 3.4.

We will first define R–LINE in terms of those constants. We then go on to show that R–LINE is C-competitive if the system of inequalities is satisfied. In the final sections of this chapter we describe how to find a solution to these inequalities.

3.3 Algorithm Description

We now define R–LINE. Between rounds, the configuration of servers is always an S-configuration. When the adversary makes a request at a point r, R–LINE responds by making a sequence of *moves*, each consisting of the movement of one or more servers to r. Thus, during a round, R–LINE makes at most n moves. Not all configurations can arise during execution of R–LINE; in fact, we define two classes of configurations, D-configurations and R-configurations, such that every intermediate configuration of R–LINE belongs to one of those two classes. If the current configuration is a D-configuration, then R–LINE's next move is to move one or more servers deterministically to r, while if the current configuration is an R-configuration, then R–LINE's next move is to choose, using randomization, a set of servers to move to r. In this case there are always two choices – to move one or more servers from the previous request point to r, completing the round, or to move just one

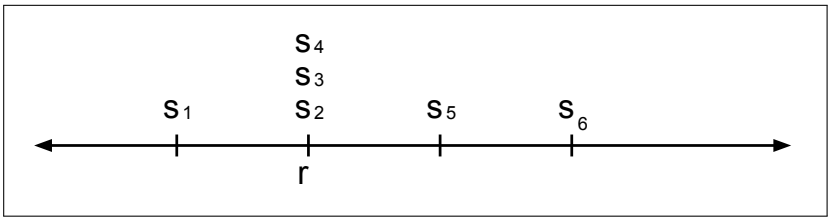


Figure 3.2: An example of a satisfying configuration (an S-configuration) for the $(6, 3)$ case. Servers s_2, s_3, and s_4 are located at the current request r, and so the request is satisfied.

server from the other side, possibly not completing the round.

We now define the classes of configurations. Note that, before the current round began, there must have been n algorithm servers at the previous request point, which we call r'. Without loss of generality, $r' \neq r$.

1. **S-Configuration**: there are n algorithm servers at r. See Figure 3.2

2. **D-Configuration**: the following two conditions hold.

   (a) There are more than n algorithm servers either strictly to the left or strictly to the right of r; that is, $r > s_{n+1}$ or $r < s_n$.

   (b) If there are fewer than n algorithm servers at r', then there is no algorithm server strictly between r' and r, and furthermore, there are at least n algorithm servers at the points r' and r combined.

   The D-configuration can be explained in the following way. In a D-configuration suppose without loss of generality that there are m algorithm servers to the left of r for some $m > n$. We can argue that we must move the servers $s_{n+1}, \ldots s_m$ to r. Suppose otherwise that a server from among $s_{n+1}, \ldots s_m$ does not move to r. Then, since servers do not pass one another, there are only $n - 1$ servers available to serve the request. Thus, the request cannot be satisfied. Refer to Figures 3.3 and 3.4 to understand this reasoning for the $(6, 3)$-server problem.

3. **R-Configuration**:

   (a) There are exactly n algorithm servers on the same side of r as r', that is, either $r' = s_n < r$ or $r < r' = s_{n+1}$.

   (b) There is no algorithm server strictly between r' and r, and furthermore, there are at least n algorithm servers at the points r' and r combined.

   See Figure 3.5 for an example.

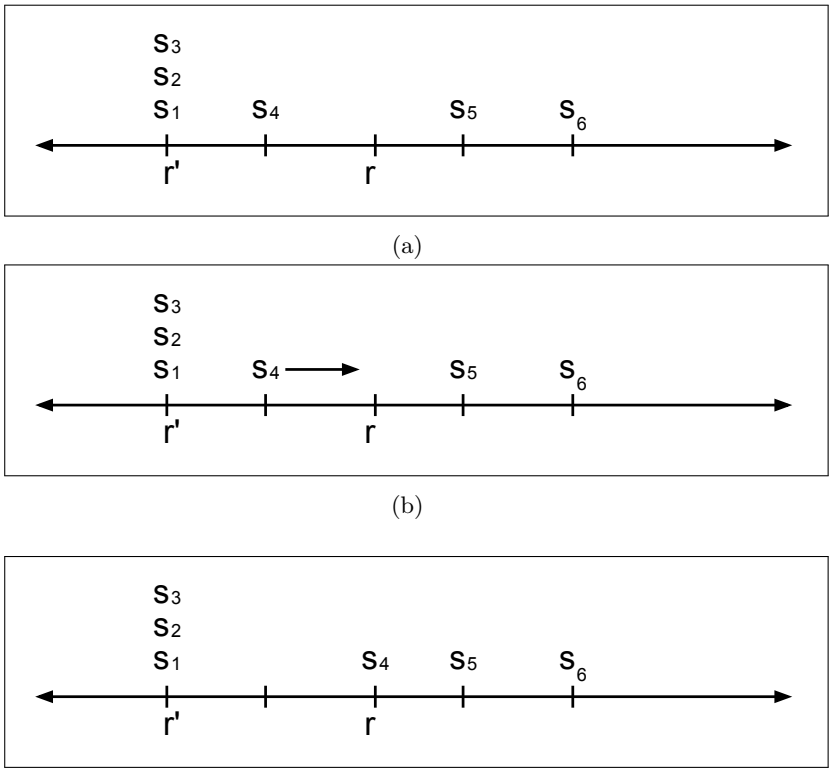


Figure 3.3: An example of a deterministic move (a D-configuration) for the $(6, 3)$ case. (a) At the beginning of the round, there are three servers s_1, s_2, s_3 located at the previous request point r' and one more server s_4 to the left of the current request r. Two servers s_5 and s_6 are located to the right of the current request. (b) We know that s_4 must move to r. If s_4 did not move then only s_5 and s_6 are able to serve r, as s_1, s_2, and s_3 are not allowed to pass s_4. (c) Server s_4 has moved to r and we are now in an R-configuration.

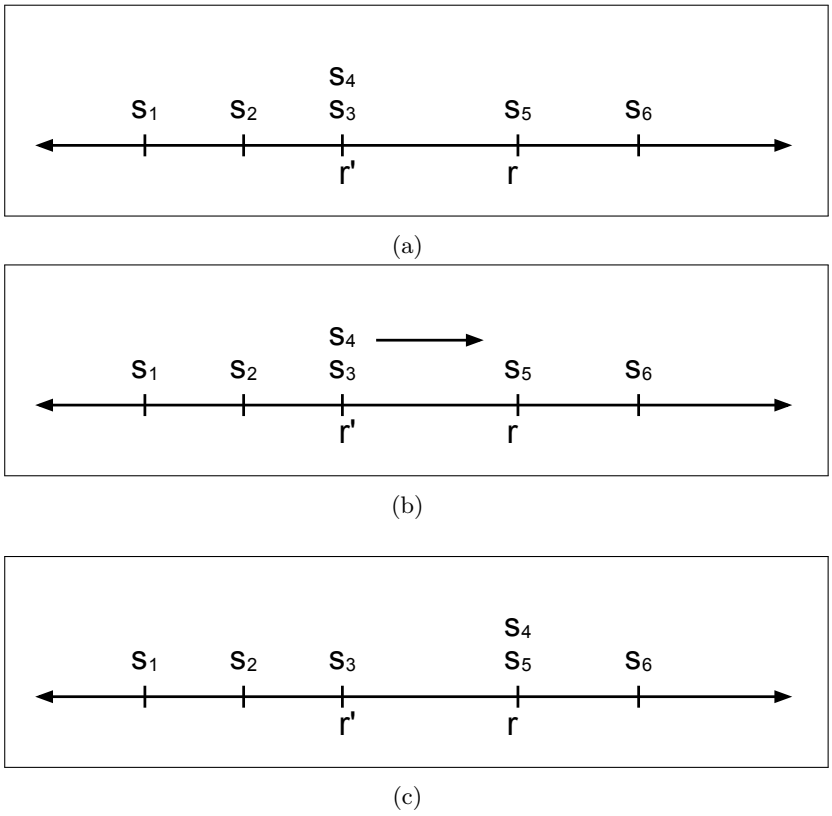


Figure 3.4: A second example of a deterministic move (a D-configuration) for the $(6, 3)$ case. (a) At the beginning of the round, there are two servers s_3 and s_4 located at the previous request point r' while s_1 and s_2 are further to the left. Two servers s_5 and s_6 are located to the right of the current request r. (b) Here we know that s_3 or s_4 must move to r. If neither of them move then only s_5 and s_6 are able to serve r, as s_1 and s_2 are not allowed to pass s_4 and s_3. (c) Server s_4 has moved to r and we are now in an R-configuration.

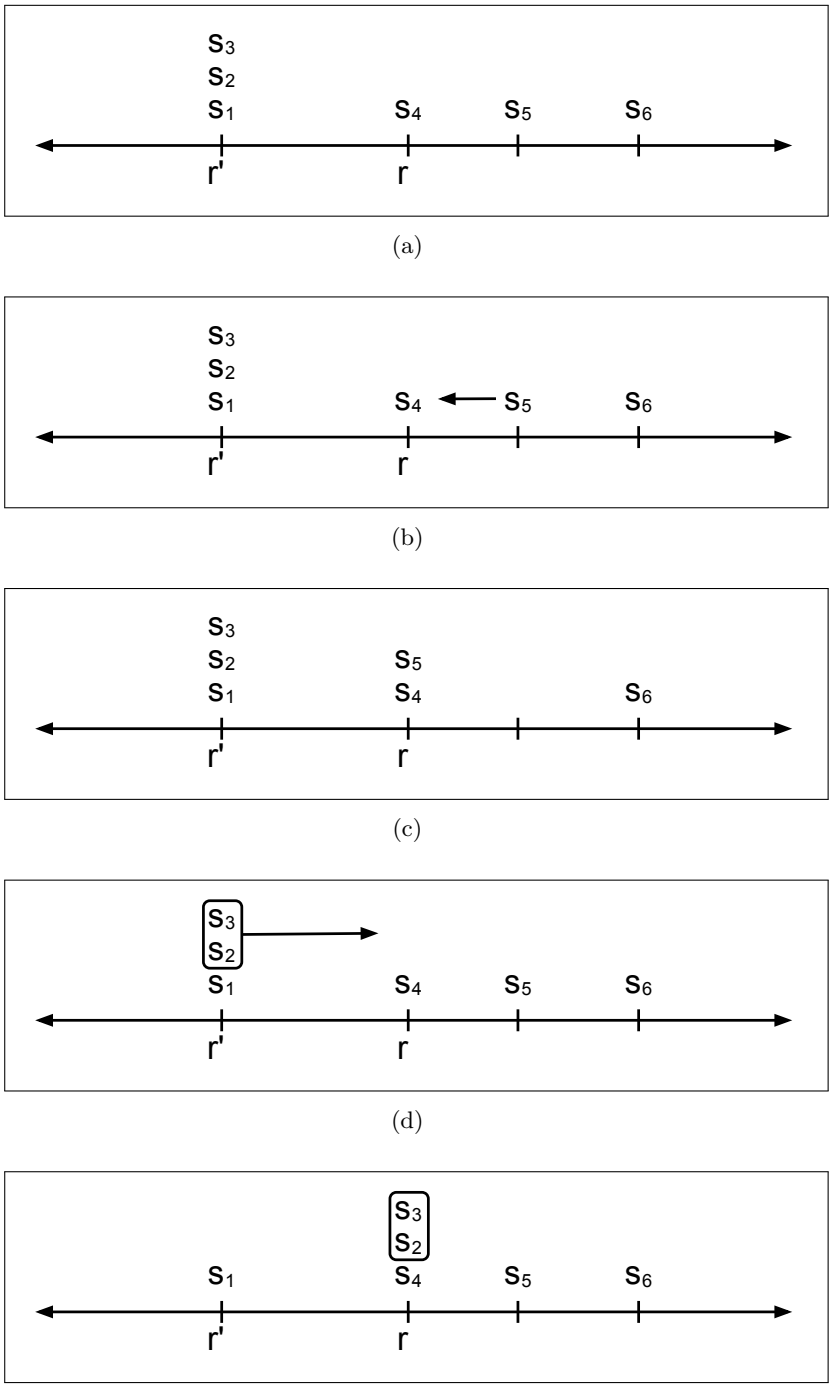


Figure 3.5: An example of a randomized move (an R-configuration) for the $(6, 3)$ case. (a) There are three servers s_1, s_2, s_3 located at the previous request point r' and one server s_4 at the current request r. Two servers s_5 and s_6 are located to the right of the current request. (b) One possibility is that s_5 will move to the current request point. (c) Server s_5 has moved to r and we are still in an R-configuration. (d) Another possibility is that we serve r using s_2 and s_3. (e) Servers s_2 and s_3 have moved to r and we are in an S-configuration.

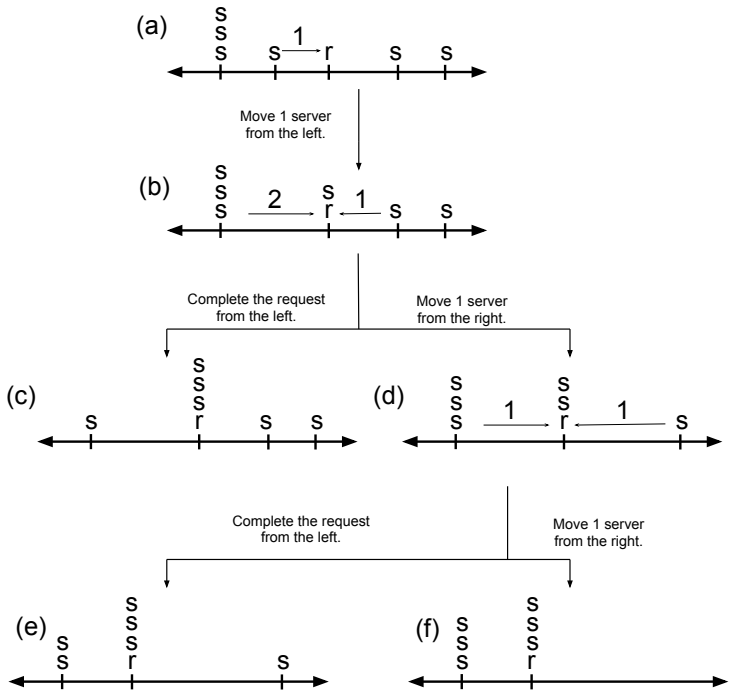


Figure 3.6: Possible executions of R–LINE. (a) A D-configuration, where $n = 3$. The request is r, there are three servers located at $r' < r$. The next move is deterministic. (b) An R-configuration. One server has moved to r from the left. The next move is randomized; either move two servers from the left or one from the right. (c) An S-configuration, after two servers moved from the left. The round is over. (d) An R-configuration, after one server moved from the right. The next move is randomized; either move one server from the left or one from the right. (e) An S-configuration, after one server moved from the right. The round is over. (f) An S-configuration, after one server moved from the left. The round is over.

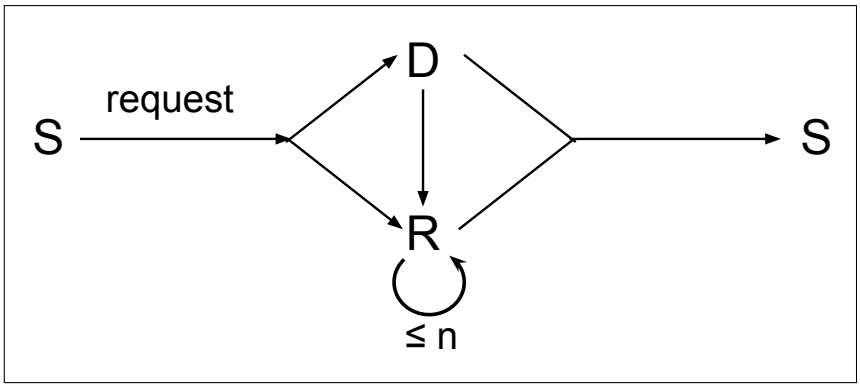


Figure 3.7: A state transition diagram for R–LINE. We begin a round in an S-configuration. When we receive a request, we might end up in either a D-configuration or an R-configuration. From a D-configuration we might possibly go to an S-configuration or an R-configuration. Whenever the algorithm is in an R-configuration there can be no more than n randomized moves. We end the round when we have reached an S-configuration and R-LINE is ready fro the next request.

We now give an explicit definition of R–LINE. By symmetry, we can assume, without loss of generality, that $r' < r$. The reader might also consult Figure 3.6 where we illustrate R–LINE through a single round, in a case where $n = 3$. The general flow of a single round is illustrated in Figure 3.7.

1. If the current configuration is a D-configuration, then there are m algorithm servers to the left of r for some $m > n$. Move the servers $s_{n+1}, \ldots s_m$ to r. If the resulting configuration is an S-configuration, the round is over. Otherwise, the resulting configuration is an R-configuration, and we proceed to the next step.

2. If the current configuration is an R-configuration, then $r' = s_n < r \leq s_{n+1} < s_{2n}$. Let p be the number of algorithm servers at r. Then $s_{n+p+1} > r$. R–LINE executes one of two moves; each move is executed with a probability that is determined by solving a 2-person zero-sum game. We compute those probabilities below. The two choices of move are:

   (a) Move s_{n+p+1} to r.

   (b) Move the servers $s_{p+1} \ldots s_n$ to r.

   If the resulting configuration is an S-configuration, the round is over. Otherwise, the resulting configuration is an R-configuration, and repeat this step. The reader may refer to Figure 3.5.

For the randomized step, one of the two choices is selected by using the optimum strategy for a 2-person zero sum game, where R–LINE is the column player, and $\mathcal{A}dv$ is the row player; the choice of the row player is where to place the hidden server. As we show later, we can assume, without loss of generality, that the hidden server is located at either s_n or s_{n+p+1}. Thus, each player has exactly two strategies. Each entry of the payoff matrix is equal to $\Delta\phi + cost = \phi' - \phi + cost$, where ϕ and ϕ' are the potentials before and after the move; and *cost* is the cost of the move, which is equal to the number of servers moved times the distance moved, either $(s_{n+p+1} - r)$ or $(n - p)(r - s_n)$.

The payoff matrix is as follows:

$$G =$$

	Move s_{n+p+1}	Move $s_{p+1} \ldots s_n$
$a = s_n$	$(\eta_{n+p+1,2} - \eta_{n+p,2} + 1)(s_{n+p+1} - r)$	$(\eta_{p,1} - \eta_{n,1} + n - p)(r - s_n)$
$a = s_{n+p+1}$	$(\eta_{n+p+1,1} - \eta_{n+p,1} + 1)(s_{n+p+1} - r)$	$(\eta_{p,0} - \eta_{n,0} + n - p)(r - s_n)$

The entries of the game matrix G depend only on the values of the isolation index coefficients, the locations of the algorithm servers, and the current request. This matrix and the isolation index coefficients will play an important role in proving the competitiveness of R–LINE.

3.4 Proof of Competitiveness

We now present a system of inequalities, which we denote $\mathbb{S}$, which suffice for R–LINE to be C-competitive. We will prove, in Theorem 3.4.1, that $\mathbb{S}$ implies C-competitiveness of R–LINE.

$$\forall\ 0 \le i \le 2n:\ |\eta_{i,1} - \eta_{i,0}| \le n \cdot C \tag{3.1}$$

$$\forall\ 1 \le i \le n \text{ and } \forall\ 1 \le j \le 2:\ \eta_{i,j} + 1 \le \eta_{i-1,j} \tag{3.2}$$

$$\forall\ 1 \le i \le n \text{ and } \forall\ 1 \le j \le 2:\ \eta_{i-1,j-1} \le \eta_{i,j-1} + 1 \tag{3.3}$$

$$\forall\ 1 \le i \le n:\ (\eta_{i-1,1} - \eta_{i,1} + 1)(\eta_{n-i,1} - \eta_{n,1} + i) \le (\eta_{i-1,0} - \eta_{i,0} + 1)(\eta_{n-i,0} - \eta_{n,0} + i) \tag{3.4}$$

Theorem 3.4.1 *For any assignment of values to C and $\eta_{i,j}$ for $0 \le i \le 2n$ and $0 \le j \le 2$ that satisfies the system $\mathbb{S}$,* R–LINE *is C-competitive.*

We prove Theorem 3.4.1 with a sequence of lemmas. We will prove that if the system of inequalities $\mathbb{S}$ is satisfied, then the following properties hold. We write $\Delta\phi = \phi' - \phi$, where ϕ is the potential before the move and ϕ' is the potential after the move.

1. For any move by the adversary, $\Delta\phi \le C \cdot cost_{\mathcal{A}dv}$. (Recall that the adversary pays n times the distance moved.)

2. For any deterministic move by R–LINE, $\Delta\phi + cost \le 0$.

3. We may assume the adversary's hidden server is at one of at most two possible locations during a given round, namely at the closest algorithm server to either the left or the right of r.

4. For any randomized move by R–LINE, $\mathrm{E}(\Delta\phi + cost) \le 0$.

We say that a move is *simple* if the move consists of moving a single server (either an algorithm or an adversary server) across an interval, and there is no other server (of either type) located strictly between the end points of that interval. We also refer to a simple move as a *step*; in general, every movement of servers is a concatenation of steps.

Lemma 3.4.2 *If $\mathbb{S}$ holds, then Property 1 holds.*

Proof: By the symmetry of the $\eta_{i,j}$, inequality (3.1) implies that $|\eta_{i,j} - \eta_{i,j-1}| \le n \cdot C$ for $j = 1, 2$. Without loss of generality the move is simple, since every move which is not simple is the concatenation of simple moves. Without loss of generality, the adversary server a_j moves to the right, from x to y, where $x < y$. Since the move is simple, $s_i \le x$ and $y \le s_{i+1}$ for some $0 \le i \le 2n$,. (Recall the default values $s_0 = -\infty$ and $s_{2n+1} = \infty$.) Thus, $\alpha_{i,j}$ decreases by $y - x$ and $\alpha_{i,j-1}$

increases by $y-x$. The cost to the adversary of this move is $n(y-x)$. By definition of the potential, $\Delta\phi = (\eta_{i,j} - \eta_{i,j-1})(y-x) \leq n \cdot C \cdot (y-x) \leq C \cdot cost_{\mathcal{A}dv}$. ■

Lemma 3.4.3 *If $\mathbb{S}$ holds, then Property 2 holds.*

Proof: For convenience, we assume that $r < r' = s_{n+1}$. There are exactly m algorithm servers to the right of r, for some $m > n$. Servers $s_{2n-m+1} \ldots s_n$ move to r. The move is the concatenation of steps, and it suffices to show that $\Delta\phi \geq cost_{\text{R–LINE}}$ for each of those steps.

Fix one step. During the step, s_i moves from x to y, where $y < x$, for some $2n-m+1 \leq i \leq n$. The algorithm cost of the step is $x-y$. Pick the maximum j such that $a_j \leq y$. Since $r \leq y$, j is either 1 or 2. The move causes $\alpha_{i,j}$ to increase by $x-y$ and $\alpha_{i-1,j}$ to decrease by the same amount. By inequality (3.1), and the definition of the potential: $\Delta\phi + cost_{\text{R–LINE}} = (x-y)(\eta_{i,j} - \eta_{i-1,j} + 1) \leq 0$. ■

Lemma 3.4.4 *If $1 \leq i \leq 2n$ and $j = 1, 2$, then $\eta_{i,j} + \eta_{i-1,j-1} \leq \eta_{i,j-1} + \eta_{i-1,j}$*

Proof: Suppose $i \leq n$. Then $1 + \eta_{i,j} \leq \eta_{i-1,j}$ by (3.2), while $-1 + \eta_{i-1,j-1} \leq \eta_{i,j-1}$ by (3.3). Adding the two inequalities, we obtain the result.

If $i > n$, then $\eta_{2n-i+1,3-j} + \eta_{2n-i,2-j} \leq \eta_{2n-i+1,2-j} + \eta_{2n-i,3-j}$ by the previous case. By symmetry, we are done. ■

Lemma 3.4.5 *If $\mathbb{S}$ holds, then Property 3 holds.*

Proof: Since a could be any point on the line, the payoff matrix of the game has infinitely many rows. We need to prove that just two of those rows, namely $a = s_n$ and $a = s_{n+p+1}$, dominate the others.

By batching the row strategies, we illustrate the $\infty \times 2$ payoff matrix below.

		Move s_{n+p+1}	Move $s_{p+1} \ldots s_n$
I	$a \leq s_n$	$(\eta_{n+p+1,2} - \eta_{n+p,2} + 1)(s_{n+p+1} - r)$	$(\eta_{p,1} - \eta_{n,1} + n - p)(r - s_n)$
II	$s_n \leq a \leq r$	$(\eta_{n+p+1,2} - \eta_{n+p,2} + 1)(s_{n+p+1} - r)$	$(\eta_{p,1} - \eta_{n,1} + n - p)(r - a)$ + $(\eta_{p,0} - \eta_{n,0} + n - p)(a - s_n)$
III	$r \leq a \leq s_{n+p+1}$	$(\eta_{n+p+1,2} - \eta_{n+p,2} + 1)(s_{n+p+1} - a)$ + $(\eta_{n+p+1,1} - \eta_{n+p,1} + 1)(a - r)$	$(\eta_{p,0} - \eta_{n,0} + n - p)(r - s_n)$
IV	$a \geq s_{n+p+1}$	$(\eta_{n+p+1,1} - \eta_{n+p,1} + 1)(s_{n+p+1} - r)$	$(\eta_{p,0} - \eta_{n,0} + n - p)(r - s_n)$

The row strategy $a = s_n$ trivially dominates all row strategies in Batch I. It also dominates all row strategies in Batch II, because

$$\begin{aligned}
\eta_{p,1} - \eta_{n,1} &= \sum_{i=p+1}^{n} (\eta_{i-1,1} - \eta_{i,1}) \\
&\geq \sum_{i=p+1}^{n} (\eta_{i-1,0} - \eta_{i,0}) \quad \text{by Lemma 3.4.4} \\
&= \eta_{p,0} - \eta_{n,0}
\end{aligned}$$

The row strategy $a = s_{n+p+1}$ trivially dominates all row stages in Batch IV. It also dominates all row strategies in Batch III, because $\eta_{n+p+1,1} - \eta_{n+p,1} \geq \eta_{n+p+1,2} - \eta_{n+p,2}$, which we can similarly prove using Lemma 3.4.4. ■

Lemma 3.4.6 *If* $\mathbb{S}$ *holds, then Property 4 holds.*

Proof: Consider the 2×2 payoff matrix G of Section 3.3. By $\mathbb{S}$, the upper left and lower right entries of G are negative, while the upper right and lower left entries are positive. By Theorem 1.4.2, the value of our game is

$$\frac{\det(G)}{(\eta_{n+p+1,2} + \eta_{n+p+1} - \eta_{n+p,2} - \eta_{n+p+1,1}) \cdot (s_{n+p+1} - r) + (\eta_{p,0} + \eta_{n,1} - \eta_{n,0} - \eta_{p,1}) \cdot (r - s_n)}$$

The numerator is non-negative by inequality 3.4. The denominator is negative, which we can prove by combining inequalities of $\mathbb{S}$ labeled (2) and (3). Thus, $E(\Delta\phi + cost_{\text{R–LINE}}) = v(G) \leq 0$ as claimed. ■

Theorem 3.4.1 follows immediately from Lemmas 3.4.2, 3.4.3, 3.4.5, and 3.4.6.

3.5 Simplifying the Inequalities

We wish to find a solution to the system $\mathbb{S}$ that minimizes the competitiveness, C. We shall make a series of substitutions in order to transform $\mathbb{S}$ into a new system of inequalities that is more readily solved using the method described here. First, recall the inequalities labeled (4) in $\mathbb{S}$.

$$(\eta_{i,1} - \eta_{i-1,1} - 1)(\eta_{n,1} - \eta_{n-i,1} - i) \leq (\eta_{i,0} - \eta_{i-1,0} - 1)(\eta_{n,0} - \eta_{n-i,0} - i).$$

Notice that we may bound the left hand side. Using the inequalities (2) we have that

$$(\eta_{i,1} - \eta_{i-1,1} - 1) \leq -2 \text{ and } (\eta_{n,1} - \eta_{n-i,1} - i) \leq -2i.$$

Thus the left hand side of the equation is bounded by $4i$:

$$4i \leq (\eta_{i,1} - \eta_{i-1,1} - 1)(\eta_{n,1} - \eta_{n-i,1} - i).$$

Now, for $0 \leq i \leq n$, use the bound of $4i$ for the left hand side of the equation, let $\delta_i = 3i - \eta_{i,0}$, let $\epsilon_i = \delta_i - \delta_n$, and let $\delta_n = 2\delta$. Making the substitutions we get:

$$(2i + \epsilon_{n-i})(2 - \epsilon_i + \epsilon_{i-1}) \geq 4i.$$

We make the assumption that the first n inequalities are exact and that the nth inequality may not be. In the eventual solution of the system, we will see that this is in fact the case. We then seek a solution to the following system:

$$(2i + \epsilon_{n-i})(2 - \epsilon_i + \epsilon_{i-1}) = 4i, \quad \forall i : 0 < i < n \tag{3.5}$$

$$(2n + \epsilon_0)(2 - \epsilon_n + \epsilon_{n-1}) \leq 4n \tag{3.6}$$

By making the following substitutions, a solution to the above described system that maximizes δ also provides a solution to S that minimizes C.

$$\begin{aligned}
\eta_{i,0} &= 3i - \delta_i && \forall i : 0 \leq i \leq n \\
\eta_{i,0} &= 2n + i - \delta_n && \forall i : n < i \leq 2n \\
\eta_{i,1} &= 2n - i - \delta && \forall i : 0 \leq i \leq n \\
\eta_{i,1} &= i - \delta && \forall i : n < i \leq 2n \\
\eta_{i,2} &= \eta_{2n-i,0} && \forall i : 0 \leq i \leq 2n
\end{aligned}$$

3.6 A Differential Difference Equation

We may convert the above recurrence relation of inequalities 3.5 and 3.6 into a differential equation in the following way. For any fixed $0 \leq t \leq 1$, let

$$g(t) = \lim_{n \to \infty} \epsilon_{n,[t \cdot n]}/n$$

where $[x]$ denotes the nearest integer to x. In the limiting case as $n \to \infty$, the recurrence relation then becomes

$$(2t + g(1 - t)) \cdot (2 - g'(t)) = 4t$$

for $0 \leq t \leq 1$. We can substitute variables to make the equation look more symmetric. If we let $t(x) = \frac{x+1}{2}$ and $f(x) = g(t(x))$, we have

$$(x + 1 + f(-x)) \cdot (1 - f'(x)) = x + 1 \tag{3.7}$$

for $-1 \leq x \leq 1$.

Notice that the differential equation contains the terms $f(-x)$ and $f'(x)$. A differential equation that relates $f'(x)$ to $f(-x)$ has come to be known as a *differential equation with reflected argument*. Particular versions of this problem have been studied, but no general analytic method of solution has been found [Rob72]. Solving this equation analytically would theoretically provide an exact value for the competitiveness of R–LINE, but all attempts to produce such a solution have unfortunately been unsuccessful. Therefore, we have settled for an approximation, using a modified first-order numeric method.

3.7 Approximating the Differential Equation Numerically

In order to approximate the solution to the differential equation, we use a modified Euler shooting method. Suppose that we want to approximate the solution of the initial value problem

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0.$$

The approximation proceeds in steps. We choose a value h for the size of every step and set $t_n = t_0 + n \cdot h$. A single step of the Euler method from t_n to $t_n + h$ is given by

$$y_{n+1} = y_n + h \cdot f(t_n, y_n),$$

where y_n is the approximated value of y at time t_n. The error in this approximation is $O(h^2)$. By making h small we can typically achieve small error.

Approximating the solution to our differential is complicated by the reflected argument. We can then modify the standard Euler method slightly to take the reflection into account. Now, suppose that we seek to approximate a solution for

$$y'(t) = f(t, y(-t)), \qquad y(t_0) = y_0.$$

Our method proceeds similarly to the Euler method, by alternating approximations in both directions. At every step we approximate the solution for two time values equidistant from t_0: $t_n = t_0 + n \cdot h$ and $t_{-n} = t_0 - n \cdot h$. We then compute the following two updates.

$$y_{n+1} = y_n + h \cdot f(t_{-n}, y_{-n}),$$

$$y_{-n-1} = y_{-n} + h \cdot f(t_n, y_n),$$

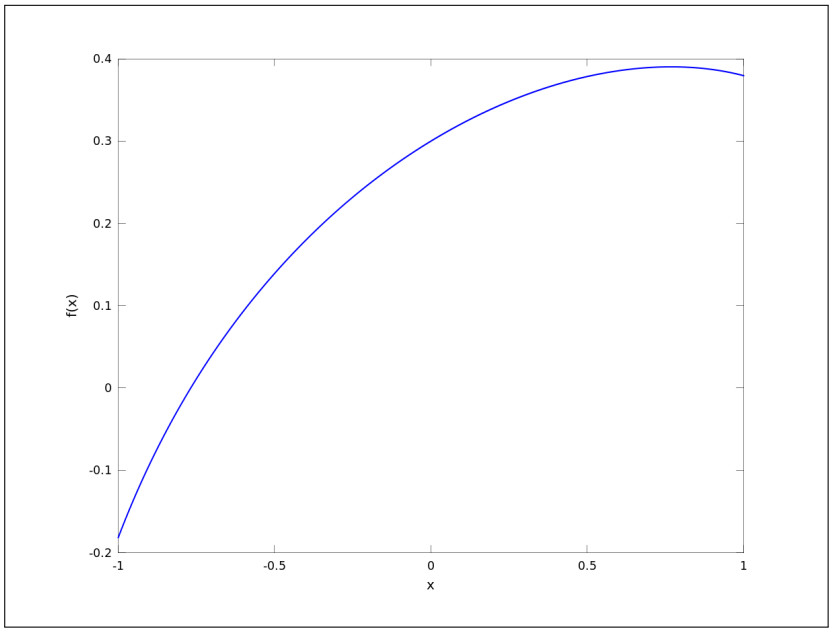
By straightforward observation we see that the error in this method is still $O(h^2)$.

We can now approximate the differential equation in 3.7 using this method, for any initial value $f(0)$, by discretizing the interval $(-1, 1)$ into n pieces. Given such an approximation we can back–substitute to find the corresponding values of ϵ_i from the recurrence relation defined in 3.5 and 3.6, from which we can determine the value of C. We then use the following algorithm to find an initial value $f(0)$ and the corresponding approximation to $f(x)$ that minimizes C.

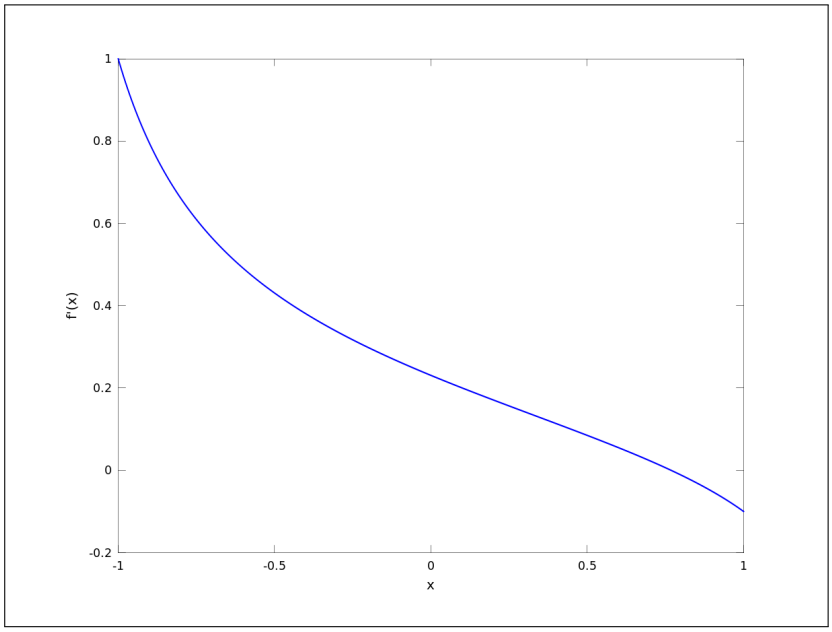
Algorithm: Search for minimum C

1. Choose a range of initial values R.
2. For every value of $f(0)$ in R:
   (a) Approximate $f(x)$ using the modified Euler method.
   (b) Compute the corresponding value of $C = 2 - \frac{\delta_n}{2n}$ using the substitutions of Section 3.5.
3. Choose the solution that minimizes C.
4. Use the substitutions of Section 3.5 to confirm that $f(x)$ corresponds to values of $\eta_{i,j}$ that satisfy $\mathbb{S}$.

This algorithm was implemented in the GNU Octave language which is an open source software package specifically designed for solving numeric problems. The code for this algorithm is given in Appendix A. Using a discretization of 10000 intervals, we achieve a competitive ratio of $C = 1.90079728$. A plot of the approximations of $f(x)$ and $f'(x)$ are given in Figure 3.8. A plot of the competitiveness of R–LINE verses the discretization size for the approximation is given in Figure 3.9. In addition, the specific values of δ_i and $\eta_{i.j}$ are given for the case where $n = 30$ in Table 3.1. For $n = 30$ we compute $C = 1.9153104$.



(a)



(b)

Figure 3.8: Differential equation approximations. (a) The approximation of $f(x)$ and (b) the approximation of $f'(x)$ for $n = 10000$ which minimizes C. The corresponding value of C is 1.90079728.

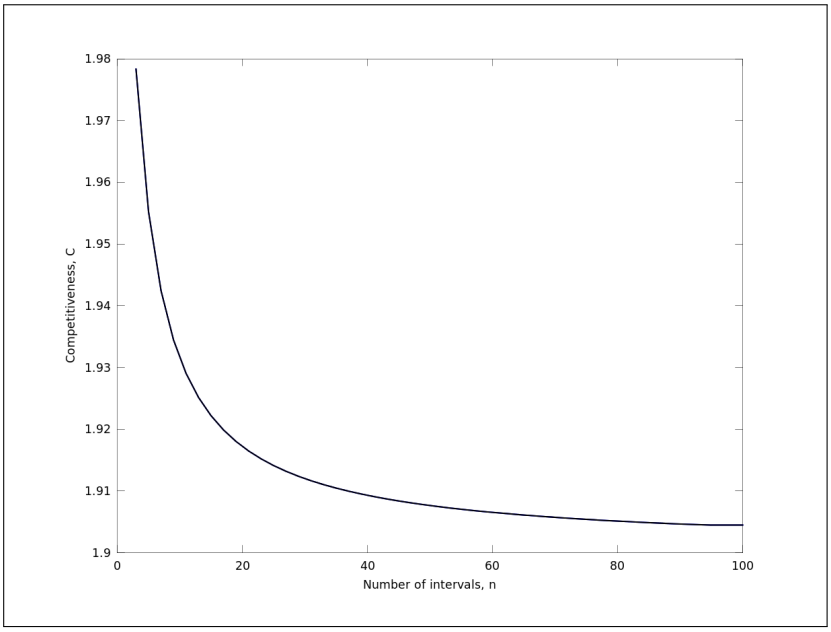


Figure 3.9: Competitiveness vs. discretization size. The competitiveness C as determined by the approximated solution of the differential equation D is plotted each value of n. It can be see that as n increases, the competitiveness of our algorithm decreases.

i	δ_i	$\eta_{i,0}$	$\eta_{n+i,0}$	$\eta_{i,1}$
0	0.0000	0.0000	84.917	57.459
1	1.5826	1.4174	85.917	56.459
2	2.8970	3.1030	86.917	55.459
3	4.0217	4.9783	87.917	54.459
4	5.0035	6.9965	88.917	53.459
5	5.8723	9.1277	89.917	52.459
6	6.6488	11.351	90.917	51.459
7	7.3475	13.653	91.917	50.459
8	7.9793	16.021	92.917	49.459
9	8.5523	18.448	93.917	48.459
10	9.0729	20.927	94.917	47.459
11	9.5461	23.454	95.917	46.459
12	9.9757	26.024	96.917	45.459
13	10.365	28.635	97.917	44.459
14	10.716	31.284	98.917	43.459
15	11.050	33.950	99.917	42.459
16	11.366	36.634	100.92	41.459
17	11.649	39.351	101.92	40.459
18	11.901	42.099	102.92	39.459
19	12.121	44.879	103.92	38.459
20	12.311	47.689	104.92	37.459
21	12.471	50.529	105.92	36.459
22	12.600	53.400	106.92	35.459
23	12.698	56.302	107.92	34.459
24	12.764	59.236	108.92	33.459
25	12.796	62.204	109.92	32.459
26	12.793	65.207	110.92	31.459
27	12.751	68.249	111.92	30.459
28	12.667	71.333	112.92	29.459
29	12.534	74.466	113.92	28.459
30	5.0830	84.917	114.92	27.459

Table 3.1: A list of all coefficients $\eta_{i,j}$ that are used to define an algorithm for the $(60, 30)$-server problem. In this case $C = 1.9153104$.

3.8 Future work

There are several possible avenues for carrying this research further. Here we have developed an algorithm for the $(2n, n)$-server problem on the line and by Theorems 2.4.1, 2.4.3, and 2.4.2 we have a competitive randomized algorithm for the 2-server problem on the line. These theorems may be extended in a straightforward way with analogous results for the (kn, n)-server problem on the line. Therefore, if we are able to find a competitive algorithm for the (kn, n)-server problem for some $k > 2$ we might possibly improve current results for the k-server problem on the line.

Yet another possibility is to generalize our approach for the $(2n, n)$-server problem for other metric spaces. In fact, preliminary investigation indicates that the system of inequalities $\mathbb{S}$ is also sufficient to show that our algorithm will work for continuous tree metric spaces. This is possible using the same potential function and a more general notion of isolation indices. In addition, some effort has been made to extend R–LINE to other metric spaces including the circle and the Manhattan plane. This line of research has been so far unsuccessful using the same potential function extended to the Manhattan plane in the most obvious way. However, it is conceivable that an appropriately defined potential function would allow for improved results in this area.

Of course, there is also the open problem of the k-server conjecture–that there is a k-competitive algorithm for the server problem in any metric space. Even showing that the 2-server problem is 2-competitive in an arbitrary metric space would be a significant breakthrough in the theory of online algorithms.

There is also a possible direction of future research that is interesting outside the k-server problem. In Section 3.6 we derived a differential equation with a reflected argument. This type of differential equation has been studied, but only results on uniqueness and existence of solution have been found. Our attempts to solve our differential equation analytically were unsuccessful. An analytic solution to this differential equation would provide the exact competitiveness of R–LINE, but would also be an interesting result in the field of differential equations.

Appendix A

Differential Equation Approximation Code Listing

```
clc;
clear all;
close all;
k = 1;
%N = [3, 5,10, 15, 20, 30, 50, 100]%, 200, 400, 800, 1200, 2000]
%N = [3:2:59,65:6:100];
N = 9999;
for n = N

  plot(1,1);
  title(num2str(n));
  drawnow;
  d = 2/n;
  t = [-1:d:1];

  minC = inf;

  for i = 0.10:0.02:0.30

  f = zeros(1,length(t));

  fprime = f;


  slope = -1/(1+i) + 1;

  f(tequals(t,d/2)) = i + slope*d/2;
  f(tequals(t,-d/2)) = i - slope*d/2;


  fprime(tequals(t,d/2)) = f(tequals(t,-d/2))/(d/2 + 1 + f(tequals(t,-d/2)));
  fprime(tequals(t,-d/2)) = f(tequals(t,d/2))/(-d/2 + 1 + f(tequals(t,d/2)));

  for tk = d/2:d:1-d
    %[t, tk+d, -tk-d]
```

```
        f(tequals(t,tk+d)) = fprime(tequals(t,tk))*d+f(tequals(t,tk));
        f(tequals(t,-tk-d)) = -fprime(tequals(t,-tk))*d+f(tequals(t,-tk));

        fprime(tequals(t,tk+d)) = f(tequals(t,-tk-d)) / ...
                                  (tk+d + 1 + f(tequals(t,-tk-d)));
        fprime(tequals(t,-tk-d)) = f(tequals(t,tk+d)) / ...
                                   (-tk-d + 1 + f(tequals(t,tk+d)));
    end

    eps_from_f = (n+1)*f;
    delta_n = eps_from_f(1)

    possibleC = 2 - delta_n/(2*(n+1));

    if(possibleC < minC)
        minC  = possibleC;
        fmin = f;
        fprimemin = fprime;
        mini = i;
    end

    hold on;

    end

    C(k) = minC;
    k += 1;

end

% use substitutions to compute all isolation index coefficients
delta = n*(2-C);
e0 = -2*delta;
en = 0;
eps = [e0 n*fmin en];
deltas = eps + 2*delta;
deltas = deltas(2:end);
deltas(1) = 0;
is = 0:n+1;
etas = 3*is - deltas;
etasnpi = etas(end) + is;
etaai =  etasnpi(end)/2 - is;

% create a chart of thee values
CHART = [is' deltas' etas' etasnpi' etaai'];

alletai = [etas etasnpi(2:end)])

%Determine if system S is satisfied
Ssatisfied = prod(1.0*(etaai - etas <= (n+2)*C)) & ...
             prod(1.0*(etaai(2:end) + 1 <= etaai(1:end-1))) & ...
             prod(1.0*(alletai(1:end-1) + 1 <= alletai(2:end)));
```

Listing A.1: Octave code to approximate reflective differential equation.

Bibliography

[Bar08] E. N. Barron. *Game Theory: An Introduction.* John Wiley & Sons, New Jersey, 2008.

[BBM01] Yair Bartal, Béla Bollobás, and Manor Mendel. A Ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In *Proc. 42st Symp. Foundations of Computer Science (FOCS)*, pages 396–405. IEEE, 2001.

[BBMN11] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Proc. 52nd Symp. Foundations of Computer Science (FOCS)*. 10 pages. IEEE Computer Society, 2011.

[BCL98] Yair Bartal, Marek Chrobak, and Lawrence L. Larmore. A randomized algorithm for two servers on the line. In *Proc. 6th European Symp. on Algorithms (ESA)*, Lecture Notes in Comput. Sci., pages 247–258. Springer, 1998.

[BCL02] Wolfgang Bein, Marek Chrobak, and Lawrence L. Larmore. The 3-server problem in the plane. *Theoret. Comput. Sci.*, 287:387–391, 2002.

[BD92] Hans-Jürgen Bandelt and Andreas Dress. A canonical decomposition theory for metrics on a finite set. *Adv. Math.*, 92:47–105, 1992.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[BIK08] Wolfgang Bein, Kazuo Iwama, and Jun Kawahara. Randomized competitive analysis for two-server problems. *Algorithms*, 1:30 – 42, 2008.

[BIK+11] Wolfgang Bein, Kazuo Iwama, Jun Kawahara, Lawrence L. Larmore, and James A. Oravec. A randomized algorithm for two servers in cross polytope spaces. *Theoretical Computer Science*, 412(2):563–572, 2011.

[BLNR11] Wolfgang Bein, Lawrence Larmore, John Noga, and Rüdiger Reischuk. Knowledge state algorithms. *Algorithmica*, 60(3):653–678, 2011.

[CDRS90] Don Coppersmith, Peter G. Doyle, Prabhakar Raghavan, and Marc Snir. Random walks on weighted graphs and applications to online algorithms. In *Proc. 22nd Symp. Theory of Computing (STOC)*, pages 369–378. ACM, 1990.

[CL91] Marek Chrobak and Lawrence L. Larmore. An optimal online algorithm for k servers on trees. *SIAM J. Comput.*, 20:144–148, 1991.

[CLLR97] Marek Chrobak, Lawrence L. Larmore, Carsten Lund, and Nick Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Inform. Process. Lett.*, 63:79–83, 1997.

[KP94] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. In *Proc. 35th Symp. Foundations of Computer Science (FOCS)*, pages 394–400. IEEE, 1994.

[Lar72] Richard C. Larson. *Urban Police Patrol Analysis*. Kingsport Press, 1972.

[MMS90] Mark Manasse, Lyle A. McGeoch, and Daniel Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11:208–230, 1990.

[Rob72] Muril L. Robertson. The equation y'(x) = f(t, y(g(t))). *Pacific Journal of Mathematics*, 42:483–491, 1972.

Vita

Graduate College
University of Nevada, Las Vegas

Lucas A. Bang

Degrees:

Bachelor of Arts in Computer Science 2010
University of Nevada Las Vegas

Bachelor of Science in Mathematics 2010
University of Nevada Las Vegas

Thesis Title: An Online Algorithm for the 2-Server Problem on the Line with Improved Competitiveness

Thesis Examination Committee:

Chairperson, Professor Lawrence Larmore
Committee Member, Professor Wolfgang Bein
Committee Member, Professor Matt Pedersen
Graduate Faculty Representative, Professor Ebrahim Salehi