UNLV | UNIVERSITY LIBRARIES

UNLV Retrospective Theses & Dissertations

1-1-1999

Trackless online two-server problems and red-black games

Anna N Naydenova
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation

Naydenova, Anna N, "Trackless online two-server problems and red-black games" (1999). *UNLV Retrospective Theses & Dissertations*. 1065.
http://dx.doi.org/10.25669/3u4f-oikm

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TRACKLESS ONLINE TWO SERVER PROBLEMS AND RED BLACK GAMES

by

Anna N. Naydenova

Bachelor of Science
University of Nevada, Las Vegas
1997

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree
Department of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 1999

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 1397971

Copyright 2000 by
Naydenova, Anna N.

All rights reserved.

UMI®

UMI Microform 1397971
Copyright 2000 by Bell & Howell Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Copyright by Anna N. Naydenova 2000
All Rights Reserved

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UNLV

Thesis Approval
The Graduate College
University of Nevada, Las Vegas

7/12/ , 19 99

The Thesis prepared by

ANNA N NAYDENOVA

Entitled

TRACKLESS ONLINE TWO SERVER PROBLEMS AND RED-BLACK GAMES

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ABSTRACT

Trackless Online 2-server Problems

and Red-Black Games

by

Anna N. Naydenova

Dr. Wolfgang W. Bein, Examination Committee Chair
Assistant Professor of Computer Science
University of Nevada, Las Vegas

The online 2-server problem presents a number of challenges in the search for simple competitive algorithms for solving it. Finding the optimal off-line solution involves costly dynamic programming. Looking for more efficient algorithms, researchers have studied how restriction on the input information given to the algorithm affects its competitiveness. One such restriction is tracklessness. Trackless algorithms for the 2-server problem include many known server algorithms including BALANCE_SLACK and some paging algorithms. It is demonstrated that the trackless 2-server optimization problem has a deterministic lower bound of $\frac{23}{11} > 2$ for competitiveness, thus proving that tracklessness is a significant restriction. The optimally competitive online non-trackless algorithm for the 2-server problem is 2-competitive. Other current research on the topic is also discussed.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE OF CONTENTS

ABSTRACT iii

LIST OF TABLES vi

LIST OF FIGURES vii

ACKNOWLEDGMENTS viii

CHAPTER 1 INTRODUCTION 1
Off-line Computation 1
Online Computation 1
Optimization Problem 2
Competitiveness 3
The k-server Problem 3
Background for the k-server Problem 4

CHAPTER 2 SERVER PROBLEMS AND WORK FUNCTIONS 7
The 2-server Problem 7
The Work Function Algorithm 16

CHAPTER 3 TRACKLESSNESS AND THE TRACKLESS 2-SERVER PROBLEM 23
What is Tracklessness? 23
The Trackless 2-server Problem 24
Example of an Online 2-server Algorithm 25
BALANCE_SLACK Algorithm 27
BALANCE Algorithm 29
HARMONIC Algorithm 30
Paging Algorithms 31

CHAPTER 4 PROOF OF COMPETITIVENESS FOR THE TRACKLESS 2-SERVER PROBLEM 32
Two Preliminary Lemmas 32
Proof of Lower Bound for Competitiveness 35

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 5 THE TRACKLESS WORK FUNCTION AND RED-BLACK GRAPHS 43
The Trackless Work Function 43
Red-Black Graphs 51

REFERENCES 54

VITA 56

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF TABLES

Table 2.1: Optimal Service for r_1 9
Table 2.2: Optimal Service for sequence r_1, r_2 10
Table 2.3: Costs Resulting from servicing r_3 12
Table 2.4: Optimal costs for all configurations after r_4 has been serviced 13
Table 2.5: Configurations optimal costs after servicing r_5 14
Table 2.6: Determining the optimal cost for the entire sequence 15
Table 2.7: Work function values for the servers in Example 2.2 18
Table 5.1: Trackless Work Function τ 46
Table 5.2: Function $\upsilon = \tau((1,2))$ 47
Table 5.3: $\upsilon' = \upsilon - b$ after offset 48
Table 5.4: Trackless work function υ^r 50
Table 5.5: Function υ^s 51

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF FIGURES

Figure 2.1: Metric space M for dynamic programming example 8
Figure 2.2: Achieving a configuration containing r_2 10
Figure 2.3: Service from configuration $\{r_2, p_1\}$ to configuration $\{r_3, p_3\}$ 11
Figure 2.4: Service of configuration $\{r_4, p_7\}$ from configuration $\{r_3, p_4\}$ 12
Figure 2.5: Service of configuration $\{r_4, p_8\}$ to configuration $\{r_5, p_2\}$ 13
Figure 2.6: Final configuration and optimal solution 15
Figure 2.7: Metric space for Work Function Algorithm example 17
Figure 3.1: Illustration of tracklessness 23
Figure 3.2: Illustration of Move Closest Server algorithm 25
Figure 3.3: BALANCE_SLACK Algorithm illustration 28
Figure 3.4: Two steps of the BALANCE algorithm 29
Figure 4.1: Illustration of the Forcing Lemma 33
Figure 4.2: Metric space M used in proof of lower bound for competitiveness 35
Figure 4.3: A mapping of M onto a torus 36
Figure 5.1: Triangle Inequality 49
Figure 5.2: Red-black graph representation of one step of TWF 52
Figure 5.3: Illustration of critical cycle in red-black graph 53

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEGDMENTS

First and foremost I want to thank my family – Mom, Dad, Lubo, Zimbee, and Ike. They have always been there for me. Even when I decided to go into Computer Science instead of business. You are the best!

UNLV and the Department of Computer Science in particular have played an influential part of my development providing me with opportunities to teach, research, and learn. To all professors who taught me or worked with me – thank you!

This thesis would not have been possible without the hard work and help of my advisor, the chair of my examination committee, Dr. Wolfgang W. Bein.

His collaborator and a person I began to think of as my bonus advisor, Dr. Lawrence L. Larmore, spent hours proofreading and explaining the sometimes horrendous concepts.

Thanks go out to Dr. Laxmi P. Gewali, a member of my examination committee, who thinks of me as his product. There is a basis for this: I have taken every single class he teaches at UNLV.

Dr. John Wang is the final member of the examination committee. I am indebted to him because he joined the committee on very short notice.

I appreciate the readiness of Dr. George Miel to support this project and regret he did not remain a member of my committee due to scheduling conflicts.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 1

INTRODUCTION

Much research is concerned with situations where all input information is supplied to an algorithm prior to its execution. The algorithm then produces output based on the complete knowledge available to it. Such problems are known in computation as off-line problems.

Off-line Computation

Given a set of inputs $x^1, ..., x^n$, an algorithm produces an output

$$y = F(x^1, ..., x^n),$$

such that the entire set of inputs is available prior to the algorithm's execution.

Online Computation

In practice, the off-line model might not be realistic as data might only become available during computation. For example, paging algorithms must evict pages according to some rule, without knowledge of future paging requests. Such algorithms are commonly known as online algorithms. More formally, online computation can be defined as follows:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Given a sequence of inputs $x^1, ..., x^n$, an online algorithm produces a sequence of outputs $y^1, ..., y^n$, such that input x^t is not available to the algorithm prior to step t in its execution sequence and thus

$$y^t = F^t (x^1, ..., x^t, y^1, ..., y^{t-1}),$$

where F^t is some function, $x^1, ..., x^t$ are the available inputs, and $y^1, ..., y^{t-1}$ are the outputs produced in the previous t-1 steps, i.e. the current output y^t is a function of the first t inputs from the input sequence and all previously calculated outputs. A problem fitting the above description is known as an online problem.

While the two prior definitions seem quite different, it is easy to show that the off-line problem is a special case of the online computation problem. If there is only one time step in the execution of an online problem, all input becomes available at that time step and thus the algorithm produces an output, which is a function of all input. Upon closer examination we observe that this result matches the definition of the off-line problem.

Online problems are generally more difficult than their corresponding off-line problems. This fact becomes apparent when we phrase our problems in terms of optimization.

Optimization Problem

An instance I of an optimization problem is a pair (P, c), with $c: P \rightarrow R^+$, where P is any set. The problem is to find p in P such that $c(p) < c(y)$ for all y in P. The optimization problem is the collection of all instances and is denoted by **P.**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Given an optimization problem **P** with instance I, let $cost_{opt}(I)$ be the optimal cost for instance I. We assume the instance I is described in terms of $x^1, ..., x^n$ as an input to algorithm A, which will in turn compute a solution given as output $y^1, ..., y^n$, with associated cost $cost_A(I)$.

For many online problems, no online algorithm which computes the optimal solution to every instance of **P** exists. The performance of an online algorithm is measured by how close it is to optimal. Toward that end, the concept of competitiveness is defined. Basically, competitiveness is the ratio of the algorithm cost to the optimal cost.

Competitiveness

Let A be an online algorithm for solving **P**. The online algorithm A is C-competitive if for any instance I of **P** $cost_A(I) \leq C * cost_{opt}(I) + b$, where b is a constant independent of I. An algorithm A is called competitive if it attains a constant competitive ratio C. The infimum over the set of all values C such that A is C-competitive is called the competitive ratio of A.

Now we turn our attention to a specific optimization problem, the k-server Problem.

The k-server Problem

Given a metric space M, an online algorithm manages k mobile servers, each of which resides at one point in the metric space at any given time. The algorithm's input is a sequence of requests $\rho = r_1, ..., r_n$, where each r_i is a point in M. To serve a request, the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

algorithm needs to move a server to the request point unless there already is a server at that point. Whenever a server is moved, the distance that the server has moved is incurred as a cost.

Background for the k-server Problem

The k-server problem was formulated in 1988 by Manasse, McGeoch, and Sleator [19], as a natural abstraction of the paging problem. Soon researchers realized that the server problem was important to the field of competitive analysis.

Manasse, McGeoch, and Sleator's early publications established some bounds for the competitiveness of the server problem. They were able to prove a deterministic lower bound for the competitiveness of online server algorithms with k servers in terms of off-line algorithms with h servers:

$$C = \frac{k}{k-h+1},$$

where $h \leq k$ and the space of the problem contains at least k+1 points. Two other important results were the establishment of the 2-competitiveness of the online 2-server problem in an arbitrary metric space and the k-competitiveness of the k-server problem in a space containing k + 1 points.

Another significant contribution by Manasse, McGeoch, and Sleater is the posing of the k-server conjecture: In any space there exists a deterministic online algorithm for the k-server problem, which is k-competitive. This conjecture holds in a uniform metric space (paging problems), for the 2-server problem, and for any space containing k + 1 points, where k is the number of servers. This conjecture contributed to the interest in the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

server problem, but for some time progress was made only for special cases and not for general algorithms.

An important step in the research of the server problem was the proving of the "weak" k-server conjecture around 1991. Fiat, Rabani, and Ravid [11] constructed a deterministic algorithm for the k-server problem which is $O((k!)^3)$-competitive, thus proving that there is a fixed function of k that bounds the competitive ratio (competitiveness) of every server system. This was the first upper bound shown for the server problem.

The upper bound was improved in subsequent years by Grove [14] using deterministic versions of randomized algorithms. Later, in 1994, Koutsoupias and Papadimitriou [16] established the upper bound as (2k - 1)-competitive for k servers in any metric space. This is the best currently known upper bound. Koutsoupias and Papadimitriou have also proven that the k-server problem is k-competitive for any metric space consisting of k + 2 points.

In 1991 Chrobak and Larmore [9] formally defined the Work Function Algorithm (WFA) discussed later in Chapter 2. It is an algorithm based on the use of the optimal cost up to the current request to make a decision regarding service of the request. This algorithm was given form after other researchers came with algorithms using the same concept but resulting in larger competitive ratios. WFA was proven to be 2-competitive for 2 servers. Chrobak and Larmore also proved that the work function algorithm is k-competitive for k servers against a lazy adversary, which informs the algorithm whenever their configurations are matched. Several researchers have shown independently that WFA is k-competitive for k servers on a line.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This paper discusses algorithms for the online 2-server optimization problem. Some known algorithms are reviewed and the results from the trackless input restriction on this class of algorithms restriction is described in detail.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 2

SERVER PROBLEMS AND WORK FUNCTIONS

This chapter introduces the online 2-server optimization problem, describes the optimal off-line algorithm for the problem, and outlines the proofs of competitiveness for some online algorithms for the 2-server problem.

The 2-server Problem

Let M be a metric space in which there are 2 mobile servers that can occupy points of M. At each time step, an algorithm A is given a request specified by a location in M and A must choose which one of the two servers will move to the point of the request. This move constitutes servicing of the request. The measure of cost for A is the total distance the servers have to travel to service a finite sequence of requests. The objective is to choose A to minimize costs. Moreover, the requests must be served online; i.e. the 2-server problem is an online optimization problem.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Optimal Solution for the 2-server Problem

The optimal off-line solution to the 2-server problem can be obtained using dynamic programming. Although dynamic programming may take a relatively large amount of space and memory compared to other algorithms, it represents a thorough analysis of the problem and guarantees an optimal solution.

Example 2.1: Optimal solution for a two sever problem using dynamic programming

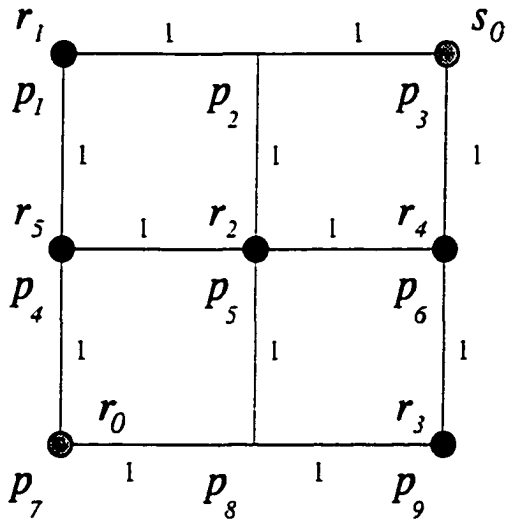

Figure 2.1: Metric space M for the dynamic programming example

Given a metric space M consisting of 9 points, p_1, ..., p_9, as shown on Figure 2.1, and a sequence of 5 requests $\rho = \{r_1, r_2, r_3, r_4, r_5 = p_1, p_5, p_9, p_6, p_4\}$, the optimal service is calculated with the help of the following current configurations and tables representing the minimization process. It is a common convention that the server, which served the most recent request, is referred to as the r server. The other server is referred to as the s server.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Thus a string of r's and s's will represent the optimal service. Initially, the servers are named arbitrarily. For this example, r_0 is at point p_7 and s_0 is at p_3, thus the initial configuration is $\{p_3, p_7\}$.

At the first step in the execution r_1 must be serviced by one of the two servers. The dynamic programming model calculates the cost for every possible configuration of the two servers after the request has been serviced. A service constitutes a move by one of the servers to the request point r_1. The other server can be at any point in M. The results of the calculations are shown in Table 2.1, assuming that the numbers represent the best possible cost for the configuration. The server, which served the request to achieve that cost, is shown in parentheses next to that cost. If moving either server results in the same cost, the server names are omitted.

Table 2.1: Optimal Service for r_1

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
opt	4	3	2(r)	3(s)	4	3	2(s)	3(s)	4

One of the next possible configurations is shown on Figure 2.2 and the results are summarized in Table 2.2. The cost is cumulative, i.e. newly incurred costs are added to the optimal costs from Table 2.1. The new table is 2-dimensional because it calculates costs from a given server configuration (r at the previous request point, s at any point in M) to a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

final configuration in which one of the servers has serviced the current request (in this case r_2) and the other server can be at any point in M.

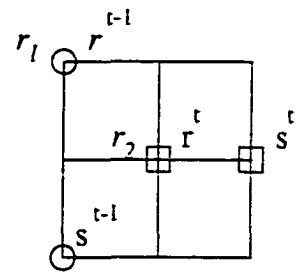

Figure 2.2: Achieving a configuration containing r_2

Table 2.2: Optimal service for sequence r_1, r_2

	p_1	P_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	6	7	8	7	8	9	8	9	10
p_2	4(s)	5(r)	6	5(s)	6	7	6(s)	7	8
p_3	4(s)	5	4(r)	5(s)	6	5(s)	6(s)	7	6(r)
p_4	4(s)	5(s)	6(s)	5	6	7	6	7	8
p_5	4(s)	5(s)	6(s)	5(s)	6	7(s)	6(s)	7(s)	8(s)
p_6	4(s)	5(s)	6	5(s)	6	5(r)	6(s)	7	6(r)
p_7	4	5(s)	6(s)	5	6	7	4(r)	5(r)	6(r)
p_8	4(s)	5(s)	6(s)	5(s)	6	7	6	5(r)	6(r)
p_9	6(s)	7(s)	8	7(s)	8	7(r)	8	7(r)	6(r)
opt	4_2(s)	5_2(r)	4_3(r)	5_2(s)	6_2(r)	5_3(s)	4_7(r)	5_7(r)	6_3(r)

In Table 2.2, if the server configuration is $\{p_1, p_7\}$, meaning one of the servers has just serviced r_1, and the other server is at p_7, the cost incurred to move the servers to the new

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

configuration $\{p_5, p_6\}$ is 4. Adding the cost to achieve $\{p_1, p_7\}$ from the initial configuration, the cumulative cost of servicing the request sequence r_1, r_2 is 4 + 3 = 7.

The subscripts in the opt row of the table show which prior configuration yielded the optimal result for the current configuration.

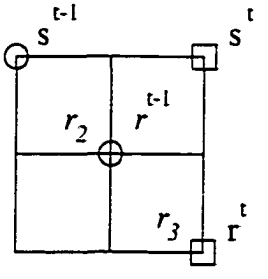

Figure 2.3: Service from configuration $\{r_2, p_1\}$ to configuration $\{r_3, p_3\}$

The service of request r_3 is calculated based on the optimal service of r_2. Again, a sample previous and a sought current configuration are shown on Figure 2.3. The results of the complete calculations are in Table 2.3. The optimal cost for servicing r_3 with the above restrictions is 4 + 4 = 8, where the optimal cost for configuration $\{r_2, p_1\}$ is taken from Table 2.2 and added to the optimal cost to move from previous configuration $\{r_2, p_1\}$ to the current configuration $\{r_3, p_3\}$. Each move between two configurations is represented by one cell of the dynamic programming table for the current time step.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 2.3: Costs resulting from servicing r_3

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	6(r)	7(r)	8(r)	7(r)	8	9	8(r)	9	10
p_2	8(r)	7(r)	8(r)	9	8	9	10	9	10
p_3	8	7(r)	6(r)	7(s)	6(s)	7	8(s)	7(s)	8
p_4	8(s)	9(r)	10	7(r)	8	9	8(r)	9	10
p_5	10	9	10	8	8	8	10	9	10
p_6	8(s)	7(s)	8	7(s)	6(s)	7(r)	8(s)	7(s)	8
p_7	8	7(s)	8(s)	7	6(s)	7(s)	6(r)	7	8
p_8	8(s)	7(s)	8(s)	9	6(s)	7(s)	8	7(r)	8
p_9	8(s)	7(s)	8(s)	7(s)	6(s)	7(s)	8(s)	7(s)	8
opt	6_1(r)	7_1(r)	6_3(r)	7_1(r)	6_3(s)	7_3(r)	6_7(r)	7_3(s)	8_3(r)

The configuration and optimal solutions for all configurations after serving the sequence r_1, ..., r_4 follow in Figure 2.4 and Table 2.4.

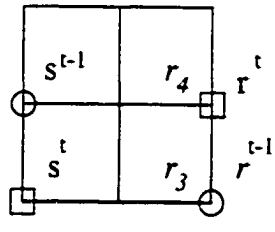

Figure 2.4: Service of configuration $\{r_4, p_7\}$ from configuration $\{r_3, p_4\}$

For the configurations of Figure 2.4, the optimal total cost of servicing the request sequence is the cost for servicing the previous configuration from Table 2.3 plus any

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

newly incurred costs of servicing r_4 and achieving the current configuration. This cost is $7+2=9$.

Table 2.4: Optimal costs for all configurations after r_4 has been serviced

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	7(r)	8(r)	9(r)	8(r)	9(r)	10	9(r)	10	9(s)
p_2	9(r)	8(r)	9(r)	10	9(r)	10	11	10	9(s)
p_3	9(r)	8(r)	7(r)	10	9	8	9(s)	8(s)	7(s)
p_4	9(r)	10(r)	11	8(r)	9(r)	10	9(r)	10	9(s)
p_5	9(r)	8(r)	9	8(r)	7(r)	8	9	8	7(s)
p_6	11(s)	10(s)	9(r)	10	9(s)	8	9(s)	8(s)	7(s)
p_7	9(r)	10(r)	11	8(r)	9(r)	10	7(r)	8(r)	9(s)
p_8	11(r)	10(r)	11	10	9(r)	10	9(r)	8(r)	9(s)
p_9	13	12	11	11	11	10	10	10(s)	9
opt	7_1(r)	8_1(r)	7_3(r)	8_1(r)	7_5(r)	8_3(r)	7_7(r)	8_3(s)	7_3(s)

Servicing the last request, r_5, results in the solutions in Table 2.5. A sample configuration is shown on Figure 2.5.

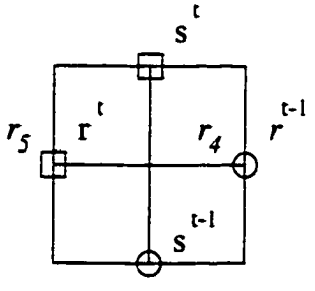

Figure 2.5: Service from configuration $\{r_4, p_8\}$ to configuration $\{r_5, p_2\}$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For the current configuration on Figure 2.5 the cost is 8 + 4 = 12 which corresponds to cell [8, 2] of Table 2.5.

Table 2.5: Configurations optimal costs after servicing r_5

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	9(r)	10	9	10	9	8(s)	11	10	9
p_2	11(r)	10	11	12	11	10	13	12	11
p_3	11(r)	10	9	12	11	10	13	12	11
p_4	11(s)	10	9	10	9	8(s)	11	10	9
p_5	11	10	9	10	9	8(s)	11	10	9
p_6	13	12	11	12	11	10	13	12	11
p_7	11	10	9	10	9	8(s)	9	10	9
p_8	13	12	11	12	11	10	11	10	11
p_9	13	12	11	12	11	10	11	10	9
opt	9_1(r)	10_1(r)	9_1(r)	10_1(r)	9_1(r)	8_1(s)	9_7(r)	10_1(r)	9_1(r)

To obtain the final result, all opt rows from the above tables are recorded into a new table (see Table 2.6). The optimal service is selected based on the optimal cost for each row. Ties are broken arbitrarily. In this case the first occurrence of the best cost is selected and the r server moves. The first request is serviced by r_0 = r, which moves from point p_7 to point p_1. After the service r_0 remains the r server, s_0, the s server. The names of the servers are updated after each service.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 2.6: Determining the optimal cost for the entire sequence

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
r_1	4	3	2(r)	3	4	3	2	3	4
r_2	4(s)	5	4	5	6	5	4	5	6
r_3	6(r)	6	6	7	6	7	6	7	8
r_4	7(r)	8	7	8	7	8	7	8	7
r_5	9	10	9	10	9	8(s)	9	10	9

The selections for service in Table 2.6 yield the final configuration shown on Figure 2.6.

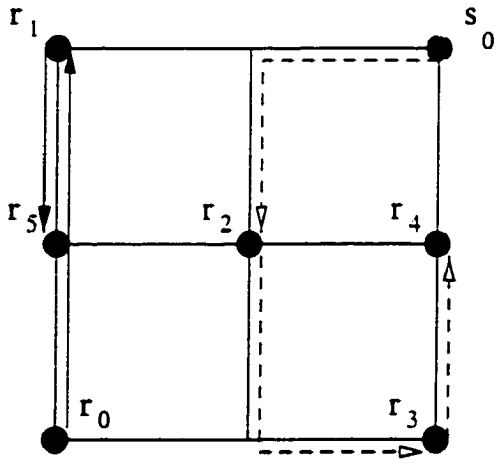

Figure 2.6: Final configuration and optimal solution

Note that the solution is not unique.

The run time for the dynamic programming algorithm is $O(n^3)$, where n is the number of requests. The memory requirement is $O(n^2)$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Work Function Algorithm

As previously mentioned, a number of researchers [11, 14, 16] used similar approaches in search for an optimally competitive solution to the server problem, but the Chrobak and Larmore [9] model and name for the Work Function Algorithm are the commonly accepted in the field.

Dynamic programming enables the calculation of the optimal off-line solution to the off-line server problem. The Work Function Algorithm (WFA) uses a similar approach, but is constrained by the online information flow. For the online 2-server problem the competitiveness of the work function algorithm is proven to be 2, which is optimally competitive [9].

Given a metric space M, two servers, s and r, and a request sequence $\rho = r^1, ..., r^n$, let $\omega_n(p)$ be the minimum cost for two servers starting at points r_0 and s_0 in M to service ρ and achieve final configuration in which one of the servers is at r^n and the other is at point p in M. At each time step t, updates are calculated in the following manner (ω_t denotes the updated value):

$$\omega_t(x) = \min \{\text{dist}(r^{t-1}, r^t) + \omega_{t-1}(x), \text{dist}(r^{t-1}, x) + \omega_{t-1}(r^t)\}.$$

A simple example follows, after which Example 2.1 is discussed within the scope of WFA. The metric space and the request sequence for the example are taken from Borodin, El-Yaniv [5].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Example 2.2: Calculating the Work Function values for a 2-server Problem

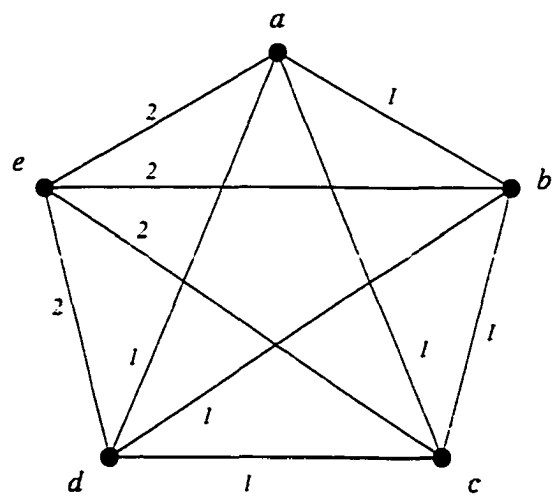

Figure 2.7: Metric space for Work Function Algorithm example

Consider the metric space represented by the complete weighted graph shown on Figure 2.7. All edge weights are 1 except those of the edges incident on node e. The two servers are initially located at nodes a and b.

The request sequence to be serviced is ρ = e, d, a, b, c, a, b, a, c, e. Table 2.7 shows the values of work functions corresponding to all 2-node configurations and all prefixes of the above request sequence. The table contains a column for each of C (5, 2) = 10 possible configurations. The first row gives the values of the initial work functions corresponding to an empty request sequence based on initial configuration ab. Subsequently, row i shows the values of the work functions after the i-th request has been serviced.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The update for one of the work functions is shown below. The results of all updates for all work functions are shown in Table 2.7. Consider request d (i=2). Here is the update for the work function from configuration bc, or ω_d (ac):

$$\omega_d (ac) = \min \{\omega_e (ac - x + d) + dist (d, x)\} =$$

$$= \min \{\omega_e (cd) + dist (d, a), \omega_e (ad) + dist (d, c)\} =$$

$$= \min \{5 + 1, 4 + 1\} =$$

$$= \min \{6, 5\} =$$

$$= 5.$$

Table 2.7: Work function values for the servers in Work Function Example

	i	ab	ac	ad	ae	bc	bd	be	cd	ce	de
0	0	0	1	1	2	1	1	2	2	3	3
e	1	4	4	4	2	4	4	2	5	3	3
d	2	5	5	4	4	5	4	4	5	4	3
a	3	5	5	4	4	6	5	5	5	5	5
b	4	5	6	6	6	6	5	5	6	6	6
c	5	7	6	7	7	6	7	7	6	6	7
a	6	7	6	7	7	7	8	8	7	8	8
b	7	7	8	8	9	7	8	8	8	9	9
a	8	7	8	8	9	8	8	9	9	10	10
c	9	9	8	9	10	8	9	10	9	10	11
e	10	11	11	11	10	11	11	10	11	10	11

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

At each request the Work Function Algorithm calculates another row of Table 2.7 and then uses this information to decide on the move. Specifically, the algorithm selects a server in such a way that a combination of the work function value of the resulting configuration and the movement of the servers in minimized, i.e.

$$s = \arg\min \{\omega_t(C - x + r^t) + \text{dist}(x, r^t)\},$$

where s is the server which moves, C is the configuration which resulted from the service of r^{t-1}, and x is a point in that configuration.

For example, using the s and r server notation and assuming that the s server is at point b and the r server is at point a, to service request e, the algorithm compares $\omega_0(ae) + \text{dist}(b, e) = 2$ and $\omega_0(be) + \text{dist}(a, e) = 2$. The two values are equal and, in general, ties are broken arbitrarily. In this example, when a tie occurs, the r server moves to the request point. Thus at this step r is selected to service e. The new configuration of the servers is r at e, s at b. The next request in the sequence is d. WFA looks at $\omega_e(bd) + \text{dist}(d, e) = 6$ and $\omega_e(de) + \text{dist}(b, d) = 4$. Clearly, it is more beneficial to service with the s server, since the sum of the service cost and the value of the corresponding work function is lower. Continuing in the same manner, the resulting service from Table 2.7 is the string α = r s r r r s r s r r. The service cost is calculated to be 10.

Now consider Example 2.1. Recall that the initial configuration is $\{p_3, p_7\}$ and the request sequence is $\rho = \{p_1, p_5, p_9, p_6, p_4\}$ in a 9-point Manhattan plane. The rows in Table 2.6 are work functions, except that only the values for configurations, which include the request, are listed. The number of all possible configurations is C (9, 2) = 36, but as shown below only configurations containing the request points are needed for the work

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

function algorithm. This simplifies the notation for the work function to $\omega_j(p_i)$ since the other point is understood to be the request point.

To determine which server will service r_1, WFA evaluates and compares $\omega_0(p_3)$ + dist (p_1, p_3) and $\omega_0(p_7)$ + dist (p_1, p_7). This results in the following update:

If $\omega_0(p_3)$ + dist $(p_1, p_7) \leq \omega_0(p_7)$ + dist (p_1, p_3)

server^1 = r.

Otherwise

server^1 = s.

At this step $\omega_0(p_3)$ + dist $(p_1, p_3) = \omega_0(p_7)$ + dist $(p_1, p_7) = 2$, so r serves. The new configuration is $\{r = p_1, s = p_3\}$. At the next step WFA looks at request r_2 located at point p_5. From the previous service the servers are at points p_1 and p_3. Based on this information, column p_1 and column p_3 are considered.

server^2 = arg min $\{\omega_1(p_1)$ + dist $(p_3, p_5), \omega_1(p_3)$ + dist $(p_1, p_5)\}$

= arg min $\{4 + 2, 2 + 2\}$ =

= arg min $\{6, 4\}$ =

= s.

The smaller value is 4, corresponding to service with the s server. The new configuration is $\{r = p_5, s = p_1\}$.

To move a server to the next request point, $r_3 = p_9$, the algorithm looks at

server^3 = arg min $\{\omega_2(p_1)$ + dist $(p_5, p_9), \omega_2(p_5)$ + dist $(p_1, p_9)\}$ =

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$= \arg\min \{4 + 2, 6 + 4\} =$

$= \arg\min \{6, 10\} =$

$= r.$

Obviously, the better choice for WFA is to move the r server to service r_3, resulting in configuration $\{r = p_9, s = p_1\}$. For the new request, $r_4 = p_6$, the values of the work function and costs considered are

$$server^4 = \arg\min \{\omega_3(p_1) + dist(p_6, p_9), \omega_3(p_9) + dist(p_1, p_6)\} =$$

$= \arg\min \{6 + 1, 8 + 3\} =$

$= \arg\min \{7, 11\} =$

$= r.$

The decision is to move the r server again to obtain configuration $\{r = p_6, s = p_1\}$. The last request results in the following calculation:

$$server^5 = \arg\min \{\omega_4(p_1) + dist(p_4, p_6), \omega_4(p_6) + dist(p_1, p_4)\} =$$

$= \arg\min \{7 + 2, 8 + 1\} =$

$= \arg\min \{9, 9\} =$

$= r.$

The r server moves to service r_5. The string denoting the service is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$\alpha = r\,s\,r\,r\,r.$

Notice that the cost for the algorithm to service the request sequence is higher than the previously calculated optimal cost.

WFA is the best known general algorithm for the online k-server problem. For the 2-server problem, WFA is 2-competitive, which is optimal. For k servers it is conjectured that WFA is k-competitive [19], which also would be optimal. However, only an upper bound of (2k - 1) is known, as shown by Papadimitriou and Koutsoupias [16]. The space and time complexity are at least as large as the space and time complexity for dynamic programming. Because of the large overhead for dynamic programming it is desirable to construct "simpler" online algorithms which do not use all the information available, but which maintain an acceptable level of competitiveness.

One of the approaches to achieving this simplification is to impose restrictions on the type of input the algorithm can receive. Research in this area has resulted in the development of the concept of trackless algorithms, which is the subject of the subsequent chapters of this work.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 3

TRACKLESSNESS AND THE TRACKLESS 2-SERVER PROBLEM

What is Tracklessness?

Bein and Larmore [1, 18] introduced the concept of tracklessness. Tracklessness is an input restriction imposed on an algorithm. Input goes through a so-called referee where it is processed. The algorithm only sees the processed version of the input and based on it produces its output. Moreover, in this case the problem is also online.

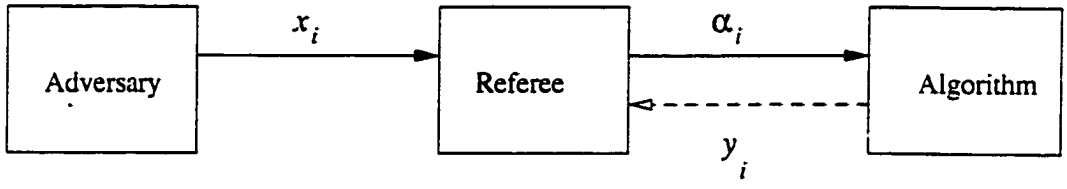

Figure 3.1: Illustration of tracklessness

In the example of Figure 3.1 the input stream is $x_1, ..., x_n$ and is passed from the adversary to the referee one x_i at a time. The referee uses some function of all currently

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

available inputs and all currently produced outputs to process the input after which it passes the new input

$$\alpha_t = A^t(x_1, ..., x_t, y_1, ..., y_{t-1})$$

to the algorithm. A^t is the aforementioned function at step t, y_i's are all outputs produced thus far.

From here the study of the trackless server problems is restricted to lazy algorithms. An algorithm is lazy if it only moves one of its servers in response to a request. This restriction is easily justified because since each request can be serviced by one server, if the algorithm wants to move the other servers to other locations, it can store these locations in memory and only move the servers there when these locations are specifically requested. By the triangle inequality, the total cost incurred with a lazy algorithm is no more than the total distance of any other type of algorithm.

The Trackless 2-server Problem

In the case of the 2-server problem, a trackless algorithm can be described in the following way:

At every time step the algorithm is given a pair of numbers which represent the distances between each server and the current request point [18]. Based on these distances alone the algorithm makes a decision as to which server will serve the request.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Example of an Online 2-server Algorithm

A number of trackless algorithms have been developed to solve the online 2-server problem. Some of these algorithms are competitive, some are not. While some simpler algorithms work well for specific cases, they are not competitive as illustrated by this example of the Move Closest Server algorithm.

Example 3.1: Move Closest Server is not competitive

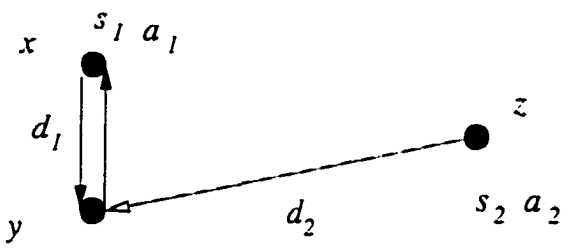

Figure 3.2: Illustration of Move Closest Server algorithm

Given is a configuration of three points, x, y, and z, as shown on Figure 3.2, where the distance between x and y is d_1, dist (x, z) = dist (y, z) = d_2 and d_2 is significantly larger than d_1. The algorithm has server s_1 at x and server s_2 at z. The adversary also has servers at x and z. The adversary can design a sequence of requests such that, under the Move Closest Server strategy, the algorithm will incur an infinite cost while the cost for the adversary is d_2. That sequence consists of alternating requests of x and y. The algorithm always moves s_1 to service the requests. The adversary only needs to move its server from z to y at cost d_2 and all of its subsequent costs are zero.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

On the other hand, Move Closest Server works well for Example 2.1. Refer to Figure 2.1 for the metric space, request sequence and server positions. The first request is r_1 located at point p_1. This information is given to the referee which passes on to the algorithm $\alpha_1 = (2, 2)$ -- the distances from the two servers to r_1. The first number in the pair is associated with the server who has serviced more recently (the r server).

The algorithm responds by indicating which server moved to service the request. In this case the two severs are equidistant to the request point, so it can be chosen arbitrarily which one will service. To be consistent with the Chapter 2 example, let r_0 service at cost 2. It is assumed that servers are again referred to as s and r with the stipulation that the r server has serviced the most recent request.

When r_2 is requested, the algorithm receives pair $\alpha_2 = (2, 2)$. Choose s to service the request at cumulative cost 4. For r_3 the pair is $\alpha_3 = (2, 2)$. Server r moves to the request point and the algorithm cost is 6. Request r_4 comes in and the algorithm receives $\alpha_4 = (1, 2)$ as its input information and services the request with r at cost 7.

Finally, r_5 comes in resulting in $\alpha_5 = (2, 1)$. Upon evaluating this input information, the algorithm chooses to serve the request with the s server thus achieving total cost of 8 for this service.

Note that in Example 3.1 the only information used by the algorithm is the distances between the servers and the current request. This is the main characteristic of a trackless algorithm. It does not have to know where in space the request is located. With this in mind, we list some additional known trackless algorithms.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BALANCE_SLACK Algorithm

The BALANCE_SLACK algorithm [6] is only defined for two servers. There is a slack value associated with each server. Initially, the slack values (e_1 for s_1, e_2 for s_2) are both 0. The slack represents the total cumulative slack work each server has done up to the current time step, where slack work is defined below. As each new request r^t comes in, prospective new slack values are calculated:

$$e_1' = e_1 + \{\text{dist}(s_1, r^t) + \text{dist}(s_1, s_2) - \text{dist}(s_2, r^t)\} / 2$$

$$e_2' = e_2 + \{\text{dist}(s_2, r^t) + \text{dist}(s_1, s_2) - \text{dist}(s_1, r^t)\} / 2,$$

where d_1^t is the distance between the location of server s_1 at time t and request r^t, d_2^t is the distance between s_2's location at time t and r^t. The update of the slack happens in the following way:

If $e_1' < e_2'$,

(1) s_1 services the request,

(2) e_1 is updated: $e_1 \leftarrow e_1'$

Otherwise,

(1) s_2 services r^t,

(2) $e_2 \leftarrow e_2'$

This process is illustrated on Figure 3.3.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

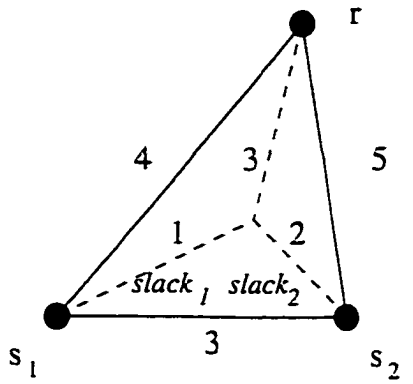

Figure 3.3: BALANCE_SLACK Algorithm illustration

The slack update and request servicing continues until the request sequence has been completed. This algorithm has been proven by Chrobak and Larmore to be 4-competitive. It uses O(1) memory and O(1) time at each step.

The BALANCE_SLACK algorithm uses the distances between the two servers in the calculation of the current slack. This does not contradict the tracklessness condition because upon closer review it is observed that the distance between the two servers was the distance between the previous request and the server which did not service the request. Since the algorithm is not memoryless, it can store this information for future use.

BALANCE_SLACK is an example of a specific algorithm for the server problem. It is defined only for two servers, but achieves relatively good competitiveness and thus presents an interesting case.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BALANCE Algorithm

The BALANCE algorithm was formulated by Irani and Rubinfeld [15] who proved it to be 10-competitive. Larmore and Chrobak [9] proved that 6 is a lower bound for its competitiveness.

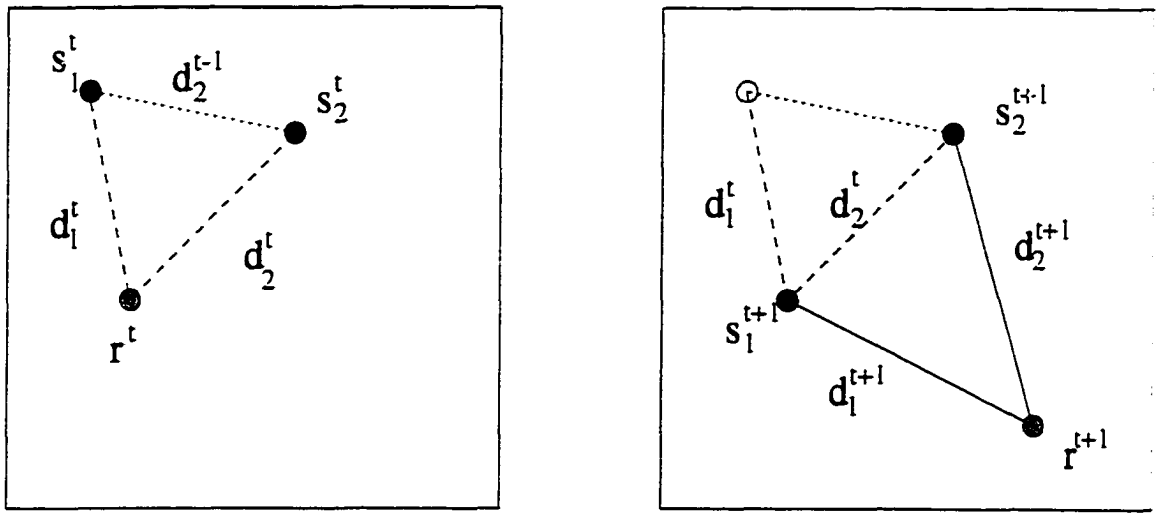

Figure 3.4: Two steps of the BALANCE Algorithm

Let e_i denote the work performed by server s_i up to the current time step. Initially, $e_i = 0$ for all i. Let d_i^t be the distance between s_i and the current request r^t. In the BALANCE algorithm the following sequence of steps is performed:

(1) Select i to minimize $e_i + d_i^t$;

(2) Move s_i to r^t;

(3) Update e_i: $e_i \leftarrow e_i + d_i^t$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In essence, the BALANCE Algorithm strives to minimize the maximum of e_i. BALANCE is illustrated on Figure 3.4. In the figure d_2^{t-1} denotes the distance between the two servers at time t. It is also the distance between s_2 and r^{t-1} from the previous step of the algorithm. It has been assumed that s_1 served the (t-1)-st request.

As previously stated, for two servers the competitiveness of the BALANCE Algorithm is known to be

$$6 \leq C_{BALANCE} \leq 10.$$

The BALANCE algorithm (also known as the Irani Rubinfeld Algorithm) uses O(k) memory and O(k) time at each step, where k is the number of servers.

HARMONIC Algorithm

HARMONIC is a randomized trackless algorithm for the k-server problem in an arbitrary metric space. Which server services a request is chosen randomly with server s_i chosen with probability

$$\frac{\frac{1}{di}}{\sum_{j=1}^{k} \frac{1}{dj}} = \frac{1}{di \sum_{j=1}^{k} \frac{1}{dj}},$$

where d_i is the distance between the i-th server and the current request, k is the number of servers. HARMONIC uses O(k) time at each step and is memoryless; it does not need to store the locations of prior requests. Its competitiveness is conjectured to be C (k+1, 2).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chrobak and Larmore [10] proved the 3-competitiveness of HARMONIC for three servers.

Paging Algorithms

It is easily observed that paging is a special case of the server problem. If pages are considered as requests and memory location are considered as servers, when a page is requested, a location (server) must be available to process the page. In the context of the server problem this will constitute a request and its service. There are a number of known competitive paging algorithms. For example, the Least Recently Used (LRU) paging algorithm evicts the page which has been in memory the longest without being requested again. LRU is a trackless k-competitive algorithm, where k is the cache size and is equivalent to the k-server problem in a uniform space.

The next chapter presents a detrministic lower bound, greater than 2, for the competitiveness of the trackless 2-server problem.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 4

PROOF OF COMPETITIVENESS FOR THE TRACKLESS 2-SERVER PROBLEM

In this chapter a lower bound for any trackless algorithm for the 2-server problem is established. The results shown demonstrate that tracklessness is an important restriction on input since it raises the lower bound for competitiveness for the online 2-server problem. The online 2-server problem is 2-competitive as proven for the Work Function Algorithm. We prove a lower bound for the trackless case of $\frac{23}{11} \approx 2.09 > 2$ [3]. The proof begins with two necessary lemmas.

Two Preliminary Lemmas

Lemma 1: (Hammering Lemma) If the adversary servers are located at distinct points x and y, there exists a sequence of requests which will force the algorithm to move its servers to x and y at cost zero for the adversary.

Proof: The adversary requests an alternating sequence of x's and y's. This process continues until the algorithm moves one of its servers to x and the other to y. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

algorithm must move its servers to these points to stay competitive, otherwise it will incur an infinite cost. The adversary already has its servers at the request points and its total cost is zero.

A sequence of requests alternating between two points is known as a hammering sequence. If the adversary and the algorithm have their servers at the same points, it is said that their servers are matched at these points.

Lemma 2: (Forcing Lemma) Given points x and y at distance d_1 from each other, a point z at distance d_2 from both x and y, adversary servers Adv_1 and Adv_2 at x and y respectively, and algorithm servers Alg_1 and Alg_2 at x and y respectively, the following cost are incurred if the next request is z followed by a hammering sequence (see Lemma 1) between z and either x or y:

$$Cost_{Alg} \geq d_1 + d_2$$

$$Cost_{Adv} = d_2.$$

Proof: (See Figure 4.1)

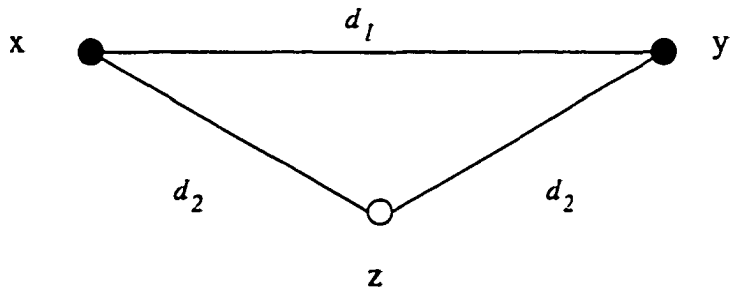

Figure 4.1: Illustration of the Forcing Lemma

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Case 1: Point z is requested. The algorithm moves server Alg_1 from x to z to service the request at cost d_2. The adversary moves server Adv_2 from y to z to service the request at cost d_2. The adversary initiates a hammering sequence between z and x. The adversary cost for that sequence is 0. The algorithm needs to move one of its servers to x. In the best possible scenario the algorithm immediately moves server Alg_2 from y to x at cost d_1 after which the cost to the algorithm for the remainder of the hammering sequence is 0. Adding up all incurred costs yields

$$Cost_{Alg} \geq d_2 + d_1$$

$$Cost_{Adv} = d_2 + 0.$$

Case 2: Point z is requested. The algorithm moves server Alg_2 from y to z to service the request at cost d_2. The adversary moves server Adv_1 from x to z to service the request at cost d_2. The adversary initiates a hammering sequence between z and y. The adversary cost for that sequence is 0. The algorithm needs to move one of its servers to y. In the best possible scenario the algorithm immediately moves server Alg_1 from x to y at cost d_1 after which the cost to the algorithm for the remainder of the hammering sequence is 0. Adding up all incurred costs yields

$$Cost_{Alg} \geq d_2 + d_1$$

$$Cost_{Adv} = d_2 + 0.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Proof of Lower Bound for Competitiveness

Theorem 1: There is no deterministic trackless algorithm for the 2-server problem which is C-competitive for any $C < \frac{23}{11} \approx 2.09$.

Proof: (see Figure 4.2)

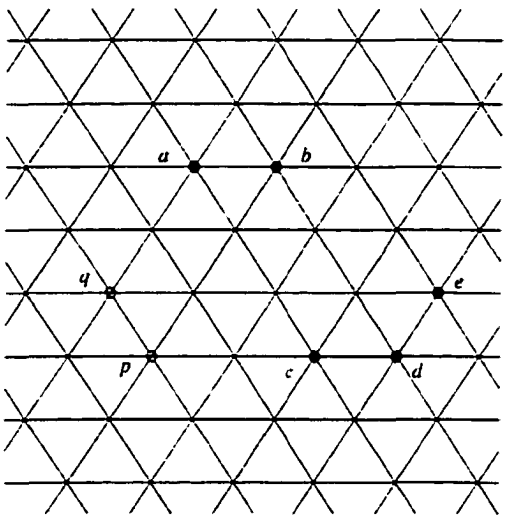

Figure 4.2: Metric space M used in the proof of lower bound for competitiveness

Consider a metric space M, which is defined as follows:

(a) M is infinite in all directions in a 2-dimensional plane;

(b) Each point in M has 6 neighbors all at distance 1 from it.

M is represented by the set of vertices of tiles in tiling the plane into uniform equilateral triangles. If M is thought of as an infinite graph, The points in M are the vertices of the graph and the distance between any two points is defined to be the length of the shortest path between them. All edges have length 1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

It is possible to reduce M to a finite metric space by identifying points to obtain the graph on a torus on Figure 4.3. This graph contains 64 distinct points.

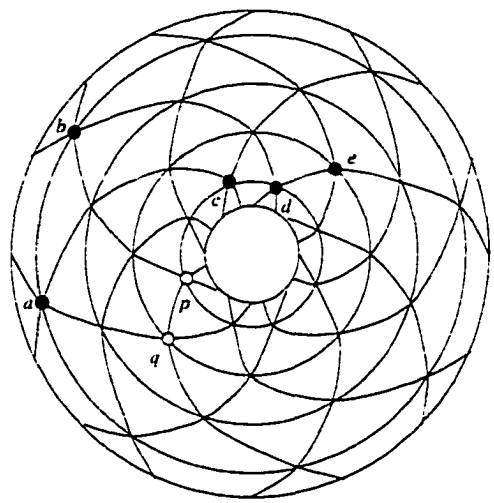

Figure 4.3: A mapping of M onto a torus

Consider any trackless algorithm for the 2-server problem in M. The initial positions of the algorithm servers (Alg_1, Alg_2) and the adversary servers (Adv_1, Adv_2) are matched at points a and b in M which are 1 apart. Assume Alg_1 and Adv_1 are at point a. There exists a request sequence ρ such that after the algorithm services ρ and moves its servers so they are 1 apart, its cost will be no less than $\frac{23}{11} * Cost_{Adv}$. Such sequence of moves constitutes one phase of the algorithm. The request sequence ρ is described as follows:

Select points c, d, e, p, q in M such that

(a) cd = de = 1, ac = bc = bd = be = 3, and ad = ae = 4;

(b) pq = 1, aq = 2, and ap = bp = bq = 3.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Such points exist in M due to its geometry. Their locations are shown on Figure 4.2 for the infinite 2-dimensional case and on Figure 4.3 for the toroidal finite case.

Six classes of possible algorithms for servicing a request sequence in M are defined as follows:

Case 1 -- 5: Move Alg_1 i times, then move Alg_2, for i = 1, 2, 3, 4, 5;

Case 6: Move Alg_1 six times.

These six cases will be discussed in detail. Six more cases, when Alg_2 services the first request, are summarized towards the end of the proof.

Each of the described cases will be used to examine one phase of the algorithm. A phase is the execution of a request sequence from a starting position in which the algorithm servers and the adversary servers are matched at two neighboring point in M to a final position symmetric (but not necessarily identical) to the starting one in which the servers of the algorithm and the adversary are again matched at two neighboring points in M. After servicing the sequence the Hammering Lemma and the Forcing Lemma are used to calculate the cost for reaching a phase completing final server configuration.

Case 1: (Move Alg_1 once, then move Alg_2) The adversary presents request sequence ρ^1 = cd[ad] where [ad] denotes a hammering sequence between point a and point d.

The sequence of moves for the algorithm is: Alg_1 to c, Alg_2 to d, Alg_1 to a.

The sequence of moves for the adversary is: Adv_2 to c, Adv_2 to d, adversary does not move.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

After the hammering sequence, to complete the phase, the servers need to reach a configuration where they are 1 apart. In the current configuration (servers at a and d) they are 4 apart. The transition to 1 apart happens in 2 steps.

Step 1: Cause the servers to be matched and 2 apart. There exists a point g in M which is at distance 2 from both a and d. By applying the Forcing Lemma the cost incurred by the algorithm to move a server to g is 2 + 4 = 6, the cost to the adversary is 2. The Hammering Lemma guarantees that the algorithm will indeed move a server to g.

Step 2: Cause the servers to be matched and 1 apart. There exists a point h in M, which is at distance 1 from (without loss of generality) a and g. By the Hammering Lemma, the algorithm will move a server to h. By the Forcing Lemma, the cost to the algorithm to move a server there is 1 + 2 = 3, and the cost to the adversary is 1.

This completes the phase. Now consider algorithm and optimal costs for this phase.

The cumulative costs for the algorithm are calculated as follows. Each term of the sum represents the cost of a move or the result of an application of the Forcing Lemma.

$$Cost_{Alg} = 3 + 3 + 3 + 6 + 3 = 18$$

$$Cost_{Adv} = 3 + 1 + 0 + 2 + 1 = 7.$$

The cost ratio C is defined as the ratio between the cost of the algorithm and the cost of the adversary, thus, for this phase,

$$C^1 = \frac{Cost_{Alg}}{Cost_{Adv}} = \frac{18}{7} \approx 2.57.$$

Case 2: (Move Alg_1 two times, then move Alg_2) The adversary presents request

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sequence ρ^2 = cde[ae]. The algorithm's sequence of moves is: Alg_1 to c, Alg_1 to d, Alg_2 to e, Alg_1 to a. The adversary sequence is: Adv_2 to c, then to d, then to e, no move. To complete the phase we apply the Forcing Lemma twice in a way similar to the approach in Case 1.

Cumulative costs incurred are:

$$Cost_{Alg} = 3 + 1 + 3 + 4 + 6 + 3 = 20$$

$$Cost_{Adv} = 3 + 1 + 1 + 0 + 2 + 1 = 8.$$

The cost ratio is

$$C^2 = \frac{20}{8} = 2.5.$$

Case 3: (Move Alg_1 three times, then move Alg_2) The sequence presented to the algorithm is ρ^3 = cded[ad]. The algorithm's moves are: Alg_1 to c, to d, to e, Alg_2 to d, Alg_1 to a. The adversary's: Adv_2 to c, to d, to e, to d, no move. Applying the Forcing Lemma twice results in the following total costs:

$$Cost_{Alg} = 3 + 1 + 1 + 3 + 4 + 6 + 3 = 21$$

$$Cost_{Adv} = 3 + 1 + 1 + 1 + 0 + 2 + 1 = 9.$$

Resulting in cost ratio

$$C^3 = \frac{21}{9} \approx 2.33.$$

Case 4: (Move Alg_1 four times, then move Alg_2) For this case the adversary picks

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ρ^4 = cdede[ae]. The move sequence for the algorithm is: Alg_1 to c, to d, to e, to d, Alg_2 to e, Alg_1 to a. Adversary responds with: Adv_2 to c, to d, to e, to d, to e, no move. After two applications of the Forcing Lemma, the cost are:

$$Cost_{Alg} = 3 + 1 + 1 + 1 + 3 + 4 + 6 + 3 = 22$$

$$Cost_{Adv} = 3 + 1 + 1 + 1 + 1 + 0 + 2 + 1 = 10.$$

The resulting cost ratio is

$$C^4 = \frac{22}{10} = 2.2.$$

Case 5: (Move Alg_1 five times, then move Alg_2) The request sequence selected by the adversary is ρ^5 = cdeded[ad]. The algorithm moves are as follows: Alg_1 to c, to d, to e, to d, to e, Alg_2 to d, Alg_1 to a. Again, the adversary only moves one of its servers: Adv_2 to c, to d, to e, to d, to e, to d, no move. Two applications of the Forcing Lemma yield:

$$Cost_{Alg} = 3 + 1 + 1 + 1 + 1 + 3 + 4 + 6 + 3 = 23$$

$$Cost_{Adv} = 3 + 1 + 1 + 1 + 1 + 1 + 0 + 2 + 1 = 11.$$

The cost ratio is

$$C^5 = \frac{23}{11} \approx 2.09.$$

Case 6: (Alg_1 moves 6 times) This is the case where the adversary uses the tracklessness condition imposed on the algorithm to maximize the algorithm's service cost. Based on the information restriction on the algorithm (point locations not available, only distances between current server positions and current request), the adversary picks

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a sequence which to the algorithm is indistinguishable from the previous sequences, namely ρ^6 = pqpqpq[pq] (see Figure 4.2). The distances from Alg_1 and Alg_2 to point p are the same as those to point c. Thus tracklessness prevents the algorithm from distinguishing a request at p from a request at c. On the other hand the adversary cost is reduced significantly. The only adversary moves are Adv_2 to p, Adv_1 to q. After 6 moves, the algorithm is forced to move Alg_2 to p to stay competitive. There is no need to apply the Forcing Lemma in this case, because after the hammering all servers are at p and q, which are 1 apart, thus the phase is complete. The corresponding costs are:

$$Cost_{Alg} = 3 + 1 + 1 + 1 + 1 + 1 + 3 = 11$$

$$Cost_{Adv} = 3 + 2 + 0 + 0 + 0 + 0 + 0 = 5.$$

The cost ratio is

$$C^6 = \frac{11}{5} = 2.2.$$

Consider now the situation where, if c is the first request, the algorithm serves that request with server Alg_2. By the tracklessness condition, the algorithm will also have to service p with Alg_2 if p is the first request. M is symmetric in a way such that a and b can be interchanged and c and p can be interchanged. By symmetry the request sequence can be chosen in a way that $Cost_{Alg} \geq \frac{23}{11} * Cost_{Adv}$ for a phase.

Upon examination of the above 6 cases, the best possible cost ratio for the algorithm is observed to be $C = \min\{\frac{18}{7}, \frac{20}{8}, \frac{21}{9}, \frac{22}{10}, \frac{23}{11}, \frac{11}{5}\} = C^5 = \frac{23}{11} > 2$. Thus the cost ratio

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

over any number of phases is at least $\frac{23}{11}$. It can be concluded that no trackless online 2-server algorithm can be 2-competitive; in fact its competitiveness must be at least $\frac{23}{11}$.

This completes the proof of lower bound for the competitiveness of a trackless algorithm for the 2-server problem. As a fundamental result this proof shows that there cannot exist a k-competitive trackless online algorithm for the k-server problem, for $k = 2$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 5

THE TRACKLESS WORK FUNCTION AND RED-BLACK GRAPHS

Looking back at the Work Function Algorithm, it is easy to realize that the approach, although optimal, is complicated and demands large amounts of space and time. This fact has prompted researchers to look for simpler algorithms, which still retain high competitiveness. One of the simplifications is the trackless concept discussed in previous chapters. As seen, there are a number of known trackless algorithms. Another trackless algorithm, the trackless work function algorithm, is expected to perform well.

The Trackless Work Function

Bein and Larmore [1, 4, 18] have defined a class of trackless algorithms, Trackless Work Function Estimator Algorithms, which calculate an estimator $\tau(r, x)$ of the optimal service $\omega(x)$ for the online server problem, where r is the current request and x is the point in the metric space M for the problem where the other server is located after r has been serviced. This $\tau(r, x)$ is the largest value for which the algorithm can prove that the optimal cost is at least $\tau(r, x)$ for any service, which culminates with configuration $\{r, x\}$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Since the algorithm is trackless, in reality x represents a class of points, which are at distance e from the r server and distance f from the s server. In mathematical notation, this is expressed as [x] = (e, f). In cases, where it is clear that [x] is in question, the brackets are omitted. In essence, to obtain $\tau(r, x)$, the algorithm minimizes over all work functions for the class of points [x]. This approach is justified because the algorithm cannot distinguish between two points from the same class and therefore there cannot exist different values for work functions within the same class. The domain of the trackless work function is expressed in terms of classes of points in the metric space of the problem.

A remark on notation: in most cases, it is understood that one of the servers is at the current request point. The only variable in those cases is the location of the other server. In such cases the notation for the work functions is simplified to $\tau(x)$ and $\omega(x)$, i.e. it is a given that one of the servers has serviced and the only variable on which the values of ω and τ depend is the location of the other server.

More formally the trackless work function can be defined as follows:

Given a current request sequence $\rho = r^1, ..., r^n$, and algorithm A with servers at points r and s such that $r = r^n$ and $s = r^i$, where r^i is some request in the sequence $r^1, ..., r^{n-1}$, A calculates a trackless estimator of the work function $\omega(x)$

$$\tau([x]) = \tau((e,f)) = \min_{\rho=\sigma} \min_{x=y=e,f} \omega_\sigma(y) \leq \omega_\rho(x).$$

The request sequence σ is such that at any step of its execution A (being trackless) cannot distinguish between σ and ρ for the first n-1 steps of its execution. Such request sequences are known as equivalent with respect to the first n-1 steps.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In general, there are exponentially many equivalent sequences and classes of points for which the algorithm needs to calculate $\tau([x])$ even in relatively small metric spaces. This is because the algorithm's tracklessness prevents it from distinguishing between points in ρ and points in any σ. The minimum could occur at a point y in σ indistinguishable from point x in ρ. Yet, using dynamic programming this minimum can be calculated in polynomial time.

As stated, $\tau(x)$ is an estimator of the work function $\omega(x)$. It is also at most as large as ω: $\tau(x) \leq \omega(x)$. It is important for the estimator to be as close as possible to the optimal cost for the service to be useful in the estimation of bounds for competitiveness.

$$Cost_A \leq C * Cost_{EST} \leq C * Cost_{OPT}$$

The precision of the estimator is a factor in the computation of lower bound for the competitiveness of the work function. A bad estimator underestimates the cost for the adversary (optimal) and therefore results in a poor lower bound.

The work function $\omega(x)$ can be computed by examining all services of request sequence ρ. This result is equivalent to the dynamic programming table at time n (see Chapter 2). Similarly, the trackless work function $\tau(x)$ can be computed by examining all services of all i request sequences σ^i, which are equivalent to ρ and obtain the dynamic programming table at time n.

Example 5.1: Computing the Trackless Work Function

Consider a metric space M where distances between points are 0, 1, or 2. If an algorithm A has servers at points r and s in M, then all other points in the space are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

divided into classes based on their distances from r and s. A work function τ for a configuration in M where the two servers are 1 apart is shown in Table 5.1. Included in the table are costs for some service, which has ended in the configuration show.

Table 5.1:Trackless work function τ

distance to s				
	2		1	0
	1	2	1	1
	0		2	
		0	1	2
		distance to r		

The adversary selects a request point r' from the class of points, which are at distance 1 from r and 2 from s; e = dist (r, r') = rr' = 1, f = dist (s, r') = sr' = 2. This pair of distances is all the information the algorithm receives.

A new function υ is defined on the domain of the trackless work function such that

$$\upsilon(x) = \min \{\tau^{t-1}(x) + rr', \tau^{t-1}(r') + rx\}$$

Applying υ to the trackless work function of Table 5.1 results in a new trackless work function shown in Table 5.2. For each class only the minimum value for $\upsilon(x)$ is recorded. For instance, for the class of points [x] = (2, 2)

$$\upsilon(2, 2) = \min \{\tau^{t-1}(x) + rr', \tau^{t-1}(r') + rx\} =$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$= \min \{0 + 1, 1 + 2\} =$

$= \min \{1, 3\} =$

$= 1.$

In general, the value of $\upsilon((e, f))$ is a lower bound for the value of $\omega'(x)$, where ω' is the work function after r' has been serviced, and x is any point in the class [x].

Table 5.2 Function $\upsilon = \tau((1, 2))$

distance to s				
	2		2	1
	1	1	2	2
	0		2	
		0	1	2
		distance to r		

At this stage there has been no move by the algorithm to service r'. Before the algorithm services, an offset operation is performed. The purpose of this operation is to reduce the sample space by decreasing all work function values by the largest possible value. This value is the amortized cost for the adversary for the request. The result of the offset is a new function $\upsilon' = \upsilon - b$, where b is the offset. For this example $b = 1$. The resulting υ' is shown in Table 5.3.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Now the algorithm is ready to make a decision regarding which server will service r'. Different decisions result in different work functions.

Table 5.3: Function $\upsilon' = \upsilon$ - b after offset

distance to s				
	2		1	0
	1	0	1	1
	0		1	
		0	1	2
		distance to r		

Consider the case when the server at r moves to r'. It is known that r' is in the class (1, 2). Therefore if r services, the cost for the algorithm for this step is 1. After the service, r' becomes the r server. The two servers are now 2 apart, which accounts for the changed appearance of Table 5.4 compared to the previous tables. The distances between r and s and the points in M are updated and points are shuffled into new classes. Now the new values for the trackless work function need to be determined. One of the new classes contains the old location of r. At the previous step, the value of the work function at the old r was 0. It is known that this class is at distance 1 from r and 1 from s. Since the algorithm always takes the minimum value for a class of points, for the class (1, 1), $\tau((1,1)) = 0$. This is entered into Table 5.4.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Now consider class [x] = (2, 2) at time t-1. At time t the points in [x] are still at distance 2 from s, but their distance from r needs to be updated. An estimate of their location with respect to the new r is made with the help of the triangle inequality as illustrated on Figure 5.1.

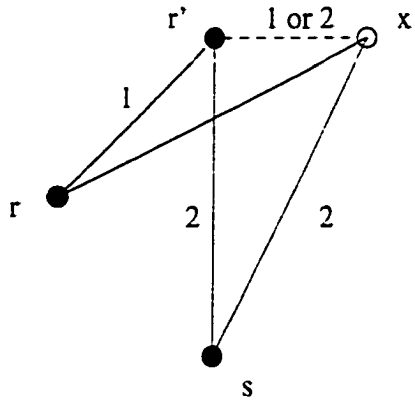

Figure 5.1: Triangle inequality

From Figure 5.1 it can be deduced that the points in [x] are divided into two classes: $[x_1] = (1, 2)$ and $[x_2] = (2, 2)$. The value of $\tau^{t-1}(x)$ was 0, which determines the value of $\upsilon^r = \tau^t(x_1) = \tau^t(x_2) = 0$.

For all other points in M the updated trackless work function is $\upsilon^r = 1$, because this is the minimum possible value of the trackless work function at these points.

Complete results are shown in Table 5.4.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 5.4: Trackless work function υ^r

distance to s				
	2	1	0	0
	1		0	1
	0			1
		0	1	2
		distance to r		

The other choice available to the algorithm is moving the server at s to r'. Since r' is at distance 2 from s, the additional cost for the algorithm to service r' is 2. The new trackless work function υ^s needs to be defined in terms of distances between all points and the updated locations of the servers. The new r server is at r', the new s server is at the old location of r.

Consider again class [x] = (2, 2) at time t-1. At time t the points in the class will be distributed between the two new classes $[x_1] = (1, 2)$ and $[x_2] = (2, 2)$. This distribution is justified by the triangle inequality. The values of u^s for these two classes are 0.

For all other points the value of $\upsilon^s = 1$. Minimization over work function values within the class justifies this result.

Table 5.5 shows the complete results of the calculations.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 5.5: Function υ^s

distance to s				
	2		0	0
	1	1	1	1
	0		0	
		0	1	2
		distance to r		

This completes the update for one step of the trackless work function.

It is important to note that the Trackless Work Estimator Algorithm is expected to perform better than most known trackless algorithms, but its competitiveness has not yet been proven. However, Larmore [4, 18] has conjectured its competitiveness to be 3, which, if true, is an improvement over the competitiveness of BALANCE_SLACK, which is 4.

Red-Black Graphs

A red-black graph is a research tool used to estimate bounds for the trackless work function. It represents a sequence of alternating configurations and moves for an adversary and an algorithm in servicing a request sequence. In work with red-black graphs the common convention is that red nodes and edges represent algorithm configurations and moves, while black nodes and edges represent adversary configurations and moves.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 5.2 illustrates a red-black graph, which is a representation of one time step of the trackless work function.

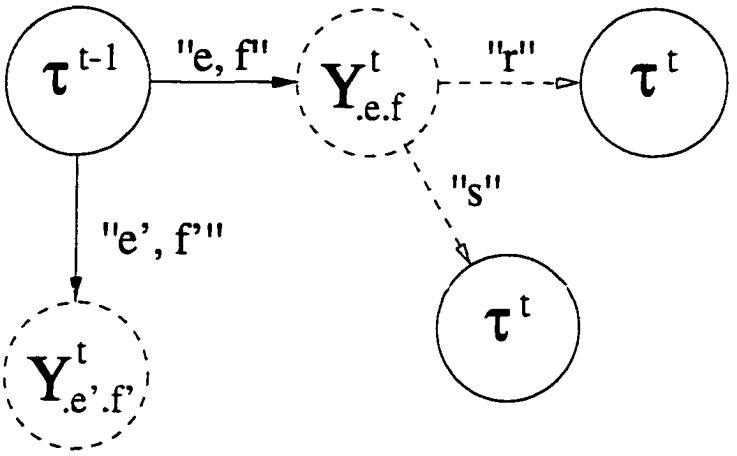

Figure 5.2: Red-black graph representation of one step of TWF

One of the applications of the red-black graph is that it can be used as method to estimate the competitiveness of a TWF algorithm. This is done by examining the cycles of the graph. In this scope a cycle is described in terms of configurations indistinguishable to the algorithm. For the purpose of estimating the competitiveness, a cycle in the graph is identified as a critical cycle. By definition a critical cycle is a cycle in the red-black graph for which some balance is achieved between the cost for the adversary and the cost for the algorithm. If either the adversary or the algorithm selects a service outside the cycle, its cost will be increased. In essence, the critical cycle determines the competitiveness of the algorithm since it is a measure for the ratio between the algorithm cost and the adversary cost, which is the definition of competitiveness. A critical cycle is the analog of a saddle point in classic game theory.

Example 5.2: Critical cycle in a red-black graph

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A simple red-black graph is shown on Figure 5.3. Upon examination it can be concluded that the critical cycle is abcd. The cost for the algorithm in the cycle is 3, the cost for the adversary is 2. Therefore this particular algorithm has competitiveness $C = \frac{3}{2}$.

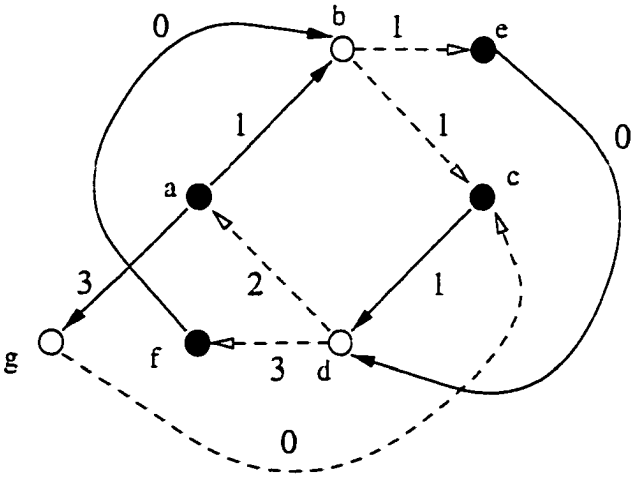

Figure 5.3: Illustration of critical cycle in a red-black graph

Consider the case when the algorithm takes the alternate path at node b. For the cycle abed $Cost_{Alg} = 3$, which is the same as before, but the adversary cost has been reduced: $Cost_{Adv} = 1$. By making this decision the algorithm has worsened its competitiveness which is now 3. Similarly, if the adversary takes the alternate edge at node a, its cost for cycle agcd will be increased to 4, while the algorithm cost is reduced to 2.

The competitiveness estimated with a red-black graph is no better than the actual competitiveness for the algorithm.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

REFERENCES

[1] Bein, Wolfgang. 1999. Personal Communication.

[2] Bein, Wolfgang, and Lawrence Larmore. 1998. The Trackless Server Problem Tutorial. Available from World Wide Web: <http://www.cs.unlv.edu/~bein/research/trackless/>

[3] Bein, Wolfgang, and Lawrence Larmore. 1999. Trackless online algorithms for the server problem. Submitted for publication in Information Processing Letters.

[4] Bein, Wolfgang, and Lawrence Larmore. 1999. Estimators for the Server Problem. Manuscript.

[5] Borodin, Allan, and Ran El-Yaniv. 1998. Online Computation and Competitive analysis. Cambridge University Press.

[6] Chrobak, Marek, and Lawrence Larmore. 1991. On fast algorithms for two servers. Journal of Algorithms 20: 607-614.

[7] Chrobak, Marek, and Lawrence Larmore. 1991. An optimal online algorithm for k servers on trees. SIAM Journal on Computing 20: 144-148.

[8] Chrobak, Marek, and Lawrence Larmore. 1991. A note on the server problem and a benevolent adversary. Information Processing Letters 38: 173-175.

[9] Chrobak, Marek, and Lawrence Larmore. 1992. HARMONIC is three-competitive for two servers. Theoretical Computer Science 98: 339-346.

[10] Chrobak, Marek, and Lawrence Larmore. 1992. The server problem and online games. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 7: 11-64.

[11] Fiat, Amos, Y. Rabani, and Y. Ravid. 1990. Competitive k-server algorithms. Proceedings of the 31-st Annual Symposium on Foundations of Computer Science 454-463.

[12] Fiat, Amos, Richard Karp, Michael Luby, Lyle McGeoch, Daniel Sleator, and Neal Young. 1991. Competitive paging algorithms. Journal of Algorithms 12: 685-699.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[13] Fiat, Amos, and Gerhard Woeginger (Eds.). 1998. Online Algorithms. Springer-Verlag Berlin Heidelberg New York.

[14] Grove, E. 1991. The harmonic online k-server algorithm is competitive. Proceedings of the 23-rd Annual ACM Symposium on Theory of Computing 260-266.

[15] Irani, Sandy, and R. Rubinfeld. 1991. A competitive 2-server algorithm. Information Processing Letters 39: 85-91.

[16] Koutsoupias, Elias, and Christos Papadimitriou. 1995. On the k-server conjecture. Journal of the ACM 42: 971-983.

[17] Koutsoupias, Elias, and Christos Papadimitriou. 1996. The 2-evader problem. Information Processing Letters 57 (5): 249-252.

[18] Larmore, Lawrence. 1999. Personal Communication.

[19] Manasse, M., Lyle McGeoch, and Daniel Sleator. 1988. Competitive algorithms for on-line problems. Proceedings of the 20-th Annual ACM Symposium on Theory of Computing 322-333.

[20] Manasse, M., Lyle McGeoch, and Daniel Sleator. 1990. Competitive algorithms for server problems. Journal of Algorithms 11: 208-230.

[21] Motwani, Rajeev, and Prabhakar Raghvan. 1995. Randomized Algorithms. Cambridge University Press.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

VITA

Graduate College
University of Nevada, Las Vegas

Anna N. Naydenova

Home Address:
3639 E Villa Knolls
Las Vegas NV 89120

Degree:
Bachelor of Science, Computer Science, 1997
University of Nevada, Las Vegas

Special Honors and Awards:
Upsilon Pi Epsilon International Honor Society for the Computing Sciences

Thesis Title: Trackless Online Server Problems and Red-Black Games

Thesis Examination Committee:
Chairperson, Dr. Wolfgang W. Bein, Ph.D.
Committee Member, Dr. Lawrence L. Larmore, Ph.D.
Committee Member, Dr. Laxmi P. Gewali, Ph.D.
Graduate Faculty Representative, Dr. John Wang, Ph.D.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.