T-theory and analysis of online algorithms

James Oravec
University of Nevada, Las Vegas
T-THEORY AND ANALYSIS
OF ONLINE ALGORITHMS

by

James Oravec

Bachelor of Arts in Computer Science
University of Nevada, Las Vegas
2005

Master of Science in Computer Science
University of Nevada, Las Vegas
2007

A thesis submitted in partial fulfillment of the requirements for the

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

Graduate College
University of Nevada, Las Vegas
August 2007
The Thesis prepared by

JAMES A. ORAVEC

Entitled

T-THEORY AND ANALYSIS OF ONLINE ALGORITHMS

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

MASTER OF SCIENCE IN COMPUTER SCIENCE

Laurence J. Lamarre
Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative
ABSTRACT

T-Theory and Analysis of Online Algorithms

by

James Oravec

Lawrence L. Larmore, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

Several advancements in Online Algorithms can be credited to T-theory, a field of discrete mathematics. T-theory has aided in the development of several online algorithms for the $k$-server problem, although the standard notation of T-theory was not used at the time of their creation.

A summary of the $k$-server problem, and some important concepts of T-theory, are given. A number of known $k$-server results are restated using the established terminology of T-theory. Included is a 3-competitiveness proof, using T-theory, for the HARMONIC algorithm for two servers, which was presented in a paper by Larmore and Oravec [71].

Previously, the Knowledge State Method was documented in Kurlinski’s thesis [70]. Additional research and analysis was done by Larmore and Oravec. Summaries of that work, as well as prior work by Larmore and Bein are given.

Research supported by NSF grant CCR-0312093.
# Table of Contents

ABSTRACT ................................................................. iii

ACKNOWLEDGMENTS .............................................. v

CHAPTER 1 INTRODUCTION ....................................... 1
Thesis Introduction .................................................. 1

CHAPTER 2 ONLINE ALGORITHMS ............................. 10
Introduction .......................................................... 10
Deterministic versus Randomized .............................. 10
The Work Function Algorithm ................................. 11

CHAPTER 3 PARTICULAR PROPERTIES OF T-THEORY .... 12
Introduction .......................................................... 12
Basic Definitions ..................................................... 12
Examples .............................................................. 13
Properties .............................................................. 14

CHAPTER 4 THE k-SERVER PROBLEM AND T-THEORY ... 16
Introduction to the k-Server Problem ......................... 16
Elementary T-Theory Concepts ................................ 20
The Virtual Server Construction .............................. 26
Tree Algorithms ...................................................... 27
Balance Algorithms ............................................... 31
Server Algorithms in the Tight Span ......................... 33
Definition and Analysis of HANDICAP ..................... 37
Harmonic Algorithms ............................................. 44
Summary and Future Applications of T-theory. .......... 50

BIBLIOGRAPHY ....................................................... 53

APPENDIX A MATHEMATICA CALCULATIONS ............. 61

APPENDIX B FIGURES AND TABLES ............................ 64

VITA ................................................................. 105
ACKNOWLEDGMENTS

The \LaTeX\ document class was made by Steve Lumos. I would also like to give thanks to Dean Bartkiw and Marek Chrobak for reviewing the final manuscript coauthored by Larmore and Oravec.

This thesis is dedicated to those who make a difference in the life of others. I have been blessed in having a great group of friends and mentors.

To my family and friends, for their love, encouragement and support. To my advisor, Lawrence L. Larmore, for giving me a genuine academic experience. To Angel Muleshkov, for explaining mathematics to me in depth. To Ronald Corbett, for encouraging me to think outside the box. To Chad Lexis, a childhood friend who has encouraged me to achieve more in life. To all of my computer science friends, for making my time at UNLV enjoyable. To Colonel Robert Luberacki and the men and women of the United States Military Services, for all they have done for this great country.

To CJ Dowell, who left us before his time. He is a reminder to me and others that life is short, so enjoy your life and those who are important to you, while you can.

Lastly, I would like to thank all of the members of my committee: Ajoy Datta, Lawrence Larmore, John Minor, and Angel Muleshkov.
CHAPTER 1

INTRODUCTION

Thesis Introduction

This thesis surveys the research performed by Larmore and Oravec between February 2006 and April 2007. Chapter 4 is the heart of the thesis. This chapter is based on the topics covered in the paper by Larmore and Oravec titled “T-Theory Applications to Online Algorithms for the Server Problem” [71], which is currently available on arXiv, a prepublishing service provided by Cornell University.

The remainder of the thesis covers research topics that Larmore and Oravec have worked on. Those topics may lead to future published papers. In order for the reader to better understand the research in this paper, we provide additional background information.

Lastly, Eppstein [50] suggested a possible generalization of a previous results by Bein, Chrobak, and Larmore [15]. The suggested generalization would extend previous results for Manhattan planes to Manhattan orbifolds, a class of metric spaces introduced by Eppstein. No publishable results on this question have been obtained by Larmore and Oravec, and the details of this research are beyond the scope of this thesis.

Dynamic Programming

Some online algorithms use dynamic programming to obtain an optimal solution, which is then used in competitive analysis of the given algorithm. An online algorithm can be analyzed using a potential function, which can sometimes be obtained by dynamic
programming. Potential functions are widely used to calculate the competitivenesses of online algorithms. A potential function is a mapping of possible states of the optimal configuration and the configuration of an online algorithm, to the set of real numbers. The potential is normally denoted by the symbol $\phi$, thus we could write the mapping as: $\phi : S_{online} \times S_{opt} \rightarrow \mathbb{R}$, where $S_{online}$ and $S_{opt}$ are sets of possible configurations for the online algorithm and the optimal offline algorithm [21].

In certain cases, a minimum potential can be calculated using dynamic programming, using a process called boosting. Boosting has been used in programs written by Chrobak and Larmore, and, in particular, was used to obtain a lower bound result in [28]. When using boosting, the program can loop many times. This looping process does not guarantee convergence. There is good news though; in some cases, there are tests which will detect divergence.

Boosting fails if a positive cycle exists, just as the Bellman-Ford algorithm fails with negative cycles as shown in Rivest et. al. [39].

The $k$-Path Problem

I was introduced to the application of offline algorithms to online algorithms via a challenge problem in Larmore’s Introduction to Algorithms class. The problem was titled the “Two Path Problem.” The problem description is as follows: Given a weighted directed acyclic graph and a source vertex, find, if possible, a minimum cost pair of paths which cover all vertices.

My solution used dynamic programming and is easily generalized to the $k$-path version of the problem using $O(n^k)$ time complexity. The algorithm is offline, however it has applications to online algorithms.

I presented the solution during Math Fest 2006 in Knoxville, TN (August 10-12, 2005), during the Pi Mu Epsilon Session 2. The title of the talk was “A Dynamic Solution to the $k$-Path Problem”.
A trivial solution to the $k$-path problem can be solved in $O((k+1)^n)$ time. For the 2-path problem this means that the time complexity would be $O(3^n)$. Although this trivial solution would be feasible for small values of $n$, it becomes infeasible quickly, as $n$ grows.

To aid the readers intuition to the 2-path problem, we give Figure 1, which illustrates a simple case that can be worked by hand. We give more details later on how the dynamic programming solution to this problem works.

We provide a quick analysis for a small graph of 21 vertices (See Figure 4 for an illustration). The start vertex is included in both paths. Since the other vertices may be included in either path, or both, we have approximately $3^{20} = 10,460,353,203$ computation steps to solve the problem. Since the graph is small, this is practical. However the dynamic programming approach takes $O(n^3)$, an order of polynomial time.

For a directed acyclic graph of 21 vertices, the dynamic programming solution can be solved using a $21 \times 21$ matrix and solve the problem with roughly 441 computation steps.

The first solution quickly grows beyond practical computation. Assume that $n = 10,000$. The first vertex is in both paths. Thus there are $9,999$ vertices can be in the first path, the second path, or both. Thus we have $3^{9999}$ many possible solutions. This means that to solve the problem would take approximately

$$543783361412206247677224360539400604441511041765331163067753272524940773024238385210228261103224417749407732561444468527656393204102$$

$$641271370112721410775101648199937194607571140964618989519868568373080691494215766365377581418145725604194679060226353597785314408438$$

$$957620671262853105861998940158885502201399623083572575787418816878556117541729350462236106934904649497797100813971796503061011619226$$

$$48100139812079337787245824151690191089099740745101825272393999476578778950978995329765998298215491764565152139443656411390137477$$

$$9885026879577061276439233996636137656938366653646268068583687820348928312077992297193568556782086434604907320217800414404646054$$

$$40186478801614488384996087010673045928236383713760497936000541472267815269617333020776713761903714119632918964002094480437949746973609448$$

$$87426232030092353360010293585491146803964982264076046564421500461960233230204141666642653201612314225524798990790630633208322781042665111087$$

$$281320491969042014706248268711914101490003746243431231288012566767677774546177277414851359490153261435757375320834143188461765511526$$

$$532389278878489865326096624617301765203011649467878670487862925905906315671983943547696266031775877664360308809046892770573548333$$

$$5156962912192089290282016486453128658851924892478496177489489348307016020064499234169989934701436994254428228783494735289725906$$

$$0210147461230356113400599565352283226881110130501984600123749496820326871037383055786079949656411630755557464851509203412728$$

$$270707097993100700809241156704856771626455704618996313628610182672583466866961756310981229005771951399116066530239765169$$

$$0436115977503661047408254449915472952656104899402596850068927849585551857717949000394267253169002000000301326985319444$$

$$39159471611466938282971946892090304565390201810662035295111512556510997596642549072001728383557619702603737254824821960396155$$

$$1533984207562577572299681959727614444481432052676687678973041007478604000005136840101696937666654108923175714711900296422290381789676$$

$$55779944670007328001631891122976893403752012283575584113803837797991808939138970391796711073382422444997951101326427476460535917$$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
computation steps, whereas the dynamic programming solution would take, at most
\[ n^3 \approx 10,000^3 = 1,000,000,000,000 \text{ computation steps.} \]

Dynamic program implicitly uses the principle of optimality. In the minimum path
problem, the principle of optimality is that every subpath of an optimal path must be
optimal [57].

The proof of the principle of optimality is by contradiction. Any improvement to
a subproblem that is used for the final solution leads to an improvement of the final
solution. This contradicts the optimality of the final solution.

An example to aid intuition is given in Figure 9. In this figure, we are given
that the optimal path from S to H is the ordered tuple \( \{ S, A, B, C, E, H \} \). We now
assume that there is an optimal path between A and E, which is not the ordered tuple
\( \{ A, B, C, E \} \). This means that \( \{ A, D, E \} \) would cost less than \( \{ A, B, C, E \} \). If
this were true this would mean that the optimal path from S to H would be the ordered
tuple \( \{ S, A, D, E, H \} \), since \( \{ S, A, D, E, H \} \) would cost less than \( \{ S, A, B, C, E, H \} \).

This is a contradiction to the information we are provided, thus a better solution than
the optimal solution cannot exist.

To aid the reader's understanding of the 2-path problem, we start with a simpler
problem, the shortest path problem. For an instance of this problem we are given a graph and a source vertex in that graph. The goal is calculate the shortest path from the source vertex to every other vertices. We can use the principle of optimality to help us avoid repeated computations. Figure 10 gives an initial graph. Figure 11 shows the first step of calculating the shortest path from vertex 0 to vertex 1. Since there is only one possible path to vertex 1, that is the shortest path. Figure 12 shows the next step which shows the minimum cost path to vertex 2 from vertex 0. Notice that we can use the fact that the minimum cost to vertex 1 has already been calculated. By using this information, the principle of optimality, we are able to determine the least cost path to vertex 2 by being the minimum cost of the edge weight from vertex 0 to vertex 2, or the sum of the cost to get to vertex 1 plus the edge weight from vertex 1 to vertex 2. In this example, it is cheaper to take the path from vertex 0 to vertex 2. Figure 13 illustrates the calculation of the minimum cost from 0 to 3. In this case there is only one incoming edge to vertex 3. Since we know the minimum cost to vertex 2 is 3, we can calculate that the minimum cost to vertex 3 has the minimum cost to get to vertex 2 plus the edge weight from vertex 2 to vertex 3. Similarly, Figure 14, Figure 15, and Figure 16 show the steps in calculating the minimum costs to vertices 4, 5 and 6 respectively. The order of the calculation is important to avoid recalculations.

We now revisit the 2-path problem given earlier. Figure 1 gives an example graph. Figure 2 represents all combinations of path endings for the five vertices. The sub-problems are worked in topological order.

To further describe Figure 2, the SS at the top left represents the all possible configurations which have S as the ending vertex for both paths. Since S is the starting vertex, both paths have length zero. Below SS is SA; this represents two paths, where one path’s ending is still at S and the other is ending at A. Since there is only one way to get from S to A, the path is implied at this point. We can create AA by moving the path ending is S to A. Now, we consider AB, which does not uniquely determine the
paths to it. AB can be create in one of two ways. The first way is by having one path that goes from S to A to B and the other path going from S to A. The second way is to have one path go from S to B and the other path to go from S to A. Both paths are represented in the overall calculation at this location. Back pointers can be used to recover the correct path back and the principle of optimality mentioned above is used to prove the result correct. We continue the calculations through the rest of the path endings in order to finish the problem.

Figure 3 gives the graph of Figure 1 in a representation similar to Figure 2. In Figure 3 there is an edge weight from SS to SA. This represents the cost to move one path ending from S to A. Similarly, the edge weight from SA to AA represents moving a path ending from S to A as well. Now in order to get to AB, we can go from SS to SA to AB or from SS to SA to AA to AB. It costs less to go from SS to SA to AB. Thus, using proper back pointers, AB would point to SA to keep track of the optimal paths. AB should also store the value of $2 + 4 = 6$ since that is the minimum cost to get to AB. This value will be passed on to AC, which will add the 6 and the edge weight between AB and AC. By doing this, we are able to avoid recalculations of path costs.

We continue the calculations to the bottom row. We then select the minimum value on the bottom row, which represents the minimum 2-path cost of this graph. If we need to state the paths, then we can use the back pointers to recover the paths of the solution. Continuing this process, the optimal solution is obtained.

Looking at Figure 4, the reader will find that trying to “eyeball” a solution to an instance of this magnitude is extremely difficult at best. However the optimal solution can be calculated by hand using the dynamic programming approach.

Figure 5 is another example showing path endings. The order of calculation can be done for a row by doing all horizontal calculations first followed by any calculations that go vertically to the next row.
Table 1 gives another example graph for the two path problem. Figure 6 shows the calculations for each cell of the matrix and the paths that are created from these calculations.

The dynamic programming approach to the 2-path problem above can be used to calculate the optimal offline algorithm cost for the two path problem. A variation of the 2-path problem is the online version of the problem. Instead of getting the optimal solution, the goal is to get a good competitive online algorithm. Figure 7 gives an example of an online algorithm, relating to the 2 path problem. The vertical dashed line represents a cutoff between the information the algorithm knows up to that point and future input which is unknown. At this point, the algorithm would have to give output before it would receive the next input. In this example, the vertices are labeled with numbers and there are two paths which are represented by their endings which we call A and B. If a path letter is at a vertex, then this means that this vertex is part of that path and it is currently the ending vertex of that path. Figure 8 gives an example of an intermediate position of the online version of the two path problem.

We continue our discussion of online algorithms in Chapter 2. Additional reference to online algorithms can also be found in [21].

Knowledge State Algorithms

Larmore and Oravec started their research on the Knowledge State Algorithms, which is outlined in Joshua J. Kurlinski’s thesis [70]. In the Fall of 2001, Larmore intuitively selected a set of Knowledge States that he believed would yield a $\frac{71}{36}$ competitiveness for the randomized 2 server problem on the line. If this was true, this would be a result of interest because it would be a proof that a randomized algorithm exists with a competitiveness of less than two on the line, which would be better than any deterministic algorithm could achieve, using a method would allow for generalization. Bartal, Chrobak and Larmore were able to find a less than 2 competitiveness for
the line [7], unfortunately there does not seem to be an obvious way to generalize their results.

The set of all knowledge states used in the algorithm consisted of six one-parameter families, each represented by an interval. The set of all cases, where each case consisted of a knowledge state and a request, was broken into 20 2-parameter families, each represented by a triangle. Each case needed to be validated by a proof.

Since there were infinitely many cases, it would seem that infinitely many proofs were needed. However, these proofs could themselves be classified into families, and each proof family proved the validity of a set of cases represented by a region in a triangle.

Figures 17 and 18 shows a triangle representing a family of cases, where the green area represents those cases validated by just one family of proofs. Our goal was to find enough proofs to cover all the triangles.

After going through all of the cases, we found that everything was covered except for three particular areas. When these areas were plotted, they looked like slivers. These slivers were enough to show that the set of knowledge states were not \( \frac{21}{36} \) competitive. These tests were used to get a quick intuition of the problem.

After finding this result, we changed the competitiveness to be closer to two, however still less than two. By doing so, we found that the slivers got smaller by visual inspection. Even with additional manipulation, we were not able to modify the competitiveness to make the selected knowledge states work.

Wolfgang Bein previous presented preliminary work on the \( \frac{21}{36} \)-Competitive Knowledge State Algorithm for the 2-Server Problem on the line at the Dagstuhl Workshop on Online Algorithms in June 2002.

Since the original set of knowledge states failed, we had a few options; either select a new group of knowledge states and test them, or try to automate the process and select the best knowledge states. Larmore had done previous research on the second
option. He called the approach the “magic approach”. The algorithm would take advantage of linear programming and pseudo-convex hulls, which are discussed later in this paper.

Originally the pseudo-convex hull program was written in Pascal by Lawrence Larmore. Later Edward Larmore, a programmer at Lockheed-Martin, reprogrammed the code into Java. Lastly, Lawrence Larmore and Oravec translated the Pascal code into Mathematica code, in order to take advantage of the mathematically feature rich environment. Funding by the National Science Foundation allowed for us to used mathematica on multiple computers.

Although this method would not work for general metric spaces, it would however be able to be used on metric spaces of interest. Some of these metric spaces include the CNN metric, which is used for the CNN problem, and also for the server problem in metric spaces, such as $M_{24}$.

Before Larmore and Oravec started the general implementation of the program, they used these methods and mathematica to test a few specific metric spaces. The results were confirmed for the classes of metric spaces $M_{13}$ and $M_{24}$. Results for special cases of the CNN problem, a variation of the server problem which has important applications in memory access, were also proven. Progress was made on general implementation of this process. Research on this problem was interrupted by the research involving T-theory, which is discussed later in this thesis. If future results are found, they are planned to be presented in a paper called “Construction of Knowledge State Algorithms by Geometric Techniques”.
CHAPTER 2

ONLINE ALGORITHMS

Introduction

An offline algorithm is given the sequence of "requests" which it must "serve", prior to it having to serve any requests. An optimal offline algorithm uses the sequence to calculate an optimal way of serving it.

Online algorithms are required to give output between inputs of the request sequence. An offline algorithm can calculate optimal solutions, but online algorithms cannot. Thus the "goodness" of an online algorithms have to be measured in a different way. We measure online algorithms by the competitiveness of the algorithm, where competitiveness is the ratio of the cost of the online algorithm to the cost of the optimal offline algorithm which receives the same request sequence.

Deterministic versus Randomized

For a particular input sequence, a deterministic algorithm will always give the same output.

A barely random algorithm uses randomization once in its execution at the beginning. Randomization is used to pick one deterministic algorithms out of a collection of such algorithms, after which the algorithm will behave like the selected deterministic algorithm.

A behavioral randomized algorithm uses randomization in a stronger sense, using randomization at every step.
When dealing with the $k$-server problem, it is necessary to understand some basic concepts. One such concept is a server. A server is an abstract data type which is used to serve requests. A request is a point in a Metric space. To serve a request, in the classical $k$-server problem, means that the server moves to the location of the request. A request sequence is a sequence of requests.

For any request sequence, there exists an optimal way of serving the request sequence. We can compute how much the optimal offline algorithm would have to pay to serve the request sequence. Using this information, we can compare effectiveness of deterministic algorithms against the optimal offline algorithm. After considering all possible request sequences, we are able to use the worst cost case. By using the worst, we can guarantee that the algorithm will not have to pay more than a certain multiple of the optimal offline algorithm; that multiple is the competitiveness of the algorithm.

The cost of a randomized algorithm is defined to be its expected cost, which is calculated using the probabilities of the algorithms moves and the associated costs of those moves.

The Work Function Algorithm

The Work Function Algorithm (WFA) is used on deterministic problems, and in some cases is proven to be the best possible algorithm to use for particular problems. WFA uses dynamic programming and can be used for a large variety of online algorithms including the server problem.
CHAPTER 3

PARTICULAR PROPERTIES OF T-THEORY

Introduction

The purpose of this section is to provide the reader with a brief overview of T-theory and the type of problems it can be applied to. These topics were part of the Summer 2006 independent study course that Oravec took under Larmore.

The following sections give a brief formal definition of Metric on $X$ and Pseudo-Metric.

Definition of a Metric on $X$

Let $X$ be any set. A metric on $X$ is defined to be a function $d : X \times X \rightarrow R \cup \{\infty\}$ where the following conditions are satisfied for all $x, y, z \in X$ (see [38]).

1) Positiveness: $d(x, y) > 0$ if $x \neq y$, and $d(x, x) = 0$.
2) Symmetry: $d(x, y) = d(y, x)$
3) Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

Definition of a Pseudo-Metric

A pseudo-metric, satisfies all the properties of definition of a metric except it relaxes the requirement of $d(x, y) = 0$ implies $x = y$. This means that different points can have distance of zero. As given in [38].

In most papers in the area of T-theory, the word metric is used to mean pseudo-metric. Henceforth, we will do the same.
Basic Definitions

The Metric on a Graph

Suppose $V$ is a set and $E \subseteq V \times V$. Then $G = (V, E)$ is a graph. If $G$ is connected, there is a standard metric on $V$, which is the number of edges in the shortest path between two vertices. We will freely refer to a graph as a metric space whenever convenient, where the graph metric is understood. The definition above applies to unweighted graph, however there is also a definition for weighted graphs.

Cuts and Splits

A metric $d$ is a cut metric if there are exactly two subsets of $X$ in the set $X/d$ of equivalence classes of elements of $X$ under the equivalence relation $x \sim y \iff d(x, y) = 0$. Splits and cuts are the same as per the definition in [41].

A trivial cut is a cut in which a vertex is by itself and is separated from the rest of the graph.

The relevance is explained as follows: The simplest building stones are the cut (pseudo-) metrics associated to cuts of the set $V$: for a cut $S=(A,B)$ of $V$. After Chepoi [23], we define

$$\delta_S(x, y) = \begin{cases} 
0 & \text{if } x, y \in A \text{ or } x, y \in B \\
1 & \text{otherwise, i.e., if } S \text{ separates } x \text{ and } y
\end{cases}$$

Examples

The following objects mentioned in this section, will be used throughout the rest of this chapter. They are used to help illustrate some properties of T-Theory.

The Bipartite Graph $K_{p,q}$

Let $K_{p,q}$ be the bipartite graph whose shores have $p$ and $q$ vertices, respectively. For example, $K_{2,3}$ is illustrated in Figure 38, and its metric is represented by the matrix in Table ??

13
**The Hypercube Graph $H^n$**

Let $H^n$ be the hypercube graph of dimension $n$. The vertices of $H^n$ are all binary $n$-tuples. There is an edge from one vertex to another if and only if they two tuples differ at exactly one position. Its metric is the Hamming distance.

**The Complete Graph $K_n$**

Let $K_n$ be the complete graph of $n$ vertices. Its metric is the uniform metric.

**The Orthoplex Graph**

The orthoplex graph of order $n$ has $2n$ vertices, namely $a_i$ and $b_i$ for all $1 \leq i \leq n$. If $i \neq j$, there is an edge from $a_i$ to $a_j$, from $b_i$ to $b_j$, and from $a_i$ to $b_j$. There is no edge from $a_i$ to $b_i$. The orthoplex graph of order 3 is the classic octahedral graph with 6 vertices. The orthoplex is also known as the cross-polytope [40]. The metric on an orthoplex graph is a finite metric space in the class $M_{24}$ if and only if it is a subspace of some orthoplex graph.

**Properties**

**Property Implication**

Let $X$ and $Y$ be metric spaces, and let $f : X \to Y$ be a function. We will give the relations among the following properties:

- a) $f$ is continuous.
- b) $f$ is non-expansive.
- c) $f$ is an isometric embedding.
- d) $f$ is an isomorphism.
- e) $f$ is 1-Lipschitz.
- f) $f$ is a retraction.
We will use these letters the following sections to illustrate implication. The following relations summarize the properties above:

1. \( b \equiv c \)

2. \( d \Rightarrow c \Rightarrow b \Rightarrow a \)

3. \( d \Rightarrow f, \text{ if } x = y \)

4. \( f \Rightarrow e \)

**Visual Representation**

To aid in the understanding of the above properties, we illustrate the implication by a directed graph. If a path exists from a vertex to the another vertex, then the latter vertex is implied from your initial property.

See Figure 39 to see the directed graph representation of implication amongst these properties.
CHAPTER 4

THE K-SERVER PROBLEM AND T-THEORY

Introduction to the k-Server Problem

The k-server problem was introduced by Manasse, McGeoch, and Sleator [72], while T-theory was introduced by John Isbell [62] and independently rediscovered by Andreas Dress [42][43]. The communities of researchers in these two areas have had little interaction. The tight span, a fundamental construction of T-theory, was later defined independently, using different notation, by Chrobak and Larmore, who were unaware of the work of Isbell, Dress, and others.

Bartal [6], Chrobak and Larmore [26][28][29][32], and Teia [79], have used the tight span concept to obtain results for the k-server problem. In this paper, we summarize those results, using the standard notation of T-theory. We then suggest ways to use T-theory to obtain additional results for the k-server problem.

The k-Server Problem

Let $M$ be a metric space, in which there are $k$ identical mobile servers. At each time step a request point $r \in M$ is given, and one server must move to $r$ to serve the request. The measure of cost is the total distance traveled by the servers over the entire sequence of requests.

An online algorithm is an algorithm which must decide on some outputs before knowing all inputs. Specifically, an online algorithm for the server problem must decide which server to move to a given request point, without knowing the sequence of future requests, as opposed to an offline algorithm, which knows all requests in advance.
For any constant $C \geq 1$ we say that an online algorithm $\mathcal{A}$ for the server problem\footnote{Or for any of a large number of other online problems.} is $C$-competitive if there exists a constant $K$ such that, for any request sequence $\varrho$, where $\text{cost}_{\text{opt}}(\varrho)$ is the optimum cost for serving that sequence:

$$\text{cost}_{\mathcal{A}}(\varrho) \leq C \cdot \text{cost}_{\text{opt}}(\varrho) + K$$

If $\mathcal{A}$ is randomized, the expected cost $E\text{cost}_{\mathcal{A}}(\varrho)$ is used instead of $\text{cost}_{\mathcal{A}}(\varrho)$.

The $k$-server conjecture, posed by Manasse, McGeoch, and Sleator [72], is that there is a deterministic $k$-competitive online algorithm for the $k$-server problem in an arbitrary metric space. Since its introduction by Manasse et al., substantial work has been done on the $k$-server problem [1] [2] [6] [7] [8] [9] [10] [11] [12] [13] [15] [16] [18] [19] [20] [21] [31] [33] [34] [36] [49] [51] [52] [53] [54] [55] [61] [63] [69] [67] [79] [80].

The $k$-server conjecture has been solved for $k = 2$, and for some special cases for $k \geq 3$, but the general problem remains open.

It is traditional to analyze the competitiveness of an online algorithm by imagining the existence of an adversary, who creates the request sequence, and must also serve that same sequence. Since we assume that the adversary has unlimited computational power, it will serve the request sequence optimally; thus, competitiveness can be calculated by comparing the cost incurred by the online algorithm to the cost incurred by that adversary. We refer the reader to Chapter 4 of [21] for an extensive discussion of adversarial models.

Throughout this paper, we will let $s_1, \ldots, s_k$ denote the algorithm's servers, and also, by an abuse of notation, the points where the servers are located. Similarly, we will let $a_1, \ldots, a_k$ denote both the adversary's servers and the points where they are located. We will also let $r$ be the request point.
Memoryless, Fast, and Trackless $k$-Server Algorithms

Let $A$ be an online algorithm for the $k$-server problem.

$A$ is called memoryless if its only memory between steps is the locations of its own servers. When a new request is received, $A$ makes a decision, moves its servers, and then forgets all information except the new locations of the servers.

$A$ is called fast if, after each request, $A$ can make its decision using $O(1)$ operations, where computing the distance between two points counts as one operation.

$A$ is called trackless if $A$ initially knows only the distances between its various servers. When $A$ receives a request, it is only told the distances between that request and each of its servers. $A$’s only allowed output is an instruction to move a specific server to the request point. $A$ may not have any naming system for points. Thus, it cannot tell how close a given request is to any point on which it does not currently have a server. See [17] for further discussion of tracklessness.

The Lazy Adversary

The lazy adversary is an adversary that always makes a request that costs it nothing to serve, but which forces the algorithm to pay, if such a request is possible. For the $k$-server problem, the lazy adversary always requests a point where one of its servers is located, provided the algorithm has no server at that point. When all algorithm servers are at the same points as the adversary servers, the lazy adversary may move one of its servers to a new point. Thus, the lazy adversary never has more than one server that is in a position different from that of an algorithm server. Some online algorithms, such as HANDICAP, introduced in Section 4.7, perform better against the lazy adversary than against an adversary without that restriction.

$T$-Theory and its Application to the $k$-Server Problem

Since the pioneering work by Isbell and Dress, there have been many contributions to the field of $T$-theory [3] [4] [5] [22] [23] [24] [41] [44] [45] [46] [47] [48] [56] [58] [59]
The original motivation for the development of T-theory, and one of its most important application areas, is phylogenetic analysis, the problem of constructing a phylogenetic tree showing relationships among species or languages.

It was first discovered by Chrobak and Larmore [26] that T-theory can aid in the competitive analysis of online algorithms for the k-server problem. Since then, work by Telia [79] and Bartal [6], and additional work by Chrobak and Larmore [28] [32] have made use of T-theory concepts to obtain k-server results.

Many proofs of results in the area of the k-server problem require lengthy case-by-case analysis. T-theory can help guide this process by providing a natural way to break a proof or a definition into cases. This can be seen in this paper in the definitions of BALANCE SLACK §4.5.2 and HANDICAP §4.5.3, and in the proof of 3-competitiveness of HARMONIC for $k = 2$, in Section 4.8.

In a somewhat different use of T-theory, the tight span algorithm and EQUIPOISE make use of the virtual server method discussed in Section 4.3. These algorithms move servers virtually in the tight span of a metric space.

Overview of the Paper

In Section 4.2, we give some elementary constructions from T-theory that are used in applications to the server problem. We provide illustrations and pseudo code for a number of algorithms that we describe.

In Section 4.3, we give the virtual server construction, which is used for the tight span algorithm as well as for EQUIPOISE. In §4.4.2, we describe the tree algorithm (TREE) of [29], which forms the basis of a number of the other server algorithms described in this paper. In §4.4.3, we describe the Bartal’s Slack Coverage algorithm for 2 servers in a Euclidean space [6] in terms of T-theory.

In Section 4.5, we discuss balance algorithms for the k-server problem in terms
of $T$-theory. In Section 4.6 we describe the tight span algorithm [26] and \textsc{equipoise} [32] in terms of $T$-theory. In Section 4.7, we present a description of Teia's algorithm \textsc{handicap}. In §4.8.2 we describe how the algorithm \textsc{random slack} is defined using $T$-theory. In §4.8.3, we present a $T$-theory based proof that \textsc{harmonic} [73] is 3-competitive for $k = 2$. In Section 4.9, we discuss possible future uses of $T$-theory for the $k$-server problem.

We present a simplified proof that \textsc{handicap} is $k$-competitive against certain adversaries (Theorem 3), based on the proof in Teia's dissertation [79]. We also give a previously unpublished proof that \textsc{harmonic} is 3-competitive for $k = 2$.

**Elementary T-Theory Concepts**

In keeping with the usual practice of T-theory papers, we extend the meaning of the term metric to incorporate what is commonly called a pseudo-metric. That is, we define a metric on a set $X$ to be a non-negative real valued function $d : X \times X \to \mathbb{R}$ such that

1. $d(x, x) = 0$ for all $x \in X$
2. $d(x, y) = d(y, x)$ for all $x, y \in X$ [Symmetry]
3. $d(x, y) + d(y, z) \geq d(x, z)$ for all $x, y, z \in X$ [Triangle Inequality]

We say that $d$ is a proper metric if, in addition, $d(x, y) > 0$ whenever $x \neq y$. We also adopt the usual practice of abbreviating a metric space $(X, d)$ as simply $X$, if $d$ is understood.

**Injective Spaces and the Tight Span**

Isbell [62] defines a metric space $M$ to be injective if, for any metric space $Y \supseteq M$, there is a non-expansive retraction of $Y$ onto $M$, i.e., a map $r : Y \to M$ which is the identity on $M$, where $d(r(x), r(y)) \leq d(x, y)$ for all $x, y \in Y$. The real line, the
Manhattan plane, i.e., the plane $\mathbb{R}^2$ with the $L_1$ (sum of norms) metric, and $\mathbb{R}^n$ with the $L_\infty$ (sup-norm) metric, where the distance between $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ is $\max_{1 \leq i \leq n} |x_i - y_i|$, are injective. No Euclidean space of dimension more than one is injective.

The tight span $T(X)$ of a metric space $X$, which we formally define below, is characterized by a universal property: up to isomorphism, $T(X)$ is the unique minimal injective metric space that contains $X$. Thus, $X = T(X)$ if and only if $X$ is injective.

Isbell [62] was the first to construct $T(X)$, which he called the injective hull of $X$. Dress [42] independently developed the same construction, naming it the tight span of $X$. Still later, Chrobak and Larmore also independently developed the tight span, which they called the abstract convex hull of $X$.

We now give a formal construction of $T(X)$. Let

$$P(X) = \{ f \in \mathbb{R}^X \mid f(x) + f(y) \geq d(x, y) \text{ for all } x, y \in X \}$$

(4.1)

where $\mathbb{R}^X$ is the set of all functions $f : X \rightarrow \mathbb{R}$, and let $T(X) \subseteq P(X)$ be the set of those functions which are minimal with respect to pointwise partial order. $T(X)$ is a metric space where distance is given by the sup-norm metric, i.e., If $f, g \in T(X)$, we define $d(f, g) = \sup_{x \in X} |f(x) - g(x)|$. If $X$ is finite, then $P(X)$ is also a metric space under the sup-norm metric, and is called the associated polytope of $X$ [64]. There is a canonical embedding\(^2\) of $X$ into $T(X)$. For any $x \in X$, let $h_x \in T(X)$ be the function where $h_x(y) = d(x, y)$ for all $y$. By an abuse of notation, we identify each $x$ with $h_x$, and thus say $X \subseteq T(X)$.

If $X$ has cardinality $n$, then $P(X) \subseteq \mathbb{R}^X \cong \mathbb{R}^n$. For any $x, y \in X$, let $D_{x,y} \subseteq \mathbb{R}^X$ be the half-space defined by the inequality $f(x) + f(y) \geq d(x, y)$, and let $H_{x,y} \subseteq$

\(^2\)In this paper, embedding will mean isometric embedding.
$\mathbb{R}^X$ be the boundary of $D_{x,y}$, the hyperplane defined by the equation $f(x) + f(y) = d(x,y)$, which we call a bounding hyperplane of $P(X)$. Then $P(X) = \bigcap_{x,y \in X} D_{x,y}$ is an unbounded convex polytope of dimension $n$, and $T(X)$ is the union of all the bounded faces of $P(X)$.

The definition of convex subset of a metric space is not consistent with the definition of convex subset of a vector space over the real numbers. $T(X)$ is a convex subset of $P(X)$, if $P(X)$ is considered to be a metric space; but $T(X)$ is not generally a convex subset of $\mathbb{R}^X$ if $\mathbb{R}^X$ is considered to be a vector space over $\mathbb{R}$.

Dress proves [43] that the tight span of a metric space of cardinality $n$ is a cell complex, where each cell is a polytope of dimension at most $\left\lfloor \frac{n}{2} \right\rfloor$. In Figures 21, 22, and 23, we give an example of the tight span of the 3-4-5 triangle. Let $X = \{x, y, z\}$, where $d(x,y) = 3$, $d(x,z) = 4$, and $d(y,z) = 5$. The vertices of $T(X)$, represented as 3-tuples in $\mathbb{R}^3 \cong \mathbb{R}^X$, are

$$
\begin{align*}
    h_x &= (0,3,4) = H_{x,x} \cap H_{x,y} \cap H_{x,z} \\
    h_y &= (3,0,5) = H_{x,y} \cap H_{y,y} \cap H_{y,z} \\
    h_z &= (4,5,0) = H_{x,z} \cap H_{y,z} \cap H_{z,z} \\
    (1,2,3) &= H_{x,y} \cap H_{x,z} \cap H_{y,z}
\end{align*}
$$

Figure 21 shows a projection of $P(X)$ in two dimensions. The boundary of $P(X)$ consists of four vertices, three bounded edges, six unbounded edges, and six unbounded 2-faces. $T(X)$ is the union of the bounded edges. Figure 22 is a perspective showing $T(X)$ in $\mathbb{R}^3$, which we endow with the $L_\infty$ metric. Figure 23 is a rendering of the polytope obtained by intersecting $P(X)$ with a half space.
The Isolation Index and Split Decompositions

Let $(X,d)$ be a metric space. If $A,B \subseteq X$ are non-empty subsets of $X$, Bandelt and Dress [5] (page 54) define the isolation index of the pair $\{A,B\}$ to be

$$\alpha_{A,B} = \frac{1}{2} \min_{a,a' \in A \land b,b' \in B} \left\{ \max \left\{ 0, d(a,b) + d(a',b') - d(a,a') - d(b,b'), d(a,b') + d(a',b) - d(a,a') - d(b,b') \right\} \right\}$$

**Observation 1** $\alpha_{(x),(y,z)} = \frac{d(x,y)+d(x,z)-d(y,z)}{2}$ for any three points $x,y,z$.

A split of a metric space $(X,d)$ is a partition of the points of $X$ into two non-empty sets. We say that a split $A,B$ separates two points if one of the points is in $A$ and the other in $B$. We will use Fraktur letters for sets of splits. If $\mathcal{S}$ is a set of splits of $X$, we say that $\mathcal{S}$ is weakly compatible if, given any four point set $Y \subseteq X$ and given any three members of $\mathcal{S}$, namely $\{A_1,B_1\}$, $\{A_2,B_2\}$, and $\{A_3,B_3\}$, the sets $A_1 \cap Y$, $B_1 \cap Y$, $A_2 \cap Y$, $B_2 \cap Y$, $A_3 \cap Y$, and $B_3 \cap Y$ do not consist of all six two point subsets of $Y$. Figure 24 shows an example of three splits which are not weakly compatible.

If $\alpha_{A,B} = \alpha_{S} > 0$, we say that $\{A,B\} = S$ is a $d$-split of $X$. The set of all $d$-splits of $X$ is always weakly compatible. For more information regarding weak compatibility, see [24]. From Bandelt and Dress [5], we say that $(X,d)$ is split-prime if $X$ has no $d$-splits. For any split $S$ the split metric on $X$ is defined as

$$\delta_{S}(x,y) = \begin{cases} 1 & \text{if } S \text{ separates } \{x,y\} \\ 0 & \text{otherwise} \end{cases}$$

The split decomposition of $(X,d)$ is defined to be

$$d = d_0 + \sum_{S \in \mathcal{S}} \alpha_{S} \delta_{S}$$
where $\mathcal{S}$ is the set of all $d$-splits of $X$, and $d_o$ is called the split-prime residue of $d$. The split decomposition of $d$ is unique. If $d_o = 0$, we say that $d$ is totally decomposable. From Bandelt and Dress [5], we have

**Lemma 1** Every metric on four or fewer points is totally decomposable.

**Observation 2** If $X$ is totally decomposable and $x, y \in X$, then

$$d(x, y) = \sum_{s \text{ separates } (x,y)} \alpha_s \delta_s$$

More generally:

**Observation 3** If $d_o$ is the split-prime residue of $X$, and if $x, y \in X$, then

$$d(x, y) = d_o(x, y) + \sum_{s \text{ separates } (x,y)} \alpha_s \delta_s$$

In Figure 25, we show how the computations of all $d$-splits for the 3-4-5 triangle and the resulting tight span of that space.

**T-Theory and Trees**

The original inspiration for the study of T-theory was the problem of measuring how "close" a given metric space is to being embeddable into a tree. This question is important in phylogenetic analysis, the analysis of relations among species or languages [14][75], since we would like to map any set of species or languages onto a phylogenetic tree which represents their actual descent, using a metric which represents the difference between any two members of the set. We say that a metric space $M$ is a tree if, given any two points $x, y \in M$, there is a unique embedding of an interval of length $d(x, y)$ into $M$ which maps the endpoints of the interval to $x$ and $y$. An arbitrary metric space $M$ embeds in a tree (equivalently, $T(M)$ is a tree) if and only if $M$ satisfies the four point condition [42]:

24
\[ d(u,v) + d(x,y) \leq \max \{ d(u,x) + d(v,y), d(u,y) + d(v,x) \} \quad \text{For any } u,v,x,y \in M \]

\section*{Tight Spans of Finite Metric Spaces}

Two metric spaces \( X_1 \) and \( X_2 \) of the same cardinality are combinatorially equivalent if the tight spans \( T(X_1) \) and \( T(X_2) \) are combinatorially isomorphic cell complexes. A finite metric space of cardinality \( n \) is defined to be generic if \( P(X) \) is a simple polytope, i.e., if every vertex of \( T(X) \) is the intersection of exactly \( n \) of the bounding hyperplanes of \( P(X) \). Equivalently, \( (X,d) \) is generic if there is some \( \varepsilon > 0 \) such that \( T(X,d) \) is combinatorially equivalent to \( T(X,d') \) for any other metric \( d' \) on \( X \) which is within \( \varepsilon \) of \( d \), i.e., if \( |d(x,y) - d'(x,y)| < \varepsilon \) for all \( x,y \in X \).

The number of combinatorial classes of generic metric spaces of cardinality \( n \) increases rapidly with \( n \). There is just one combinatorial class of generic metrics for each \( n \leq 4 \). The tight span of one example for each \( n \leq 4 \) is illustrated in Figure 26. For \( n = 5 \), there are three combinatorial classes of generic metrics. One example of the tight span for each such class is illustrated in Figure 27. There are 339 combinatorial classes of generic metrics for \( n = 6 \), as computed by Sturmfels and Yu [77].

\section*{Motivation for Using T-Theory for the \( k \)-Server Problem}

In Figure 28, we illustrate the motivation, in the case \( k = 2 \), for using T-theory to analyze the server problem. Let \( \varepsilon_1 = \alpha_{\{s_1\},\{s_2,r\}} ', \varepsilon_2 = \alpha_{\{s_2\},\{s_1,r\}} ', \text{ and } \alpha = \alpha_{\{r\},\{s_1,s_2\}} ' \). If \( s_1 \) serves the request, the total distance it moves is \( \varepsilon_1 + \alpha \). We can say that \( \varepsilon_1 \) is the unique portion of that distance, while \( \alpha \) is the common portion. When we make a decision as to which server to move, instead of comparing the two distances to \( r \), we could compare the unique portions of those distances. In Figure 28, we assume that \( s_1 \) serves the request at \( r \). The movement of \( s_1 \) can be thought of as consisting of two phases. During the first phase, \( s_1 \) moves towards both points \( r \) and \( s_2 \). In the second
phase, further movement towards both \( r \) and \( s_2 \) is impossible, so \( s_1 \) moves towards \( r \) and away from \( s_2 \).

In the case that \( k = 2 \), this intuition leads to modification of the Irani-Rubinfeld algorithm, \textsc{balance}2, \cite{61} to \textsc{balance slack}, which we discuss in §4.5.2, and modification of \textsc{harmonic} \cite{73} to \textsc{random slack}, which we discuss in §4.8.2. For \( k > 3 \), the intuition is still present, but it is far less clear how to modify \textsc{balance}2 and \textsc{harmonic} to improve their competitiveness. Teia \cite{79} has partially succeeded; his algorithm \textsc{handicap}, discussed in this paper in Section 4.7, is a generalization of \textsc{balance slack} to all \( k \). \textsc{handicap} is trackless, and is \( k \)-competitive against the lazy adversary for all \( k \). Teia \cite{79} also proves that, for \( k = 3 \), \textsc{handicap} is 157-competitive against any adversary (Theorem 4 of this paper).

The Virtual Server Construction

In an arbitrary metric space \( M \), the points to which we would like to move the servers may not exist. We overcome that restriction by allowing servers to virtually move in \( T(M) \), while leaving the real servers in \( M \). (In an implementation, the algorithm keeps the positions of the virtual servers in memory.)

More generally, if \( M \subseteq M' \) are metric spaces and there is a \( C \)-competitive online algorithm \( A' \) for the \( k \)-server problem in \( M' \), there is a \( C \)-competitive online algorithm \( A \) for the \( k \)-server problem in \( M \). If \( A' \) is deterministic or randomized, \( A \) is deterministic or randomized, respectively. As requests are made, \( A \) makes use of \( A' \) to calculate the positions the servers of \( A' \), which we call virtual servers. When there is a request \( r \in M \), \( A \) calculates the response of \( A' \) and, in its memory, moves the virtual servers in \( M' \). If the \( i \)th virtual server serves the request, then \( A \) moves the \( i \)th real server in \( M \) to \( r \) to serve the request, but does not move any other real server.

We give a formal description of the construction of \( A \) from \( A' \):
Virtual Server Construction

Let \( \{s_i\} \) be the servers in \( M \), and let \( \{s'_i\} \) be the virtual servers in \( M' \).

Let \( s'_i = s_i \) for all \( i \).

Initialize \( A' \).

For each request \( r \):

Move the virtual servers in \( M' \) according to the algorithm \( A' \).

At least one virtual server will reach \( r \). If \( s'_i \) reaches \( r \), move \( s_i \) to \( r \). All other servers remain in their previous positions.

We can assume that the virtual servers match the real servers initially. If a server \( s_i \) serves request \( r^t \) and then also serves request \( r^{t'} \), for some \( t' > t \), then \( s_i \) does not move during any intermediate step. The corresponding virtual server can make several moves between those steps, matching the real server at steps \( t \) and \( t' \). Thus, by the triangle inequality, the movement of each virtual server is as least as great as the movement of the corresponding real server. Thus, \( \text{cost}_A \leq \text{cost}_{A'} \) for the entire request sequence. It follows that the competitiveness of \( A \) cannot exceed the the competitiveness of \( A' \).

Tree Algorithms

The tree algorithm, which we call \textsc{tree}, a \( k \)-competitive online algorithm for the \( k \)-server problem in a tree, occupies a central place in the construction of a number of the online algorithms for the \( k \)-server problem presented in this paper. The line algorithm, \textsc{Double Coverage}, given in §4.4.1 below, is the direct ancestor of \textsc{tree}.

Double Coverage

In [25], Chrobak, Karloff, Payne, and Viswanathan defined a deterministic memoryless fast \( k \)-competitive online algorithm, called \textsc{double coverage} (DC), for the
real line. If a request $r$ is to the left or right of all servers, the nearest server serves. If $r$ is between two servers, they both move toward the $r$ at the same speed and stop when one of them reaches $r$.

Double Coverage

For each request $r$:

- If $r$ is at the location of some server, serve $r$ at no cost.
- If $r$ is to the left of all servers, move the leftmost server to $r$.
- If $r$ is to the right of all servers, move the rightmost server to $r$.
- If $s_i < r < s_j$ and there are no servers in the open interval $(s_i, s_j)$, let $\delta = \min \{r - s_i, s_j - r\}$. Move $s_i$ to the right by $\delta$, and move $s_j$ to the left by $\delta$. At least one of those two servers will reach $r$.

The Tree Algorithm

DC is generalized by Chrobak and Larmore in [29] to a deterministic memoryless fast $k$-competitive online algorithm, $\text{TREE}$, for the $k$-server problem in a tree. We can then extend $\text{TREE}$ to any metric space which embeds in a tree, using the virtual server construction given in Section 4.3.
The Tree Algorithm

Repeat the following loop until some server reaches $r$:

Define each server $s_i$ to be blocked if there is some server $s_j$ such that $d(s_i, r) = d(s_i, s_j) + d(s_j, r)$, and either $d(s_j, r) < d(s_i, r)$ or $j < i$. Any server that is not blocked is active.

For each $i \neq j$, let $\alpha_{i,j} = \alpha_{\{s_i\}, \{s_j\}} = \frac{1}{2}(d(s_i, r) + d(s_i, s_j) - d(s_j, r))$.

If there is only one active server, move it to $r$.

If there is more than one active server:

- Let $\delta$ be the minimum value of all $\alpha_{i,j}$ for all choices of $i, j$ such that both $s_i$ and $s_j$ are active.
- Move each active server a distance of $\delta$ toward $r$.

Assume $M$ is a tree. If $s_1, \ldots, s_k$ are the servers and $r$ is a request, we say that $s_i$ is blocked by $s_j$ if $d(s_i, r) = d(s_i, s_j) + d(s_j, r)$, and either $d(s_j, r) < d(s_i, r)$ or $j < i$. Any server that is not blocked by another server is active. The algorithm serves the request by moving the servers in a sequence of phases. During each phase, all active servers move the same distance towards $r$. A phase ends when either one server reaches $r$ or some previously active server becomes blocked. After at most $k$ phases, some server reaches $r$ and serves the request. Figure 30 illustrates an example step (consisting of three phases) of TREE where $k = 4$. The proof of $k$-competitiveness of TREE makes use of the Coppersmith-Doyle-Raghavan-Snir potential [37], namely

$$\Phi_{\text{CDRS}} = \sum_{1 \leq i < j \leq k} d(s_i, s_j) + 2 \sum_{1 \leq i \leq k} d(s_i, a_i)$$

where $\{a_1, \ldots, a_k\}$ is the set of positions of the optimal servers and $\{s_i \leftrightarrow a_i\}$ is the
minimum matching of the algorithm servers with the optimal servers. We refer the reader to [29] for details of the proof.

More generally, if $M$ satisfies the four point condition given in Inequality (4.2), then $T(M)$ is a tree. We simply use the above algorithm on $T(M)$ to define a $k$-competitive algorithm on $M$, using the method of Section 4.3. We remark that in the original paper describing TREE [29], there was no mention of the tight span construction. The result was simply stated using the clause, “If $M$ embeds in a tree . . . .”

The Slack Coverage Algorithm

Bartal’s Slack Coverage algorithm (SC) is 3-competitive for the 2-server problem in any Euclidean space$^3$ [6].

---

**Slack Coverage**

For each request $r$:

Without loss of generality, $d(s_1, r) \leq d(s_2, r)$.

Let $\delta = \alpha_{\{s_1, s_2, r\}} = \frac{1}{2}(d(s_1, r) + d(s_1, s_2) - d(s_2, r))$.

Move $s_1$ to $r$.

Move $s_2$ a distance of $\delta$ along a straight line toward $r$.

---

The intuition behind SC is that a Euclidean space $E$ is close to being injective. Figure 31 illustrates one step of SC. First, construct $T(X)$, where $X = \{s_1, s_2, r\}$. In $X$, the response of the algorithm TREE would be to move $s_1$ to $r$ to serve the request, and to move $s_2$ a distance of $\delta = \alpha_{\{s_1, s_2, r\}}$ towards $r$, in $T(X)$. SC approximates that move by moving $s_2$ that same distance in $E$ towards $r$. We refer the reader to pages

---

$^3$A parametrized class of Slack Coverage algorithms is described in Borodin and El-Yaniv [21]. Our definition of SC agrees with the case that the parameter is $\frac{1}{2}$. 

30

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
159–160 of [21] for the proof that SC is 3-competitive.4

Balance Algorithms

Informally, we say that a server algorithm is a balance algorithm if it attempts, in some way, to balance the work among the servers. Three algorithms discussed in this paper satisfy that definition: BALANCE2, BALANCE SLACK, and HANDICAP.

**BALANCE2**

The Irani-Rubinfeld algorithm, also called BALANCE2 [61], tries to equalize the total movement of each server. More specifically, when there is a request \( r \), BALANCE2 chooses to move that server \( s_i \) which minimizes \( C_i + 2 \cdot d(s_i, r) \), where \( C_i \) is the total cost incurred by \( s_i \) on all previous moves. BALANCE2 is trackless and needs \( O(k) \) memory.

---

**BALANCE2**

Let \( C_i = 0 \) for all \( i \)

For each request \( r \):

Pick the \( i \) which minimizes \( C_i + 2 \cdot d(s_i, r) \).

\[
C_i = C_i + d(s_i, r).
\]

Move \( s_i \) to \( r \).

---

From [28] and [61] we have

**Theorem 1** The competitiveness of BALANCE2 for \( k = 2 \) is at least 6 and at most 10.

The competitiveness of BALANCE2 for \( k > 2 \) is open.

---

4The slack is defined to be the isolation index in [28], while in [21], slack is defined to be twice the isolation index.
BALANCE SLACK

BALANCE SLACK [28], defined only for $k = 2$, is a modification of BALANCE2. This algorithm tries to equalize the total slack work; namely, the sum, over all requests, of the Phase I costs, as illustrated in Figure 28.

Let $e_i = 0$ for $i = 1, 2$.

For each request $r$:

- $e_1 = \alpha\{s_1, s_2, r\}$
- $e_2 = \alpha\{s_2, s_1, r\}$

Pick that $i$ which minimizes $e_i + e_i$

$e_i = e_i + e_i$

Move $s_i$ to $r$.

We associate each $s_i$ with a number $e_i$, the slack work, which is updated at each move. If $r$ is the request point, let $X = \{s_1, s_2, r\}$, a 3-point subspace of $M$. Let $e_1 = \alpha\{s_1, s_2, r\}$ and $e_2 = \alpha\{s_2, s_1, r\}$, as shown in Figure 32. We now update the slack work values as follows. If $s_i$ serves the request, we increment $e_i$ by adding $e_i$, while the other slack work remains the same. We call $e_i$ the slack cost of the move if $s_i$ serves the request. The algorithm BALANCE SLACK then makes that choice which minimizes the value of $\max\{e_1, e_2\}$ after the move.

BALANCE SLACK is trackless, because it makes no use of any information regarding any point other than the distances between the three active points, namely the points of $X$, but it is not quite memoryless, as it needs to remember one number, viz, $e_1 - e_2$.

From [28] we have

\footnote{It is incorrectly stated on page 179 of [21] that BALANCE SLACK requires unbounded memory.}

32
Theorem 2 BALANCE SLACK is $4$-competitive for $k = 2$.

HANDICAP

Teia's algorithm, HANDICAP [79], is also a balance algorithm, a rather sophisticated generalization of BALANCE SLACK. HANDICAP is defined for all $k$ and all metric spaces, and is $k$-competitive against the lazy adversary. We postpone discussion of HANDICAP until Section 4.7.

Server Algorithms in the Tight Span

The tight span algorithm, TREE, and EQUIPOISE [26] [29] [32] permit movement of virtual servers in the tight span of the metric space. The purpose of using the tight span is that an algorithm might need to move servers to virtual points that do not exist in the original metric space. The tight span, due to the universal property described in §4.2.1, contains every virtual point that might be needed, and no others.

Virtual Servers in the Tight Span

The tight span algorithm, TREE, and EQUIPOISE [26] [29] [32], described in this paper in Sections 4.4 and 4.6, are derived using the embedding $M \subseteq T(M)$, from algorithms defined on $T(M)$. One problem with that derivation is that, in the worst case, $O(|M|)$ numbers are required to encode a point in $T(M)$, which is impossible if $M$ is infinite. Fortunately, we can shortcut the process by assuming the virtual servers are in the tight span of a finite space. If $X \subseteq X'$ are metric spaces and $X' = X \cup \{x'\}$, there is a canonical embedding $\iota: T(X) \subseteq T(X')$ where, for any $f \in T(X)$:

$$(\iota(f))(x) = \begin{cases} f(x) & \text{if } x \in X \\ \sup_{y \in X} \{d(x', y) - f(y)\} & \text{if } x = x' \end{cases}$$

By an abuse of notation, we identify $f$ with $\iota(f)$. In Figure 33, $X$ consists of three points, and $T(X)$ is the union of the solid line segments, while $T(X')$ is the entire
Continuing with the construction, let $s_0^0, \ldots, s_k^0 \in M$ be the initial positions of the servers, and $r^1 \ldots r^n$ the request sequence. Let $X^t = \{s_0^0, \ldots, s_k^0, r^1, \ldots, r^t\}$, a set of cardinality of at most $k + t$, for $0 \leq t \leq n$. Before the $t^{th}$ request all virtual servers are in $T(X^{t-1})$.

Let $A'$ be an online algorithm for the $k$-server problem in $T(M)$ and $A$ the algorithm in $M$ derived from $A'$ using the virtual server construction of Section 4.3. When the request $r^t$ is received, $A$ uses the canonical embedding $T(X^{t-1}) \subseteq T(X^t)$ to calculate the positions of the virtual servers in $T(X^t)$, then uses $A'$ to move the virtual servers within $T(X^t)$. At most, $A$ is required to remember the distance of each virtual server to each point in $X^t$.

**The Tight Span Algorithm**

TREE of §4.4.2 generalizes to all metric spaces in the case that $k = 2$, essentially because $T(X)$ is a tree for any metric space $X$ with at most three points. This generalization was first defined in [29], but was not named in that paper. We shall call it the tight span algorithm. As we did for TREE, we first define the tight span algorithm as a fast memoryless algorithm in any injective metric space. We then use the virtual construction of Section 4.3 to extend the definition of the tight span algorithm to any metric space.

---

**The Tight Span Algorithm**

For each request $r$:

- Let $X = \{s_1, s_2, r\}$.
- Pick an embedding $T(X) \subseteq M$.
- Execute TREE on $T(X)$.
Assume that $M$ is injective, i.e., $M = T(M)$. We define the tight span algorithm on $M$ as follows: let $X = \{s_1, s_2, r\} \subseteq M$. Since $M$ is injective, the inclusion $X \subseteq M$ can be extended to an embedding of $T(X)$ into $M$. Since $T(X)$ is a tree, use \textsc{tree} to move both servers in $T(X)$ such that one of the servers moves to $r$. Since $T(X) \subseteq M$, we can move the servers in $M$. In Figure 40, we show an example consisting of two steps of the tight span algorithm, where $M$ is the Manhattan plane.

Finally, we extend the tight span algorithm to an arbitrary metric space by using the virtual server construction given in Section 4.3. We refer the reader to [26] for the proof of 2-competitiveness for $k = 2$, which also uses the Coppersmith-Doyle-Raghavan-Snir potential.

\textsc{Equipoise}

In [32], a deterministic algorithm for the $k$-server problem, called \textsc{Equipoise}, is given. For $k = 2$, \textsc{Equipoise} is the tight span algorithm of [26] discussed in §4.6.2, and is 2-competitive. For $k = 3$, \textsc{Equipoise} is 11-competitive. The competitiveness of \textsc{Equipoise} for $k \geq 4$ is unknown.
EQUIPOISE

For each request $r$:

Let $G$ be the complete graph whose nodes are $S = \{s_1, \ldots, s_k\}$
and whose edges are $E = \{e_{i,j}\}$.

For each $1 \leq i < j \leq k$, let $w_{i,j} = d(s_i, s_j) + d(s_i, r) + d(s_j, r)$ be the weight of $e_{i,j}$.

Let $E_{MST} \subseteq E$ be the edges of a minimum spanning tree of $G$.

For each $e = e_{i,j} \in E_{MST}$:

Let $T_e$ be the tight span of $\{s_i, s_j, r\}$. Choose an embedding $T_e \subseteq M$.

Emulate $\text{tree}$ on $T_e$ for two servers at $s_i$ and $s_j$ and request point $r$. One of those servers will move to $r$, while the other will move to some point $p_e \in T_e \subseteq M$.

Let $S' = \{r\} \cup \{p_e \mid e \in E_{MST}\}$, a set of cardinality $k$.

Move the servers to $S'$, using a minimum matching of $S$ and $S'$. One server will move to $r$.

Let $M$ be an arbitrary metric space. We first define EQUIPOISE assuming that $M$ is injective. Let $S = \{s_1, \ldots, s_k\}$, the configuration of our servers in $M$, let $r$ be the request point, and let $X = \{s_1, \ldots, s_k, r\}$. Let $G$ be the complete weighted graph whose vertices are $S$ and whose edge weights are $\{w_{i,j}\}$, where $w_{i,j} = d(s_i, s_j) + d(s_i, r) + d(s_j, r)$ for any $i \neq j$. Let $E_{MST}$ be the set of edges of a minimum spanning tree for $G$.

For each $e = e_{i,j} = \{s_i, s_j\} \in E_{MST}$, let $X_e = \{s_i, s_j, r\}$, and let $T_e = T(X_e)$, and choose an embedding $T_e \subseteq M$. We then use the algorithm $\text{tree}$, for two servers, as a subroutine. For each $e = e_{i,j} \in E_{MST}$, we consider how $\text{tree}$ would serve the request $r$ if its two servers were at $s_i$ and $s_j$. It would move one of those servers to $r$, and the
other to some other point in \( M \), which we call \( p_e \). Let \( S' = \{ r \} \cup \{ p_e \mid e \in E_{\text{MST}} \} \), a set of cardinality \( k \). EQUIPOISE then serves the request at \( r \) by moving its servers from \( S \) to \( S' \), using the minimum matching of those two sets. One server will move to \( r \), serving the request. Figure 34 shows one step of EQUIPOISE in the case \( k = 3 \), where \( M \) is the Manhattan plane. By using the virtual server construction of Section 4.3, we extend EQUIPOISE to all metric spaces.

Definition and Analysis of HANDICAP

In this section, we define the algorithm HANDICAP, a generalization of BALANCE SLACK, given initially in Teia’s dissertation [79], using slightly different notation. HANDICAP is trackless and fast, but not memoryless.

The algorithm given in [27] is \( k \)-competitive against the lazy adversary, but only if the adversary is benevolent i.e., informs us when our servers matches his; HANDICAP is more general, since it does not have that restriction.

For \( k \leq 3 \), HANDICAP has the least competitiveness of any known deterministic trackless algorithm for the \( k \)-server problem. The competitiveness of HANDICAP for \( k \geq 4 \) is unknown.

We give a proof that, for all \( k \), HANDICAP is \( k \)-competitive against the lazy adversary; in fact, against any adversary that can have at most one open server, i.e., a server in a position different from any of the algorithm’s servers. This result was proved in [79]. The proof given here is a simplification inspired by Teia [78].

Definition of HANDICAP

HANDICAP maintains numbers \( E_1, \ldots, E_k \), where \( E_i \) is called the handicap\(^6\) of the \( i^{\text{th}} \) server. The handicap of each server is updated after every step, and is used to decide which server moves. The larger a server’s handicap, the less likely it is to move.

\(^6\)In [79], the handicap was defined to be \( H_i \). The value of \( H_i \) is twice \( E_i \).
Since only the differences of the handicaps are used, the algorithm remembers only $k - 1$ numbers between steps.

**HANDICAP**

Let $E_j = 0$ for all $j$.

For each request $r$:

Pick that $i$ for which $E_i + d(s_i, r)$ is minimized.

For all $1 \leq j \leq k$:

$$E_j = E_j + \alpha_{(r),(s_i,s_j)}$$

Move $s_i$ to $r$.

Initially, all handicaps are zero. At any step, let $s_1, \ldots, s_k$ be the positions of our servers, and let $r$ be the request point. For all $1 \leq i, j \leq k$, define $\alpha_{ij} = \alpha_{(r),(s_i,s_j)}$, the isolation index. Choose that $i$ for which $E_i + d(s_i, r)$ is minimized, breaking ties arbitrarily. Next, update the handicaps by adding $\alpha_{ij}$ to $E_j$ for each $j$, and then move the $i$th server to $r$. The other servers do not move. It is a simple exercise to prove, for $k = 2$, that HANDICAP $=\text{BALANCE SLACK}$. Simply verify that $e_1 - e_2 = E_1 - E_2$, and that $e_1 + \varepsilon_1 \leq e_2 + \varepsilon_2$ if and only if $E_1 + d(s_1, r) \leq E_2 + d(s_2, r)$.

Let $a_1, \ldots, a_k$ be the adversary's servers. We assume that the indices are assigned in such a way that $\{s_i \leftrightarrow a_i\}$ is a minimum matching. If $s_i \neq a_i$, we say that $s_i \leftrightarrow a_i$ is an open matching, and $a_i$ is an open server. If $s_i = a_i$ for all $i$, we can arbitrarily designate any $a_i$ to be the open server. We now prove that HANDICAP is $k$-competitive against any adversary which may not have more than one open server, using the Teia potential defined below, a simplification of the potential used in [79].

**Competitiveness of HANDICAP Against the Lazy Adversary**

In order to aid the reader's intuition, we define the Teia potential $\Phi$, to be a sum.
of simpler quantities.

<table>
<thead>
<tr>
<th>Name</th>
<th>Notation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Diversity</td>
<td>$D$</td>
<td>$\sum_{1 \leq i &lt; j \leq k} d(s_i, s_j)$</td>
</tr>
<tr>
<td>$k$-Minimum Matching</td>
<td>$M$</td>
<td>$k \cdot \sum_{i=1}^{k} d(a_i, s_i)$</td>
</tr>
<tr>
<td>Coppersmith-Doyle-Raghavan-Snir Potential</td>
<td>$\Phi_{CDRS}$</td>
<td>$D + M$</td>
</tr>
<tr>
<td>Tension (Spannung) Induced by $s_i$ on ${a_j, s_j}$</td>
<td>$\varepsilon^i_j$</td>
<td>$\alpha_{{s_j, a_j, a_i}}$ for all $i, j$</td>
</tr>
<tr>
<td>Total Tension Induced by $s_i$</td>
<td>$\varepsilon^i$</td>
<td>$\sum_{j=1}^{k} \varepsilon^i_j$ for all $i$</td>
</tr>
<tr>
<td>Net Handicap of $s_i$</td>
<td>$e_i$</td>
<td>$E_i - \varepsilon^i$ for all $i$</td>
</tr>
<tr>
<td>Maximum Net Handicap</td>
<td>$e_{max}$</td>
<td>$\max_{i \leq i \leq k} e_i$</td>
</tr>
<tr>
<td>Handicap Portion of Potential</td>
<td>$\mathcal{H}$</td>
<td>$2 \cdot \sum_{i=1}^{k} (e_{max} - E_i)$</td>
</tr>
<tr>
<td>Teia Potential</td>
<td>$\Phi$</td>
<td>$\Phi_{CDRS} + \mathcal{H}$</td>
</tr>
</tbody>
</table>

We prove that $\Phi$ is non-negative, and that the following update condition holds for each step:

$$\Delta \Phi - k \cdot \text{cost}_{adv} + \text{cost}_{alg} \leq 0 \quad (4.3)$$

where $\text{cost}_{alg}$ and $\text{cost}_{adv}$ are the algorithm's and the adversary's costs for the step, and where $\Delta \Phi$ is the change in potential during that step.

**Lemma 2** $\Phi \geq 0$.

**Proof:** For all $i$:

$$2\varepsilon^i_i = d(s_1, s_i) + d(s_1, a_i) - d(a_i, s_i) \quad \text{by Observation 1}$$

$$\leq d(s_i, s_i) + d(s_i, a_i)$$

39
Thus

\[
\Phi = D + M + H = \sum_{1 \leq i < j \leq k} d(s_i, s_j) + \sum_{1 \leq i < j \leq k} d(s_i, s_j) + 2 \cdot \sum_{i=1}^{k} (e_{\text{max}} - E_i)
\geq \sum_{i=1}^{k} d(s_i, s_1) + k \cdot d(a_1, s_1) + 2 \cdot \sum_{i=1}^{k} (e_{\text{max}} - E_i)
= \sum_{i=1}^{k} (d(s_i, s_i) + d(s_i, a_i) + 2(e_{\text{max}} - e_i - \varepsilon_i))
= \sum_{i=1}^{k} \left( (d(s_i, s_i) + d(s_i, a_i) - 2\varepsilon_i^i) + 2(e_{\text{max}} - e_i) \right)
\geq 0
\]

\[
\square
\]

Every step can be factored into a combination of two kinds of moves:

1. The adversary can move its open server to some other point, but make no request. We call this a cryptic move.

2. The adversary can request the position of its open server, without moving any server. We call this a lazy request.

To prove that Inequality (4.3), the update condition, holds for every step, it suffices to prove that it holds for every cryptic move and for every lazy request.

**Lemma 3** Inequality (4.3) holds for a cryptic move.

**Proof:** We use the traditional \( \Delta \) notation throughout to indicate the increase of any quantity.

Without loss of generality, \( r = a_1 \), the open server. Let \( \hat{a}_i \) be the new position of the adversary’s server. Then \( \Delta D = 0 \), since the positions of the algorithm’s servers do not change, \( \Delta M = k \cdot (d(s_i, \hat{a}_i) - d(s_i, a_1)) \), and \( \Delta e_i = -\Delta \varepsilon_i^i \) for each \( i \). Since \( \Delta e_{\text{max}} \leq \max_i \Delta e_i \), there exists some \( j \) such that
\[ \Delta H \leq -2k \cdot \Delta \varepsilon_i^j \]
\[ = k(d(s_i, a_j) - d(a_i, s_j) - d(s_i, \hat{a}_i) + d(\hat{a}_i, s_j)) \text{ by Observation 1} \]

Thus
\[ \Delta \Phi - k \cdot \text{cost}_{\text{adv}} = \Delta M + \Delta H - k \cdot d(a_i, \hat{a}_i) \]
\[ \leq k(d(\hat{a}_i, s_j) - d(a_i, s_j) - d(a_i, \hat{a}_i)) \]
\[ \leq 0 \]

\[ \square \]

**Lemma 4** *Inequality (4.3) holds for a lazy request.*

*Proof:* Without loss of generality, \( a_i \) is the open server. Then \( r = a_i \), the request point.

Case I: \( s_i \) serves the request.

As illustrated in Figure 36, \( \alpha_{i_1} + \varepsilon_i^i = d(s_i, a_i) \), for all \( i \), \( \Delta D = \sum_{i=2}^{k} (\alpha_{i_1} - \varepsilon_i^i) \), and \( \alpha_{i_1} = d(s_i, a_i) \). Then

\[ \Delta E_i = \alpha_{i_1} \quad \text{for all } i \]
\[ \Delta e_i = \alpha_{i_1} + \varepsilon_i^i = d(s_i, a_i) \quad \text{for all } i \]
\[ \Delta e_{\text{max}} = d(s_i, a_i) \quad \text{since } \min \Delta e_i \leq \Delta e_{\text{max}} \leq \max \Delta e_i \]

Thus
\[ \Delta \Phi + \text{cost}_{\text{alg}} = \Delta D + \Delta M + \Delta H + d(s_i, a_i) \]
\[ = \sum_{i=2}^{k} (\alpha_{i_1} - \varepsilon_i^i) - k \cdot d(s_i, a_i) + 2 \left( k \cdot d(s_i, a_i) - \sum_{i=1}^{k} \alpha_{i_1} \right) \\
+ d(s_i, a_i) \]

41

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[ (k + 1) \cdot d(s_1, a_1) - \sum_{i=2}^{k} (\alpha_{i1} + \varepsilon_i^i) - 2 \cdot \alpha_{11} \]
\[ = 0 \quad \text{since} \quad \alpha_{i1} + \varepsilon_1^i = d(s_1, a_1) = \alpha_{11} \]

Case II: For some \( i > 1 \), \( s_i \) serves the request. Without loss of generality, \( i = 2 \).

Using the carat notation to indicate the updated values after the move, we have \( \hat{s}_2 = a_1, \hat{s}_i = a_2, \hat{s}_i = s_i \) for all \( i > 2 \), and \( \hat{a}_i = a_i \) for all \( i \neq 2 \).

Claim A: \( \Delta e_i = \alpha_{12} \) for all \( i \neq 2 \).

Using Observation 1:

\[
\begin{align*}
\hat{e}_i &= E_i - \varepsilon_i^i \\
\hat{\varepsilon}_i &= E_i + \alpha_{2i} - \varepsilon_i^i \\
\varepsilon_i^i &= \frac{d(s_i, s_i) + d(s_i, a_i) - d(s_i, a_i)}{2} \\
\alpha_{2i} &= \frac{d(s_2, a_i) + d(s_i, a_i) - d(s_i, s_i)}{2} \\
\hat{\varepsilon}_i &= \frac{d(s_i, s_i) + d(s_i, a_i) - d(s_i, s_i)}{2} \\
\alpha_{12} &= \frac{d(s_1, a_i) + d(s_2, a_i) - d(s_1, s_2)}{2}
\end{align*}
\]

Combining the above equations, we obtain \( \hat{e}_i - e_i - \alpha_{12} = 0 \), which verifies Claim A.

Claim B: \( \hat{e}_{\max} = \hat{e}_i \) for some \( i \neq 2 \).

Since HANDICAP moves \( s_2 \), we know that \( E_2 + d(s_2, a_1) \leq E_1 + d(s_1, a_1) \).

\[
\begin{align*}
\hat{e}_1 &= \hat{E}_1 + \alpha_{12} = E_1 + \alpha_{12} \quad \text{by Claim A} \\
\hat{e}_2 &= \hat{E}_2 - \hat{\varepsilon}_1^2 = E_2 + d(s_2, a_1) - \hat{\varepsilon}_1^2
\end{align*}
\]

Thus
\[ \hat{e}_1 - \hat{e}_2 = E_1 + \alpha_{12} - E_2 - d(s_2, a_1) + \tilde{e}_1^2 \geq \alpha_{12} - d(s_1, a_1) + \tilde{e}_1^2 = 0 \]

Since \( \hat{e}_1 \geq \hat{e}_2 \), we have verified Claim B.

We now continue with the proof of Case II of Lemma 4. From Claims A and B, \( \Delta e_{\text{max}} \leq \alpha_{12} \). Recall that \( s_2 = a_2 \) and \( r = a_1 \). Thus \( \alpha_{22} = \alpha_{(r),(s_2,s_2)} = d(s_2, a_1) \).

Then

\[
\Delta M = k(d(s_1, a_2) - d(s_1, a_1)) = k(d(s_1, s_2) - d(s_1, a_1))
\]

\[
\Delta H \leq 2k \cdot \alpha_{12} - 2 \sum_{i=1}^{k} \alpha_{2i}
\]

\[
\Delta \Phi + \text{cost}_{\text{alg}} = \Delta D + \Delta M + \Delta H + d(s_2, a_1)
\]

\[
\leq \sum_{i \neq 2} (d(s_i, a_1) - d(s_i, s_2)) + k(d(s_1, s_2) - d(s_1, a_1)) + 2k \cdot \alpha_{12}
\]

\[
-2 \sum_{i=1}^{k} \alpha_{2i} + d(s_2, a_1)
\]

\[
= k(2\alpha_{12} - d(s_1, a_1) + d(s_1, s_2))
\]

\[
- \sum_{i \neq 2} (\alpha_{2i} - d(s_i, a_1) + d(s_i, s_2)) - 2\alpha_{22} + d(s_2, a_1)
\]

\[
= kd(s_2, a_1) - (k - 1)d(s_2, a_1) - 2d(s_2, a_1) + d(s_2, a_1)
\]

\[
= 0
\]

This completes the proof of Lemma 4, since the left-hand side of the update condition is less than or equal to zero.

\[\square\]

**Theorem 3** HANDICAP is \( k \)-competitive against any adversary which can have at most
one open server.

Proof: Lemma 2 states that the Teia potential is non-negative, while Lemmas 3 and 4 state that the update condition, Inequality (4.3), holds for every step.

Teia also obtains a competitiveness of HANDICAP against any adversary, for \( k = 3 \).

From Section 8.4 of Teia’s dissertation [79], on page 59:

**Theorem 4** For \( k = 3 \), HANDICAP is 157-competitive.

Harmonic Algorithms

In this section, we present the classical algorithm HARMONIC, as well as RANDOM SLACK, an improvement of HARMONIC which uses T-theory. In §4.8.3 we present a T-theory based proof that HARMONIC is 3-competitive for \( k = 2 \).

**HARMONIC**

HARMONIC is a memoryless randomized algorithm for the \( k \)-server problem, first defined by Raghavan and Snir [73] [74]. HARMONIC is based on the intuition that it should be less likely to move a larger distance than a smaller. HARMONIC moves each server with a probability that is inversely proportional to its distance to the request point.
HARMONIC

For each request \( r \):

For each \( 1 \leq i \leq k \), let

\[
p_i = \frac{1}{\frac{1}{d(s_i, r)} + \cdots + \frac{1}{d(s_k, r)}} \quad (4.4)
\]

Pick one \( i \), where each \( i \) is picked with probability \( p_i \).

Move \( s_i \) to \( r \).

HARMONIC is known to be 3-competitive for \( k = 2 \) [30] [35]. Raghavan and Snir [73] [74] prove that its competitiveness cannot be less than \( \binom{k+1}{2} \), which is greater than the best known deterministic competitiveness of the \( k \)-server problem [68] [69]. For \( k > 2 \), the true competitiveness of HARMONIC is unknown but finite [55]. HARMONIC is of interest because it is simple to implement.

RANDOM SLACK

RANDOM SLACK, defined only for two servers, is derived from HARMONIC, but moves each server with a probability inversely proportional to the unique distance that a server would move to serve the request, namely the Phase I cost (see Figure 28.)
RANDOM SLACK

For each request $r$:

$$\varepsilon_1 = \alpha(s_1 \cup \{s_2, r\})$$

$$\varepsilon_2 = \alpha(s_2 \cup \{s_1, r\})$$

Let $p_1 = \frac{\varepsilon_1}{\varepsilon_1 + \varepsilon_2}$

Let $p_2 = \frac{\varepsilon_2}{\varepsilon_1 + \varepsilon_2}$

Pick one $i$, where each $i$ is picked with probability $p_i$.

Move $s_i$ to $r$.

We refer the reader to [28] for the proof that RANDOM SLACK is 2-competitive.

Analysis of HARMONIC using Isolation Indices

The original proof that HARMONIC is 3-competitive for $k = 2$ used T-theory, but was never published. In this section, we present an updated version of that unpublished proof.

We will first show that the lazy potential $\Phi$, defined below, satisfies an update condition for every possible move. In a manner similar to that in the proof of Theorem 3, we first factor all moves into three kinds, which we call active, lazy, and cryptic. After each step, HARMONIC's two servers are located at points $s_1$ and $s_2$, and the adversary's servers are located at points $a_1$ and $a_2$. Without loss of generality, $s_2 = a_2$ is the last request point. In the next step, the adversary moves a server to a point $r$ and makes a request at $r$, and then HARMONIC moves one of its two servers to $r$, using the probability distribution given in Equation (4.4). We analyze the problem by requiring that the adversary always do one of three things:

1. Move the server at $a_2$ to a new point $r$, and then request $r$. We call this an active request.
2. Request a₁ without moving a server. We call this a lazy request.

3. Move the server from a₁ to some other point, but make no request. We call this a cryptic move.

If the adversary moves its server from a₁ to a new point r and then requests r, we consider that step to consist of two moves: a cryptic move followed by a lazy request. Our analysis will be simplified by this factorization.

If \( x, y, z \in M \), we define \( \Phi(x, y, z) \), the lazy potential, to be the expected cost that HARMONIC will pay if \( x = s_1 \), \( y = a_1 \), and \( z = s_z = a_z \), providing the adversary makes only lazy requests henceforth. The formula for \( \Phi \) is obtained by solving the following two simultaneous equations:

\[
\Phi(x, y, z) = \frac{2 \cdot d(x, y) \cdot d(y, z)}{d(x, y) + d(y, z)} + \frac{d(x, y)}{d(x, y) + d(y, z)} \cdot \Phi(x, z, y) \tag{4.5}
\]
\[
\Phi(x, z, y) = \frac{2 \cdot d(x, z) \cdot d(y, z)}{d(x, z) + d(y, z)} + \frac{d(x, z)}{d(x, z) + d(y, z)} \cdot \Phi(x, y, z) \tag{4.6}
\]

Obtaining the solution

\[
\Phi(x, y, z) = \frac{2 \cdot d(x, y)(2 \cdot d(x, z) + d(y, z))}{d(x, y) + d(x, z) + d(y, z)} \tag{4.7}
\]

**Theorem 5** HARMONIC is 3-competitive for 2 servers.

*Proof:* We will show that the lazy potential is 3-competitive. For each move, we need to verify the update condition, namely that the value of \( \Phi \) before the move, plus three times the distance moved by the adversary server, is at least as great as the expected distance moved by HARMONIC plus the expected value of \( \Phi \) after the move. The update condition holds for every lazy request, by Equation (4.5). We need to verify the update inequalities for active requests and cryptic moves.

If HARMONIC has servers at x and z and the adversary has servers at y and z, the update condition for the active request where the adversary moves the server from z
If HARMONIC has servers at $x$ and $z$ and the adversary has servers at $y$ and $z$, the update condition for the cryptic move where the adversary moves the server from $y$ to $r$ is:

$$\Phi(x, y, z) + 3 \cdot d(z, r) - 2 \cdot \frac{d(x, r) \cdot d(z, r)}{d(x, r) \cdot d(z, r)} - \frac{d(x, r)}{d(x, r) \cdot d(z, r)} \cdot \Phi(x, y, r) - \frac{d(z, r)}{d(x, r) \cdot d(z, r)} \cdot \Phi(z, y, r) \geq 0 \quad (4.9)$$

Let $X = \{x, y, z, r\}$. It will be convenient to choose a variable name for the isolation index of each split of $X$. Let:

$$a = \alpha(x, \{y, z, r\})$$
$$b = \alpha(y, \{x, z, r\})$$
$$c = \alpha(z, \{x, y, r\})$$
$$d = \alpha(r, \{x, y, z\})$$
$$e = \alpha(x, y, \{z, r\})$$
$$f = \alpha(x, z, \{y, r\})$$
$$g = \alpha(x, r, \{y, z\})$$

By Lemma 1 and Observation 2, we have
\[ d(x, y) = a + b + f + g \]
\[ d(x, z) = a + c + e + g \]
\[ d(y, z) = b + c + e + f \]
\[ d(x, r) = a + d + e + f \]
\[ d(y, r) = b + d + e + g \]
\[ d(z, r) = c + d + f + g \]

The three non-trivial splits of \( X \) do not form a coherent set; thus, at least one of their isolation indices must be zero. Figure 37 shows the three generic possibilities for \( T(X) \).

Let \( \text{MaxMatch} = \max \{ d(x, y) + d(z, r), d(x, z) + d(y, r), d(x, r) + d(y, z) \} \). If \( \text{MaxMatch} = d(x, y) + d(z, r) \), then \( e = 0 \), as shown in Figure 37(a). If \( \text{MaxMatch} = d(x, z) + d(y, r) \), then \( f = 0 \), as shown in Figure 37(b). If \( \text{MaxMatch} = d(x, r) + d(y, z) \), then \( g = 0 \), as shown in Figure 37(c). In any case, the product \( efg \) must be zero. Substituting the formula for each distance, and using the fact that \( efg = 0 \), we compute the left hand side of Inequality (4.9) to be

\[
\text{numerator}_1 = \frac{(a + b + c + e + f + g)(a + b + d + e + f + g)(b + c + d + e + f + g)(a + c + 2d + e + 2f + g)}{(a + b + c + e + f + g)(a + c + d + e + f + g)(a + c + 2d + e + 2f + g)}
\]

where \( \text{numerator}_1 \) is a polynomial in the literals \( a, b, c, d, e, f, \) and \( g \), given in Appendix A.

Similarly, the left hand side of Inequality (4.10) is

\[
\text{numerator}_2 = \frac{\text{numerator}_1}{(a + b + c + e + f + g)(a + c + d + e + f + g)(a + c + 2d + e + 2f + g)}
\]

where \( \text{numerator}_2 \) is also a polynomial in the literals \( a, b, c, d, e, f, \) and \( g \), given in Appendix A.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The denominators of these rational expressions are clearly positive. The proof that $\text{numerator}_1$ and $\text{numerator}_2$ are non-negative is given in Appendix A. Thus, the left hand sides of both inequalities are non-negative, thus verifying that HARMONIC is 3-competitive for two servers. □

Summary and Possible Future Applications of T-Theory to the k-Server Problem

We have demonstrated the usefulness of T-theory for defining online algorithms for the server problem in a metric space $M$, and proving competitiveness by rewriting the update conditions in terms of isolation indices. In this section, we suggest ways to extend the use of T-theory to obtain new results for the server problem.

Using T-Theory to Generalize RANDOM SLACK

A memoryless trackless randomized algorithm for the $k$-server problem must act as follows. Given that the servers are at $\{s_1, \ldots, s_k\}$ and the request is $r$, first compute $T(X)$, where $X = \{s_1, \ldots, s_k, r\}$ and then use the parameters of $T(X)$ to compute the probabilities of serving the request with the various servers.

We know that this approach is guaranteed to yield a competitive memoryless randomized algorithm for the $k$-server problem, since HARMONIC is in this class. HARMONIC computes probabilities using the parameters of $T(X)$, but we saw in RANDOM SLACK in §4.8.2 that, for $k = 2$, a more careful choice of probabilities yields an improvement of the competitiveness. We conjecture that, for $k \geq 3$, there is some choice of probabilities which yields an algorithm of this class whose competitiveness is lower than that of HARMONIC.

Using T-Theory to Analyze HARMONIC for Larger $k$

We know that the competitiveness of HARMONIC for $k = 3$ is at least $\binom{4}{2} = 6$ [73] [74]. As in Section 4.8, we could express the lazy potential in closed form, and then
attempt to prove that it satisfies all necessary update conditions.

In principle, the process of verifying that the lazy potential suffices to prove 6-competitiveness for HARMONIC for $k = 3$ could be automated, possibly using the output of Sturmfels and Yu's program [77] as input. However, the complexity of the proof technique used in Section 4.8 rises very rapidly with $k$, and may be impractical for $k > 2$. There should be some way to simplify this computation.

Generalizing the Virtual Server Algorithms and the $k$-Server Conjecture

The $k$-server conjecture remains open, despite years of effort by many researchers. The most promising approach to date appears to be the effort to prove that the work function algorithm (WFA) [31], or perhaps a variant of WFA, is $k$-competitive. This opinion is explained in depth by Koutsoupias [66]. To date, for $k \geq 3$, it is only known that WFA is $(2k - 1)$-competitive [69], and that it is $k$-competitive in a number of special cases.

HANDICAP represents a somewhat different approach to the $k$-server problem. Teia conjectures that HANDICAP can be modified in such a way as to obtain a 3-competitive deterministic online algorithm for the 3-server problem against an arbitrary adversary, thus settling the server conjecture for $k = 3$. He suggests that this can be done by maintaining two reference points in the tight span. The resulting algorithm would not be trackless.

From the introduction (pp. 3-4) of Teia's dissertation [79]:

For the case of more than one open matching, the memory representation would have to be augmented by additional components. One possibility would be to introduce reference points in addition to handicaps. We are convinced that, for $k = 3$, by careful case analysis and the introduction of two reference points, a 3-competitive algorithm can be given.
Für den Fall mehr als eines offenen Matchings müßte die Gedächtnisrepräsentation um zusätzliche Komponenten erweitert werden. Eine Möglichkeit wäre, zusätzlich zu den Handicaps Bezugspunkte einzuführen. Wir sind überzeugt, daß sich für $k = 3$ durch sorgfältige Fallunterscheidungen und die Einführung zweier Bezugspunkte ein 3-kompetitiver Algorithmus angeben läßt.
BIBLIOGRAPHY


60

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

MATHEMATICA CALCULATIONS

We used Mathematica 5.2 to rewrite the left hand side of Inequality (4.9) as a single rational expression with the least common denominator. We then substituted zero for $efg$ throughout. Then $\text{numerator}$, the numerator of the resulting rational expression, is the following polynomial:

$$
\begin{align*}
4a^3bc + 9a^2b^2c + 5ab^3c + 4a^3c^2 + 14a^2bc^2 + 13ab^2c^2 + 3b^3c^2 + 5a^2c^3 + 9abc^3 + 4b^3c^3 + ac^4 + bc^4 + 2a^2bd + 3a^2b^2d + ab^3d + 6a^3cd + 23a^2bcd + 24ab^2cd + 7b^3cd + 17a^2c^2d + 36abc^2d + 17b^2c^2d + 10ac^3d + 9bc^3d + c^4d + 2a^3d^2 + 7a^2bd^2 + 5ab^2d^2 + 18a^2cd^2 + 37abcd^2 + 17b^2cd^2 + 21ac^2d^2 + 20bc^2d^2 + 5c^3d^2 + 6a^2d^3 + 10abd^3 + 4b^2d^3 + 16acd^3 + 16bcd^3 + 8c^2d^3 + 4ad^4 + 4bd^4 + 4c^3d + 17a^2bce + 17a^2bce + 13a^2ce^2 + 26abcde + 25a^2cde + 50abcde + 21b^2cde + 30ac^2de + 26bc^2de + 7c^3de + 10a^2d^2e + 18abd^2e + 6b^2d^2e + 37abcd^2e + 34bcd^2e + 16c^2d^2e + 14ad^3e + 14bd^3e + 4d^4e + 8a^2ce^2 + 16abcce^2 + 6b^2cc^2 + 10ac^2ce^2 + 7bc^2e^2 + 2c^3e^2 + 6a^2d^2e^2 + 10abd^2e^2 + 2b^2d^2e^2 + 22acd^2e^2 + 17bcde^2 + 9c^2de^2 + 12ad^2e^2 + 10bd^2e^2 + 13cd^2e^2 + 6d^2e^2 + 4ace^3 + 2bce^3 + c^2e^3 + 4ade^3 + 2bde^3 + 3cde^3 + 2d^3e^3 + 2a^3bf + 4a^2b^2f + 2ab^3f + 6a^3cf + 30a^2bcf + 33ab^2cf + 8b^3cf + 22a^2ce^2 + 46abc^2f + 21b^2ce^2f + 13ac^3f + 12bc^3f + c^4f + 4a^3df + 16a^2bdf + 13ab^2df + 2b^3df + 39a^2cdf + 88abcdf + 43b^2cdf + 54ac^2df + 54bc^2df + 13c^3df + 16a^2d^2f + 29abd^2f + 10b^2d^2f + 59acd^2f + 59bcd^2f + 31c^2d^2f + 18ad^3f + 16bd^3f + 24cd^3f + 4d^4f + 2a^3ef + 9a^2b^2ef + 8ab^2ef + 5ab^2ef + 30a^2ce^2f + 67abc^2f + 30b^2ce^2f + 42ac^2ef + 39bc^2ef + 10c^3ef + 20a^2def + 41abcdef + 14b^2def + 90acdef + 85bcdef + 45c^2def + 42ad^2ef + 39bd^2ef + 58cd^2ef + 22d^3ef + 5a^2ce^2f + 9abc^2f + 2b^2ef + 30ace^2f + 26bce^2f + 15c^2e^2f + 26ade^2f + 21bc^2ef + 37cde^2f + 23d^2ef + 3ae^3f + be^3f + 6c^3f + 7de^3f + 2a^3f^2 + 10a^2bf^2 + 10ab^2f^2 + 2ab^2f^2 + 23a^2ef^2 + 57abc^2f + 28b^2ef^2 + 35ac^2f^2 + 35bc^2f^2 + 8c^2f^2 + 16a^2d^2f + 32abcd^2f + 12b^2df^2 + 70acdf^2 + 74b^2df^2 + 40c^2df^2 + 28ad^2f^2 + 24bd^2f^2 + 46cd^2f^2 + 12d^2f^2 + 11a^2ef^2 + 25abc^2f + 10b^2ef^2 + 57ace^2f + 58bc^2ef^2 + 31c^2ef^2 + ... 
\end{align*}
$$

61
Since there are no negative coefficients in the polynomial \textit{numerator}_1, its value must be non-negative.

Similarly, we used Mathematica to find a polynomial expression for \textit{numerator}_2, the numerator of the left hand side of Inequality (4.10):

\[4a^2b + 4ab^2 + 9abc + 4b^2c + 6bc^2 + 2a^2d + 6abd + 4b^2d + 3acd + 6bcd + 2ad^2 + 2bd^2 + 2cd^2 + 2a^2e + \]

Each of the variables \(a, b, c, d, e, f, g\) is an isolation index, hence cannot be negative.
11abe + 4b^2e + 2ace + 12bce + 6ade + 9bde + 3cde + 2d^2e + 3ae^2 + 7be^2 + ce^2 + 3de^2 + e^3 + 8abf + 
4b^2f + 8bcf + 4adf + 6bdf + 4cdf + 2d^2f + 5acf + 11bce + 3cde + 7def + 4e^2f + 4bf^2 + 2df^2 + 3e^2f + 
4a^2g + 12abg + 4b^2g + 10acg + 15bce + 6c^2g + 7adg + 9bdg + 6cdg + 2d^2g + 12aeg + 15beg + 12ceg + 
9deg + 7c^2g + 7afg + 11bfg + 9cfg + 7dfg + 3f^2g + 9ag^2 + 9bg^2 + 11cg^2 + 5dg^2 + 11eg^2 + 8fg^2 + 5g^3

Every term is non-negative, hence \( \text{numerator}_2 \) is non-negative.
Figure 1: Example 2 Path Problem with Edge Weights Labeled
Figure 2: 2 Path Problem In Terms Of End Vertices and Illustration of Order of Steps on How to Work the Dynamic Program for a Dense Acyclic Graph
Figure 3: 2 Path Problem In Terms Of End Vertices. Example of a Sparse Acyclic Graph, as in Figure 1
Figure 4: Illustration of the complexity of the 2 Path Problem as it grows. This figure has 21 vertices.
Illustration of Path Endings

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>S,S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>S,A</td>
<td>A,A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>S,B</td>
<td>A,B</td>
<td>B,B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>S,C</td>
<td>A,C</td>
<td>B,C</td>
<td>C,C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>S,D</td>
<td>A,D</td>
<td>B,D</td>
<td>C,D</td>
<td>D,D</td>
</tr>
</tbody>
</table>

Figure 5: 2 Path Problem Matrix view of Path Endings, where $n = 5$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table 1: Weights of a particular 2 Path Problem, where $n = 5$.

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>
SS -> SA -> AB -> AC -> AD
Path 1: S -> A
Path 2: S -> B -> C -> D

Figure 6: Illustrated Calculation of Table 1
Figure 7: Initial Position of Online 2 Path Problem
Figure 8: Example of Positions of Path Endings in Online 2 Path Problem
Figure 9: Graph used in Optimality Proof
Figure 10: Shortest Path: Initial Graph
Figure 11: Shortest Path: From vertex 0 to 1
Figure 12: Shortest Path: From vertex 0 to 2
Figure 13: Shortest Path: From vertex 0 to 3
Figure 14: Shortest Path: From vertex 0 to 4

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 15: Shortest Path: From vertex 0 to 5
Figure 16: Shortest Path: From vertex 0 to 6
Figure 17: The Knowledge State Family $B_{xyz}$ and its required area of validity.
Figure 18: One example combination of Knowledge States and its coverage for Bxyz.
Figure 19: Example calculation of Excess.
Figure 20: Illustration of $L_1$, $L_2$, and $L_\infty$ metrics on the plane. The distance between $x$ and $y$ is 9 with the $L_1$ metric, $\sqrt{41}$ with the $L_2$ metric and 5 with the $L_\infty$ metric.
Figure 21: A two-dimensional projection of the three-dimensional complex $P(X)$, in the case where $X$ is the 3-4-5 triangle. $T(X)$ is the subcomplex consisting of the vertices and the bold line segments.
Figure 22: A view of the tight span of the 3-4-5 triangle, embedded in \((\mathbb{R}^3, \mathcal{L}_\infty)\).

Figure 23: A view of the associated polytope of the 3-4-5 triangle, with the tight span in bold.
Figure 24: Three splits which are not weakly compatible.
Figure 25: Step-by-step calculation of the tight span of the 3-4-5 triangle.
Figure 26: Examples of the decomposition given by Observation 2 of metrics on four or fewer points. In the upper figures, distances between points are shown. The lower figures show the tight spans, where the edge lengths are isolation indices.
Figure 27: Examples of tight spans for the three generic cases of spaces with five points. Observation 2 applies only to the first case; Observation 3 applies to all cases.
Figure 28: The movement phases of $s_1$ to $r$. 

Phase I

Phase II
Figure 29: The DOUBLE COVERAGE algorithm.
Figure 30: The phases of one step of TREE.
Figure 31: One step of Bartal's Slack Coverage algorithm in a Euclidean space: $s_1$ serves, and $s_2$ moves $\alpha_{(s_1),(s_2,r)} = 7$ towards $r$.
Figure 32: Computing the moves for BALANCE SLACK and RANDOM SLACK.
Figure 33: The inclusion $\iota : T(X) \subseteq T(X')$ where $|X| = 3$ and $X' = X \cup \{x'\}$. 
Figure 34: One step of EQUIPOISE in the Manhattan plane

- Figure 34(a) shows the computation of \( w_{12} \).
- Figure 34(b) shows the computation of \( w_{13} \).
- Figure 34(c) shows the computation of \( w_{23} \).
- Figure 34(d) shows the weighted graph \( G \).
- Figure 34(e) indicates \( E_{MST} = \{e_{13}, e_{23}\} \), the minimum spanning tree of \( G \).
- Figure 34(f) shows \( T(X) \), with the two-dimensional cell of \( T(X) \) shaded.
- Figure 34(g) shows \( T_{e_{13}} \), and the position \( s_3 \) would move to if \( \text{TREE} \) for two servers were executed on \( T_{e_{13}} \).
- Figure 34(h) shows \( T_{e_{23}} \), and the position \( s_2 \) would move to if \( \text{TREE} \) for two servers were executed on \( T_{e_{23}} \).
- Figure 34(i) shows the minimum matching movement of \( S \) to \( S' \).
- Figure 34(j) shows the positions of the three servers after completion of the step.
Figure 35: Definitions of $\alpha_{ij}$ and $\varepsilon_j^i$ for HANDICAP.
Figure 36: Illustration of the proof of Lemma 4.
Figure 37: Three possible pictures of the tight span of $X$. 

(a) 

(b) 

(c)
Figure 38: $K_{2,3}$
Figure 39: Implication of properties
Figure 40: The Tight Span Algorithm in the Manhattan Plane
\begin{tabular}{|c|c|c|c|c|}
\hline
0 & 2 & 1 & 1 & 1 \\
\hline
2 & 0 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 2 & 2 \\
\hline
1 & 1 & 2 & 0 & 2 \\
\hline
1 & 1 & 2 & 2 & 0 \\
\hline
\end{tabular}
Local Address:
P.O. Box 80402
Las Vegas, Nevada 89180

Degrees:
Bachelor of Arts, Computer Science, 2005
University of Nevada, Las Vegas

Master of Science, Computer Science, 2007
University of Nevada, Las Vegas

Publications:

Thesis Title:
T-Theory and Analysis of Online Algorithms

Thesis Examination Committee:
Chairperson, Lawrence L. Larmore, Ph. D.
Committee Member, Ajoy Datta, Ph. D.
Committee Member, John Minor, Ph. D.
Graduate Faculty Representative, Angel Muleshkov, Ph. D.