Competitive Algorithms for On-line Problems and Their Application to Mobile Agents

By

Ma Weimin

A Thesis Submitted to

The Hong Kong Polytechnic University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Philosophy

[Temporary Binding for Examination Purposes]

Department of Computing

The Hong Kong Polytechnic University

March 2002

Abstract of the thesis entitled "Competitive Algorithms for On-line Problems and Their Application to Mobile Agents".

Submitted by Ma Weimin for the degree of Master of Philosophy at the Hong Kong Polytechnic University, March 2002.

Abstract

Most traditional optimization theories deal with finding an optimal solution to a problem with initial conditions that are known and fixed. If the initial conditions change, most solutions found using these algorithms will not remain optimal. Researchers investigating competitive algorithms for on-line problems are exploring strategies for producing solutions that are proportional to the optimal solution (within a certain range), even in the worst cases. In contrast to the conventional (model-based) on-line approaches and the existing off-line strategies for optimization, we propose several new on-line competitive algorithms. These algorithms apply competitive analysis to on-line performance evaluation. In these proposed algorithms, we focus on several on-line problems, which include: (1) the on-line k-truck problem, (2) the on-line k-sever problem with twin-request, and (3) the on-line number of snacks problem. For each of these problems, several research steps are involved: (1) to establish and describe a model of the problem; (2) to construct a relevant on-line algorithm to solve the problem; (3) to derive a competitive ratio, which is an important measure of an on-line algorithm; (4) to provide a report on performance analysis and preliminary results; (5) to describe future work on the application of the proposed algorithm to task scheduling for systems based on mobile agents.

With rapid growth in the use of on-line systems, on-line algorithms are playing an increasingly important role in decision-making for higher performance. A general on-line problem requires an optimized solution to either maximize the benefit or minimize the cost based on the existing information associated with the system. There are three key issues to consider: (1) the identification/specification of an on-line problem, (2) the decision strategy of an on-line algorithm, and (3) the performance measurement used for optimization. In this thesis, three new kinds of on-line problems are proposed, and several competitive algorithms (with good competitive ratios) are provided to solve these problems. Application of these algorithms to systems with mobile agents is also discussed.

Based on the existing results concerning on-line problems and competitive algorithms, we first propose the on-line k-truck problem, which is an extension of the famous k-server problem. With the help of the Position Maintaining Strategy (PMS), we give different competitive algorithms according to the different cases of the k-truck problem. Next, we extend the k-server problem. Traditionally, the k-server problem assumes that at any given time there is only one request occurring, and no new request can occur until the previous request has been satisfied. Based on this observation, we propose the concept of a k-sever problem with multi-request. Specifically, we focus on the k-server problem with twin-request and employ the work function algorithm to obtain some minor results. We also give a lower bound for the k-server problem with twin-request, but it needs further improvement. Next, the on-line number of snacks problem is proposed, which is a realistic noshery management problem. We discuss four versions of this problem, and also provide relevant competitive algorithms. This is followed by analysis and evaluation of the algorithms for the on-line number of snacks problem. Next, we employ the theory of on-line problems and competitive algorithms to handle the dynamic allocation of mobile agents in an on-line task-scheduling problem. A new approach is presented to solve this problem for high performance in Internet computing. Of course, there are many unsolved problems concerning on-line problems and competitive algorithms. How to combine the existing theories with more realistic problems is still a challenging question. Finally, we discuss future work and further research directions.

Keywords: on-line problem, competitive algorithm, *k*-server problem, *k*-truck problem, on-line number of snacks problem, on-line *k*-server problem with twin-request, dynamic allocation of mobile agents.

Acknowledgement

The completion of this thesis would not have been possible without the help of many teachers and friends, to whom I would like to express my heartfelt appreciation.

First, I would like to express my deepest thanks to my supervisor, Dr. Jane You. I gratefully acknowledge her help in adopting a correct attitude to my academic research, in choosing a suitable direction for my research, and in improving my writing of more professional academic documents. I would like to thank her for her kind supervision and continuous support during my M. Phil study at the Hong Kong Polytechnic University. I believe that I learned a lot from her, both in terms of the technology involved and in terms of a general research methodology. I also believe that my experience as her student for two years will be of great benefit to me throughout my life.

Another excellent person, whom I would like to express my deep gratitude to, is Dr. James Liu. Although he was not my supervisor or co-supervisor, we had many discussions that were helpful to me and he gave further useful suggestions when I wrote papers and this thesis. On each occasion, he read my draft documents carefully and usually found a constructive way to correct any errors using his acuminous insight. Without his help, I believe that I would not have had the opportunity to finish two journal papers as well as this thesis.

Next, I would like to express my heartfelt thanks to Prof. Xu Yingfeng, the supervisor of my Master and Doctor of Management studies at Xi'an Jiaotong University. He gave me many valuable suggestions for my academic research even though I was studying in Hong Kong. He introduced me to the subject of on-line problems and competitive algorithms.

I also would like to thank Dr. Raymond Lee, Mr. Xinfeng Yang, Mr. Hui Chen, and Mr. Yanchao Xing for useful discussions and suggestions. In particular, I need thank Mr. Xinfeng Yang and Mr. Hui Chen since some parts of this thesis have benefited from their work.

I would like to express my thanks to Prof. Xizhao Wang, Prof. Kanliang Wang, Prof. Dianhui Wang, and Prof. Minghu Ha, who shared with me not only their expertise in computational linguistics and mathematics, but also their enjoyment of life. We had many useful discussions that helped me towards finishing this thesis.

Also I would like to express my thanks to Dr. Fengwu Shen, Dr. Lihui Sun, and Dr. Zhijun Zhu, my buddies in Xi'an Jiaotong University, who generously made available to me useful discussions concerning the on-line problems and competitive algorithms. Without their help, my research work would surely have suffered.

I would like to thank many of my friends in Hong Kong Polytechnic University: Prof. Dapeng Zhang, Dr. Jiannong Cao, Dr. Allan Wang, Dr. Huaping Chen, Dr. Kuanquan Wang, Mr. Renguo Xiao, and Mr. Ruifeng Xu, who continuously encouraged me during the past two years.

In addition to those involved directly with my research work, I would like to thank many of my friends: Ms. Yan Li, Ms. Ye Yang, Ms. Sheng Liang, Mr. Guoqing Cao, and Mr. Lei Zhang, who share the pleasure of life with me.

At last, but not the least, I wish to express my deepest appreciation to my parents, and especially to my dearest wife and daughter, for their endless love and unwavering support. This thesis is dedicated to them.

Table of Contents

Chapter 1 1

Introduction 1

1.1 Motivation and Objectives 1

1.2 Preliminary 5

1.2.1 Basic Concepts 5

1.2.2 A Simple Example 7

1.2.3 Application of the On-line Algorithm 9

1.3 Literature Review of On-line Algorithms 10

1.4 Structure of Thesis 14

1.5 List of Contributions 15

1.6 List of Publications 17

Chapter 2 19

An Overview of the *k*-Server Problem 19

2.1 The *k*-Server Problem 19

2.2 Literature of the *k*-Server Problem 19

2.3 Some Important Results on the *k*-Server Problem 20

Chapter 3 23

The *k*-Truck Problem 23

3.1 Problem Statement 23

3.2 The Model of the *k*-Truck Problem 25

3.3 Relationship Between the Server and Truck Problems 26

3.4 Several Lemmas and Position Maintaining Strategy 27

3.4.1 Two Lemmas 27

3.4.2 Position Maintaining Strategy 28

3.5 Off-line *k*-Truck Problem 29

3.5.1 Dynamic Programming (DP) Solution 30

3.5.2 Minimum Cost Maximum Flow (MCMF) Solution 32

Chapter 4 35

Computational Analysis of the k-Truck Problem 35

4.1 The Results Concerning the Competitive Ratio for the k-Truck Problem P 35

4.2 Discussion on the Constraint Graphs 39

4.3 Scheduling k Trucks on a Special Graph 41

4.4 Comparison of the Two Algorithms for Solving Problem $P1$ 44

4.5 A Lower Bound for the k-truck Problem 45

4.6 Concluding Remarks 50

Chapter 5 51

The k-Server Problem with Twin-Request 51

5.1 Introduction 51

5.2 Preliminaries for the k-Server Problem with Twin-Request 55

5.2.1 An Optimal Off-line Algorithm 58

5.2.2 Work Function Algorithm 59

5.3. The Special Case of a Metric Space with $k+2$ Points 65

5.4 A Lower Bound 68

5.5. Open Problems 72

Chapter 6 74

On the On-line Number of Snacks Problem 74

6.1 Introduction 74

6.2 The Number of Snacks Problem 76

6.2.1 Problem Statement 76

6.2.2 Assumptions about the Fluctuation in the Number of Customers 79

6.2.3. Results 79

6.3 The Degenerate Version of P — Problem $P1$ 81

6.3.1 Extreme Numbers Harmonic Algorithm 81

6.3.2 Competitive Ratio of the Extreme Numbers Harmonic Algorithm 81

6.3.3 A Lower Bound of the Competitive Ratio 83

6.3.4 The Upper Bound of the Competitive Ratio 84

6.4 The General Version — Problem P 84

6.4.1 Transformative Extreme Numbers Harmonic Algorithm 84

6.4.2 Competitive Ratio 85

6.4.3 Lower Bound 85

6.5 The Degenerate Special Version of P' — Problem P'' 86

6.5.1 Local Extreme Numbers Harmonic Algorithm 86

6.5.2 Competitive Ratio 86

6.5.3 A Lower Bound of the Competitive Ratio 88

6.6 The Special Version — Problem P' 88

6.6.1 Transformative Local Extreme Numbers Harmonic Algorithm 88

6.6.2 Competitive Ratio 89

6.6.3 A Lower Bound of Competitive Ratio 89

6.7 Evaluation of the Results 89

6.7.1 Evaluation of the Competitive Ratios 89

6.7.2 Comparison of Problems $P1$ and P'' 90

6.8 Conclusions and Future Work 91

Chapter 7 93

Dynamic Allocation of Mobile Agents by On-line Task Scheduling 93

7.1 Introduction 93

7.2 Agent-oriented Computing Paradigm 96

7.3 Multi-agent System Structure 99

7.4 Online Task-Scheduling Algorithm 103

7.4.1 General Harmonic Algorithm (GHA) 105

7.4.2 Competitive Ratio 106

7.4.3 Lower Bound for the Competitive Ratio 107

7.5 Two More Corollaries 109

7.6 Conclusions 111

Chapter 8 112

FUTURE STUDY 112

8.1 For the *k*-Server Conjecture 112

8.2 Other On-line Problems 112

8.3 Application to Management and Economic Problems 112

8.4 Continue Research into the *k*-Truck Problem 113

8.5 Application to Agent-based E-commerce 113

BIBLIOGRAPHY 114

List of Figures

Figure 1.1. On-line 2-server problem on a 3-node graph....................................7

Figure 6.1. The curve of competitive ratio with the fluctuation Φ..........................90

Figure 6.2. The difference of the problem $P1$ and problem P''(n days game).........91

Figure 7.1. Aglet major life events...98

Figure 7.2. Multi-agent system structure enhanced by online task-scheduling algorithm 100

Figure 7.3. Relation between d/m and p/q..109

Figure 7.4. Relation between α and ϕ...111

Figure 7.5. Relation between ϕ and d/m...112

Chapter 1

Introduction

The purpose of this research is to study several on-line problems and their associated competitive algorithms in order to enrich the relevant on-line algorithm theories and to exploit some wider and more realistic applications. In this chapter, we explain the motivation and objectives of this research, provide introductory knowledge on the relevant topics, and present an overview of this thesis.

Section 1.1 explains the motivation for conducting this research work on on-line problems and competitive algorithms and describes the objectives of this thesis. Some preliminary information concerning on-line problems and competitive algorithms is presented in section 1.2, and section 1.3 briefly reviews the existing literature. The organization of this thesis is outlined in section 1.4. Next, the contributions of this research are summarized in section 1.5. Finally, the publications arising from this research are listed in section 1.6.

1.1 Motivation and Objectives

With the discovery of more on-line problems, and deeper research into competitive algorithms, many results concerning on-line algorithms theory have been applied in many different fields; for example, management, military affairs, computing, economics, and so on. Decision-making can be considered in two different contexts:

making decisions with complete information, and making decisions based on partial information. A major reason for the study of different algorithms is to try to answer the question "which is the better algorithm" for solving a particular problem. The study of the computational complexity of algorithms is useful for distinguishing between characteristics of algorithms, in terms of the computational resources used and the quality of the solutions they compute. However, the computational complexity of algorithms may be irrelevant, or a secondary issue, when dealing with algorithms that operate in an uncertain environment. Competitive analysis has been found useful in the study of algorithms that operate in a state of uncertainty.

Most people have this kind of experience when dealing with some general issues: they say to themselves after the issue has happened "if only I had known *that* before, then I could have done better". In fact, if one knows enough conditions concerning an issue one can always find an optimal solution, at least theoretically (for many NP-hard problems, here the word "theoretically" was used). A famous person once said, "If one could grow from eighty to one, instead of from one to eighty, most people of this world would be great people". Why is that the case? This is because people could see the whole course of their life, and this foreknowledge would allow them to make the best decision. Like this example, most traditional optimization theories deal with finding the optimal solution to a problem when the known conditions are fixed. Optimality in most cases is lost when those initial conditions change. Research into on-line problems and competitive algorithms aims at exploring strategies that can produce solutions that are proportional to the optimal solution (within a certain range), even in the worst cases.

If we were to compare two arbitrary algorithms, A and B, on an instance-by-instance basis, we could expect that sometimes A is better than B, and sometimes B is better than A. Thus, such a comparison would not impose a total ordering on the quality of the two algorithms, and it would not answer the basic question, "Which is the better algorithm?" There are many ways to choose a quality function that imposes a total ordering on these algorithms. One obvious possibility is simply to consider the worst-case behavior of an algorithm; another is to make some assumption on the input distribution. As discussed later, both these measures are problematic, especially in the context of algorithms that make decisions when conditions are uncertain.

Competitive analysis is useful in the analysis of systems that have some notion of temporal progression, that have an environment, and that also have a memory state. That is, there is some notion of a configuration that varies over time, and dealing with future changes in the system's configuration depends in some way on the current configuration. There are a great many problems that can be phrased in this manner, and it does not matter whether time is an inherent part of the problem. For example, in stock transactions the timing of events is important: the bid/asking price of a share is valid for a very short period of time. In Lunar exploration, using a mobile probe, the timing of events depends on the probe's behavior; for example, an impassable crevice in the lunar terrain is revealed only after going around a boulder. If the probe does not go around the boulder then it will not learn of the problem.

Competitive analysis is certainly useful for "on-line algorithms" that have to respond to events over time (e.g., fluctuating stock prices), but it can be used in many other contexts as well. Competitive analysis is used whenever the nature of a problem concerns making decisions with incomplete information. Information can be incomplete because some event has not yet occurred (e.g., stock price tomorrow at noon is unknown now), or obtaining the missing information requires action by the algorithm (e.g., move the probe around a boulder), or it could be because an algorithm is distributed and no single node has global information. Nonetheless, the largest body of work involving competitive analysis deals with on-line problems, to the extent that many uses of competitive analysis are mistakenly referred to as on-line.

Despite the existence of many unsolved on-line problems, and unresolved issues concerning competitive algorithms, there are many real life on-line problems yet to be discovered. Therefore further research can follow two directions, one of which is to solve the current open problems, and the other is to discover new on-line problems. New problems have to be modeled, and competitive algorithms have to be constructed to solve them. In summary, the main objectives of this research/thesis are to:

- Discover some new on-line problems or transform exiting on-line problems.
- Propose some relevant on-line algorithms to solve these problems.
- Prove that these proposed on-line algorithms are competitive algorithms and obtain the relevant competitive ratio.
- Evaluate the performance of the competitive algorithms according to their competitive ratio.
- Apply the theories of on-line problems and competitive algorithms to mobile agents.

1.2 Preliminary

In this section, some introductory knowledge concerning on-line problems and competitive algorithms is introduced.

1.2.1 Basic Concepts

What is an on-line problem? What is a competitive algorithm? In this section, we answer these questions, and we present several basic concepts that are very significant for this field of research.

Originally, on-line problem referred to a problem where the solver had to respond to events over time to solve that problem. The concept of an on-line problem has now been broadenedto include problems that need the solver to make decisions with partial information. There is a famous Chinese idiom, "Why can one not see the real face of Lu mountain? This is because one is on the mountain". To solve an on-line problem, one must make decisions "within" a problem (one cannot obtain all the information about the problem, but one can obtain partial information) instead of from "outside" of the problem. Another Chinese idiom can express the difficulty of the solving an on-line problem, "Blundering are those who are concerned, while the onlookers see most clearly".

Algorithms for solving on-line problems are called on-line algorithms. Obviously, a competitive algorithm is a kind of on-line algorithm. What is a competitive algorithm? A very general model for the type of algorithms and problems we consider is that an algorithm *A* is always associated with a configuration, and after a new portion of the input becomes available the algorithm moves to a new configuration. We consider two basic types of problems: maximization of benefit problems, and minimization of cost problems. For benefit problems, there is a function to represent the benefit that may be obtained. The value of this function depends on the input and on the sequence of configurations occupied over time by an on-line algorithm A. For cost problems, there is a cost function that depends on the input and on the sequence of configurations occupied over time by A. This function represents the cost associated with the solution that A finds. This informal model is a generalization of the task system formalization, developed by Borodin, Linial & Saks [12] to include benefit problems and other partial information problems that are not necessarily on-line problems.

To avoid the problems associated with probabilistic models, we seek a worst-case model that will hold for any distribution. In addition, to avoid the problem that the standard worst-case measure might be entirely insensitive to the algorithm used, and give us no useful information, we must refine the worst-case measure. This is the underlying idea of a competitive algorithm: we do not consider the absolute behavior of the algorithm; instead, we consider the ratio between the algorithm's behavior and the optimal behavior for a given problem instance.

Let $\text{cost}_A(\text{I})$ denote the cost of algorithm A on problem instance I. We define the competitive ratio of an algorithm A for a cost problem P to be the value:

$$\inf\{c \mid cost_A(I) \le c \cdot cost_B(I), \forall I \in P, \forall B\}$$

Similarly, let $\text{benefit}_A(I)$ denote the benefit of algorithm A on problem instance I, then we define the competitive ratio of algorithm A on benefit problem P to be:

$$\sup\{c \mid c \cdot benefit_A(I) \ge benefit_B(I), \forall I \in P, \forall B\}$$

Then on-line algorithm A is called the competitive algorithm and c is the competitive ratio.

1.2.2 A Simple Example

To better illustrate the relevant concepts, we present a simple example of an on-line problem and a competitive algorithm. Given a line on which there are three points, initially two servers are located on the line, one each at points B and C, as shown in Figure 1.1. The distance from A to B is one unit and from B to C is two units. We assume when a new request occurs the two servers are both in their ready states.

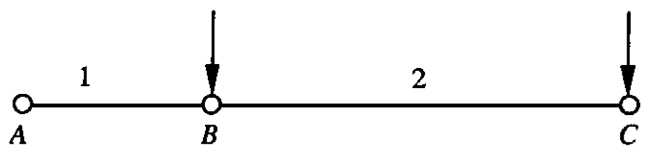

Figure 1.1. On-line 2-server problem on a 3-node graph.

To begin with, what is an on-line problem? For the above problem, if we know the service request sequence σ in advance, then the problem is off-line and we can easily solve it using dynamic programming. However, if the service requests arrive sequentially, and whenever we make a decision we only know part of the sequence σ which has occurred, the problem changes to an on-line problem. The decision maker must make a choice without any knowledge of the future.

Next, we explain competitive algorithms. First, we illustrate why greedy algorithms do not perform well. Despite their widespread application, from the perspective of their optimization capability, we find these algorithms perform very badly under certain conditions. For example, consider an input request sequence $\underbrace{ABAB\cdots AB}_{m}$, with a greedy algorithm a server will travel between the points A and B on m occasions, and the total cost is valued as m. However, for this input sequence, we can obtain an optimal solution (when m is greater than 3), by moving the server on the point B to the point A and then moving the other server from the point C to the point B. This is done once, and the optimal solution is only 3. Obviously, the ratio of the values of the greedy algorithm's solution to the optimal solution is $(m-1)/3$. This value tends to infinity when m tends to infinity. Therefore, this greedy algorithm is not a competitive algorithm, because its solution may be very far from the optimal solution for an instance of this problem.

The weakness of this greedy algorithm is that it may let one server become very busy while another is idle. We assume that the server on point i has moved D_i units, the shortest distance from point i to point j is d_{ij}, and the server request occurs at the point k. We can give a new algorithm (Balance Algorithm)as follows: (1) if $D_i + d_{ki} < D_j + d_{kj}$ then move the server on point i to the point to satisfy the new server request, (2) if $D_i + d_{ki} > D_j + d_{kj}$ then move the server on point j to the point k to serve the new request, and (3) if $D_i + d_{ki} = D_j + d_{kj}$ then choose one of them (i or j) at random.

Using this kind of strategy, for the sequence $\underbrace{ABAB\cdots AB}_{m}$, we can get a solution of 5 units. In fact, for any request sequence for the graph shown in Figure 1.1, if we schedule the servers according to the Balance Algorithm, the total distance that all servers move will be less than or equal to double the value of the optimal solution.

1.2.3 Application of the On-line Algorithm

One class of problems that seems to be inherently suitable for competitive analysis is the class of on-line problems where a sequence of events ($\sigma=\sigma_1,\sigma_2, \ldots$) appears over time and has to be dealt with immediately. An on-line algorithm has to make a decision in response to an event without knowledge of future events.

A typical example of such a problem is the paging problem. When a page fault occurs the paging algorithm has to decide what page is to be evicted from memory without knowledge of future page access patterns. Many different flavors of on-line problems have been considered in last two decades; these include data structure problems, paging problems, distributed problems, packing problems, routing problems, graph problems, scheduling problems, load balancing problems, financial problems, and many others.

Partial information in the input to a problem instance is not only limited to lack of knowledge about future events. In various exploration and navigation problems, the issue is how to obtain the goal (e.g., search for a missing child, or reaching the beach) while traversing the terrain efficiently without a map. Depending on the sensors allowed (e.g., vision and touch), some of the missing information may be obtained by performing a movement.

Another class of problems, where only partial information is available, is in a distributed setting where the global configuration may be unavailable to individual processors that are only aware of their own local view of the world. Again, as with exploration/navigation in an unknown terrain, it may be possible to improve the local understanding of the global environment at the expense of performing appropriate communications between processors.

Since decisions have to be made under uncertainty, and because we consider a worst-case measure, good algorithms seem indecisive with respect to the competitive ratio. Certainly, such algorithms seem to procrastinate before making any decision, which is also difficult to repair if it proves to be a wrong decision. In the famous *k*-server problem, algorithms with good competitive ratios will move multiple servers to an active area only after a great length of time, even if no further activity takes place. However, we note that a misunderstanding of the worst-case measure can lead to difficulties in assessing this area.

With the rapid development of the Internet, an increasing number of disciplines are incorporating Internet technologies. For example, agent-based methods and e-commerce are based on Internet technologies. However, they need be combined with many other disciplines, such as financial, management, mathematics and so on, in order to develop further. In this report, we try our best to apply some of the techniques from on-line algorithms to agent-based e-commerce.

1.3 Literature Review of On-line Algorithms

On-line problems and competitive algorithms are new areas in the field of optimization; they have been paid much attention in the last three decades. On-line algorithms have been implicitly and explicitly studied for approximately 30 years in the context of scheduling, optimization, data structures, and in other computational topics. The roots of competitive analysis can be found in established combinatorial optimization theory [1]. Graham introduces a simple deterministic greedy algorithm (the so-called "List Scheduling" algorithm) for a scheduling problem on parallel machines, and he performs a complete worst-case analysis of this on-line algorithm. However, Graham's paper did not use either of the terms "on-line" or "competitive".

In Volume 2 of "The Art of Computer Programming", (Section 4.7 in [2]) Knuth discusses the computation of the Cauchy product of two power series. The *n*th coefficient of the Cauchy product can be computed based solely on the first *n* coefficients of the two multiplicands. In the first edition of [2] in 1968, Kunth defines "off-line" to be the complement of "on-line".

The term "on-line" in the context of approximation algorithms was first used in the early 1970s, when it was used to describe bin-packing algorithms. The first description of on-line approximation algorithms, that were also called on-line approximation algorithms, can be found in a Ph.D. thesis [3] and also in the journal article [4] of Johnson. Johnson [5] suggests that the origin of the words "on-line" and "off-line" lies in cryptographic systems, in which decryption was either done as part of the communications system (i.e., on the communication line) or after the fact by using other facilities (i.e., off the communication line).

The adversary method for deriving lower bounds on the competitive ratio is used implicitly by Woodall [6] in the analysis of the so-called Bay restaurant problem (which is essentially a linear storage allocation problem with an objective of annoying the owner of the restaurant as much as possible). Kierstead and Trotter [7] also use the adversary method in their investigation of on-line interval graph coloring. Yao [8] formulates a theorem, beginning with the words "For any on-line algorithm...", which proves the impossibility of an on-line bin-packing algorithm having a competitive ratio that is strictly better than 3/2. This seems to be the first result stated for the class of all on-line algorithms, applied to a certain optimization problem, which exploited the distinction between on-line and off-line algorithms.

In 1985, Sleator and Tarjian published two papers, which triggered the boom in on-line algorithms, in the journal Theoretical Computer Science. In [9], they describe competitive algorithms for the list update problem and for the paging problem. In [10], they introduce self-adjusting binary trees and the famous dynamic optimality conjecture for splay trees.

The phrase "competitive analysis" was coined by Karlin, Manasse, Rudolph, and Sleator in their paper [11] on competitive snoopy caching. The paper [12] by Borodin, Linial and Saks on task systems gives a general formalism for a great many on-line cost problems. In 1988, Manasse, McGeoge, and Sleator published their *k-server* paper [13] that contains their famous *k-server* conjecture. While the terminology used is quite different, the first competitive analysis of on-line financial problems can definitely be associated with Cover in [14]. The first sets of problems to be studied in terms of competitive analysis, which are not inherently on-line problems, are the navigation and exploration problems of Papadimitriou and Yannakakis [15].

Much of the work in the field of on-line algorithms has been motivated because of two open problems: the dynamic optimality conjecture for splay trees of Sleator and

Tarjian [10], and the *k-server* conjecture of Manasse, McGeoch, and Sleator [13]. Both of these problems are still open, although it seems that very considerable progress has been made. Even if these problems are very far away from being decided, the study of these problems has resulted in a great body of useful research. What is so special about both these conjectures is that both can be explained in a few minutes, yet they are seemingly very hard to prove (or disprove). These two conjectures, with their misleading simplicity, explains why so much work has been done in this field over the past few years.

Although many fundamental combinatorial optimization problems (e.g., bin packing, machine scheduling, and load balancing) do not always have to be solved on-line (i.e., they are traditionally considered off-line problems), for such problems on-line algorithms constitute an interesting restricted class of algorithms that may sometimes yield surprisingly good approximate solutions. Approximation algorithms for NP-hard combinatorial problems are of particular importance. Here the classic and perhaps most studied problem is (one-dimensional) bin packing, where simple on-line algorithms such as first-fit or best-fit provide reasonable approximations. Here reasonable approximation means within a factor of 1.7 of the optimal solution, which is computed off-line and requires non-polynomial time to compute in the worst case (if $P \neq NP$). Perhaps the first explicit application of competitive analysis (although not named as such) in the computer science literature was by Yao [8], who studied how well on-line bin packing algorithms could perform. At approximately the same time (but using the terminology of "recursive combinatorics " rather than "online algorithms"), Kierstead and Trotter [7] demonstrated an optimal 3-competitive algorithm for online coloring of interval graphs. There are many more "classical" results relating to problems in combinatorial optimization that, at least implicitly, relate to competitive analysis. In addition, this type of analysis can also be found in the literature of decision theory as well as in the area of finance.

1.4 Structure of Thesis

This chapter introduces the motivation and objectives of the research work and presents a summary of the current research situation concerning on-line problems, competitive algorithms, and their applications in many different fields. In order to illustrate the relevant concepts, a simple example of on-line theory is shown.

In Chapter 2, an overview of the *k*-server problem is presented. The statement of the *k*-server problem, the literature concerning the *k*-server problem, and some important results on the *k*-server problem are introduced in this chapter.

Chapter 3 focuses on the introduction of the proposed *k*-truck algorithm. The model of the *k*-truck problem, the relationship between the k-truck and *k*-server problem, and two important results that are used to solve the k-truck problem, are introduced.

In Chapter 4, we present a report on performance analysis and preliminary results concerning *k*-truck problem. Some competitive algorithms, relevant competitive ratios and their proofs are given.

Next, in Chapter 5, a new on-line problem is proposed, the on-line *k*-server problem with twin-request, which is a variant of the traditional *k*-server problem. Some relevant competitive analysis is given. One special case of this problem is analyzed by employing the famous Work Function Algorithm (WFA). Furthermore, a lower bound for the competitive ratio and some open issues concerning this problem are given.

In the Chapter 6, we discuss another new on-line problem, the on-line number of snacks problem, which was originally proposed by us. Some competitive analysis is given and an evaluation for the performance of some on-line algorithms is introduced.

Chapter 7 involves explaining some applications of the theories of on-line problems and competitive algorithms to Internet technologies: dynamic allocation of mobile agents by on-line task scheduling. In that chapter, a complete procedure is presented on how to apply the relevant theory to specific techniques.

Finally, in Chapter 8 suggestions for future work are given.

1.5 List of Contributions

The main contributions of this thesis are as follows:

- Based on the relevant theory of the on-line k-server problem and its competitive solutions, the on-line k-truck scheduling problem is proposed. Furthermore, the model of the k-truck problem is established on any metric space. Next, several solutions, including the Position Maintaining Strategy (PMS) and partial-greedy algorithms, for this on-line problem are presented. We present a lower bound for the competitive ratio, the solutions for the off-line k-truck problem, and some competitive solutions for some special cases.

The different solutions for the on-line k-truck problem are compared and some open problems are involved.

- The on-line k-server problem is generalized to the on-line k-server problem with multi-request. This generalization brings this theoretical model closer to reality. Employing the work function algorithm, a special case of the on-line k-server problem with twin-request is considered and a competitive ratio is obtained. Furthermore, a lower bound for the competitive ratio of k-server problem with twin-request is presented.

- The on-line number of snacks problem is proposed, and four variants of it are discussed. First of all, the model of this problem is established and the necessary assumptions are presented. We then present for each variant: a competitive algorithm, the relevant competitive ratio, and the relevant lower bound for the competitive ratio. Finally, the competitive ratios of the four variants are comprehensively analyzed, and the comparison is made between two of these variants.

- A new approach is presented to the dynamic allocation of mobile agents, by on-line task scheduling, for high performance Internet computing. A guided allocation scheme, based on a system's competitive ratio, is developed to optimize the allocation of mobile agents. The proposed system combines the push-based technology with an innovative on-line task-scheduling scheme to speed up system response and minimize system overheads. System analysis and simulation is used to demonstrate the feasibility and effectiveness of our approach.

1.6 List of Publications

Journal Papers:

1. W. M. Ma, Y.F. Xu, and K. L. Wang. On-line k-truck problem and its competitive algorithm. *Journal of Global Optimization*, **21** (1): 15-25, September 2001.
2. W. M. Ma, J. You, Y. F. Xu, J. Liu, and K. L. Wang. On the on-line number of snacks problem. Accepted by the *Journal of Global Optimization* in October 2001.

Conference Papers:

1. W. M. Ma, H. Chen, J. You, J. Liu, and Y. F. Xu. On-line *k*-server problem with twin-request. ISCA 17th International Conference on Computers and Their Applications. Pages: 277-282. San Francisco, California USA, 2002.
2. X. F. Yang, W. M. Ma, J. You, and J. Liu. On the Dynamic Allocation of Mobile Agents by On-line Task Scheduling. Second International Workshop on Internet Computing and E-Commerce (ICEC'02). Pages: 217, Fort Lauderdale, Florida, USA, 2002.
3. X. F. Yang, W. M. Ma, J. You, and J. Liu. On-line Agent Scheduling in Internet Computing. ISCA 17th International Conference on Computers

and Their Applications Pages: 293-296. San Francisco, California USA, 2002.

4. W .M. Ma, Y. F. Xu, J. You, J. Liu and K. L. Wang. New Result on the *k*-Truck Scheduling Problem. Accepted By The Eighth Annual International Computing and Combinatorics Conference (COCOON'02).

Chapter 2

An Overview of the *k*-Server Problem

In this chapter an overview of the *k*-server problem, which is the basis for the *k*-truck problem and *k*-server problem with twin-request, is given.

2.1 The *k*-Server Problem

The k-server problem can be stated as follows. We are given a metric space *M*, and *k* servers which move among the points of *M*, each occupying one point of *M*. Repeatedly, a request appears at a point $x \in M$. To serve *x*, each server moves some non-negative distance, after which the point *x* must be occupied by one of the servers. The cost incurred to serve the request is the sum of the *k* distances moved. We must serve this request by considering only past requests. In other words, the choice of the server to meet a request at each step must be made on-line, that is, without knowledge of future requests. Therefore (except for degenerate situations) no algorithms are able to achieve the optimal cost on each request sequence. Our goal is to design a solution strategy with a small competitive ratio.

2.2 Literature of the *k*-Server Problem

Sleator and Tarjan [9] introduce a worst-case complexity analysis technique for on-line algorithms called competitive analysis. An on-line algorithm deals with events that require immediate responses. Future events are unknown when the current event is dealt with. Task systems [12], the *k-server* problem [13], layered graph traversal [15], and on-line/off-line games [16] are all attempts to model on-line problems and algorithms.

For the *k*-server problem, Manasse et al [13] prove that for all *k* and any metric space of at least *k*+1 points, no competitive ratio less than *k* is possible. On-line *k*-server algorithms with a competitive ratio of *k* are known for specific metric spaces [9, 13, 17, 18], and Fiat et al [19] give a competitive ratio that depends only upon *k* for all metric spaces. Fiat et al [20] deal with a generalization of the *k-server* problem described in [13], in which the servers are unequal. Several authors independently proposed the Work Function Algorithm (*WFA*) for the *k*-sever problem. *WFA* was investigated by Chrobak and Larmore [18], who suggest that competitive analysis of *WFA* can be done by estimation of the pseudo-cost. They also prove that WFA is 2-competitive for 2 servers. Koutsoupias and Papadimitriou [21] proved that WFA is (*2k-1*)-competitive for all k. The *k*-taxi problem is presented in [22]. The authors provide a good strategy called *PMS* (*Position Maintaining Strategy*) to deal with the on-line *k*-taxi problem.

2.3 Some Important Results on the *k*-Server Problem

In this subsection, we will introduce some important results on the *k*-server problem. Although many useful results have been obtained, the famous *k*-server conjecture is still open:

The k-server conjecture: Any metric space allows for a deterministic k-competitive, k-server algorithm.

After considerable effort, researchers have nearly solved the k-server conjecture. Koutsoupias and Papadimitriou [21] have shown that there is a generic k-server algorithm, the work function algorithm, as follows:

Algorithm *WFA*: let σ_i be the request sequence thus far and let C be the configuration of *WFA* after servicing the last request. Then, given the next request $r=r_{i+1}$, *WFA* serves r with a server $s \in C$ satisfying:

$$s = \arg\min_{x \in C}\{\omega(C - x + r) + d(x, r)\},$$

Ties are broken with an arbitrary choice.

In fact, with algorithm *WFA*, the following theorem holds:

Theorem 2.1[21]: for any k and any metric space, algorithm *WFA* is ($2k$-1)-competitive.

Also important for my research work is the Balance Algorithm (BAL) [13]:

Algorithm *BAL*: For each server s_i, maintain its total distance traveled thus far as D_i. To serve a request r, shuttle the server s_i that minimizes $D_i+d(s_i, r)$.

In fact, the algorithm gives the following theorem:

Theorem 2.2[13]: *BAL* is k-competitive in any metric space with $n=k+1$ points.

Other results for the k-server problem are omitted since they are not related to the research presented in this thesis.

Chapter 3

The *k*-Truck Problem

In this chapter we introduce a new on-line problem, the *k*-truck problem, which is an improved version of the *k-server* problem. We also present the problem statement, the model of the problem, comparison between the *k*-server problem and the *k*-truck problem. Several lemmas and the Position Maintaining Strategy are discussed. Finally, we discuss the solutions of the off-line *k*-truck problem.

3.1 Problem Statement

The *k*-truck problem can be stated as follows. We are given a metric space *M*, and *k* trucks that move among the points of *M*, each occupying one point of *M*. Repeatedly, a request appears at a pair of points $x, y \in M$. To service a request, an empty truck must first move to *x* and then move to *y* with the goods loaded at *x*. How to minimize the total cost of all trucks? Let us first consider following problems:

(1) *Given a service request sequence, how can we schedule trucks so as to minimize the cost?*

(2) *If the service requests are received sequentially, in the process of servicing them without any knowledge of the future requests, how do we minimize the cost?*

Problem (1) is an *off-line* problem, and (2) is an *on-line* problem. The difference between them lies in whether the service request sequence is known in advance. The former can be solved easily with dynamic programming, but the latter is difficult to solve because we must serve the request based only on information concerning the previous requests.

The k-truck problem aims at minimizing the cost of all trucks. Since the cost of moving trucks carrying goods is different from moving empty trucks, the total distance cannot be considered as the objective to minimize. For simplicity, we assume that the cost of the truck carrying goods is θ times that of an empty truck traveling the same distance. Then we take $(1+\theta)$ times the empty truck distance as the objective.

The k-truck problem is a generalization of the k-taxi problem [23] and well-known *k-server* problem [13]. In the *k-server* problem, each request contains only a point x in M. We must move a server to service a request at x. For the *k-server* problem, several on-line algorithms have been proposed [13, 24, 21, 25, 26]. The k-taxi problem is presented in [23]; it is a special case of the k-truck problem. The authors provide a good strategy (*Position Maintaining Strategy*) to deal with the on-line k-taxi problem. Some competitive algorithms for the k-taxi problem, which have good competitive ratios, are given. The same strategy is also used in dealing with the on-line k-elevator problem [61].

Let M be a class of metric spaces. We call an on-line strategy c-competitive for M if, for every metric space in M and for every request sequence, the total cost incurred by that on-line strategy is at most c times the optimal off-line cost of serving the same request sequence. A strategy is competitive for M if it is c-competitive for M, for a given value of c. Note that if a strategy is competitive then the respective definitions hold for all metric space. More results, positive and negative, for on-line algorithms can be found in [21,59].

3.2 The Model of the *k*-Truck Problem

Let $G=(V,E)$ denote an edge weighted graph with n vertices, and the weights of the edges satisfy the triangle inequality, where V is a metric space consisting of n vertices, and E is the set of all weighted edges. We assume that the weight of edge (x,y) is denoted by $d(x,y)$ and the weights are symmetric; that is, for all edges (x, y), $d(x,y)=d(y,x)$. We assume that k trucks occupy k vertices, which are a subset of V. A service request $r=(a,b)$, $a,b\in V$ implies that there are some goods at vertex a that must be moved to vertex b (and for simplicity, we assume that the weight of goods is the same for every request). A service request sequence R consists of service requests in turn, $R=(r_1,\ldots,r_m)$ where $r=(a_i,b_i)$, $a_i,b_i\in V$. The on-line k-truck scheduling problem has to decide which truck to move when a new service request occurs, on the basis that we have no information about any future requests.

The following discussion is based on these essential assumptions:

i) Graph G is connected.

ii) When a new service request occurs, all k trucks are available.

iii) All trucks move the same load weight for each request, and the cost of the truck carrying goods is θ times that of an empty truck covering the same distance, and $\theta \geq 1$.

For a known sequence $R=(r_1,...,r_m)$, let $C_{\text{OPT}}(R)$ be the optimal total cost after the trucks finish serving all requests. For a new service request, if scheduling algorithm *A* can schedule without information concerning the requests following r_i, we call *A* an on-line algorithm. For on-line algorithm *A*, if there are constants α and β satisfying:

$$C_{\text{A}}(R) \leq \alpha \cdot C_{\text{OPT}}(R) + \beta \ .$$

Then, for any possible *R*, *A* is called a competitive algorithm, where $C_{\text{A}}(R)$ is the total cost found by algorithm *A* to satisfy the sequence *R*.

For a request $r_i=(a_i,b_i)$, the procedure for scheduling of a truck to move to a_i is called *empty load*, and to move from a_i to b_i is called *heavy load*. If there is no limit for the *R* and θ, the on-line truck problem is called *P*. In problem *P*, if for any $r_i=(a_i,b_i)$, $d(a_i,b_i)>0$, and $\theta>1$ holds, the problem is called *P*1. Problem *P*1 is also called the normal *k*-truck problem. In problem *P*, if there is no limit for any $r_i=(a_i,b_i)$, but $\theta=1$, the problem is *P*2. Problem *P*2 is also called the *k*-taxi problem. In *P*2, if $d(a_i,b_i)>0$, that is $a_i \neq b_i$, then problem P2 is called *P*3. Problem *P*3 is also called the normal *k*-taxi problem. In problem *P*, if $d(a_i,b_i)=0$, that is $a_i=b_i$, then P is called *P*4. Problem *P*4 is also called the *k*-server problem.

3.3 Relationship Between the Server and Truck Problems

Both the *k*-server problem and the *k*-truck problem are on-line problems. Obviously, the *k*-truck problem is a generalization of the *k*-server problem and we can look at the *k*-server problem as a special case of the *k*-truck problem. Some of the results that were obtained in this research for the *k*-truck problem are based on the results of the *k*-server problem. Finally, both problems can be represented by models in the metric space.

As for their differences, the most important point concerns the way a request is serviced. For the *k*-server problem, the service request is satisfied on a single point of the graph. However, for the k-truck problem, the cost function for the service consists of two parts, the empty load cost and heavy load cost. In other words, for the *k*-server problem, we schedule a server to the point where a service request occurs. Whereas for the *k*-truck problem, we need to schedule a truck to the point where the service request occurs and also allow the truck to carry some goods from this point to another point to finish the service request. Moreover, the cost (per unit distance) is different before and after the truck picks up the goods. In summary, their models are different, their cost functions are different and their solution approach and results are different.

3.4 Several Lemmas and Position Maintaining Strategy

3.4.1 Two Lemmas

The on-line *k-server* problem is presented as a special case of the on-line truck problem in [13]. There are many results on the competitive ratio for the *k-server* problem. Koutsoupias & Papadimitriou show that there exists an on-line algorithm for the *k-server* problem with competitive ratio 2*k*-1[21]. Since the *k-server* problem is a special case of the *k*-truck problem, our research concerning the on-line truck problem has to discover the relationship between them. Here we first give the following lemma [21]:

Lemma 3.1. *There exists an on-line algorithm for the k-server problem with the competitive ratio* $2k-1$.

For the k-server problem, we obviously have the following lemma:

Lemma 3.2. *For any algorithm A for a request sequence* $R=(r_1,\ldots,r_m)$, *we have* $C_A(R)\geq\sum_{i=1}^{m}\theta\cdot d(a_i,b_i)$ *and* $C_{OPT}(R)\geq\sum_{i=1}^{m}\theta\cdot d(a_i,b_i)$.

Proof. For any request sequence *R*, $\sum_{i=1}^{m}\theta\cdot d(a_i,b_i)$ is the necessary cost because it is the cost of the heavy load. Namely, any algorithm that completes the sequence *R* must at least take the cost $\sum_{i=1}^{m}\theta\cdot d(a_i,b_i)$. The inequalities hold.□

3.4.2 Position Maintaining Strategy

In [23], the *PMS* is proposed and used to obtain several useful results for the *k*-taxi problem. For our investigation of the *k*-truck problem, we outline the *PMS* as it applies to the *k*-truck problem as follows.

Position Maintaining Strategy is defined as follows. For the present request $r_i = (a_i, b_i)$, after a_i is reached, the truck reaching a_i must move from a_i to b_i with the goods to complete r_i. When the service for r_i is finished, the PMS moves the truck at b_i back to a_i (empty) before the next request arrives.

Lemma 3.3. *Let OPT be an optimal algorithm for a request sequence $R = (r_1, \ldots, r_m)$, then we have*

$$C_{\mathrm{OPT}}(R) \geq C_{\mathrm{OPT}}(\sigma) + \sum_{i=1}^{m} (\theta - 1) \cdot d(a_i, b_i),$$

where $\sigma = ((a_1, a_1), \ldots, (a_m, a_m))$ and $r_i = (a_i, b_i)$.

Proof. For any request sequence R, the scheduling procedure for serving it must finish the sequence σ, and at same time $\sum_{i=1}^{m} \theta \cdot d(a_i, b_i)$ is the necessary cost because it is the cost of the loaded trucks. Furthermore, the overlap of the cost to finish the sequence σ and $\sum_{i=1}^{m} \theta \cdot d(a_i, b_i)$ is at most $\sum_{i=1}^{m} d(a_i, b_i)$.

Thus any algorithm which completes the sequence R must at least take the cost at the sum of $C_{\mathrm{OPT}}(\sigma)$ and $\sum_{i=1}^{m} (\theta - 1) \cdot d(a_i, b_i)$. The inequality holds.□

3.5 Off-line *k*-Truck Problem

In this section, two solution approaches for the off-line k-truck problem are discussed. One is Dynamic Programming (DP) and the other is Minimum Cost Maximum Flow (MCMF) in an acyclic network. Fist of all, we need to define a configuration as follows.

Definition. 3.1. (Configuration) *In the metric space M, a possible position of k trucks is called a configuration. That is, a configuration is a special k-multiset of at least one and at most k points in the space M. Here 'special' means that in the multiset the same node can be repeated from one to k times.*

Obviously, the configuration is different from that of the k-server problem for which the configuration is defined as the multiset of exactly k different points in metric space M.

3.5.1 Dynamic Programming (DP) Solution

In [13], a DP solution is given for the k-server problem. Similarly, we can develop a DP solution for the k-truck problem. However, because there are some differences between the k-server problem and k-truck problem, the relevant DP solutions are also different. First of all, we introduce a lemma with which to show the number of possible configurations k-trucks on a given graph with n nodes.

Lemma 3.4. *On a given graph G with n nodes, the number of possible configurations of all k trucks is* $\binom{n+k-1}{n-1}$, *where* $(k \le n)$.

Proof. Assume that all k trucks and all n nodes line up in a line from left to right, thus there are $n+k$ locations on which there is either a truck or a node. Next, we move all of the trucks that are between two nodes, given as i and j, to the node i. We assume that the node i is to the right of the node j, and there are no nodes between them. Obviously, if there are no trucks between the two nodes, no trucks are moved to the node i. In addition, we need to let the extreme right location $(n+k)$be a node. Finally, we choose $n-1$ locations, on which we arrange all the remaining $n-1$ nodes, from the $n+k-1$locations. Obviously, we have $\binom{n+k-1}{n-1}$choices.□

Let function $C_{\mathrm{OPT}}(R,S)$ denote the cost of the minimum cost algorithm that handles request sequence R and ends up in configuration S. Similar to [13], we can compute this function recursively, assuming that the trucks are initially in configuration S_0, as follows:

$$C_{\mathrm{OPT}}(\varepsilon, S) = \begin{cases} 0, & \text{if } S = S_0 \\ \text{undefined}, & \text{otherwise} \end{cases}$$

$$C_{\mathrm{OPT}}(Rr, S) = \begin{cases} \min\limits_{T}\left(C_{\mathrm{OPT}}(R,T) + d\left(T \xrightarrow{r=\theta\cdot(a_i,b_i)} S\right)\right), & \text{if } b_i \in S \\ \text{undefined}, & \text{otherwise} \end{cases}$$

where $d\left(T \xrightarrow{r=\theta\cdot(a_i,b_i)} S\right)$ is the cost of transition from configuration T to configuration S, and the last operation of the transition is $a_i \rightarrow b_i$ (satisfying the request $r_i = (a_i, b_i)$). T and S denote the configurations at time $i-1$ and time i, respectively, and ε denotes the empty request sequence.

Theorem 3.5. The *above optimal off-line algorithm for the k-truck problem can obtain the optimal solution with computation time proportional to* $m \cdot \binom{n+k-1}{n-1}^2$, *where m is the length of the request sequence (i.e., the number of requests).*

Proof. Let $|R| = m$. We can develop a table-building method according to the above discussion. Build a table with $|R|+1$ rows, each of which corresponds to a subsequence of the input request sequence *R*, and $\binom{n+k-1}{n-1}$ columns, each of which denotes a possible configuration of the trucks. The entry in row *i* and column *j* is $C_{\text{OPT}}(R_i, S_j)$, where R_i is the subsequence of *R* of length *i*. Each row of the table can be built from the previous row within computation time $\binom{n+k-1}{n-1}^2$. Furthermore, only $|R| = m$ rows need these computations. The proof is completed.□

In addition, the above table-building procedure can produce an optimal sequence of moves. See the relevant discussion in [13].

3.5.2 Minimum Cost Maximum Flow (MCMF) Solution

In [17], MCMF is used to resolve the off-line *k*-server problem. Similar to the discussion in that paper, we can also use this method to handle the case of the off-line *k*-truck problem. The main difference lies in how we construct the network for the *k*-truck problem on which the MCMF algorithm is used to find the optimal solution. We study the problem of finding an optimal strategy to serve a sequence of *m* requests with *k* trucks, and with the request sequence given in advance. We assume that the *k* trucks initially occupy one point, the *origin*, and the *i*-th request is denoted by the binary tuple (a_i, b_i). When there are *m* requests, the inputs to our problem are the superdiagonal entries of an $(m+1)\times(m+1)$ matrix. In the matrix, the $(0, j)$ entry is the sum of the cost of moving from the origin to the location of the *j*-request, starting at a_j (empty) and then moving to the request destination b_j (with the goods), $j = 1,2,\dots,m$. The (i,j) entry is the sum of the cost of moving from the location of the i-request destination to the location of the j-request starting location and then moving to the relevant destination with the goods, $1 \le i < j \le m$.

Theorem 3.6. *There is a $O(km^2)$-time off-line algorithm able to find an optimal schedule for k trucks to serve a sequence of m requests (whether or not the triangle inequality holds).*

Proof. We reduce the k-truck problem, with or without a triangle inequality, to the problem of finding a minimum cost maximum flow in an acyclic network. If we have k trucks $t_1,\dots,t_k$ and m requests $r_1,\dots,r_m$, where $r_i = (a_i,b_i)$, and $i = 1,\dots,m$, we build the following $(2+k+3m)$ -node acyclic network. The vertex set is $V = \{s, s_1,\dots,s_k,a_1,b_1,b_1',\dots,a_m,b_m,b_m',t\}$. Nodes s and t are the source and sink, respectively. Each arc has capacity one. There three sets of arcs of cost 0: (1) from s to each vertex s_i, (2) form each vertex b_i' to t, and (3) from each s_i to t. From each vertex s_i, there is an arc to a_j of cost equal to the distance from the origin to the location of a_j. From each vertex a_j, there is only an arc to b_j of cost equal to $\theta \cdot d(a_i,b_i)$. For $i < j$, there is an arc from vertex b_i' to a_j of cost equal to the distance between b_i and a_j. Furthermore, form vertex b_i to b_i' there is an arc of cost $-K$, where K is an extremely large real number.

The value of the maximum flow in this network is k. Since all capacities are integral, and because the network is acyclic, we can use minimum-cost augmentation [60] to find an integral minimum-cost flow of value k in time $O(km^2)$. An integral $s \to t$ flow of value k can be decomposed into k arc-disjoint $s \to t$ paths, the ith one passing through s_i. Because $-K$ is so small, an integral minimum-cost flow of value k saturates all of the (b_i, b_i') arcs, and hence this corresponds to an optimal schedule for serving the requests, with the *i*th server servicing exactly those requests contained in the $s \rightarrow t$ path that passes through s_i.□

Chapter 4

Computational Analysis of the *k*-Truck Problem

In this chapter, several competitive algorithms and their competitive ratios are presented. The interrelationships among the *k*-truck, *k*-taxi and *k-server* problems are given. Based on the *Position Maintaining Strategy* (PMS), an on-line scheduling algorithm for the *k*-truck problem is presented, which is a generalization of the *k-server* problem. This is a competitive algorithm whose competitive ratio is $2k+1/\theta$ for any $\theta \geq 1$. Furthermore, if $\theta \geq (c+1)/(c-1)$ holds, then there must exist a ($2k-1$)-competitive algorithm for the *k*-truck problem; and if $\theta \leq (c+1)/(c-1)$ holds, then there must exist a ($\frac{2k}{\theta}+1$)-competitive algorithm, where *c* is the competitive ratio of the on-line algorithm from the relevant *k*-server problem. A greedy algorithm with competitive ratio $1+\lambda/\theta$ is given, where λ is a parameter related to the structural property of a given graph. In addition, competitive algorithms with ratios of $1+1/\theta$ are given for two special families of graphs. Finally, a lower bound $\frac{(1+\theta)k}{\theta \cdot k+2}$ for the competitive ratio of the *k*-truck problem is presented.

4.1 The Results Concerning the Competitive Ratio for the *k*-Truck Problem *P*

Using Lemma 3.2 and *PMS* from Chapter 3, the close relationship between the *k-server* and the *k*-truck problem can be shown below.

Theorem 4.1. *For a given graph G, if there is a c-competitive algorithm for the k-server problem on G, then there is a $(c+1+1/\theta)$-competitive algorithm A for the k-truck problem on G, where θ has the same meaning given previously and $\theta \geq 1$.*

Proof. For any $R=(r_1,\dots,r_m)$, considering the *k-server* problem's request $\sigma=(a_1,\dots,a_m)$, let A_σ be a *c*-competitive algorithm for the on-line *k-server* problem on graph *G*. We design the relevant algorithm *A* for the on-line *k*-truck problem as follows.

For the current service request $r_i=(a_i,b_i)$, we first schedule a truck to a_i using algorithm A_σ, then we complete the request r_i using *PMS*. Thus algorithm *A*'s total cost is:

$$\begin{aligned} C_A(R) &= \sum_{i=1}^{m} C_A(r_i) \\ &= \sum_{i=1}^{m} [C_A(a_i)+(\theta+1)\cdot d(a_i,b_i)] \qquad (1), \\ &= C_{A_\sigma}(\sigma)+\left(1+\frac{1}{\theta}\right)\cdot\sum_{i=1}^{m}\theta\cdot d(a_i,b_i) \end{aligned}$$

where θ is defined above and $\theta \geq 1$. From Lemma 3.2 and algorithm A_σ, we have:

$$\begin{aligned} C_{A_\sigma}(\sigma) &\leq c\cdot C_{OPT}(\sigma)+\beta \\ &\leq c\cdot\left(C_{OPT}(R)-\sum_{i=1}^{m}((\theta-1)\cdot d(a_i,b_i))\right)+\beta \qquad (2), \\ &\leq c\cdot C_{OPT}(R)+\beta \end{aligned}$$

and

$$\sum_{i=1}^{m}\theta\cdot d(a_i,b_i)\leq C_{OPT}(R) \quad (3).$$

Combining expressions (1), (2) and (3), we get:

$$C_A(R)\leq\left(c+1+\frac{1}{\theta}\right)\cdot C_{OPT}(R)+\beta\ ,$$

where c and β are constants.□

As a special case of the k-truck problem, we can let $\theta = 1$ and then get a $(c+2)$-competitive algorithm, which is given in [23].

From Theorem 4.1 and Lemma 3.1, the following Corollary holds.

Corollary 4.2. *For a given graph G, there exists a* $(2k + 1/\theta)$*-competitive algorithm for the k-truck problem on G, where* θ *is a constant and* $\theta \geq 1$.

Based on the above discussion, we have no restrictions except that $\theta \geq 1$. In fact, if we further restrict θ, according to two conditions, we can improve the result as follows.

Theorem 4.3. *For the on-line k-truck problem and a given graph G, if there is a c-competitive* (c≥1) *on-line algorithm for the k-server problem on G, then*

(1) If $\theta > (c+1)/(c-1)$, *then PMS is a c-competitive algorithm.*

(2) If $1 \leq \theta \leq (c+1)/(c-1)$, *then PMS is a* $\left(\frac{c}{\theta} + \frac{1}{\theta} + 1\right)$*-competitive algorithm.*

Proof. Similar to Theorem 4.1, we can design an algorithm A' (PMS) so that:

$$C_{A'}(R) = C_{A_\sigma}(\sigma) + \left(1 + \frac{1}{\theta}\right) \cdot \sum_{i=1}^{m} \theta \cdot d(a_i, b_i) \qquad (4),$$

where θ is defined above and $\theta \geq 1$. From Lemma 3.2 and algorithm A_σ, we have:

$$\begin{aligned} C_{A_\theta}(\sigma) &\le c \cdot C_{\mathrm{OPT}}(\sigma) + \beta \\ &\le c \cdot \left(C_{\mathrm{OPT}}(R) - \sum_{i=1}^{m} ((\theta - 1) \cdot d(a_i, b_i)) \right) + \beta \end{aligned} \quad (5).$$

Combining (4) and (5), we obtain:

$$C_{A'}(R) \le c \cdot C_{\mathrm{OPT}}(R) + \left[1 + \frac{1}{\theta} - c \cdot \left(\frac{\theta - 1}{\theta} \right) \right] \cdot \sum_{i=1}^{m} \theta \cdot d(a_i, b_i) + \beta \ (6).$$

If $\theta > (c+1)/(c-1)$ namely $\left[1 + \frac{1}{\theta} - c \cdot \left(\frac{\theta - 1}{\theta} \right) \right] < 0$, we get

$$C_{A'}(R) \le c \cdot C_{\mathrm{OPT}}(R) + \beta \ .$$

If $1 \le \theta \le (c+1)/(c-1)$, and with Lemma 3.2 and 3.3, we obtain:

$$\sum_{i=1}^{m} \theta \cdot d(a_i, b_i) \le C_{\mathrm{OPT}}(R) \ \ (7).$$

We have:

$$C_{A'}(R) \le \left(\frac{c}{\theta} + \frac{1}{\theta} + 1 \right) \cdot C_{\mathrm{OPT}}(R) + \beta \ ,$$

where c and β are some constants.□

From Theorem 4.3 and Lemma 3.1, the following Corollary holds.

Corollary 4.4. *For the on-line k-truck problem on a given graph G, if* $\theta > (c+1)/(c-1)$ *holds, then there exists a* $(2k-1)$ *-competitive algorithm; if* $1 \le \theta < (c+1)/(c-1)$ holds, *then there exists a* $(2 \cdot k/\theta + 1)$ *-competitive algorithm.*□

Obviously, for the k-taxi problem, which is a special case of the k-truck problem, according to the above corollary, PMS is a $(2k+1)$-competitive algorithm as long as we let $\theta = 1$.

4.2 Discussion on the Constraint Graphs

An extreme case of the k-truck problem occurs when $k = n$ (i.e., the number of trucks is equal to the number of vertices of G), or when $k = n-1$. For the *k-server* problem, when $k = n$ any request sequence (of equal length) can be served at the same cost. For the case when $k = n-1$, there exists an $(n-1)$-competitive algorithm [13]. For the k-truck problem, we can use PMS to handle the case when $k = n$ with competitive ratio $1+1/\theta$. For the case when $k = n-1$, if each request $r_i = (a_i, b_i)$ satisfies $d(a_i, b_i) > 0$, then using *PMS* we can also obtain a $(1+1/\theta)$-competitive algorithm. We give the following lemma that can be used to obtain the above results.

Using Lemma 3.2 and *PMS*, we can obtain the following theorem concerning the two cases.

Theorem 4.5. *For a given graph G with n vertices, if* $k = n$, *then there is an on-line algorithm for the k-truck problem on G with a competitive ratio* $1+1/\theta$, *and if each request* $r_i = (a_i, b_i)$ *satisfies* $d(a_i, b_i) > 0$, *and if* $k = n-1$, *then with PMS one can obtain a* $(1+1/\theta)$*-competitive algorithm.*

Proof. We assume that on each vertex there is at most one truck. Otherwise, we can establish a precondition for the truck locations so that each vertex has at most one truck. Furthermore, the cost of this precondition is at most $(n-1)\cdot d$, where $d = \max_{x,y\in v} d(x, y)$.

For the case when $k = n$, we design the on-line k-truck problem algorithm $A1$ as follows. For any $R = (r_1,\dots,r_m)$, we consider the current service request $r_i = (a_i, b_i)$:

(1) If $a_i = b_i$, there is no need to schedule any trucks and the cost is 0.

(2) If $a_i \neq b_i$, because there is a truck at both a_i and b_i, we can schedule the truck at a_i to carry goods to b_i, and the empty truck at b_i to move to a_i. Thus, the cost to finish r_i is $(1+\theta)\cdot d(a_i, b_i)$, and there is still only one truck at each vertex.

For the on-line algorithm $A1$, we have:

$$\begin{aligned}
C_{A1}(R) &= \sum_{i=1}^{m} C_{A1}(r_i) + \beta \\
&\le (\theta + 1)\sum_{i=1}^{m} d(a_i, b_i) + \beta \\
&= \left(1 + \frac{1}{\theta}\right)\sum_{i=1}^{m} \theta \cdot d(a_i, b_i) + \beta \\
&\le \left(1 + \frac{1}{\theta}\right) C_{OPT}(R) + \beta
\end{aligned}$$

where β is the cost of the precondition for starting truck locations, and $\beta \le (n-1)\cdot d$, where $d = \max_{x,y\in v}(x, y)$.

For the case when $k = n-1$, we design the on-line k-truck problem algorithm $A2$ as follows. For any $R = (r_1, \ldots, r_m)$, we consider the current service request $r_i = (a_i, b_i)$ that satisfies $d(a_i, b_i) > 0$:

(1) If there are trucks at both a_i and b_i, we can schedule the truck on a_i to carry goods to b_i and the empty truck at b_i to move to a_i. Thus, the cost to finish r_i is $(1+\theta)\cdot d(a_i, b_i)$, and there is still only one truck at each vertex.

(2) If there is a truck at a_i but not at b_i, we can schedule the truck at a_i to carry goods to b_i. The cost is $\theta \cdot d(a_i, b_i)$. In addition, there is no vertex with more than one truck.

(3) If there is a truck at b_i but not at a_i, we can schedule the truck at b_i to move (empty) to a_i and then return carrying goods to b_i. The cost is $(1+\theta)\cdot d(a_i, b_i)$. In addition, no vertex has more than one truck.

In a similar manner to $A1$, we can prove that the on-line algorithm $A2$ is a $(1+1/\theta)$-competitive algorithm.□

4.3 Scheduling *k* Trucks on a Special Graph

In this section, we consider the problem of scheduling k trucks on a special graph. Let $d_{\max} = \max d(v_i, v_j)$, $d_{\min} = \min d(v_i, v_j)$, $i \neq j$, $v_i, v_j \in V$, and let

$$\lambda = \frac{d_{\max}}{d_{\min}}.$$

We study the *k*-truck problem with following constraints: (a) there is a constant C such that $\lambda \leq C$, (b) each request $r_i = (a_i, b_i)$ satisfies $d(a_i, b_i) > 0$, and (c) there is at most one truck located at a vertex.

Given the above constraints, we present the following on-line algorithm *B* (Partial-Greedy, PG for short) to schedule the *k*-truck problem. Let $r_i = (a_i, b_i)$ be the present request. We give algorithm *B* as follows:

(1) If there is a truck at a_i and there is also a truck at b_i, then *B* moves the truck at a_i to b_i, to service the request, and at the same time *B* moves the empty truck at b_i to a_i. The cost of servicing r_i is $(1+\theta) \cdot d(a_i, b_i)$ and no vertex has more than one truck.

(2) If there is a truck at a_i and no truck at b_i, then *B* moves the truck at a_i to b_i, to service the request. The cost of servicing r_i is $\theta \cdot d(a_i, b_i)$ and no vertex has more than one truck.

(3) If there is no truck at a_i and there is a truck at b_i, then *B* moves the empty truck at b_i to a_i, and moves the truck carrying goods from a_i to b_i, to service the request. The cost of servicing r_i is $(1+\theta) \cdot d(a_i, b_i)$ and no vertex has more than one truck.

(4) If there is no truck at a_i or b_i, then *B* moves an empty truck that is closest to location a_i to a_i, let us suppose that this truck is located at c_i, and then it carries goods to b_i, to service the request. The cost of servicing r_i is $d(c_i, a_i) + \theta \cdot d(a_i, b_i)$ and yet again no vertex has more than one truck.

Using algorithm B, we can obtain the following result.

Theorem 4.6. *Under the assumption (a), (b) and (c) specified at the beginning of this section, scheduling algorithm B (PG) for the k-truck problem achieves competitive ratio* $1+\lambda/\theta$.

Proof. We have four possible cases for dealing with a request. For cases (1), (2) and (3), the cost of B is at most $(1+\theta)$ times the optimal cost for any request. For case (4), the extra cost is $d(c_i,a_i)$. Since c_i is the closest occupied vertex to a_i, we have: $d(c_i,a_i)\le d_{\max}$. Let $C_B(R)$ denote the cost of B for R, then we have:

$$C_B(R)\le \sum_{i=1}^{m}\{\max[d(b_i,a_i),d(c_i,a_i)]+\theta\cdot d(a_i,b_i)\}+\beta$$

where β is the cost for preconditioning the starting truck positions so that each vertex has at most one truck and it is bounded by a constant related to G.

Since $a_i\ne b_i$ and $d(a_i,b_i)>0$, we have:

$$\frac{C_B(R)}{\sum_{i=1}^{m}d(a_i,b_i)}\le\theta+\frac{\sum_{i=1}^{m}\max[d(b_i,a_i),d(c_i,a_i)]}{\sum_{i=1}^{m}d(a_i,b_i)}+\frac{\beta}{\sum_{i=1}^{m}d(a_i,b_i)}.$$

Also, since $d(c_i,a_i)<=d_{\max}$, and $d_{\min}<=d(a_i,b_i)<=d_{\max}$, we have:

$$\frac{C_B(R)}{\sum_{i=1}^{m}d(a_i,b_i)}\le\theta+\frac{\sum_{i=1}^{m}d_{\max}}{\sum_{i=1}^{m}d_{\min}}+\frac{\beta}{\sum_{i=1}^{m}d(a_i,b_i)}.$$

$$=\theta+\lambda+\frac{\beta}{\sum_{i=1}^{m}d(a_i,b_i)}$$

This implies that:

$$\begin{aligned} C_B(R) &\le \left(1+\frac{\lambda}{\theta}\right)\cdot\theta\cdot\sum_{i=1}^{m} d(a_i,b_i)+\beta \\ &\le \left(1+\frac{\lambda}{\theta}\right)\cdot C_{\mathrm{OPT}}(R)+\beta \end{aligned}.$$

□

4.4 Comparison of the Two Algorithms for Solving Problem *P*1

In sections 4.1 and 4.3, we gave two algorithms *A*' and *B* respectively, both of which are competitive algorithms for *P*1. We might be confronted with the problem of choosing one of these algorithms for use in different contexts. In this section, we consider how to make that choice.

The competitive ratiocan be used to judge which on-line algorithm is better. The competitive ratios of algorithms *A*' ($c_{A'}$) and *B* (c_B) are:

$$c_{A'} = \begin{cases} 2k-1, & \text{if } \theta > (c+1)/(c-1), \\ \dfrac{2k}{\theta}+1, & \text{if } 1 \le \theta \le (c+1)/(c-1). \end{cases}$$

and

$$c_B = 1+\lambda/\theta .$$

If we let $c_A = c_B$, we can obtain an expression for k that identifies conditions when the performance of algorithm *A*' is equivalent to that of algorithm *B* as follows:

$$k = \begin{cases} 1+\dfrac{\lambda}{2\theta}, & \text{if } \theta > (c+1)/(c-1), \\ \dfrac{\lambda}{2}, & \text{if } 1 \le \theta \le (c+1)/(c-1). \end{cases}$$

Obviously, the following theorem holds:

Theorem 4.7. *For the on-line k-truck problem P1, denoting the PMS and PG algorithm by A and B, respectively. Examining the competitive ratios: if $\theta > (c+1)/(c-1)$ holds compared with $k \le 1+\lambda/(2\theta)$, then algorithm A performs better than B. However, if $k > 1+\lambda/(2\theta)$ holds, then B performs better than A. If $1 \le \theta \le (c+1)/(c-1)$ holds compared with $k \le \lambda/2$, then A performs better than B. However, if $k > \lambda/2$ holds, then B performs better than A.* □

4.5 A Lower Bound for the *k*-truck Problem

In this section, we derive a lower bound for the competitive ratio for the *k*-truck problem in a symmetric metric space. We propose that, any general on-line algorithm for solving this problem, either a deterministic or a randomized algorithm, must have a competitive ratio of at least $\frac{(\theta+1)k}{\theta \cdot k+2}$. Actually, we have proven a slightly looser lower bound on the competitive ratio. Suppose, we wish to compare an on-line algorithm, for solving a problem with *k* servers, with an off-line algorithm, for solving a problem with h servers, where $h \le k$. Naturally, the competitive ratio decreases when the on-line algorithm deals with a problem containing more servers, compared with the off-line algorithm. We obtain a lower bound equal to $(\theta+1)\cdot k/((\theta+2)\cdot k-2h+2)$. A similar approach was taken in [13], where the lower and upper bound are given for the *k*-server problem.

Theorem 4.8. *Let A be an on-line algorithm for the symmetric k-truck problem on a graph G with at least k nodes. Then, for any* $1 \le h \le k$, *there exist request sequences* $R_1, R_2, \ldots$ *such that:*

(1) For all i, R_i *is an initial subsequence of* R_{i+1}, *and* $C_A(R_i) < C_A(R_{i+1})$.

(2) There exists an h-server algorithm B (which may start with its servers anywhere) such that for all i,

$$C_A(R_i) > \frac{(\theta+1)k}{(\theta+2)k-2h+2} \cdot C_B(R_i)$$

Proof. Without loss of generality, assume that *A* is an on-line algorithm and that the *k* trucks start at different nodes. Let *H* be a sub-graph of *G* of size $k+2$ created from the *k* initial truck positions from *A*, plus one other vertex.

Define *R*, as the nemesis sequence of *A* on *H*, such that *R*(*i*) and *R*(*i*-1) are the two unique vertices in *H* that are not covered by *A*, and a request $r_i = d(R(i), R(i-1))$ occurs at time *i*, for all $i \ge 1$. Then we have:

$$\begin{aligned} C_A(\sigma,t) &= \sum_{i=1}^{t} \left(d(R(i+1),R(i)) + \theta \cdot d(R(i),R(i-1)) \right) \\ &= (1+\theta) \cdot \sum_{i=1}^{t-1} d(R(i+1),R(i)) + d(R(i+1),R(i)) + \theta \cdot d(R(1),R(0)) \end{aligned}$$

because at each step, *R* requests the node that was just vacated by *A*.

Let *S* be any *h*-element subset of *H* that contains *R*(1) but not *R*(0). We can define an off-line *h*-truck algorithm *A*(*S*) as follows. The trucks initially occupy the vertices in the set *S*. To process a request $r_i = d(R(i), R(i-1))$, the following rule is applied.

If S contains $R(i)$, move the truck at node $R(i)$ to $R(i-1)$ and update S to reflect this change. Otherwise, move the truck at node $R(i-2)$ to $R(i)$, and then to $R(i-1)$, and update S to reflect this change.

Clearly, for all $i > 1$, the set S contains $R(i-2)$and does not contain $R(i-1)$ when step i begins.

The following observation is the key to the rest of the proof: if the above algorithm starts with distinct and equal-sized sets, S and T, then S and T never become equal. The reason for this is described in the following paragraph.

Suppose that, S and T differ before $R(i)$ is processed. We show that the versions of S and T also differ when they are created by processing $R(i)$, as described above. If S and T both contain $R(i)$, they both move the truck at node $R(i)$ to the empty node $R(i-1)$. No other nodes change, so S and T are still different; and S and T both contain $R(i-1)$. If either S or T contain $R(i)$ exclusively, then after that request only one of them contains $R(i-1)$, so they still differ. If neither of them contains $R(i)$, then both of them change by dropping $R(i-2)$and adding $R(i-1)$, so the symmetric difference between S and T remains the same.

Let us consider running an ensemble of algorithms $A(S)$ simultaneously, starting from each h-element subset S of H containing $R(1)$ but not $R(0)$. There are $\binom{k}{h-1}$ such sets. Since no two sets ever become equal, the number of sets remains constant. After processing $R(i)$, the collection of subsets consists of all the h-element subsets of H that contain $R(i-1)$.

By our choice of starting configuration, step 1 never costs $d(R(1),R(0))$. At step i (for $i \geq 2$), each of these algorithms either moves the truck at node $R(i)$ to $R(i-1)$ (if S contains $R(i)$), at cost $r_i = d(R(i),R(i-1))$; or they move the truck at node $R(i-2)$ to $R(i)$ and then to $R(i-1)$, at cost $d(R(i-2),R(i))+d(R(i),R(i-1))$. Of the $\binom{k}{h-1}$ algorithms being run, $\binom{k-1}{h-1}$ of them (i.e., the ones which contain $R(i-2)$ but not both $R(i)$ and $R(i-1)$) incur a cost of $d(R(i-2),R(i))+d(R(i),R(i-1))$. While the rest $\binom{k-1}{h-2}$ incur a cost of $d(R(i),R(i-1))$. So, for step i the total cost incurred by all of the algorithms is:

$$\begin{aligned}&\binom{k-1}{h-1}\cdot\left(d(R(i-2),R(i))+\theta\cdot d(R(i),R(i-1))\right)+\binom{k-1}{h-2}\cdot\theta\cdot d(R(i),R(i-1))\\&=\binom{k}{h-1}\cdot\theta\cdot d(R(i),R(i-1))+\binom{k-1}{h-1}\cdot d(R(i-2),R(i))\end{aligned}$$

The total cost of running all of these algorithms, up to and including $R(t)$, is therefore:

$$\sum_{i=1}^{t}\binom{k}{h-1}\cdot\theta\cdot d(R(i),R(i-1))+\sum_{i=2}^{t}\binom{k-1}{h-1}\cdot d(R(i-2),R(i)).$$

Thus, the expected cost of one of these algorithms chosen at random is:

$$\begin{aligned}C_{\text{EXP}}(R,t)&=\theta\cdot\sum_{i=1}^{t}d(R(i),R(i-1))+\frac{\binom{k-1}{h-1}}{\binom{k}{h-1}}\cdot\sum_{i=2}^{t}d(R(i-2),R(i))\\&\leq\frac{(\theta+2)k-2h+2}{k}\cdot\sum_{i=1}^{t-1}d(R(i),R(i+1))+\frac{(\theta+1)k-h+1}{k}\cdot d(R(1),R(0))\\&-\frac{k-h+1}{k}\cdot d(R(t-1),R(t))\end{aligned}$$

This inequality holds for the triangle inequality and expansion of the binomial coefficients.

Recall that the cost to A for the same steps was:

$$C_A(R,t) = (\theta+1)\cdot\sum_{i=1}^{t-1} d(R(i+1),R(i)) + d(R(t+1),R(t)) + \theta\cdot d(R(1),R(0))$$

Since the distances are symmetric, the two summations are identical, except that both expressions include some extra terms, which are bounded by a constant. Therefore, after suitable manipulation we obtain:

$$\frac{C_A(R,t)}{C_{EXP}(R,t)} \geq \frac{(\theta+1)k}{(\theta+2)k - 2h + 2}$$

Finally, there must be some initial set that has the property that its performance is always no worse than the average of the costs across all the algorithms. Let S be this set, and $A(S)$ be the algorithm operating on this set. Let R_i be an initial subsequence of R, for which $A(S)$ does no worse than average.□

This theorem gives a lower bound of $(\theta+1)\cdot k/((\theta+2)\cdot k - 2h + 2)$ on the competitive ratio. Even if we require our off-line algorithm to start with the trucks in particular starting locations, we can move the trucks wherever we choose at the cost of an additive constant.

Corollary 4.9. *For any symmetric k-truck problem, there is no c-competitive algorithm for* $c < (\theta+1)\cdot k/(\theta\cdot k + 2)$.□

Corollary 4.10. *For any symmetric k-taxi problem (where* $\theta=1$*), there is no c-competitive algorithm for* $c < 2k/(k+2)$.□

4.6 Concluding Remarks

All the results presented are suitable for the *k*-taxi problem, because it is a special case of the *k*-truck problem. For problems *P*1 and *P*4 in this thesis, there are many theoretical issues that need to be studied further. An interesting problem related to the *k*-truck problem considers other optimization criteria, such as minimizing the maximum waiting time, or minimizing the sum of all empty move distances.

Chapter 5

The *k*-Server Problem with Twin-Request

In this chapter, we propose the new concept of the *k*-sever problem with multi-request. We introduce and explore a model of this problem. Our analysis focuses only on the situation where two requests occur simultaneously (i.e., twin-request), and the on-line algorithm must choose two servers to satisfy both requests. By employing the Work Function Algorithm (*WFA*), we obtain results for a special case of this problem, in which the metric space has *k*+2 points. Finally, we provide a lower bound for the k-server problem with twin-request.

5.1 Introduction

In the *k*-server problem, most on-line analysis assumes that requests occur sequentially. In other words, the on-line decision-maker only chooses one server (according to the relevant on-line algorithm) to satisfy the request, and this request is satisfied before another request occurs. However, in many real life situations, it is possible for more than one request to occur at the same time. In that case, the decision-maker must simultaneously move a number of servers to satisfy all outstanding requests. The *k*-server problem with multi-request is a generalization of the *k*-server problem [28, 13]. It is defined in a metric space *M*, which is a (possibly infinite) set of points *Q* and a distance function *d*: $Q \times Q \rightarrow \Re^+$ (non-negative real function) that satisfies the triangle inequality:

$$d(x,y) \leq d(x,z) + d(z,y), \text{ for all } x, y, z \text{ in } Q.$$

If $d(x,y) = d(y,x)$, for all x, y in Q, then the metric space is called symmetric. Otherwise, the metric space is called asymmetric. In this thesis, the term "metric space" is used to denote a symmetric space. Clearly, $d(x,x) = 0$ for all x in Q.

In the metric space M, reside k servers that can move between points. A possible position of the k servers is called a *configuration*; that is, a configuration is a multi-set containing k points in M. We use similar notations from [21] to state our problem. Thus, we use capital letters to represent configurations and $D(X,Y)$ to denote the minimum distance required to move the servers from configuration X to configuration Y. We assume that the k servers are always initially in configuration A_0. For a multi-set X and a point a, we use $X + a$ for $X \cup \{a\}$ and $X - a$ for $X - \{a\}$. Finally, we use $C(X)$ to represent the sum of the distances between all pairs of points in X.

In the k-server problem with multi-request, a *request* is also a multi-set of l points of M, and we denote it by γ_i, where $\gamma_i = \{a_1, a_2, \ldots, a_l\}$, $a_i \in M$ and each a_i is one of the simultaneous requests, for $i = 1,2,\ldots,l$. We can assume that $l \leq n - k$, where n is the number of points in M (otherwise, the problem can be changed to the $l-1$ situation). A *request sequence* Γ is a sequence of *requests* (i.e., a sequence of multi-sets containing l-points in M). R*equests* are serviced by the k servers; servicing a *request* involves moving several servers to the multi-set of service request points. In particular, if $\Gamma = \gamma_1\gamma_2\ldots\gamma_n$ is a *request* sequence, then the k servers service Γ by passing through configurations A_0, A_1, A_2, ..., A_n, with $\gamma_j \subset A_j$. At step j, the cost of servicing *request* γ_j is the cost of moving the k servers from A_{j-1} to A_j; that is, the cost of moving $D(A_{j-1}, A_j)$. The cost for servicing Γ is the sum of the costs for all these steps. For convenience, we now drop the use of italics to denote a request meaning a multi-request.

Since an on-line algorithm, by definition, cannot base its decisions on future requests, our choice of A_j must depend only on A_0 and the subsequence of requests $\gamma_1\gamma_2\ldots\gamma_j$. Whereas, an off-line algorithm knows the whole request sequence in advance, and consequently A_j depends on A_0 and $\gamma_1\gamma_2\ldots\gamma_n$. Let $C_{\text{opt}}(A_0, \Gamma)$ denote the optimal off-line cost for servicing a request sequence Γ starting at the initial configuration A_0. Similarly, let $C_{\text{on}}(A_0, \Gamma)$ denote the cost for servicing Γ for an on-line algorithm. The competitive ratio of the on-line algorithm is roughly the worst-case ratio $C_{\text{on}}(A_0, \Gamma)/C_{\text{opt}}(A_0, \Gamma)$ [9]. In order to remove any dependency on the initial configuration, a more careful definition is necessary: the competitive ratio of the on-line algorithm is the infimum (greatest lower bound of a set) for all c so that for all initial configurations A_0 and for all request sequence Γ:

$$C_{\text{on}}(A_0, \Gamma) \le c \cdot C_{\text{opt}}(A_0, \Gamma) + C$$

where C may depend on the initial configuration A_0 but not on the request sequence Γ.

Obviously, in metric spaces with k or fewer points (i.e., degenerate cases), an on-line algorithm can initially cover all points with its servers; therefore, it never moves them again and its competitive ratio is 1. In addition, for the k-server problem with l-request and $l \ge k$, even metric spaces with at least k+1 points can have a competitive ratio of 1, because all k servers must be moved simultaneously. Thus, the problem becomes interesting for metric spaces if $l < k \leq n-2$, where n is the number of points in a metric space M.

There are many research results concerning the k-server problem with a single request, which is proposed in [28]. In [28], it is shown that no on-line algorithm can have a competitive ratio less that k and the *k-Server Conjecture* is posed. The authors in [19] prove that there is an algorithm with a finite competitive ratio for all metric spaces, and then answer the question regarding whether there is a finite competitive ratio for all metric spaces, although the competitive ratio in this paper increases exponentially with k. This result is improved somewhat in [29], and again in [30], which establishes a deterministic competitive ratio of $O\left(4^k \log^2 k\right)$. The authors in [31] consider memory-less randomized algorithms, which are more space-efficient than other algorithms for the k-sever problem, and show a competitive ratio of k for the special class of resistive metric spaces. Especially for the 2-server problem, the authors in [30] and [32] provide a 10-competitive and a 4-competitive efficient deterministic algorithm, respectively. In [33], the authors show that the harmonic algorithm is 3-competitive. More results concerning lower bound for the randomized version of the k-server problem are shown in a series of papers [34, 35, 36]. In [21], an upper bound of $2k-1$ for the k-server problem is established, which is the best competitive ratio of a deterministic on-line algorithm for the k-server problem. Also in [21], the work function algorithm and "potential function" proving technique are employed. More results concerning the k-server conjecture are found in [40, 41, 42, 43]. In [43], the authors use *WFA* to prove that the k-server conjecture is true for several special metric spaces.

Generalizing from the k-server concepts, we propose the new concepts of the k-server problem with multi-request. Obviously, the space and time complexity of the k-server problem with multi-request is greater than the traditional k-server problem. We employ the *WFA* to deal with a special case of the k-server problem with twin-request, where a metric space has $k+2$ points. We obtain a competitive ratio of $(k^2+3k)/2$ for our algorithm. We use the "potential function" technique to obtain this result. We also obtain the best known upper bound for the competitive ratio, which is independent of the server travel distances, $\binom{n}{k}-1$, where n is the number of points in metric space M. Although this upper bound for the competitive ratio is somewhat weak. Finally, we obtain a lower bound for the k-server problem with twin-request. We believe that this lower bound is somewhat small. Increasing this lower bound and decreasing the upper bound for the competitive ratio are challenging problems.

5.2 Preliminaries for the *k*-Server Problem with Twin-Request

We introduce some preliminaries for the k-server problem with twin-request. In the metric space M, given a sequence of requests:

$$\Gamma=(\gamma_1,\gamma_2,\cdots,\gamma_m)=(\{a_1,b_1\},\{a_2,b_2\},\cdots,\{a_m,b_m\}),$$

where each $\gamma_i=\{a_i,b_i\}$ specifies a pair of points that forms a twin-request for service, the k-server problem with twin-request is concerned with deciding how to move the servers in response to a twin-request. The initial locations of the servers are specified. For any twin-request $\gamma_i=\{a_i,b_i\}$, there are three possible cases,

Case 1. If both nodes a_i and b_i are occupied, then nothing needs to happen.

Case 2. If both nodes a_i and b_i are unoccupied, then two servers must be moved simultaneously to occupy those modes.

Case 3. If only one of the nodes a_i or b_i is already occupied, then we need to schedule one server to move to the unoccupied node and at same time block the server on the occupied node from being scheduled to move elsewhere.

The constituents of a twin-request must be satisfied in the order that the requests occur in the sequence. The cost of handling a sequence of requests is equal to the total distance moved by the servers. An on-line algorithm for solving the *k*-server problem with twin-request operates under the additional constraint that it must decide which two servers should be moved to satisfy a given twin-request without knowing what the future requests will be.

To describe the *k*-server problem with twin-request, we use the following function:

Definition 5.1. *Let i, j, h, l, be any points in metric space M, then*

$$g(i,j,h,l) = \min(d(i,h)+d(j,l), d(i,l)+d(j,h)).$$

Obviously, this function has the following properties:

(1) If we change the order of the first two arguments and the last two arguments the value of the function does not change, that is, $g(i,j,h,l) = g(h,l,i,j)$.

(2) Its value does not change if its first two arguments or the last two arguments are permuted, for example, $g(j,i,h,l) = g(i,j,h,l)$.

(3) The triangle inequality ensures that: $g(i,j,h,l)+g(h,l,m,n)\geq g(i,j,m,n)$.

(4) The triangle inequality ensures that: $g(i,j,h,l)+d(i,m)\geq g(m,j,h,l)$.

For any twin-request $\gamma_i=\{a_i,b_i\}$, an algorithm B is called *lazy* if it handles the problem using the three cases, which are described above.

Similar to the conventional *k*-server problem, the following lemma [13] shows that we may restrict our attention to *lazy* algorithms.

Lemma 5.1. *For the k-server problem with twin-request, and for any algorithm B, there is a modified algorithm B' that is lazy, does not cost more than B, and is also an on-line algorithm if B is an online algorithm.*

The proof of this lemma is trivial, and we omit it in this thesis.

The following algorithms are all lazy. In order to prove that an algorithm is competitive within a certain factor, it is sufficient to compare it to other lazy algorithms, since they outperform all other types of algorithms.

The on-line algorithms we describe have another property: they completely ignore requests for service at vertices where servers already cover both of them. These requests do not affect later decisions. Therefore, it is trivial to show that we only need to consider algorithms that have this property.

We describe a request sequence Γ as "hard" for algorithm *B*, if *B* must move some servers in response to every request sequence. For all hard Γ and all algorithms *A*, if there exists a constant α such that:

$$C_B(\Gamma) \leq c \cdot C_A(\Gamma) + \alpha$$

where $C_A(\Gamma)$ denotes the cost of the algorithm *A* to serve the request sequence Γ, then algorithm *B* is said to be *c*-competitive.

In [13], the authors proved the following lemma for all hard Γ and all algorithms B:

Lemma 5.2. *If B is a server algorithm that ignores all requests to covered vertices and is c-competitive on all of its hard request sequences, then B is c-competitive.*

The above lemma obviously holds for the *k*-server problem with twin-request provided we ignore the requests whose twin nodes are already both covered by servers.

Lemmas 5.1 and 5.2 show that in order to prove that our algorithms are competitive, it is sufficient to compare them to lazy algorithms dealing with hard sequences.

5.2.1 An Optimal Off-line Algorithm

Manasse et al give a dynamic programming procedure [13] to compute the cost of an optimal algorithm handling request sequence Γ. Let $C_{\text{opt}}(\Gamma, S)$ be a function whose value is the cost of a minimum-cost algorithm (making lazy moves only) that handles request sequence Γ and ends up in configuration *S* (covering a particular set of vertices). We can compute this function recursively as follows, assuming that the servers are initially covering a set of S_0:

$$C_{\text{opt}}(\varepsilon, S) = \begin{cases} 0, & \text{if } S = S_0, \\ \text{undefined}, & \text{otherwise,} \end{cases}$$

$$C_{\text{opt}}(\Gamma\gamma', S) = \begin{cases} \min\limits_{T}(C_{\text{opt}}(\Gamma, T) + D(T, S)), & \text{if } \gamma' = \{a', b'\} \text{and } a', b' \in S, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

where $D(T,S)$ is the cost of a transition (by a lazy move) from configuration T to configuration S. This is the correct method for computing the function because the minimum-cost algorithm reaching configuration S at time i must have been in configuration T at time $i-1$.

According to the method in [2], for $|\Gamma| = m$, one can get the optimal solution within a computation time proportional to $\binom{n}{k}^2$. And one can also obtain an optimal sequence of moves by tracking the steps made during the computation.

5.2.2 Work Function Algorithm

To obtain competitive ratios for some special cases, in which the metric space has exactly $k+2$ points, we employ the *Work Function Algorithm* (WFA). This natural idea for the on-line k-server problem was first proposed in [37]. It had already been successfully applied to other problems [38, 39]. In particular, Koutsoupias and Papadimitriou [21] use the work function algorithm to make significant progress towards proving the *k-server conjecture*, which is posed in [13]. In [21], they give the best result found so far, a $2k-1$ competitive ratio for the general k-server problem. In this thesis, we also use *WFA* to analyze the k-server problem with twin-request in a metric space with $k+2$ points. As for the other special cases and general cases of the k-server problem with twin-request, these problems remain unresolved. We also use the concept of *extended cost*, which is proposed in [21], in our proof. It is necessary to define the work function again because the k-server problem with twin-request is different from the k-server problem.

Definition 5.2. (Work Function) *Fix a metric space M and an initial configuration A_0. For any configuration X, which consist of k points in M, and any request sequence $\Gamma=(\gamma_1,\gamma_2,\cdots,\gamma_m)=(\{a_1,b_1\},\{a_2,b_2\},\cdots,\{a_m,b_m\})$, let $w_i(X)$ denote the optimal cost of serving a subsequence of request sequence $\Gamma^i=(\gamma_1,\gamma_2,\cdots,\gamma_i)$, where $i\leq m$, starting at A_0 and finishing in configuration X. The non-negative real function w_i defined on the set of k-configurations is called the work function for serving Γ^i.*

From the definition of the work function, one can find a very important property: all the useful information concerning the past scheduling is encapsulated in the work function. Therefore, an on-line algorithm needs to only remember w_i, and not Γ^i, because any other algorithm can be transformed to one with this property without any deterioration in competitiveness.

Initially, the work function $w_e(X)$ of a configuration X is only the cost of moving the servers from the initial configuration A_0 to the configuration X: $w_e(X)=D(A_0,X)$. According to the statements of section 5.2 and the definition of the work function, we can obtain the following corollary:

Corollary 5.3. *The work function has the following properties:*

1) For every configuration X, if $\gamma_i = \{a_i, b_i\}$, then

$$w_i(X) = \begin{cases} w_{i-1}(X) & \text{if } \gamma_i \subset X \text{ namely}, a_i, b_i \in X \\ \min\limits_{x \in X}\{w_i(X - x + a_i) + d(x, a_i)\} & \text{if } a_i \notin X, b_i \in X \\ \min\limits_{y \in X}\{w_i(X - y + b_i) + d(y, b_i)\} & \text{if } a_i \in X, b_i \notin X \\ \min\limits_{x,y \in X}\{w_i(X - x - y + a_i + b_i) + g(x, y, a_i, b_i)\} & \text{if } a_i, b_i \notin X \end{cases}$$

In fact, we can combine the above four cases to one formula, that is:

$$\begin{aligned} w_i(X) &= \min_{x,y \in X}\{w_i(X - x - y + a_i + b_i) + g(x, y, a_i, b_i)\} \\ &= \min_{x,y \in X}\{w_{i-1}(X - x - y + a_i + b_i) + g(x, y, a_i, b_i)\} \end{aligned}$$

where $(X - x - y + a_i + b_i)$ indicates the k-configuration resulting from X by replacing the points x or y, by a_i or b_i.

2) For every configuration X,

$$w_i(X) \geq w_{i-1}(X)$$

3) For every configurations X and Y,

$$w_i(X) \leq w_i(Y) + D(X, Y)$$

4) For every configuration and any sequence of twin-request $\Gamma = (\gamma_1, \gamma_2, \cdots, \gamma_m)$,

$$C_{\text{opt}}(\Gamma) \leq w_m(X)$$

Proof. Properties 1 and 4 follow trivially because of the definition of the work function. For property 3, according to the definition of $w(X)$, we know that all k

servers must end in configuration X. Therefore, property 3 holds. Combining properties 3 and 1, we can easily obtain property 2.□

From the definition of the work function and the above properties of corollary 5.3, one can understand the advantages of the *WFA*. Consider a request sequence Γ and let A be the configuration of an on-line algorithm after servicing Γ. Intuitively, a *greedy algorithm*, which moves the closest servers to satisfy a twin-request γ_i (i.e., it moves its servers to a new configuration A', with $\gamma_i \subset A'$, that minimizes $D(A, A')$) appears to be a good algorithm choice. However, it is easy to see that a greedy algorithm for the k-server problem with twin-request, is too conservative and has an unbounded competitive ratio; just as it is too conservative for the k-server problem. At the other extreme there is the *retrospective algorithm*, which moves servers to a configuration A', with $\gamma_i \subset A'$, that minimizes $w_i(A')$. The idea behind this algorithm is that the off-line algorithm that has its servers at A' seems to be the best so far. It would seem that a combination of these two algorithms might be a good idea; the work function algorithm combines the virtues of both of them.

Definition 5.3. (Work Function Algorithm) *Let Γ^{i-1} be a request sequence and let A_{i-1} be the configuration of an on-line algorithm after servicing Γ^{i-1}. The work function algorithm services a new request $\gamma_i = \{a_i, b_i\}$ by moving its servers to a configuration A_i, with $\gamma_i \subset A_i$, that minimizes $w_i(A_i) + D(A_{i-1}, A_i)$.*

According to the above definition of *WFA*, because of the triangle inequality of function g, we can assume that $A_i = A_{i-1} - s_i - t_i + a_i + b_i$ minimizes $w_i(A_i) + D(A_{i-1}, A_i)$ for $s_i, t_i \in A_{i-1}$. Using this, we see that:

$$w_i(A_{i-1}) = \min_{x,y \in A}\{w_i(A_{i-1} - x - y + a_i + b_i) + g(x, y, a_i, b_i)\} = w_i(A_i) + g(s_i, t_i, a_i, b_i).$$

Obviously, the cost of the work function algorithm to service request $\gamma_i = \{a_i, b_i\}$ is simply $g(s_i, t_i, a_i, b_i)$. Thus, the cost of $C_{\text{on}}(\Gamma)$ that WFA pays for serving a sequence of requests $\Gamma = (\gamma_1, \gamma_2, \cdots, \gamma_m) = (\{a_1, b_1\}, \{a_2, b_2\}, \cdots, \{a_m, b_m\})$ is:

$$C_{\text{on}}(\Gamma) = \sum_{i=1}^{m} D(A_{i-1}, A_i) = \sum_{i=1}^{m} g(s_i, t_i, a_i, b_i)$$

where A_i, s_i, and t_i are computed as described in their definitions. In order to derive a bound for the competitive ratio of the work function algorithm, the cost of an optimal off-line algorithm must be considered. Herein, we introduce the concept of an *off-line pseudo-cost* [21], which is a simple and surprisingly accurate estimation of the off-line cost. The off-line *pseudo-cost* of the move from configuration A_{i-1} to A_i is defined as:

$$w_i(A_i) - w_{i-1}(A_{i-1}).$$

Obviously, the total off-line pseudo-cost is equal to the total off-line cost; the optimal cost of the off-line algorithm for serving the same sequence Γ is:

$$C_{\text{opt}}(\Gamma) = \sum_{i=1}^{m} (w_i(A_i) - w_{i-1}(A_{i-1})).$$

Clearly, we can assume that in the worst case, the final configuration of the on-line algorithm is the same as the final configuration of the optimal off-line algorithm. Otherwise, by extending the request sequence made to the on-line algorithm with requests to reach the final configuration of the off-line algorithm, the off-line cost remains unaffected while the on-line cost increases.

The sum of the off-line pseudo-cost and the on-line cost at the *i*-th step is:

$$w_i(A_i) - w_{i-1}(A_{i-1}) + g(s_i, t_i, a_i, b_i),$$

which is equal to $w_i(A_{i-1}) - w_{i-1}(A_{i-1})$. This quantity is obviously bounded by its maximum value over all possible configurations. Therefore, the off-line pseudo-cost plus the on-line cost is bounded above by:

$$\max_X \{w_i(X) - w_{i-1}(X)\}.$$

Next, we introduce the concept of the *extended cost.*

Definition 5.4. (Extended Cost) *The extended cost of serving the i-th request in the sequence of Γ is:*

$$\psi_i = \max_X \{w_i(X) - w_{i-1}(X)\},$$

while the total extended cost of serving the entire sequence Γ is:

$$\Psi(\Gamma) = \sum_{i=1}^{m} \psi_i .$$

The extended cost *occurs on a configuration A* when *A* maximizes the quantity in the extended cost.

Clearly, by definition of the competitive ratio, and the definition of the extended cost, we have the following Lemma 5.4:

Lemma 5.4. *For all request sequences Γ, if the total extended cost is bounded above by $c+1$ times the off-line cost plus a constant then the work function algorithm is c-competitive.*

Proof. Obviously, it is enough to show that $\Psi(\Gamma) \geq C_{\text{on}}(\Gamma) + C_{\text{opt}}(\Gamma)$.

$$\begin{aligned}
\Psi(\Gamma) &= \sum_{i=1}^{m} \max_{X}\{w_i(X) - w_{i-1}(X)\} \\
&\geq \sum_{i=1}^{m} (w_i(A_{i-1}) - w_{i-1}(A_{i-1})) \\
&= \sum_{i=1}^{m} (w_i(A_i) + g(s_i, t_i, a_i, b_i) - w_{i-1}(A_{i-1})) \\
&= \sum_{i=1}^{m} g(s_i, t_i, a_i, b_i) + \sum_{i=1}^{m} (w_i(A_i) - w_{i-1}(A_{i-1})) \\
&= C_{\text{on}}(\Gamma) + C_{\text{opt}}(\Gamma)
\end{aligned}$$

□

In fact, the extended cost is an overestimation of the actual on-line cost (plus the optimal off-line cost). It was first introduced in [37] in a different form (they called it an *on-line pseudo-cost*). Later, in [21] the authors propose the concept of the extended cost to replace the on-line pseudo-cost. The advantage of using the extended cost, instead of real cost, is that we do not have to deal with all the configurations of the servers. Instead, in order to prove that the work function algorithm is competitive, we only have to show that a certain inequality holds for all work functions. Its disadvantage, of course, is that it may overestimate the cost of the work function algorithm.

5.3. The Special Case of a Metric Space with $k+2$ Points

In the special case of a metric space with $k+2$ points, we can quickly show the work function algorithm is $\left(\frac{k^2+3k}{2}\right)$-competitive using a potential function.

Theorem 5.5 *In a metric space with $k+2$ points, for the k-server problem with twin-request, the work function algorithm is $\left(\frac{k^2+3k}{2}\right)$-competitive.*

Proof. Considering the following potential function:

$$\Phi = \sum_X w(X).$$

When the request $\gamma_i = \{a_i, b_i\}$ is serviced, we assume that there are two algorithms to schedule servers to satisfy the request. One of the algorithms is an off-line algorithm (called the *off-line adversary*), while the other is the work function algorithm (an on-line algorithm). According to the analysis of section 5.2, it is easy to show that:

$$\Delta\Phi \geq w_i(A) - w_{i-1}(A),$$

where A maximizes the difference $w_i(X) - w_{i-1}(X)$. Then, summing over all the requests we can obtain:

$$\Phi_f - \Phi_0 \geq C_{\text{opt}}(\Gamma) + C_{\text{on}}(\Gamma),$$

where Φ_f and Φ_0 denote the final and initial potential functions, respectively. In addition, we have:

$$\Phi_f \leq \binom{k+2}{2} \cdot \max_X w_f(X),$$

where $w_f(X)$ denotes the work function for any final configuration, and there are exactly $\binom{k+2}{2}$ possible configurations. Next, let the maximum of $w_f(X)$ be achieved at A. Thus, because of Corollary 5.3 (3):

$$w_f(A) \le w_f(C_f) + D(A, C_f),$$

where C_f denotes the final configuration in which both WFA and the relevant optimal off-line algorithm finish serving requests. Then we have,

$$\Phi_f \le \binom{k+2}{2} \cdot w_f(C_f) + \binom{k+2}{2} \cdot D(A, C_f).$$

Note that the last term in this sum is bounded by $\binom{k+2}{2}$ times the distance between any two points in the metric space, and hence it is bounded by a constant. Thus, we have shown that:

$$C_{\text{opt}}(\Gamma) + C_{\text{on}}(\Gamma) \le \Phi_f - \Phi_0 \le \binom{k+2}{2} \cdot w_f(C_f) + \text{constant}.$$

According to the definition of the work function and the optimal solution, we have:

$$C_{\text{opt}}(\Gamma) = w_f(C_f).$$

Finally, we have,

$$C_{\text{on}}(\Gamma) \le \left(\frac{k^2+3k}{2}\right) \cdot C_{\text{opt}}(\Gamma) + \text{constant}.$$

The proof is complete.□

In fact, for the k-sever problem with twin-request, in a metric space with n points (of course, $n \geq k > 2$, otherwise the problem is trivial), we have a similar result. That is:

Theorem 5.6 *For the k-server problem with twin-request in a metric space with n points, if* $n \geq k > 2$ *holds, the work function algorithm is* $\left(\binom{n}{k}-1\right)$*-competitive.*

The proof of this theorem is similar to the proof of Theorem 5.5. Therefore, we omit the proof. Although this result is very weak, it is the best bound we know of which is dependent on the distances between points in a metric space.

5.4 A Lower Bound

In this section, we prove that any general algorithm for the symmetric k-server problem with twin-request must have a competitive ratio of at least $k/2$. In fact, we have actually proven a slightly more general lower bound. Suppose we wish to compare an on-line algorithm solving a problem with k servers to an off-line algorithm solving a problem with $h \leq k$ servers. Naturally, the competitive ratio decreases for the on-line algorithm because it has to handle more servers than the off-line algorithm. We obtain the lower bound as $\frac{k}{k-h+2}$. A similar approach was taken in [13], where the lower bound and matching upper bound are given for the traditional k-server problem.

Theorem 5.7 *Let A be an online algorithm for the symmetric k-server problem with twin-request on a graph G with at least k+2 nodes. Then, for any* $1 \leq h \leq k$*, there exists request sequences* Γ^1, Γ^2, *..., such that:*

1) For all i, Γ^i is an initial subsequence of Γ^{i+1}, and $C_A(\Gamma^{i-1}) < C_A(\Gamma^i)$.

2) There exists an h-server algorithm B (which may start with its servers anywhere), such that for all i, $C_A(\Gamma^i) > \frac{k}{k-h+2} \times C_B(\Gamma^i)$.

Proof. Without loss of generality, assume *A* is a lazy algorithm, and that the *k* servers all start out at different nodes. Let *H* be a sub-graph of *G* of size *k*+2, including the *k* initial positions of *A*'s servers, plus two other vertices. For request sequence Γ, $\gamma_i = \{a_i, b_i\}$ is the twin-request, a set containing two requesting nodes at time *i*.

Define Γ as algorithm *A*'s nemesis sequence on *H*, such that γ_i is also the set of the two vertices in *H* that are not covered by *A* at time *i*, for all $i \geq 1$. Of course, there is $\gamma_i \cap \gamma_{i+1} = \varnothing$. Without loss of generality, assume that $g(a_i, b_i, a_{i+1}, b_{i+1}) = d(a_i, a_{i+1}) + d(b_i, b_{i+1})$. Then, it follows that:

$$C_A(\sigma, t) = \sum_{i=1}^{t} g(a_i, b_i, a_{i+1}, b_{i+1}) = \sum_{i=1}^{t} \left(d(a_i + a_{i+1}) + d(b_i, b_{i+1}) \right).$$

This is because at each step in servicing the Γ requests the nodes have just been vacated by algorithm *A*: at step *i*, the server at a_{i+1} will move to a_i, and the server at b_{i+1} will move to b_i.

Let *S* be any *h*-element subset of *H* containing γ_1. We can define an off-line *h*-server algorithm *A*(*S*) as follows: the servers initially occupy the vertices in the set *S*. To process a request $\gamma_i = (a_i, b_i)$, the following rule is applied:

If a_i is not in S, move the server at a_{i-1} to a_i. If b_i is not in S, move the server at b_{i-1} to b_i. Otherwise, do nothing. Update S to reflect any change (i.e., S continues to cover all servers).

Therefore, it is easy to see that for all $i>1$, the set S contains γ_{i-1} when step i begins.

The following observation is the key to the rest of the proof: if the above algorithm starts with distinct equal-sized sets, S and T, then S and T never become equal. The reason is described in the following paragraph.

Suppose that, S and T differ before γ_i is processed. We shall show that the versions of S and T created by processing γ_i as described above must also differ.

i. $a_i \in S$, $a_i \notin T$. After it is updated, S contains a_{i-1}, and T does not contain a_{i-1}. Therefore, S and T are different.

ii. $b_i \in S$, $b_i \notin T$. After it is updated, S contains b_{i-1}, and T does not contain b_{i-1}. Therefore, S and T are also different.

iii. Both S and T contain a_i and b_i, or neither contains a_i and b_i. They will perform the same actions. They will remain different after they are updated.

Let us consider simultaneously running an ensemble of algorithms *A*(*S*), starting from each *h*-element subset *S* of *H* containing γ_1. There are $\binom{k}{h-2}$ such sets. Since no two sets ever become equal, the number of distinct sets remains constant. After processing γ_i, the collection of subsets consists of all the *h* element subsets of *H*, which contain γ_i.

By our choice of starting configuration, step 1 never costs anything. At step *i* (for *i*≥1), each of these algorithms has four choices: (1) do nothing (no cost), (2) only move a server at a_{i-1} to a_i (cost $d(a_{i-1},a_i)$), (3) only move a server at b_{i-1} to b_i (cost $d(b_{i-1},b_i)$), or (4) move two servers combining choices 2 and 3 (cost $d(a_{i-1},a_i)+d(b_{i-1},b_i)$). Of the $\binom{k}{h-2}$ algorithms begin run, $\binom{k-2}{h-2}$ of them (i.e., the ones which contain a_{i-1} and b_{i-1}, but do not contain a_i and b_i) incur the cost $d(a_{i-1},a_i)+d(b_{i-1},b_i)$. While $\binom{k-2}{h-3}$ of the algorithms (i.e., the ones which contain a_{i-1} , b_{i-1} and b_i, but do not contain a_i) incur the cost $d(a_{i-1},a_i)$. Finally, $\binom{k-2}{h-3}$ of the algorithms (i.e., the ones which contain a_{i-1}, b_{i-1} and a_i, but do not contain b_i) incur the cost $d(b_{i-1},b_i)$. The total cost from running all of these algorithms, up to and including request γ_t is:

$$\sum_{i=1}^{t-1}\binom{k-2}{h-2}\cdot\left(d(a_i,a_{i+1})+d(b_i,b_{i+1})\right)+\binom{k-2}{h-3}\cdot d(a_i,a_{i+1})+\binom{k-2}{h-3}\cdot d(b_i,b_{i+1})$$
$$=\sum_{i=1}^{t-1}\binom{k-1}{h-2}\times\left(d(a_i,a_{i+1})+d(b_i,b_{i+1})\right)$$

Thus the expected cost of one of these algorithms chosen at random is:

$$\frac{\binom{k-1}{h-2}}{\binom{k}{h-2}} \times \sum_{i=1}^{t-1} \left(d(a_i, a_{i+1}) + d(b_i, b_{i+1}) \right)$$

Recall that, the cost to A for the same steps was:

$$\sum_{i=1}^{t} \left(d(a_i, a_{i+1}) + d(b_i, b_{i+1}) \right)$$

Since the distances are symmetric, the two summations are almost identical, except the second summation includes an extra term.

By expanding the binomial coefficients we see that:

$$\frac{\binom{k-1}{h-2}}{\binom{k}{h-2}} = \frac{k-h+2}{k}$$

Finally, there must be some initial set that has the property that its performance is always no worse than the average of the costs across all the algorithms. Let S be this set, and $A(S)$ be the algorithm starting from this set. Let Γ_i be all the initial subsequences of Γ for which $A(S)$ does no worse than average.□

5.5. Open Problems

In this thesis, the concepts concerning the k-server problem are extended to the k-server problem with multi-request. This extension makes the model of the k-sever problem more realistic. In the real world service requests can occur simultaneously. However, the time complexity and the space complexity of the problem are increased. Although we only focus on the twin-request, this problem is obviously more complicated than the k-server problem with a single request. Naturally, the most obvious open problem is to improve the lower bound. We believe that there is a tighter lower bound for the k-server problem with twin-request than the lower bound presented in this thesis. Other unresolved problems include: proving that the work function algorithm has a better competitive ratio for the k-server problem with twin-request, or developing another on-line algorithm to resolve the general case or special case of the k-server problem with twin-request.

Chapter 6

On the On-line Number of Snacks Problem

In the *Number of Snacks Problem* (*NSP*), an on-line player is given the task of deciding how many snacks his noshery should prepare each day. The on-line player must make his decision and then finish the preparation before the customers come to his noshery for the snacks. His goal is to minimize the competitive ratio, defined as: $\inf_\sigma \frac{C_A(\sigma)}{C_{OPT}(\sigma)}$, where σ ranges over a sequence of customers, $C_{OPT}(\sigma)$ is the cost to satisfy σ by an optimal off-line algorithm, and $C_A(\sigma)$ is the cost to satisfy σ by the on-line algorithm. In this thesis, we present a competitive algorithm for the on-line number of snacks problem $P1$: the *Extreme Numbers Harmonic Algorithm* (*ENHA*), which has a competitive ratio $1+p\cdot\frac{M-m}{M+m}$, where M and m represent two extremes for the number of customers (over the total period of the game), and p is a ratio concerning the cost of these two extreme situations. Next, we prove that this competitive ratio is the best obtainable, if the on-line player chooses a fixed number of snacks for any sequence of customers. Furthermore, we also discuss several variants of the *NSP* and obtain some results. Finally, we propose a conjecture for the on-line *NSP*.

6.1 Introduction

Many decision-making activities, such as currency exchange, stock transactions or mortgage financing, must be carried out in an on-line fashion, with no secure knowledge of future events. Faced with this lack of knowledge, decision makers often have two ways to choose: (1) use models based on assumptions about the future distribution of relevant quantities such as exchange rates or mortgage rates, and aim for acceptable results on the average; (2) analyze the worst case and then make a decision. Unfortunately, these two approaches may give a solution that is far from the optimal solution.

An alternate approach in such situations, and the one we explore here, is to use *competitive analysis* (first applied to on-line algorithms by Sleator and Tarjian in [9]). In devising a competitive strategy, the on-line player is not able to escape the need to make some assumptions, or to have some knowledge about future events, but these do not have to be probabilistic in nature. For example, instead of knowledge concerning the distribution of future numbers of customers, an on-line strategy might be based only on knowledge of the bounds on the possible number of customers over the period in question. This strategy should work well no matter how erratically the number of customers varies each day.

The objective of the on-line NSP is to decide how many snacks should be prepared without knowing the number of customers that will arrive. We use the adversary method to solve this problem. For any on-line NSP algorithm, we imagine that there is an off-line adversary who can design the sequence of customers in an attempt to make the competitive ratio of the on-line algorithm as high as possible (i.e., worse performance). We investigate different versions of this problem by varying the on-line player's knowledge. For these different versions, we show that some surprisingly small (good) competitive ratios can be achieved under very moderate assumptions about the on-line player's knowledge: only upper and lower bounds on the possible number of customers needs to be known.

6.2 The Number of Snacks Problem

6.2.1 Problem Statement

In the number of snacks problem, an on-line player is given the task of deciding the number of snacks to prepare (for a certain time period) without knowing how many customers will actually arrive at his noshery. On any given day, the on-line player must tell his staff to prepare a certain number of snacks in the morning for the customers that arrive later (e.g., at noon). The problem is designated as an *on-line* problem because the player must make a decision without any foreknowledge. There are three cases as follows:

- The noshery prepares m snacks, and then m customers arrive. The on-line player can sell all the snacks without any being wasted or losing any potential sales. In this situation we assume that the on-line player supplies the snack at a cost of c per snack, and the total cost is $m \cdot c$.

- The noshery prepares m snacks, but there are only m_1 customers, where $m_1 < m$. The on-line player must incur a cost c_1 per snack to deal with the surplus $m - m_1$ snacks. In this situation, the on-line player's total cost is $m_1 \cdot c + (m - m_1) \cdot c_1$. Considering the economic meaning, if we assume that $c_1 > c$ (i.e., it costs more to dispose of the surplus than it costs to produce) and let $c_1 = pc$, where $p \geq 1$, the above formula is rewritten as $m_1 \cdot c + (m - m_1) \cdot pc$.

- The noshery prepares m snacks, but there are m_2 customers where $m_2 > m$. The on-line player can quickly supply the shortfall in snacks at a higher production cost, so the on-line player needs to produce the $m_2 - m$ snacks at cost c_2 per snack. In this situation, the on-line player's cost is $m \cdot c + (m_2 - m) \cdot c_2 = m_2 \cdot c + (m_2 - m) \cdot (c_2 - c)$. Obviously $c_2 > c$, if we let $c_2 = qc$, where $q \geq 1$, the above formula can be rewritten as $m_2 \cdot c + (m_2 - m) \cdot (q - 1) \cdot c$.

Considering the following two problems:

1) If the on-line player knows the exact number of customers every time, they can make an optimal production decision, provided the required number of snacks can be prepared (i.e., there is no production capacity constraint). The player always obtains the optimum cost.

2) If the number of customers arriving is unknown beforehand, then the on-line player makes the production decision for the snacks required for the next day.

Obviously, problem (1) is an *off-line* problem and (2) is an *on-line* problem. The *off-line* problem can be solved easily. The optimal solution for the *off-line* problem can be obtained if the player prepares sufficient snacks to meet the demand each day.

Problem (2) is very difficult for the decision maker; he/she never knows the actual number of customers in advance. He/she, by definition, must make the decision in an on-line manner: deciding how many snacks should be prepared without any foreknowledge.

Considering the general model for the number snacks problem, we denote the actual sequence of the number of customers by $\sigma = (d_1, d_2, \cdots, d_n)$, where d_i means the actual number of customers on the ith day. Similarly, we denote by $\sigma' = (d_1', d_2', \cdots, d_n')$ the sequence of the number of snacks produced prior to customers arriving. Therefore, the on-line decision maker prepares d_i' snacks for the ith day. Let $C_{OPT}(\sigma)$ denote the off-line optimal cost to finish servicing σ; let $C_A(\sigma)$ denote the on-line cost for the same sequence, and let c, c_1, and c_2 denote the costs defined previously for production, disposing of surplus production, and expediting production to meet a shortfall, respectively. Obviously, we can obtain the optimal for the off-line problem for a certain σ:

$$C_{opt}(\sigma) = c \cdot \sum_{i=1}^{n} d_i .$$

For any on-line algorithm A for this problem, we denote the on-line cost by:

$$C_A(\sigma) = c \cdot \sum_{i=1}^{n} d_i + (c_2 - c) \cdot \sum_{\substack{i=1 \\ d_i > d_i'}}^{n} (d_i - d_i') + c_1 \cdot \sum_{\substack{i=1 \\ d_i' > d_i}}^{n} (d_i' - d_i)$$

For any on-line algorithm A, the competitive ratio is defined as $\inf_\sigma \dfrac{C_A(\sigma)}{C_{OPT}(\sigma)}$. A small competitive ratio implies that A can do well in comparison with the optimal solution. The question then is: how can we design competitive algorithms with good competitive ratios for this on-line problem? We will answer this question in the following sections.

6.2.2 Assumptions about the Fluctuation in the Number of Customers

The on-line player's prior information about the number of customers arriving defines particular variants of the game. We define the following terms for the subsequent discussion:

M = upper bound on the possible number of customers, over the whole game, where M_i = upper bound on the ith day.

m = lower bound on the possible number of customers, over the whole game, where m_i = lower bound on the ith day.

$\Phi = M/m$ is called the *global fluctuation ratio*, where $\Phi_i = M_i/m_i$ is called the *local fluctuation ratio* on ith day.

n = the number of days in the game.

6.2.3. Results

In this thesis, optimal on-line algorithms are produced for the four variants of the number of snacks problem, as follows:

- Variant 1. The general version, problem *P*, without any constraints for c_1 or c_2, and with *M*, *m* and (of course) Φ known to the on-line player.

- Variant 2. The degenerative version of variant 1, problem *P1*, with $c_1 = c_2 - c$, (i.e., $p = q - 1$, where $p \geq 1$), and with *M*, *m* and Φ known to the on-line player.

- Variant 3. The special version, problem *P′*, with m_i and Φ_i known to the player, where $\Phi_i = \Phi_0$, and Φ_0 means a constant global fluctuation ratio, for $i=1,2,\ldots,n$, and without any constraints for c_1 or c_2.

- Variant 4. The degenerative special version, problem *P′′*, with m_i and Φ_i known to on-line player, where $\Phi_i = \Phi_0$, for $i=1,2,\ldots,n$, and with $c_1 = c_2 - c$, (i.e., $p = q - 1$, where $p \geq 1$).

For variants 1 and 2, the game progresses in the following fashion: the on-line player chooses a strategy at the beginning of the game. His/her knowledge only concerns *M*, *m* and Φ over the course of the game. For problem *P1*, we derive an optimal competitive algorithm *ENHA*. For problem *P*, we also derive a similar optimal competitive algorithm *TENHA*.

However, for variants 3 and 4, the on-line player only knows some local information (for example, m_i and Φ_i), and then chooses a strategy to solve the problem. In this paper, we only investigate the case of $\Phi_i = \Phi_0$, for $i=1,2,\ldots,n$. The problems for the more general cases are still open. We provide a competitive algorithm *LENHA* for P'' and a competitive algorithm *TLENHA* for P'.

We also discuss lower bounds for the competitive ratios for all competitive algorithms for the four variants.

6.3 The Degenerate Version of *P* — Problem *P1*

6.3.1 Extreme Numbers Harmonic Algorithm

Extreme Numbers Harmonic Algorithm: *For the on-line number of snacks problem P1, if the on-line player knows the lower (m) and upper (M) bounds of the number of customers, he can always prepare* $\bar{d} = \dfrac{2Mm}{M+m}$ *snacks each day.*

6.3.2 Competitive Ratio of the Extreme Numbers Harmonic Algorithm

Theorem 6.1. *For the on-line number of snacks problem P1, the Extreme Numbers Harmonic Algorithm is a* $\left(1+p\cdot\dfrac{M-m}{M+m}\right)$ *-competitive or* $\left(1+p\cdot\dfrac{\Phi-1}{\Phi+1}\right)$ *-competitive algorithm.*

Proof. Obviously, if M and m were known, and $c_1 = c_2 - c$, (i.e., $p = q-1$, where $p \geq 1$), the on-line cost of ENHA (denoted by A) satisfies the following formula:

$$\begin{aligned} C_A(\sigma) &= c\cdot\sum_{i=1}^{n} d_i + (c_2 - c)\cdot \sum_{\substack{i=1\\ d_i > d_i'}}^{n}(d_i - d_i') + c_1 \cdot \sum_{\substack{i=1\\ d_i' > d_i}}^{n}(d_i' - d_i) \\ &= c\cdot\sum_{i=1}^{n} d_i + pc\cdot\sum_{i=1}^{n}\left|d_i - d_i'\right| \end{aligned} \qquad (1).$$

If the on-line player chooses an ENHA strategy, the off-line adversary can obviously choose d_i=M or d_i=m, to make the competitive ratio as large (worse) as possible. Let d_i=M and let σ_M denote the sequence of customers with all d_i=M. According to ENHA, $d_i' = \bar{d} = \frac{2Mm}{M+m} \le d_i = M$, for i=1,2,..., n. Then, we obtain:

$$\begin{aligned} C_A(\sigma_M) &= c\cdot\sum_{i=1}^{n} d_i + pc\cdot\sum_{i=1}^{n}\left|d_i - \bar{d}\right| \\ &= c\cdot nM + pc\cdot n\left(M - \frac{2Mm}{M+m}\right) \\ &= \left(1 + p\cdot\frac{M-m}{M+m}\right)\cdot cnM \\ &= \left(1 + p\cdot\frac{M-m}{M+m}\right)\cdot C_{OPT}(\sigma_M) \end{aligned} \qquad (2).$$

Similarly, we can obtain the following result if we let d_i=m and denote the sequence of customers under this condition by σ_m. Considering $\bar{d} \ge m$ holds:

$$\begin{aligned} C_A(\sigma_m) &= c\cdot\sum_{i=1}^{n} d_i + pc\cdot\sum_{i=1}^{n}\left|d_i - \bar{d}\right| \\ &= c\cdot nm + pc\cdot n\left(\frac{2Mm}{M+m} - m\right) \\ &= \left(1 + p\cdot\frac{M-m}{M+m}\right)\cdot cnm \\ &= \left(1 + p\cdot\frac{M-m}{M+m}\right)\cdot C_{OPT}(\sigma_m) \end{aligned} \qquad (3).$$

The extreme situations (d_i=M or d_i=m) are the worst cases, therefore, no other σ can lead to a worse case that increases the competitive ratio. For any σ:

$$\frac{C_A(\sigma)}{C_{OPT}(\sigma)} \leq 1 + p \cdot \frac{M-m}{M+m} \quad (4).$$
$$= 1 + p \cdot \frac{\Phi - 1}{\Phi + 1}$$

The proof is complete.□

6.3.3 A Lower Bound of the Competitive Ratio

The *ENHA* gives a lower bound for the competitive ratio.

Theorem 6.2. *For the on-line number of snacks problem* $P1$, *if the off-line adversary chooses a strategy with a fixed number of customers, the competitive ratio is* $\left(1 + p \cdot \frac{M-m}{M+m}\right)$ *or* $\left(1 + p \cdot \frac{\Phi-1}{\Phi+1}\right)$, *which is given by ENHA, and this is a lower bound for the competitive ratio.*

Proof. We need to prove that if the on-line player chooses another fixed number as his decision (denoted by algorithm A'), the competitive ratio would become worse, that is the player chooses $\bar{d}' \neq \frac{2Mm}{M+m}$. Without loss of generality, we assume that $\bar{d}' < \frac{2Mm}{M+m}$. Then we prove that if an off-line adversary chooses σ_M the competitive ratio $\left(1 + p \cdot \frac{M-m}{M+m}\right)$ cannot be achieved. According to the previous statements, we have:

$$\begin{aligned} C_{A'}(\sigma_M) &= c\cdot\sum_{i=1}^{n} d_i + pc\cdot\sum_{i=1}^{n}\left|d_i - \bar{d}'\right| \\ &= c\cdot nM + pc\cdot n\left(M - \bar{d}'\right) \\ &= \left(1 + p\cdot\left(1 - \frac{\bar{d}'}{M}\right)\right)\cdot cnM \\ &= \left(1 + p\cdot\left(1 - \frac{\bar{d}'}{M}\right)\right)\cdot C_{OPT}(\sigma_M) \\ &> \left(1 + p\cdot\frac{M-m}{M+m}\right)\cdot C_{OPT}(\sigma_M) \end{aligned} \qquad (5).$$

The last inequality holds for $\bar{d}' < \frac{2Mm}{M+m}$. This means that if an off-line adversary chooses σ_M, the performance of the on-line algorithm A' is worse than A. Similarly, under the condition with $\bar{d}' > \frac{2Mm}{M+m}$, the same result can be obtained.□

6.3.4 The Upper Bound of the Competitive Ratio

If an on-line player chooses σ'_M but an off-line adversary chooses σ_m, we can obtain an upper bound for the competitive ratio for the on-line number of snacks problem, as follows:

$$1 + p\cdot\left(\frac{M}{m} - 1\right) = 1 + p\cdot(\Phi - 1).$$

Thus if an on-line player chooses any integer number from m to $\frac{2Mm}{M+n}$, then he/she will get a competitive ratiothat is between $1 + p\cdot\frac{\Phi - 1}{\Phi + 1}$ and $1 + p\cdot(\Phi - 1)$.

6.4 The General Version — Problem *P*

6.4.1 Transformative Extreme Numbers Harmonic Algorithm

Obviously, results from section 6.3 can be used to produce an on-line algorithm for *P*. In fact, we propose an on-line algorithm for *P* as follows:

Transformative Extreme Numbers Harmonic Algorithm: *For an on-line number of snacks problem P, if the on-line player knows the lower (m) and upper (M) bounds of the number of customers, then he/she can choose a fixed number* $\bar{\bar{d}} = \frac{Mm\cdot(p+q-1)}{M\cdot p+m\cdot(q-1)}$ *snacks.*

6.4.2 Competitive Ratio

From the above on-line algorithm, we can easily obtain the following theorem:

Theorem 6.3. *For the on-line number of snacks problem P*, the *Transformative Extreme Numbers Harmonic Algorithm is a* $\left(1+p\cdot(q-1)\frac{M-m}{M\cdot p+m\cdot(q-1)}\right)$*-competitive or a* $\left(1+p\cdot(q-1)\frac{\Phi-1}{\Phi\cdot p+(q-1)}\right)$*-competitive algorithm.*

The proof for theorem 6.3 is similar to that of theorem 6.1. We omit the proof in this thesis.

6.4.3 Lower Bound

For an on-line number of snacks problem *P*, we also have the following theorem:

Theorem 6.4. *For an on-line number of snacks problem P, if the off-line adversary chooses a strategy with a fixed number of customers, the competitive ratio is* $\left(1+p\cdot(q-1)\frac{M-m}{M\cdot p+m\cdot(q-1)}\right)$ *or* $\left(1+p\cdot(q-1)\frac{\Phi-1}{\Phi\cdot p+(q-1)}\right)$, *which is given by TENHA, and this is a lower bound.*

We also omit this proof since it is similar to the proof of Theorem 6.2.

6.5 The Degenerate Special Version of P' — Problem P''

6.5.1 Local Extreme Numbers Harmonic Algorithm

For problem P'', we have the following algorithm:

Local Extreme Numbers Harmonic Algorithm: *For problem P'', if the on-line player knows the lower bound (m_i) and local fluctuation ratio (Φ_i=Φ_0) of the number of customers on the ith day,* for *i=1,2,...n, and if* $c_1 = c_2 - c$ *(i.e.,* $p = q-1$, *where* $p \geq 1$*), he/she can prepare* $\bar{d}_i = \frac{2\Phi_0 \cdot m_i}{\Phi_0 + 1}$ *snacks on the ith day.*

6.5.2 Competitive Ratio

From the above algorithm, we have the following theorem:

Theorem 6.5. *For problem P'', LENHA is a* $\left(1+p\cdot\frac{\Phi_0-1}{\Phi_0+1}\right)$*-competitive algorithm.*

Proof. If m_i and $\Phi_i=\Phi_0$, for $i=1,2,\ldots n$ are known by the on-line player, and $c_1 = c_2 - c$ (i.e., $p = q - 1$, where $p \geq 1$), for the *LENHA* strategy, the off-line adversary can obviously choose $d_i = \Phi_0 \cdot m_i$ or $d_i=m_i$, to make the competitive ratio as large (worse) as possible. Let $d_i=m_i$ and let σ_{m_i} denote the sequence of the number of customers arriving. According to *LENHA* and formula (1) from section 6.3.2, the on-line cost of σ_{m_i} satisfies:

$$\begin{aligned}
C_A(\sigma_{m_i}) &= c \cdot \sum_{i=1}^{n} d_i + pc \cdot \sum_{i=1}^{n} |d_i - \bar{d}_i| \\
&= c \cdot \sum_{i=1}^{n} m_i + pc \cdot \sum_{i=1}^{n} \left(\frac{2\Phi_0 \cdot m_i}{\Phi_0 + 1} - m_i \right). \\
&= \left(1 + p \cdot \frac{\Phi_0 - 1}{\Phi_0 + 1} \right) \cdot c \cdot \sum_{i=1}^{n} m_i \\
&= \left(1 + p \cdot \frac{\Phi_0 - 1}{\Phi_0 + 1} \right) \cdot C_{OPT}(\sigma_{m_i})
\end{aligned}$$

Where the second and subsequent steps in this mathematical derivation hold for $\bar{d}_i \geq d_i = m_i$.

Similarly, we can obtain the following result if we let $d_i = \Phi_0 \cdot m_i$ and denote the sequence of the number of customers under this condition by σ_{M_i}. Considering $\bar{d}_i \leq \Phi_0 \cdot m_i$ holds:

$$\begin{aligned}
C_A(\sigma_{M_i}) &= c \cdot \sum_{i=1}^{n} d_i + pc \cdot \sum_{i=1}^{n} |d_i - \bar{d}_i| \\
&= c \cdot \sum_{i=1}^{n} (\Phi_0 \cdot m_i) + pc \cdot \sum_{i=1}^{n} \left(\Phi_0 \cdot m_i - \frac{2\Phi_0 \cdot m_i}{\Phi_0 + 1} \right) \\
&= \left(1 + p \cdot \frac{\Phi_0 - 1}{\Phi_0 + 1} \right) \cdot c \cdot \sum_{i=1}^{n} (\Phi_0 \cdot m_i) \\
&= \left(1 + p \cdot \frac{\Phi_0 - 1}{\Phi_0 + 1} \right) \cdot C_{OPT}(\sigma_{M_i})
\end{aligned}$$

These extreme situations ($d_i = \Phi_0 \cdot m_i$ or $d_i=m_i$) are the worst possible cases for the competitive ratio. For any σ:

$$\begin{aligned}\frac{C_A(\sigma)}{C_{OPT}(\sigma)} &\leq 1+p\cdot\frac{M-m}{M+m}. \\ &= 1+p\cdot\frac{\Phi_0-1}{\Phi_0+1}\end{aligned}$$

The proof is complete.□

6.5.3 A Lower Bound of the Competitive Ratio

LENHA provides a lower bound for the competitive ratio.

Theorem 6.6. *For problem P'', if the on-line player choose a fixed number as his strategy on the ith day, then the competitive ratio* $\left(1+p\cdot\frac{\Phi_0-1}{\Phi_0+1}\right)$ *is a lower bound of the competitive ratio.*

The proof is omitted in this thesis.

6.6 The Special Version — Problem P'

6.6.1 Transformative Local Extreme Numbers Harmonic Algorithm

For problem P', we give the on-line algorithm as follows:

Transformative Local Extreme Numbers Harmonic Algorithm: *For problem P', if the on-line player knows the lower bound (m_i) and local fluctuation ratio ($\Phi_i=\Phi_0$)*

of the number of customers on the ith day, for *i=1,2,...n, and without any constraints for c_1 or c_2, the optimal on-line algorithm is to choose a fixed number $\bar{\bar{d}}_i = \frac{\Phi_0 \cdot m_i \cdot (p+q-1)}{\Phi_0 \cdot p + (q-1)}$ snacks on the ith day of the game.*

6.6.2 Competitive Ratio

Theorem 6.7. *For problem P', the TLENHA is a $\left(1 + p \cdot (q-1)\frac{\Phi_0 - 1}{\Phi_0 \cdot p + (q-1)}\right)$-competitive algorithm.*

The proof is omitted in this thesis.

6.6.3 A Lower Bound of Competitive Ratio

Similarly, we have the following result:

Theorem 6.8. *For problem P', if the on-line player chooses a fixed number as his strategy on the ith day, then the competitive ratio $\left(1 + p \cdot (q-1)\frac{\Phi_0 - 1}{\Phi_0 \cdot p + (q-1)}\right)$ is a lower bound for the competitive ratio.*

The proof is omitted in this thesis.

6.7 Evaluation of the Results

6.7.1 Evaluation of the Competitive Ratios

Above, we give some competitive algorithms for the different versions of the *NSP* problem. All the competitive ratios of these algorithms are obviously functions of the

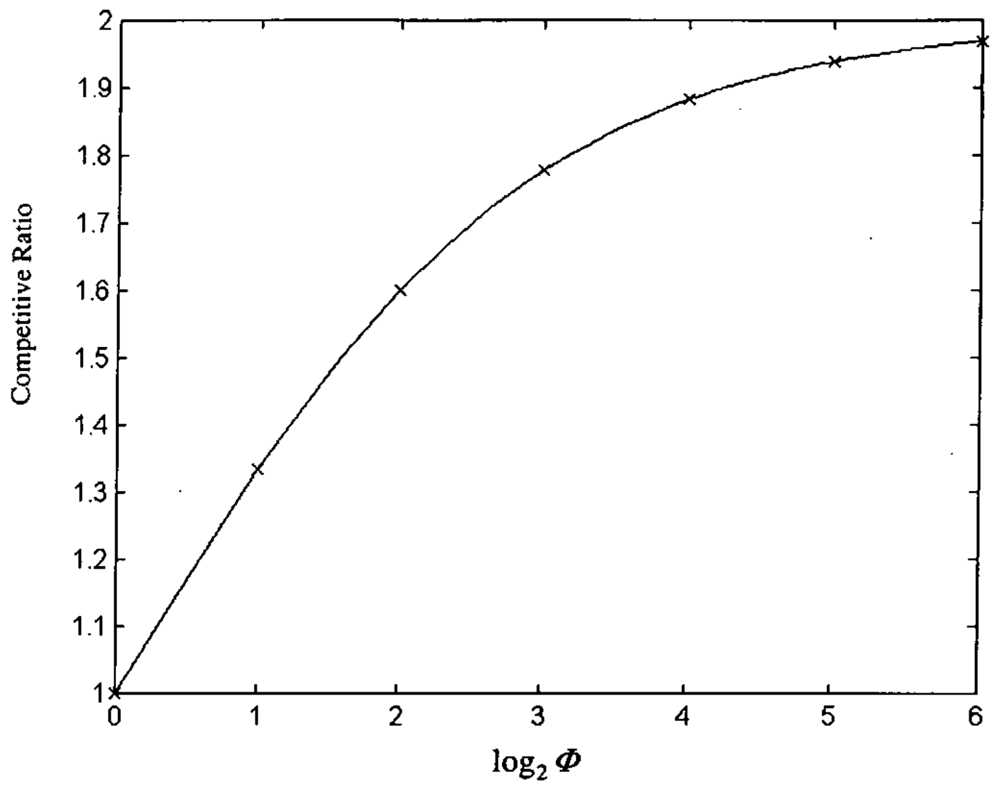

Figure 6.1. The curve of competitive ratio with changing Φ.

fluctuation in the number of customers. For example, for problem *P*1 the competitive ratio (denoted by α) is determined by the function: $\alpha = 1 + p \cdot \frac{\Phi - 1}{\Phi + 1}$. If we fix the value of p, for example, we let p=1, then we can obtain: $\alpha = 2 - \frac{2}{\Phi + 1}$. Clearly, α increases with Φ, but it has an upper bound of 2. In fact, because in the limiting case: $\lim_{\Phi \to \infty}\left(1 + p \cdot \frac{\Phi - 1}{\Phi + 1}\right) = p + 1$, we always have an upper bound for the competitive ratio. The Figure 6.1 shows how the competitive ratio varies with Φ.

6.7.2 Comparison of Problems *P*1 and *P″*

In this thesis, we investigate four variants of the *NSP* and develop some competitive algorithms. Obviously, if we use the same sequence of the number of customers, a more general cases are still open. We provide a competitive algorithm *LENHA* for P'' and a competitive algorithm *TLENHA* for P'.

We also discuss lower bounds for the competitive ratios for all competitive algorithms for the four variants.

6.3 The Degenerate Version of P — Problem $P1$

6.3.1 Extreme Numbers Harmonic Algorithm

Extreme Numbers Harmonic Algorithm: *For the on-line number of snacks problem P1, if the on-line player knows the lower (m) and upper (M) bounds of the number of customers, he can always prepare* $\bar{d} = \frac{2Mm}{M+m}$ *snacks each day.*

6.3.2 Competitive Ratio of the Extreme Numbers Harmonic Algorithm

Theorem 6.1. *For the on-line number of snacks problem P1, the Extreme Numbers Harmonic Algorithm is a* $\left(1+p\cdot\frac{M-m}{M+m}\right)$ *-competitive or* $\left(1+p\cdot\frac{\Phi-1}{\Phi+1}\right)$ *-competitive algorithm.*

Proof. Obviously, if M and m were known, and $c_1 = c_2 - c$, (i.e., $p = q - 1$, where $p \geq 1$), the on-line cost of ENHA (denoted by A) satisfies the following formula:

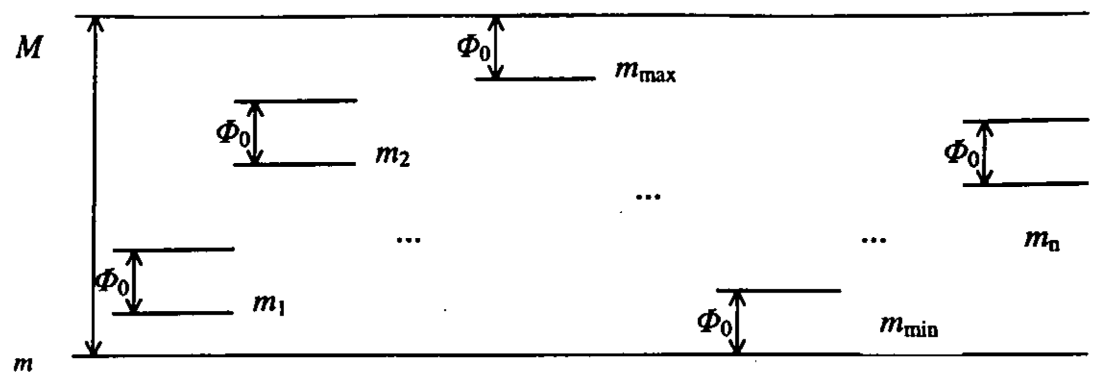

Figure 6.2. The difference between problem *P*1 and Problem *P''*(*n* days game).

comparison of problems *P1* and *P''* reveals some interesting results. The difference between problems *P1* and *P''* is that the on-line player knows the global fluctuation of the entire game, or the fluctuation of the *i*th day of the game. Intuitively, because problem *P''* provides more knowledge to the on-line player, the competitive algorithm should be better (lower) than *P1*. Figure 6.2 shows the difference between the two problems. For the same sequence, clearly we know that $M = m_{\max} \cdot \Phi_0$, $m = m_{\min}$, and therefore $\Phi = m_{\max} \cdot \Phi_0 / m_{\min}$. From Figure 6.1, if we let $\Phi = 16$ and $\Phi_0 = 2$, we obtain two competitive ratios, 1.88 and 1.33, respectively, for problems *P''* and *P*1.

6.8 Conclusions and Future Work

A striking feature of the number of snacks problem is the conceptual simplicity of the optimal strategy. To attain a given competitive ratio, the on-line player simply defends himself/herself against the threat of the adversary's choosing the worst sequence for his on-line strategy.

If we only know the lower (*m*) bound of the number of customers, then what would happen? In this case, how should the on-line player choose a strategy? For *P1*, if we think that the upper bound of the number of customers is $M \to \infty$ ($\Phi \to \infty$), the optimal on-line strategy is to always prepare $2m$ snacks. Actually, this strategy gives a $(p+1)$-competitive algorithm for *NSP P*1.

Some problems concerning the number snacks problem are still unresolved. For example, if we do not know both M and m but only the fluctuation Φ, how can we design an on-line strategy for the number of snacks problem? In this thesis, we only discuss the situation where the on-line player (and his off-line adversary) chooses a fixed number of snacks to produce. We could design other competitive algorithms in which the on-line player can choose varying production numbers that are based on analyzing historical data (as is commonly used in the real world, such as in the enormous body of knowledge related to time series analysis and forecasting techniques). With these competitive algorithms, we can obtain better competitive ratios.

We give lower bounds for the on-line snacks problems *P* and *P1* when the on-line player chooses a fixed number as his snack production strategy. We conjecture that these lower bounds hold for any cases of *P1* and *P* whether the on-line algorithm uses a fixed number or not.

Conjecture 6.1. For any competitive algorithm of the on-line number of snacks problem: $\left(1+p\cdot\frac{\Phi-1}{\Phi+1}\right)$ and $\left(1+p\cdot(q-1)\frac{\Phi-1}{\Phi\cdot p+q-1}\right)$ are the lower bounds of the competitive ratios for P1 and P, respectively.□

Chapter 7

Dynamic Allocation of Mobile Agents by On-line Task Scheduling

This chapter presents a new approach to the dynamic allocation of mobile agents, by on-line task scheduling, for high performance in Internet computing. In contrast to the existing approaches, which apply pre-defined task scheduling schemes to allocate computing resources, we introduce a new on-line competitive algorithm to achieve flexibility. A guided allocation scheme is developed to optimize the allocation of mobile agents, based on the system's competitive ratio. The proposed system combines push-based technology with an innovative on-line task-scheduling scheme to speed up system response and minimize system overheads. Analysis of the system and simulation are used to demonstrate the feasibility and effectiveness of our approach.

7.1 Introduction

With the fast development of information technology and the widespread popularity of the Internet and World Wide Web, the mobile agent paradigm is becoming increasingly important for network-based applications. It is viewed as a new trend in distributed artificial intelligence research, including information gathering, data mining, workflow and electronic commerce [54]. In general, a mobile agent is considered to be a program that represents a user in the network, which is capable of migrating autonomously from one host machine to another [55]. Mobile agents can also be viewed as a mechanism for introducing parallel activities via concurrent execution. According to [55], the design of mobile agent systems involves several key issues such as the provision of code mobility, object naming, portability, scalability and security. The mobile agent paradigm offers a variety of research topics, ranging from low-level system administration tasks to user-level applications.

Task scheduling in the design of mobile agent systems has received much attention in recent years. In [46], an agent-based workflow system is developed where a *workflow-instance agent* is used for task scheduling. The workflow agent identifies which task is to be executed next to achieve an optimal operation. The main advantage of this approach is that there is no central server involved in managing the workflow of executing tasks. However, the workflow needs to be pre-defined and the performance of the system depends on the definition of the workflow. Zakaria Maamar [54] introduces an approach to optimizing a mobile agent itinerary. A *Broker agent* is created to coordinate resource allocation among agents. The set of links that minimizes the value of network traveling, from the first task to the last task, is the optimized itinerary. This system presents a group of intelligent agents with cooperating and task scheduling capabilities, but such capability depends on complete knowledge about the resource locations and costs, which have to be specified beforehand for every link or node value. A utility driven mobile-agent-scheduling scheme is discussed in the paper of Jonathan Bredin [47]. In such a system, the agents are equipped with information about resources in the network. A demand function is defined to guide efficient resource allocation. Unfortunately, quantitative information about a resource and resource consumption relies heavily on traditional microeconomic estimation, which to some extent limits the system's flexibility in a real electronic market.

It could be said that the major advantages of using mobile agents is because of features such as mobility, portability and scalability. A task-scheduling scheme plays an important role in the design of mobile agent systems for high performance. Much of the current work has been focused on allocating resources to mobile agents intelligently, and managing business processes automatically, while mobile agents are generated statically to fulfill a given task or an off-line request sequence. However, there has not been much work on dynamically allocating mobile agents in an on-line fashion for Internet-based applications. It is essential to introduce a new system structure for dynamic allocation of mobile agents using on-line task scheduling to address the limitations of current approaches and to achieve flexibility.

In this chapter, we propose a multi-agent system structure enhanced by push-based technology and an on-line task-scheduling algorithm. Mobile agents are created and cloned dynamically, initialized with service units and pushed from remote sites to local sites that are more convenient for local clients to access, thus speeding up the system response. Push-based technology [48] is currently being proposed in response to communication asymmetry, which is exhibited by many applications, such as news delivery, software distribution, and traffic information systems. In these environments, the communication from the clients to the server is more restricted than the communication from the server to the clients, so it may make more sense to push services from the server without waiting for the clients to pull them. In the proposed system structure, attention is paid to satisfying a client's requests as soon as possible, and minimizing system-handling time, while not knowing what the future requests will be. All potential costs are identified, and an on-line task-scheduling algorithm is sought.

The remainder of this chapter is organized as follows. Section 7.2 outlines the agent-oriented computing paradigm and Section 7.3 highlights the proposed structure for the multi-agent system. An on-line task-scheduling algorithm for dynamically allocating mobile agents is introduced in Section 7.4, and two more corollaries are reported in Section 7.5. Finally, the conclusion is presented in Section 7.6.

7.2 Agent-oriented Computing Paradigm

Compared with the earlier paradigms such as process migration or remote evaluation in distributed computing, the mobile agent model is becoming popular for network-centric programming. The traditional client/server paradigm relies on a *handshake* mechanism to communicate over a network. The client requests information, while the server responds. Each request/response has to be a complete round trip on the network. The emerging mobile-agent paradigm has provided a dynamic and flexible platform for software development and redefined the way Internet-based applications work. As an autonomous software entity, with pre-defined functionality and certain intelligence, a mobile agent is capable of migrating autonomously from one host machine to another, making its request to the server directly and performing tasks on behalf of its master. Some of the advantages of this model are better bandwidth usage, more reliable network connection and reduced software application design work.

During the past few years, more than a dozen Java-based agent systems have been developed. The Java Virtual Machine (JVM), standard security manager and two other functional facilities, namely object serialization and remote method invocation have made it simple to build a mobile agent workbench. Of all these available Java-based agent systems, ObjectSpace's Voyager [50], General Magic's Odyssey [51] and IBM's Aglet [52] are the three leading commercial ones. A detailed discussion on current commercial and research-based agent systems can be found in [49].

Voyager [50] is the first platform to smoothly integrate traditional distributed computing with cutting edge agent technology. *Virtual object* is the core facility and framework in Voyager to support inter-agent communication and migration of agents. It can migrate not only between agent servers, but also to Java runtime environments of other arbitrary *virtual objects*, which is an innovative feature among existing agent systems. However, the use of a *virtual object* brings complexity as well, since most developers have to change their traditional approach to building distributed applications to make use of this new feature. General Magic's Odyssey [51] is another pure Java-based agent development platform. It utilizes Java RMI (Remote Method Invocation) as well as CORBA (Common Object Request Broker Architecture) and DCOM (Distributed Component Object Model) protocols for agent transport, and it provides a *collaboration* facility to allow agents to *meet* at a particular host on the network. However, in such a system, an *rmiregistry* name server must be installed on every machine together with an Odyssey agent server, which increases the complexity of the system's maintenance. The Aglet system [52,53] developed by IBM is chosen as the implementation example in our proposed system. Although, at present, it is not a fully-fledged platform , it has received the most press coverage and shows promise as a functional technology that fits very well into the Java world. Features that characterize an aglet are: lightweight object migration, built with support for persistence, event–driven, etc.

The design of the Aglet system is very clean. The Java Aglet API (J-AAPI) contains methods for initializing an aglet and provides all of the support services for migrating an agent from one host machine to another and for communicating between aglets. An aglet is a composite Java object that includes mobility and persistence, and it has its own thread of execution. The Aglet's mobility for roaming around the network is implemented using Java's object serialization mechanism. The communication between aglets is through message passing, which can be synchronous, one-way or future-reply. As the name of "Aglet" (a pun on agent with applet) suggests, there are several ways in which the aglet model mirrors Java's applet model. The aglet runs in an execution environment called *AgletContext*, which is much like *AppletContext* for an applet. The aglet also has a well planned *life cycle* [56]. Between starting and stopping, an aglet can experience many events such as creation, cloning, dispatching, disposal, etc. Figure 7.1 shows four fundamental operations of an Aglet.

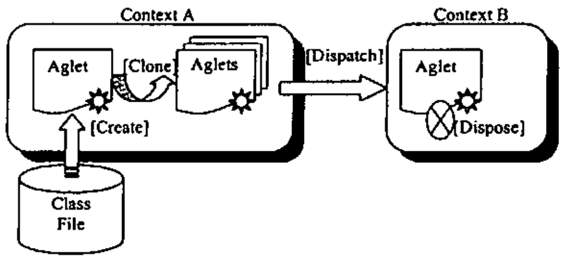

Figure 7.1. Four fundamental operations of an Aglet

Creation is the primary way to produce a new aglet in the current context; its state is initialized during this event. *Cloning* is an important way to create a copy of an original aglet in the same context. The cloned aglet has the same state but a different identifier. *Dispatch* is an active way to ship an aglet from one host to another. Upon arriving at the new host, the aglet restarts its execution. *Disposal* is a useful way to control the number of agents that reside in a context, and thereby reduce resource consumption. These four events will be implemented in our proposed system. In addition, *retraction* provides a way to return an aglet from a remote host to its home site. *Deactivated* and *activated* are ways to push an aglet out of the current context temporarily or to bring it back, respectively. When an aglet migrates from one host to another across the network, it carries with it both code and state information until it returns back to its original host, which is different to the way an applet migrates.

When cooperating to accomplish a given task, multiple aglets always follow certain working patterns [57,58]. The "master/slave" and "worker" are two patterns that are heavily employed in agent-based system design. The master/slave pattern simply means the activity of one agent is controlled by another agent. The master usually has control of the entire life cycle and all the activities of the slave. This is the fundamental working pattern used in distributed computing. Unlike the master/slave pattern, in the worker pattern a "controller" agent dispatches many worker agents to do work on its behalf. The controller does not control the life cycle of the workers but assists these agents to achieve certain fixed goals. Such a pattern demonstrates a real advantage of agents in parallel computing. These flexible working patterns and functional operations have made building multi-agent systems fairly simple.

7.3 Multi-agent System Structure

To achieve flexibility and efficiency, we propose a multi-agent system, which includes two major modules:

- A remote system module working as a remote agent server that hosts three kinds of agents: stationary agents, mobile agent controllers and mobile service agents.

- A local system module working as a local agent server hosting two kinds of agents: mobile service agents and client agents.

The following services summarize the major functionality of the proposed system: service registration, service preparation, service consumption and service completion.

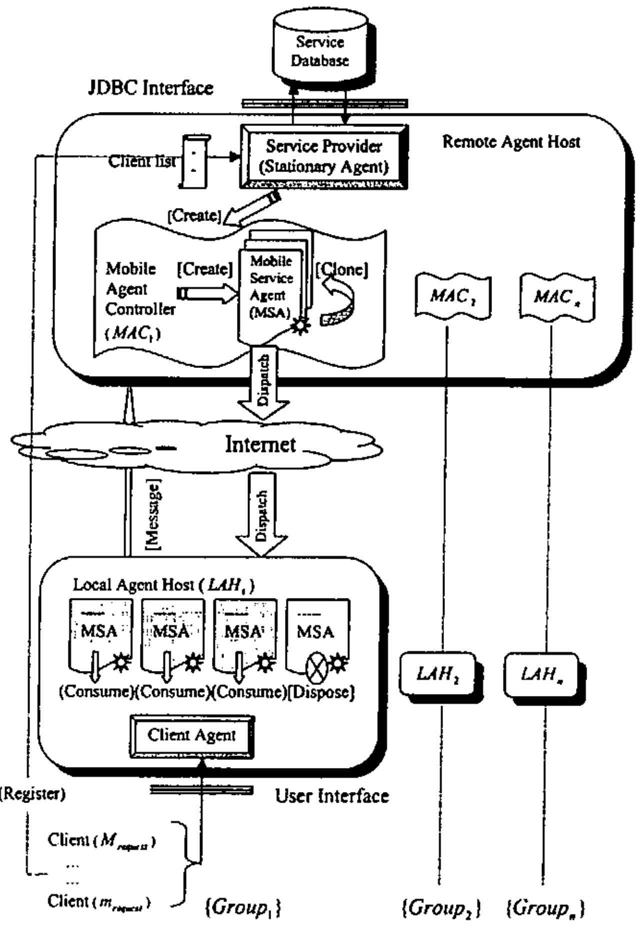

Figure 7.2. Multi-agent system structure enhanced by on-line task-scheduling algorithm

Figure 7.2 shows the general structure of the system. It also includes two interfaces: a JDBC interface resides at the remote server for retrieving service units from the back-end database, a user interface resides at the local server for accepting incoming requests, which have been grouped together according to the number of requests each client makes.

- **Service registration**

Service registration is the first step when a client wants to access a remote service. The client is required to register at the remote service provider and indicate how many "service units" are needed. The remote service provider always maintains a client list. This list changes frequently, due to the increase or decrease in a client's requests. All of the clients are divided into several groups according to the number of requests made by each client. Based on a different range for the number of requests, different groups of clients are guided to access different local agent hosts. In our proposed system, there are a total of *n* groups of clients available. For example, clients in $Group_1$ with request quantities in the range $[m, M]$ are directed to local agent host 1 (LAH_1) to obtain service units.

- **Service preparation**

Improving system response is one of our aims. In the proposed system, our strategy is to create and clone a number of mobile agents at remote agent hosts dynamically, initializing service units (i.e., ready-made units) and dispatching these agents to the local agent hosts beforehand. For serving *n* groups of clients, *n* mobile agent controllers (*MAC*) are needed. In our system, MAC_1 creates and clones a number of worker agents. These ready-made mobile service agents *(MSA)*, each equipped with one service unit, are dispatched to LAH_1 for local access.

- **Service consumption**

Initially, clients access local agent hosts instead of remote agent hosts to obtain quicker service. When a client (agent) with *x* requests *consumes* ready-made *d* service units carried by *d MSA*, three scenarios are most likely to occur:

(1). $x = d$, and the client happens to consume all the ready-made service units.

(2). $x < d$, ready-made service units exceed the client's requests, and unclaimed units are destroyed later.

(3). $x > d$, ready-made service units are insufficient, and the client sends a message to the remote *MAC* to ask for more service units. After receiving the message, the *MAC* repeats the cloning and dispatching processes to serve unsatisfied requests.

- **Service completion**

This is the last step in the whole process, where the *MAC* is informed of service completion (i.e., the client's requests have been satisfied successfully). One major concern at this stage: unclaimed mobile service agents *(MSA)* at the local host have to be disposed off explicitly before the new service process starts. For the service provider, a new process begins at the service preparation step.

The design of the proposed system is straightforward, but a problem arises when the remote agent controller prepares for service units: it needs to decide how many mobile service units (agents) to create, clone and dispatch. If insufficient service units are prepared, the agent controller has to clone more and supply them later, which causes a delay in system response. If a surplus of service units is prepared, unclaimed service agents can be disposed off explicitly, but this increases system overheads. In the proposed system, both lower overheads and faster service performance need to be taken into account. The following section presents an on-line algorithm to address the problems described above, which we call the on-line task-scheduling problem *(OTSP)*.

7.4 Online Task-Scheduling Algorithm

This section presents an on-line competitive algorithm for dynamic task scheduling. On-line problems can be found in many research areas such as data structuring, task scheduling or resource allocation [13,44,45,59]. Several examples are given to illustrate the common feature of these problems. In data structuring problems, the goal is to access elements in a given data structure at a lower cost, without foreknowledge on which elements will be accessed in the future. For the paging problem in a two-level memory system, the objective is to decide which referenced pages should be stored in fast memory. In a multi-processor network, it is a challenging job to provide network access at lower communication cost by dynamically reallocating files. All these problems are characterized by the need to satisfy requests or make decisions without foreknowledge of future requests, which is different from traditional system analysis approaches where algorithms are designed with the assumption that the complete sequence of requests is known.

In this chapter, we develop a competitive task-scheduling algorithm to allocate mobile agents to client service requests. Our aim is to minimize the total cost of service in an on-line environment. We measure each service in terms of its computing cost. For example, we use *cost(creation)* to denote the average cost of the creation service. Five types of costs are involved in our proposed system: cost(creation), cost(cloning), cost(dispatching), cost(disposal) and cost(messaging).

We examine three scenarios when a client with x requests consumes ready-made d service units, we have:

1. $x = d$, the client happens to consume all ready-made service units. In this scenario, each service agent costs c = Cost(creation) + Cost(cloning) + Cost(dispatching), and the total cost is cd or cx.

2. $x < d$, ready-made service units exceed the client's requests, so $(d - x)$ units are eliminated. In this scenario, we have c_1 = Cost(creation) + Cost(cloning) + Cost(dispatching) + Cost(disposal), and the total cost is $cx + c_1(d - x)$.

3. $x > d$, ready-made service units are insufficient to meet the client's requests, so another $(x - d)$ units are cloned and dispatched. In this scenario, we have c_2 = Cost(messaging) + Cost(cloning) + Cost(dispatching), and the total cost is $cd + c_2(x - d)$ or $cx + (c_2 - c)(x - d)$.

If the actual request sequence is denoted by $\sigma = (x_1, x_2, \ldots\ldots, x_n)$, where x_i means the actual number of requests in the *i*th period, we can obtain the optimal off-line cost of the problem (for a single client):

$$C_{OPT}(\sigma) = c \cdot \sum_{i=1}^{n} x_i \qquad (1)$$

For the same request sequence, if d_i denotes the service units that should be prepared by the on-line decision-maker for the *i*th period, we can obtain the on-line cost of competitive algorithm A as follows:

$$C_A(\sigma) = c \cdot \sum_{i=1}^{n} x_i + (c_2 - c) \cdot \sum_{\substack{i=1 \\ x_i > d_i}}^{n} (x_i - d_i) + c_1 \cdot \sum_{\substack{i=1 \\ x_i < d_i}}^{n} (d_i - x_i) \qquad (2)$$

For any on-line algorithm A, the competitive ratio is defined as:

$$\alpha = \inf_{\sigma} \frac{C_A(\sigma)}{C_{OPT}(\sigma)} \qquad (3)$$

A small competitive ratio implies that A can do well in comparison with the optimal (OPT). In designing the competitive algorithm for the *OTSP* problem, the agent controller (i.e., the on-line decision-maker) does not know beforehand the actual number of requests in the *i*th period. Instead, the controller knows the possible range in the number of requests denoted by [m, M]. The on-line competitive algorithm A needs to give the best possible choice for the number of service units *(d)* to prepare for the *i*th period, which would result in the smallest competitive ratio α.

7.4.1 General Harmonic Algorithm (GHA)

Theorem 7.1. For the on-line *OTSP* problem, the best choice that an on-line decision-maker can make is:

$$d = \frac{Mm \cdot (p + q - 1)}{Mp + m \cdot (q - 1)} \quad (4)$$

where $p = c_1/c$, $q = c_2/c$ and $[m, M]$ is the possible range in the number of requests for a client from a given group.

7.4.2 Competitive Ratio

Theorem 7.2. For the competitive *GHA* algorithm given in Theorem 1, the competitive ratio is:

$$\alpha = \left(1 + p(q-1)\frac{M - m}{Mp + m \cdot (q-1)}\right) \quad (5)$$

Proof. To prove theorem 7.2, we need to consider two possible worst-case request sequences, which an off-line *adversary* may choose: $\sigma_M = (M,M,\ldots\ldots,M)$ and $\sigma_m = (m,m,\ldots\ldots,m)$.

Firstly, let $C_A(\sigma_M)$ and $C_{OPT}(\sigma_M)$ denote the on-line cost and optimal off-line cost under the circumstances of σ_M, respectively. According to equations (3) and (5) above, we obtain:

$$\begin{aligned}
C_A(\sigma_M) &= c \cdot \sum_{i=1}^{n} M + (c_2 - c) \cdot \sum_{i=1}^{n} (M - d_i) \\
&= cMn + (c_2 - c) \cdot (M - d) \cdot n \\
&= \left(1 + (q-1)\left(1 - \frac{d}{M}\right)\right) \cdot cMn \\
&= \left(1 + (q-1)\left(1 - \frac{d}{M}\right)\right) \cdot C_{OPT}(\sigma_M) \\
&= \left(1 + p(q-1)\frac{M - m}{Mp + m \cdot (q-1)}\right) \cdot C_{OPT}(\sigma_M)
\end{aligned}$$

Similarly, let $C_A(\sigma_m)$ and $C_{OPT}(\sigma_m)$ denote the on-line cost and optimal off-line cost under the circumstances of σ_m, we obtain:

$$\begin{aligned}
C_A(\sigma_m) &= c\cdot\sum_{i=1}^{n} m + c_1\cdot\sum_{i=1}^{n}(d_i - m)\\
&= (cm + pc\cdot(d-m))\cdot n\\
&= \left(1 + p\left(\frac{d}{m} - 1\right)\right)\cdot cmn\\
&= \left(1 + p\left(\frac{d}{m} - 1\right)\right)\cdot C_{OPT}(\sigma_m)\\
&= \left(1 + p(q-1)\frac{M-m}{Mp + m\cdot(q-1)}\right)\cdot C_{OPT}(\sigma_m)
\end{aligned}$$

where σ_M and σ_m are the two worst possible cases for the given problem. For any σ:

$$\alpha = \frac{C_A(\sigma)}{C_{OPT}(\sigma)} \le 1 + p(q-1)\left(\frac{M-m}{Mp + m\cdot(q-1)}\right)$$

The proof is complete.□

7.4.3 Lower Bound for the Competitive Ratio

Theorem 7.3. For the competitive ratio of on-line *GHA* algorithm $\left(1 + p(q-1)\frac{M-m}{Mp+m\cdot(q-1)}\right)$ is the lower bound.

Proof. We need to prove that if an on-line player chooses to prepare a number of service units where the number is different to $\frac{Mm(p+q-1)}{Mp+m(q-1)}$, the competitive ratio is worse. We assume that $d' < \frac{Mm(p+q-1)}{Mp+m(q-1)}$, then we need to prove that if an off-line adversary chooses σ_M the competitive ratio $\left(1 + p(q-1)\frac{M-m}{Mp+m\cdot(q-1)}\right)$ cannot be achieved.

According to equation (3), we obtain:

$$\begin{aligned} C_{A'}(\sigma_M) &= c\cdot\sum_{i=1}^{n} M + (c_2 - c)\cdot\sum_{i=1}^{n}(M - d_i') \\ &= (cM + (q-1)\cdot c\cdot(M - d'))\cdot n \\ &= \left(1 + (q-1)\left(1 - \frac{d'}{M}\right)\right)\cdot cMn \\ &> \left(1 + p(q-1)\frac{M - m}{Mp + m\cdot(q-1)}\right)\cdot C_{OPT}(\sigma_M) \end{aligned}$$

The last inequality holds if $d' < \frac{Mm(p+q-1)}{Mp + m(q-1)}$. A similar proof can be given for the condition $d' > \frac{Mm(p+q-1)}{Mp + m(q-1)}$. □

Corollary 7.1. If we denote the fluctuation rate by $\phi = \frac{M}{m}$, for the competitive *GHA* given in Theorem 7.1, the best choice for d depends on p/q when the fluctuation rate remains constant.

Proof. According to equation (5), we obtain:

$$\frac{d}{m} = \frac{\phi\cdot p + \phi\cdot(q-1)}{\phi\cdot p + (q-1)}$$

Which means that $d \to m$ when *p>>q*, and $d \to M$ when *q>>p*.□

In the proposed system, if the cost of disposal is much higher than cloning, the number of service units we prepare approaches the lower boundary (*m*). Similarly, our choice approaches the upper boundary (*M*) if the cost of cloning is much higher than disposal. This condition is illustrated in Figure 7.3.

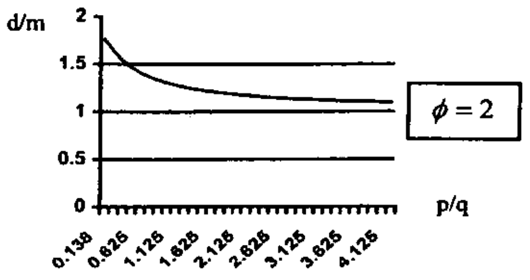

Figure 7.3. Relation between *d/m* and *p/q*

Corollary 7.2. For the on-line *OTSP* problem, if we have $c_1 = c \cdot c_2$, that is, $p = q - 1$, the best decision that an on-line player can make is:

$$\bar{d} = \frac{2Mm}{M + m}$$

where [*m*, *M*] is the possible range in the number of requests for a client in a given group.

The proof is omitted in this thesis.

Corollary 7.3. For the special case of a competitive *GHA* that supports corollary 7.2, the competitive ratio is:

$$\alpha = \left(1 + p\frac{M - m}{M + m}\right)$$

The proof is omitted in this thesis.

7.5 Two More Corollaries

Corollary 7.4. For the on-line *OTSP* problem, where $p = c_1 / c$, $q = c_2 / c$ and $\phi = M / m$, the competitive ratio α depends on ϕ if p and q are constants. When ϕ increases, α approaches q.

According to equation (5), we obtain:

$$\alpha = 1 + (q - 1)\frac{\phi \cdot p - p}{\phi \cdot p + (q - 1)}$$

This means that $\alpha \to 1$ when $\phi \to 1$, and $\alpha \to q$ when $\phi \to \infty$.

The value of ϕ is the fluctuation rate of the requests within any registered group. If we divide our clients into more groups, we obviously narrow the fluctuation rate of each new group, and we can always obtain a better competitive ratio. This is illustrated in Figure 7.4.

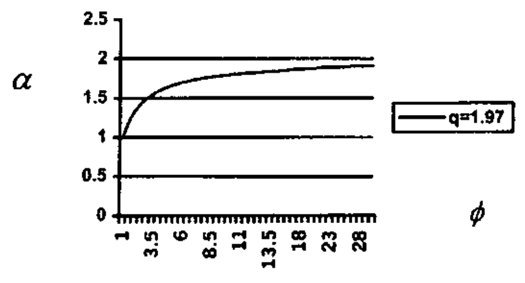

Figure 7.4. Relation between α and ϕ

Corollary 7.5. For the on-line *OTSP* problem, where $p = c_1 / c$, $q = c_2 / c$ and $\phi = M / m$, if the fluctuation rate changes due to increases or decreases in the clients' request within a given group, the on-line solution (d) should be changed accordingly using a dynamic allocation:

$$d = \frac{Mp + M \cdot (q - 1)}{\phi p + (q - 1)}$$

The proof is omitted in this thesis.

The relationship between d/m and ϕ is illustrated in Figure 7.5.

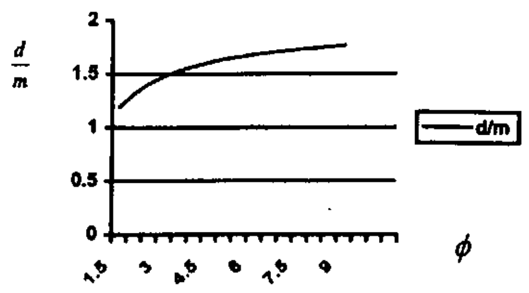

Figure 7.5. Relation between *d*/*m* and ϕ

7.6 Conclusions

Task scheduling schemes play an important role in mobile agent system design. The majority of current research has been focused on applying pre-defined schemes for allocating computing resources to fulfill a fixed off-line task. To overcome the limitations of current approaches, which lack the flexibility to dynamically allocate mobile agents in an on-line environment, we introduce a new system structure. This structure integrates push-based technology with an innovative on-line task-scheduling algorithm to improve system response time and minimize system overheads. The proposed competitive algorithm *GHA* (General Harmonic Algorithm) has the optimal competitive ratio and it has been tested through experiments to demonstrate its feasibility and effectiveness. The system has potential for a wide range of Internet-based applications, and for integration with other intelligent decision-making algorithms for high performance, flexibility and reliability.

Chapter 8

FUTURE STUDY

In this chapter, a schedule of future work is presented. In future, we will focus our attention on the application of the proposed algorithm to task scheduling for mobile agent based systems.

8.1 For the *k*-Server Conjecture

One of the main reasons why the research concerning on-line problems and competitive algorithms is so interesting is the existence of two famous conjectures (see Chapter 1). In my future study, I will do further research on the relevant problems concerning the *k-server* conjecture.

8.2 Other On-line Problems

In my future study, I will allocate time to research other on-line problems, such as on-line scheduling problems, on-line network routing and on-line financial problems. I think that I might be more interested in the on-line financial problems.

8.3 Application to Management and Economic Problems

Research concerning how to apply the theory of on-line problems and competitive algorithms to the domains of management and economics has not received enough attention. However, in real life, there are many management and economic problems that must be dealt with in an on-line manner. Therefore, how to represent and design competitive algorithms for these problems will be a major task in my future research.

8.4 Continue Research into the *k*-Truck Problem

The *k*-truck problemstill requires a great deal of work that will need some thorough research. For example, the current results for the *k*-truck problem are based on the *k*-server problem, and subsequent use of the Position Maintaining Strategy. We may be able to find some better algorithms for the *k*-truck problem that do not depend on the results from the *k*-server problem. It may be possible to obtain better results if we model the *k*-truck problem as a truck weighted problem. I will continue my research for the *k*-truck problem to obtain better results.

8.5 Application to Agent-based E-commerce

Agent-based techniques and e-commerce are two new and very active areas of academic research. I will attempt to apply the theory of the on-line problems and competitive algorithms to the field of agent-based e-commerce.

BIBLIOGRAPHY

[1] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563-1581,1966.

[2] D. E. Knuth. The Art of Computer Programming; Volume 2: Seminumerical Algorithms. Addison-Wesley,1st edition, 1968.

[3] D. S. Johnson. Near-optimal bin packing algorithm. *Ph.D. thesis*, MIT, Cambridge, MA, 1973.

[4] D. S. Johnson. Fast algorithms for bin-packing. *J. Comput. System Sci.*, 8:272-314, 1974.

[5] D. S. Johnson. Private Communication, 1998.

[6] D. R. Woodall. The bay restaurant- a linear storage problem. *American Mathematical Monthly*, 81: 240-246, 1974.

[7] H. A. Kierstead and W. T. Trotter. An external problem in recursive combinatorics. *Congressus Numerantium*, 33:143-153,1981.

[8] C. A. Yao. New algorithms for bin packing. *Journal of ACM*, 27:207-227,1980.

[9] D. D. Sleator, R.E.Tarjan, Amortized efficiency of list update and paging rules, *Comm.ACM* 28 (1985) 202-208.

[10] D. D. Sleator and Robert Endre Tarjian. Self-adjusting binary search trees. *Journal of ACM*, 32:652-686, 1985.

[11] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3 (1):79-119, 1988.

[12] A. Borodin, N.Linial, M.Sake, An optimal on-line algorithm for metrical task systems. *Proc. 19th ACM STOC* (1987) 373-382.

[13] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, Competitive algorithms for server problems, *Journal of Algorithms*, (11),208-230,1990.

[14] T. M. Cover. Universal portfolios. *Mathematical Finance*, 1 (1):1-29, January 1991.

[15] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127-150, 1991.

[16] S. Ben-david, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson. On the power if randomization in on-line algorithms. *In Proc.of the 22nd Ann. ACM* Symp.on theory of computing, pages 379-386, may, 1990.

[17] M. Chrobak, H. Karloff, T. Payne and S. Vishwanathan, new results on server problems, *Proceedings of 1st annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 1990.

[18] M. Chrobak and L.L. Larmore. The server problem and on-line games. In *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, volume 7, pages 11-64, 1992.

[19] A. Fiat, Y. Rabani, Y. Ravid, competitive k-server algorithms. *31st FOCS*, 1990.

[20] A. Fiat, M. Richlin, competitive algorithm for the weighted server problem. theory and computing system 1993, *Proceedings of the 2nd Israel symposium* page(s).294-303.

[21] E. Koutsoupias and C. Papadimitriou. on the k-server conjecture. *In proc.25th symposium on thety of computing*, pages 507-511,1994.

[22] Y. F. Xu, K. L. Wang, and B. Zhu, On the *k*-taxi problem, *Information*, Vol.2, No.4, 1999.

[23] Y. F. Xu, K. L. Wang, On-line *k*-taxi problem and competitive algorithm, *Journal of Xi'an Jiaotong University*, (1):56-61,1997.

[24] S. Ben David and A. Borodin, A new measure for the study of the on-line algorithm, *Algorithmica*, (11), 73-91,1994.

[25] N. Alon, R. M. Karp, D. Peleg, et al, A graph-theoretic game and its application to the *k-server* problem, *SIAM J. Comput.* , 24(1):78-100, 1995.

[26] D. Z. Du, *k-server* problem and competitive algorithm, *Practice and Acquaintanceship of Mathematics*, (4): 36-40, 1991.

[27] W. M. Ma, Y. F.Xu, K. L. Wang, On-line k-truck scheduling problem ad its competitive strategies, *Journal of Northwest University* (Natural Science, P. R. China), Vol.29, No.4:254-258, 1999.

[28] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. *Proceedings 20th Annual ACM Symposium on Theory of Computing*, pages 322-33, 1988.

[29] E. Grove. The Harmonic online *k*-server algorithm is competitive. *Proceedings 23rd Annual ACM Sympposium on Theory of Computing*, Pages 260-66, 1991.

[30] S. Irani and R. Rubinfeld. A competitive 2-server algorithm. *Information Processing Letters*, 39(2):85-91, July 1991.

[31] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs and applications to on-line algorithms. *Journal of the Association for Computing Machinery*, 40(3):421-53, July 1993.

[32] M. Chrobak and L. L. Larmore. On fast algorithms for 2 servers. *Journal of Algorithms*, 12(4):607-14, December 1991.

[33] M. Chrobak and L. L. Larmore. Harmonic is 3-competitive for two servers. *Theoretical Computer Science*, 98(2):339-46, May 1992.

[34] A. Blum, H. Karloff, Y. Rabani, and M. Saks. A decomposition theorem and bounds for randomized server problems. *Proceedings 33rd Annual Symposium on Foundations of Computer Science*, pages 197-207, 1992.

[35] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542-71, April 1994.

[36] H. Karloff, Y. Rabani, and Y. Ravid. Lower bounds for randomized *k*-server and motion-planning algorithms. *SIAM Journal on Computing*, 23(2):293-312, April 1994.

[37] M. Chrobak and L. L. Larmore. The server problem and on-line games. On-line algorithms: proceedings of a DIMACS workingshop. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:11-64, 1992.

[38] W. R. Burley. Traversing layered Graphs using the work function algorithm. *Technical report CS93-319*, Dept. of Computer Science and Engineering University of California, San Diego, 1993.

[39] M. Chrobak and L. L. Larmore, N. Reingold, and J. Wesbrook. Page migration algorithms using work functions. Algorithms and Computation. *4th International Symposium, IASSC'93 Proceedings*, pages 406-15, 1993.

[40] E. Koutsoupias. Weak adversaries for the *k*-server problem. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, New York City, NY, pages 444--449, 17--19 October 1999.

[41] E. Koutsoupias and D. S. Taylor. The CNN problem and other *k*-server variants. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, Lille, France, pages 581--592, 17--19 February 2000.

[42] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300--317, 2000.

[43] Y. Bartal and E. Koutsoupias. On the competitive ratio of the work function algorithm for the *k*-server problem. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, Lille, France, pages 605--613, 17--19 February 2000.

[44] S. Albers and S. Leonardi. Online algorithms. To appear in *ACM Computing Surveys* , 1999.

[45] R. El-Yaniv, A. Fiat, R. Karp,; G. Turpin, Competitive analysis of financial games. *Foundations of Computer Science, 1992. Proceedings., 33rd Annual Symposium on , 1992* , Page(s): 327 –333.

[46] H. Stormer, *Task Scheduling in Agent-Based Workflow.* International ICSC Congress INTELLIGENT SYSTEMS & APPLICATIONS ISA'2000.

[47] J. Bredin, David Kotz and Daniela Rus, *Utility Driven Mobile-Agent Scheduling.* Dartmouth Technical Report PCS-TR98-331.

[48] G. Cybenko, *The Foundations of Information Push and Pull.* A chapter in the book "Mathematics of Information", edited by D.O'Leary.

[49] J. Kiniry, D. Zimmerman, *Special feature: A Hands-on Look at Java Mobile Agents.* IEEE Internet Computing, Volume 1, No.4 July/August 1997.

[50] http://www.objectspace.com/Voyager

[51] http://www.genmagic.com/agents/odyssey

[52] http://www.trl.ibm.co.jp/aglets

[53] *ObjectSpace Voyger, General Magic Odyssey, IBM Aglets: A comparison.* ObjextSpace Inc. Technical White Paper.

[54] Z. Maamar, "An approach to Optimizing a Mobile Agent Itinerary," *International ICSC Congress Intelligent Systems & Applications ISA'2000, Networked Business II (MAMA'2000)*, 1574-350, Australia, 2000.

[55] N. M. Karnik and A. R. Tripathi, "Design Issues in Mobile Agent Programming Systems," *IEEE Concurrency, Vol. 6, No.3,* pp. 52-61, July/September 1998.

[56] D. B. Lange, and M. Oshima, , Programming and Deploying Java Mobile Agents with Aglets, *Addison-Wesley*, 1998.

[57] M. Straper, J. Baumann, and M. Schwehm, "An Agent-Based Framework for the Transparent Distribution of Computations," in *Proc. of Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Vol. I, CSREA*. pp.376-382, 1999.

[58] S. Fishchmeister and W. Lugmayr, "The Supervisor-Worker Pattern," *Programming Language of Programs (PLoP'99) Conference, USA*, 1999.

[59] A. Fiat and G. J. Woeginger. Online algorithms—the state of the art. *Lecture Note in Computer Science*. Springer, 1998.

[60] R. Tarjan, Data Structures and Network Algorithms, *SIAM*, Philadelphia, 1983, 109-111.

[61] Y. F. Xu, and K. L. Wang, On-line *k*-elevator problem and competitive algorithm. (In preparation).