ON-LINE ALGORITHMS FOR THE K-SERVER PROBLEM AND ITS VARIANTS

By

Chi-ming Wat

A Dissertation

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Division of Computer Science

The Chinese University of Hong Hong

June 1995

Acknowledgement

I would first like to thank my supervisor, Kin-hong Lee, who has help me throughout my research, from his posing the first research problem I ever solved to his careful reading of my thesis. His clarity, impeccable teaching style, and scholarly attitude toward research will always be something to aspire to.

I am grateful to have had the privilege of commenting my research work by Gilbert Young during 1993-95. It is hard to exaggerate the extent to which he has influenced my view of theoretical computer science.

While working at the Department of Computer Science, I had the pleasure of working with John Lui. I greatly appreciate his friendship and all that I have learned from him.

I have learned much from the many discussions I have had with Kam-wing Ng, John Lui and Gilbert Young and from classes they have taught here.

One of the best things about studying computer science at the Chinese University of Hong Kong is the great group of students and visitors I have had the pleasure to meet and work with. they have been more than friends. Many, many research discussions with Chi-lok Chan have strongly influenced my thinking. I have also enjoyed the opportunity of working with Hon-man Law, Wing-hee Lam, Sau-ming Lau, Yiu-leung Yeng, Wai-ting Wong, Yiu-chung Wong, and many others. Kei-Fu Mak deserves special thanks for his constant assistance. I am grateful to all of them as well as to the rest of the staff and students, for making the department what it is.

Finally, I am in debt of CUHK Graduate School for supporting my paper publications in the IASTED '95 (Austria) and ICIAM '95 (Germany).

Abstract

It is only recently that people have looked at problems from the on-line perspective. The concept itself is attractive since a large number of problems in real life have to dealt with in an on-line fashion.

In this thesis we considered the k-server problem and its variants. It is the single problem that has spurred the most interest in this field. Previous attacks on this and other on-line problems, say the metrical task systems, involved a potential function, a numerical invariant that enables the inductive proof. Our technique is based on more complex invariants, which provide valuable information about the structure of the reachable work functions. As a result, we prove an upper bound of $2k-1$ which is the best we can prove at this point, although we conjecture that the true worst case behavior of the work function algorithm is k.

In the Chapter 3 we deal with a generalization of the k-server problem, in which the servers are unequal. Each of the servers is assigned a positive weight. For any metric space, we extend our Work Function Algorithm to give a nearly optimal upper bound.

In order to study the amount of improvement that can be achieved if the future is partially known, we introduce a new model of lookahead for our k-server algorithm. We show that strong lookahead has practical as well as theoretical importance and significantly improves the competitiveness of our k-server algorithm. This is the first model of lookahead having such properties.

Finally, we study a sub-class of memoryless server algorithms with 2 servers on the real line and show that there is a unique algorithm in this sub-class whose competitive factor is best possible on the real line.

Contents

Chapter 1

Introduction

Suppose that you occasionally have to travel from your place of work to meet your business associates in a nearby town. The number of times you have to make this trip (by bus, say) is not known to you in advance. The decision you have to make is whether you should buy tickets each time you travel or buy a monthly pass. The monthly pass can be bought anytime during the month, but is valid only till the end of the month in which it was bought. The cost of the monthly pass is equal to the cost of 5 trips, say.

The problem here is that supposing you buy a monthly pass, and then it so happens that circumstances warrant your travel only twice that month, or that you decide against a monthly pass for the month and end up traveling up to 20 days that month! Clearly if you do know the number of times you are going to travel then you are in a position to spend an optimal amount of money on travel, but what do you do otherwise?

One solution here is to buy a monthly pass as soon as you know that you are going to make your $5th$ trip. It is easy to see that using this strategy, we never spend more than twice (in fact, no more than nine-fifths) the optimal amount of money. Also, it is not too difficult to show that there are situations which can force any decision maker to spend at least nine-fifths the optimal amount of

money.

The situation we discussed above is typical of the kinds of questions one studies in on-line algorithms, where one has to make decisions without knowing the full input.

In general, an on-line algorithm receives a sequence of requests and has to respond to each request as soon as it is received. In serving each request, the algorithm incurs a *cost*. An off-line algorithm, on the other hand, may wait until all requests have been received before determining its responses. The on-line model is in fact more practical in many situations: scheduling, where one must schedule jobs on processors as they arrive, or bin packing, where one must pack items into bins as soon as one receives them. Applications are numerous: analyzing the use of data structures, resource allocation in operating systems, etc.

1.1 Performance analysis of on-line algorithms

How does one compare or rank algorithms that perform the same task? This is a crucial question in the design and analysis of algorithms. We discuss the current trends, especially with respect to on-line algorithms.

The *empirical* approach to this problem consists of coding the algorithm in a programming language and running it on different instances on a computer. One can then "judiciously" choose test instances and compare algorithms by running them on these test instances.

The so called *theoretical* approach, which will be the focus in this thesis, consists of finding bounds on the amount of resources used (e.g. time, memory, space, etc.) by the algorithm as a function of the size of the input. Another parameter of interest is the equality of output produced. Typically, in this case, one restricts the class of algorithms; for instance we can restrict the time taken

by the algorithm, and then ask the question "how good a solution can one produce?".

In the classical off-line model, one usually considers only algorithms that find "optimal" solutions, and here an important criterion is time. A class of problems (the class P) that are considered "tractable" are problems for which one can produce an optimal solution in time bounded by a polynomial in the input size.

Suppose now that we are interested in algorithms that run in polynomial time but which give us a good approximate solution. We do not care about the time taken by the algorithm as long as it can be bounded by a polynomial in the input size. The parameter of importance is the quality of the approximation produced. How do we compare such algorithms? The classical solution to this problem is to compare the solution produced by the algorithm to the optimal solution produced by an algorithm that could take as much time as it likes. In fact the quantity of importance is the worst case ratio of the solution produced by the algorithm to the optimal solution. This is a quantity which can be used to compare various algorithms.

The situation is very similar when one considers on-line algorithms. The time an algorithm takes to make its decision is not of prime importance. Belady [1], Graham [29] and Sleator and Tarjan [49] suggested comparing the performance of on-line algorithms to an optimal off-line algorithm for the same problem. Such an analysis is term *competitive analysis*. Informally, in competitive analysis, we say that the on-line algorithm is good if its performance on any sequence of requests is to within some small factor of the performance of the optimal off-line algorithm.

More formally, let $\mathcal{A}$ be an on-line algorithm for a problem $\prod$. For an input instance (or a sequence of requests) σ, let $\mathcal{A}(\sigma)$ denote the value of the output produced by $\mathcal{A}$, and let $\mathcal{OPT}(\sigma)$ denote the value of the output produced by the

optimal off-line algorithm. The performance ratio denoted by $\rho_{\mathcal{A}}(\sigma)$ is defined to be $\frac{\mathcal{A}(\sigma)}{\mathcal{OPT}(\sigma)}$. The performance function or the *competitive ratio*, denoted $\rho_{\mathcal{A}}(n)$, is defined for each integer n to be the maximum of $\rho_{\mathcal{A}}(\sigma)$ over all inputs σ of size n. The term competitive ratio is more widely used, especially when $\rho_{\mathcal{A}}(n)$ is independent of n. When the competitive ratio of an algorithm is α, we will frequently refer to the algorithm as being *α-competitive.*

A more general framework for considering on-line algorithms is the *request-answer games* [2], [3], [46], [47]. In such a game, the algorithm competes against an "adversary." The adversary supplies the next chunk of input to the algorithm, who now has to process it.

1.2 Randomized algorithms

We will, in many cases, consider the power that randomization provides in on-line algorithm design. By randomized algorithms we mean algorithms which make random choices in the course of their execution. As a result, the behavior of the algorithm can be random even on a single fixed input. The idea is to prove that such an algorithm works well on *every* input with high probability. It is important to distinguish this from the probabilistic analysis of algorithms, where we study the behavior of a (possibly deterministic) algorithm when the input to the algorithm is chosen from a probability distribution. In this later style of analysis, we might be able to prove that an algorithm works well with high probability on a randomly chosen input or, in a sense, that it works well on *almost all* inputs. Such results are thus somewhat weaker than the results we wish to prove for randomized algorithms (although they are not necessarily easier to prove.)

In this case, the costs are the expected values of the associated random variables and the definition of competitive ratio now also depends on the power

given to the adversary.

1.3 Types of adversaries

We distinguish between three types of adversaries.

Oblivious adversary This adversary must determine the request sequence in advance, i.e. he could base his example on the algorithm, but must fix the entire example before the start of the game.

Adaptive on-line adversary An adaptive on-line adversary can base the next request on the algorithm's answer to the previous requests, but then serves it himself immediately.

Off-line adversary An off-line adversary makes the next request based on the algorithm's answers to previous ones and serves them optimally at the end.

First, it can be easily argued that for deterministic algorithms these adversaries are equally powerful. Also it is easily seen that we have listed the adversaries in order of their increasing power. Note that though the off-line cost for the oblivious adversary does not depend on the coin tosses of the algorithm, the off-line cost in the case of both the adaptive adversaries is a random variable, since the requests presented will depend on the previous coin-tosses of the algorithm.

In many cases, as will see, randomization seems to provide a remarkable amount of power, especially against the oblivious adversary. Throughout this thesis, whenever we deal with randomized on-line algorithms, we will implicitly be dealing with oblivious adversaries.

These various types of adversaries were defined by Raghavan and Snir [46], [47] and studied in more detail by Ben-David *et al.* [2], [3]. The following theorems are due to them. The first essentially states that randomization does not help against the off-line adversary.

Theorem 1 ([2], [3]) *If there is a randomized strategy that is α-competitive against any off-line adversary, then there also exists an α-competitive deterministic algorithm.*

The next theorem is quite surprising. It relates the power of the adversaries.

Theorem 2 ([2], [3]) *If there exists an α-competitive randomized on-line strategy against any adaptive on-line adversary and a β-competitive randomized on-line strategy against any oblivious adversary, then there is an $\alpha \cdot \beta$-competitive deterministic strategy.*

1.4 Overview of the results

In Chapter 2 we give a deterministic competitive *k-server* algorithm for all k and all metric spaces. This settles the k-server conjecture [42], [43] up to the competitive ratio at most $2k - 1$.

A generalization of the k-server problem [42], [43] in which the servers are unequal, is dealt with in Chapter 3. In this *weighted server* model each of the servers is assigned a positive weight. The cost associated with moving a server equals the product of the distance traversed and the server weight. We give an exponential $(2k^k - 1)$-competitive algorithm for any set of weights and any metric spaces.

In Chapter 4, we introduce a new model of lookahead for on-line k-server problem. We show that *strong lookahead* has practical as well as theoretical

importance and significantly improves the competitive ratios of the k-server problem.

Chapter 5 is motivated by the need for fast algorithms for on-line problems, many of which require algorithms to provide solutions in real-time. We give a sub-class of memoryless server algorithms with 2 servers on the real line and show that there is a unique algorithm in this sub-class whose competitive factor is best possible on the real line.

Finally, we conclude the thesis with some consequences of our work and some future directions for the k-server problem in Chapter 6.

Chapter 2

The k-server problem

2.1 Introduction

Competitive algorithms were introduced by Sleator and Tarjan [49] in the context of searching a linked list of elements and paging problems. They sought a worst case complexity measure for on-line algorithms that have to make decisions based upon current events without knowing what the future holds. The immediate problem is that on-line algorithms are incomparable; on-line algorithm $\mathcal{A}$ may be better than another on-line algorithm $\mathcal{B}$ for one sequence of events but $\mathcal{B}$ may be better than $\mathcal{A}$ for another sequence of events. The conceptual breakthrough in [49] was to compare the algorithms, not to each other, but to a globally optimal algorithm that knows the request sequence in advance. Formally speaking, let $\mathcal{A}$ be an on-line algorithm for a problem $\prod$. For an input instance (or a sequence of requests) σ, let $\mathcal{A}(\sigma)$ denote the value of the output produced by $\mathcal{A}$, and let $\mathcal{OPT}(\sigma)$ denote the value of the output produced by the optimal off-line algorithm. The performance ratio denoted by $\rho_{\mathcal{A}}(\sigma)$ is defined to be $\frac{\mathcal{A}(\sigma)}{\mathcal{OPT}(\sigma)}$. This performance function or *competitive ratio*, denoted by $\rho_{\mathcal{A}}(n)$, is defined for each integer n to be the maximum of $\rho_{\mathcal{A}}(\sigma)$ over all inputs σ of size n. The term competitive ratio is more widely used, especially when $\rho_{\mathcal{A}}(n)$ is independent of

n. When the competitive ratio of an algorithm is α, we will frequently refer to the algorithm as being *α-competitive.* The competitive ratio may depend on the size and parameters of the problem. Algorithms are called competitive if the competitive ratio is independent of the problem or if the dependency is probably unavoidable.

This chapter is organized as follows. We surveyed related work in the next two sections. In Section 4, we introduced some definitions needed in the rest of this chapter. We discussed the algorithm in Section 5 and investigated its competitiveness in the final section.

2.2 Related Work

Sleator and Tarjan gave competitive algorithms for managing a linked list of elements and for paging. Karlin *et al.* [38] later gave competitive algorithms for snoopy caching.

Borodin *et al.* [9], [10] generalized the concept to arbitrary *task systems.* Task systems capture a very large set of on-line problems but the generality of task systems implies that task systems cannot perform very well relative to an optimal off-line (prescient) algorithm. Borodin *et al.* [9], [10] gave an upper bound on the competitive ratio of any task system and showed that some task systems have a matching lower bound. The competitive ration upper bound for task systems depends on the number of states of the system. For a limited set of task systems, Manasse *et al.* [42], [43] later gave a decision procedure to determine if a given on-line algorithm is c-competitive.

Manasse *et al.* [42], [43] generalized the paging problem to the *k-server problem.* The on-line k-server problem may be defined as follows: We are given a metric space $\mathcal{M}$ and k servers which move among the points of $\mathcal{M}$. Repeatedly, a request, a point x in the metric space, is given. In response to this request,

we must choose one of the k servers and move it from its current location to x, incurring a cost equal to the distance from its current location to x. Note that the paging problem with k page slots in memory and n pages overall is isomorphic to the k-server problem on a metric space with n points and a uniform distance matrix (except with zeros on the diagonals). Let $\Phi = \{\mathcal{A}(k, \mathcal{M})\}$ be a family of on-line k-server algorithms for the metric space $\mathcal{M}$, where k ranges over all positive integers, and $\mathcal{M}$ ranges over all possible metric spaces. $\mathcal{A}$ is called *competitive* if there exists a sequence $\sigma_1, \sigma_2, \cdots$ such that for each metric space $\mathcal{M}$, and for each k, $\mathcal{A}(k, \mathcal{M})$ is $\alpha(\sigma_k)$-competitive. The competitive ratio for the algorithm of [9], [10] depends on the number of points in the metric space.

Another version of the k-server problem is to charge for "time" rather than "transport." If we assume that all servers move at some common speed and allow all servers to move simultaneously then the off-line problem becomes one of minimizing the total time spent to serve the requests, subject to the limitation that requests are served in order of arrival. The on-line algorithm may position its servers to deal with future events but obtain the next request only when the current event is dealt with. We call this version of the problem the *min-time server problem.*

Manasse *et al.* [42], [43] gave a lower bound for the competitive ratio of any on-line k-server algorithm: for any deterministic k-server algorithm and any metric space with more than k points there exits a sequence of requests such that the cost of the on-line algorithm is no less than k times the cost of an optimal off-line algorithm, minus an additive term.

Manasse *et al.* [42], [43] also conjectured that this lower bound is tight, up to an additive item. This conjecture is known as the *k-server conjecture.* They constructed k-competitive algorithms for all metric spaces if $k = 2$ and for all $(k + 1)$-point metric spaces. (Other competitive 2-server algorithms were later given by Chrobak and Larmore [13], [16], by Fiat *et al.* [24], [25], Irani and

Rubinfeld [34] and by Turpin [52].)

Prior to our result, only the additional case $k = 3$ was solved for general metric spaces using the randomized $\mathcal{HARMONIC}$ algorithm suggested by Raghavan and Snir [46], [47]. This is due to Berman *et al.* [4]. The competitive ratio is bounded by 3^{17000} [45]. Recently, Grove [30] showed that $\mathcal{HARMONIC}$ is $O(k2^k)$-competitive. The competitive ratio for randomized on-line algorithms is described as an expectation. It is important to make precise the definition of the worst case competitive ratio for randomized algorithms. This can be done in terms of an adversarial game with various assumptions on the strength of the adversary. $\mathcal{HARMONIC}$ uses randomization rather weakly; the randomization is used to select the next move but is not used to "hide" the on-line configuration from the adversary designing the sequence. The lower bound of k from [42] [43] holds for such randomized algorithms.

A general result of Ben-David *et al.* [2] [3] gave a non-constructive proof that the existence of any randomized on-line algorithm, which uses randomization of the form used by $\mathcal{HARMONIC}$, implies the existence of a deterministic on-line algorithm, at the cost of squaring the competitive ratio. Randomized algorithms that uses randomization to hide the on-line configuration from the adversary are dealt with by Fiat *et al.* [23], McGeocah and Sleator [44], and Karlin *et al.* [36], [37].

Competitive k-server algorithms were discovered for specific metric spaces. Specifically, Chrobak *et al.* gave k-competitive deterministic on-line algorithms for points on a line [12] and for points on a tree [15]. Randomized on-line algorithms were discovered for resistive graphs by Coppersmith *et al.* [19], [20] and points on a circle by Coppersmith *et al.* [19], [20] and Karp [40]. A deterministic competitive k-server algorithm for the circle was recently discovered by Fiat *et al.* [26]. Chrobak *et al.* [12] also proved that the optimal off-line k-server problem is equivalent to network flow problems and thus has a polynomial-time

solution.

The definition of the competitive ratio in [42], [43] allows an additive term in addition to the ratio; i.e. the on-line algorithm is allowed to perform some (constant) amount of work for free. The analysis of the line and tree algorithms above [12], [15] requires this additive term. The analysis gives an additive term if the initial configuration does not have all servers starting at one common point. This term depends on the initial distances between the servers. While the analysis is clearly overly pessimistic, neither of these algorithms is k-competitive if one discards the additive term.

Fiat *et al.* [23] introduced the concept of an on-line algorithm competitive against a set of on-line algorithms. The idea is to combine two on-line algorithms to obtain a third algorithm which has the advantage of both, at least to within some ratios. The new algorithm can be viewed as some kind of $\mathcal{MIN}$ operator on the two input algorithms. For the paging problem, Fiat *et al.* [23] showed that the $\mathcal{MIN}$ operation is possible and gave tight bounds on what is not. Performing a $\mathcal{MIN}$ operation for other metric spaces is left as an open problem.

2.3 The Evolution of Work Function Algorithm

In this section we introduce what we call the *work function* strategy for the k-server problem that is the building block of our algorithm mentioned later.

This strategy has been independently proposed by Karloff *et al.* [4], [11], [12] who called it the *Opt-based Algorithm*, and by McGeoch and Sleator [42], [43] who called it the *Better-late-than-never Algorithm.*

Chrobak and Larmore [17] extended their work and gave an algorithm for k servers that they called the *Work Function Algorithm.* Based on their overhelming empirical evidence, they conjectured that their algorithm is optimal; i.e. k competitive. However, they could only give a complete proof of the case

for $k = 2$.

They defined a *work function* to be a non-negative real-valued function $\omega : \Lambda^k \mathcal{M} \to R^+$ (where $\Lambda^k X$ is the set of all multisets of order k within a set X, $\mathcal{M}$ is a given metric space and R^+ is the set of non-negative reals) such that, for any two server configurations X and Y,

$$\omega(X) - \omega(Y) \leq XY.$$

They called this the *slope condition.* Intuitively, $\omega(X)$ can be thought as a conditional obligation of the strategy - the amount of cost the adversary must have incurred if he is at configuration X. Let $\mathbf{W}_{\mathcal{M}}$ be the set of all work function (for simplicity we write $\mathbf{W}$ when $\mathcal{M}$ is understood.)

If ω is work function and X, Y are configurations, we say that X dominates Y if $\omega(Y) = \omega(X) + XY$; i.e. the conditional obligation of the adversary if he is at Y is the maximum possible consistent with his obligation if he is at X. When that holds, the adversary is always at least as well off to be at X as to be at Y. We say that X is a *support configuration* of ω if it is not dominated by any other configuration, and we say that ω is *finitary* if $\kappa(\omega)$, the set of all support configurations, is finite, and if every configuration is dominated by some support configuration. If ω is finitary, it is characteristized by its values on its support, and an adversary can do no better than to always have its servers at a support configuration.

We say that ω is a *cone* if $\kappa(\omega)$ has just one element. If a work function is a cone, an algorithm "knows" the location of the adversary servers. A cone with support X will be denoted by χX or often simply by X it does not lead to confusion. Clearly, $\chi X(Y) = XY$ or, using the simplified notation, $X(Y) = XY$. Sometimes we also refer to χX as the *characteristic function* of X.

If ω is a work function and $r \in \mathcal{M}$, we define a work function $\omega \wedge r$, the

update of ω by r, by:

$$(\omega \wedge r)(X) = \min_{r \in Y}\{\omega(Y) + XY\}.$$

An equivalent formulation of the update operator, which seems more complex, but is actually easier to use, is:

$$(\omega \wedge r)(X) = \begin{cases} \omega(X) & if x \in X; \\ \min_{r \in Y}\{(\omega \wedge r)(Y) + XY\} & otherwise. \end{cases}$$

Property 1 *If either formulation of $\omega \wedge r$, the minimum can be taken over only the k choices of Y which lie in $Y \cup \{r\}$.*

Property 2 *IF $X \in \kappa(\omega \wedge r)$, then $r \in X$.*

Property 3 *If ω is finitary, then $\omega \wedge r$ is finitary.*

Informally, if ω is the system of conditional obligations of the adversary, then $\omega \wedge r$ is the new system of conditional obligations after one more request, r. The update operator can be extended to arbitrary sequences by setting $\omega \wedge \epsilon = \omega$ (where ϵ is the empty sequence) and

$$\omega \wedge (\varrho \cdot r) = (\omega \wedge \varrho) \wedge r.$$

Most often it is convenient to deal with functions whose infimum is zero, and thus they also define another operator,

$$(\omega \triangle r)(X) = (\omega \wedge r)(X) - \inf(\omega \wedge r).$$

As with $\wedge$, the operator $\triangle$ can be extended to sequences of requests in an obvious way. In order to simplify the notation we will often omit the parenthesis and write $\omega \wedge \varrho(X)$ and $\omega \triangle \varrho(X)$.

By an easy argument on m, we have:

Let S be the current server configuration, and r the new request point. Then update the current work function by $\omega \wedge r \to \omega$ and move one server to r so that the quantity $SS' + \omega(S')$ is minimized, where S' is the configuration after the move. Alternatively, ω can be the current offset function, since the difference is a constant. In this case, the update step is $\omega \triangle r \to \omega$.

Figure 2.1: Chrobak and Larmore's Work Function Algorithm

Property 4 *If $X \in \Delta^k \mathcal{M}$, $\varrho = r^1, r^2, \cdots, r^m \in \mathcal{M}^*$, and $Y \in \kappa(X \wedge \varrho)$, then*

$$r \in Y \subseteq X \cup \{r^1, r^2, \cdots, r^m\}.$$

We can now define the *Work Function Algorithm* in Figure 2.3.

The algorithm keeps tracks of a *current work function* ω. Initially the servers are on some configuration S^0 and $\omega = S^0$ (remember that we identify $X \in \Delta^k \mathcal{M}$ with its characteristic function χX.)

Interestingly, the update step of Chrobak and Larmore's algorithm can be done after S' is chosen instead of before, or in parallel. The reason is that $\omega \wedge r(X) = \omega(X)$ for any $r \in X$.

Theorem 3 ([17]) *The space complexity of Chrobak and Larmore's algorithm is the maximum cardinality of $\kappa(\omega)$, over all offset functions ω that occur during the computation. If $\mid \mathcal{M} \mid = n$, that does not exceed the number of multisets in $\mathcal{M}$ of order k which contain a specific point, which is $\binom{n+k}{k-1}$ while $\mathcal{M}$ is infinite, it does not exceed $\binom{M+k-1}{k-1}$ (where m is the number of requests.)*

Theorem 4 ([17]) *The time complexity is dominated by the time needed to update the current work function. If $\mid \mathcal{M} \mid = n$, that time complexity is $O(k\binom{n+k}{k-1})$ for each step. If $\mathcal{M}$ is infinite, it is $O(k\binom{t+k-1}{k-1})$ for the tth step.*

Theorem 5 ([17]) *The work Function Algorithm is 2-competitive for $k = 2$.*

2.4 Definitions

We define and describe k-server algorithms that work on any specific metric space. The underlying metric space $\mathcal{M} = (C, d)$, where C stands for a configuration and d is the distance measure, will usually be omitted from the definitions. Our definitions of 1, 2 and 6 below are equivalent to the definitions in [42] and [43]. The definition of competitiveness against other algorithms follows [23].

Definition 1 *A k-server algorithm starts in some initial k-server configuration C_0 and deals with a sequence of sequence $\sigma = \sigma_1, \sigma_2, \cdots, \sigma_{|\sigma|}$. A configuration is a multiset of k points occupied by the k servers in the metric space. The request σ is a sequence of points in the metric space. The k-server algorithm selects a sequence of configurations $C_1, C_2, \cdots, C_{|\sigma|}$ such that $\sigma_i \in C_i$. We say that such an algorithm serves σ.*

Note that configurations are denoted by capital letters. For sets A, B let $A - B$ denotes their difference. We use $A + a$ for $A \cup \{a\}$ and $A - a$ for $A - \{a\}$.

Definition 2 *An on-line k-server algorithm starts in some initial k-server configuration C_0 and deals with a sequence of requests $\sigma = \sigma_1, \sigma_2, \cdots, \sigma_{|\sigma|}$. The request sequence is presented element by element. Following the presentation of request σ_i, the on-line k-server algorithm selects a configuration C_i such that $\sigma_i \in C_i$. The configuration C_i does not depend on requests $\sigma_{i+1}, \cdots, \sigma_{|\sigma|}$.*

Definition 3 *A minimal match between two configurations C_i and C_j is the minimal distance to move from C_i to C_j. We denote the minimal match between the two configurations by $D(C_i, C_j)$.*

Definition 4 *Suppose a k-server algorithm $\mathcal{A}$ is given a request sequence $\sigma = \sigma_1, \sigma_2, \cdots, \sigma_{|\sigma|}$. and an initial configuration C_0. Let C_i, $i = 1, 2, \cdots, | \sigma |$, be the configurations selected by $\mathcal{A}$. A fixed numbering of $\mathcal{A}$'s servers is a labeling of the*

points in each C_i, $i = 1, 2, \cdots, |\ \sigma\ |$, with distinct labels chosen from $\{1, 2, \cdots, k\}$ such that for $1 \leq i \leq |\ \sigma\ |$, there exists a minimal match of C_i and C_{i-1} such that all matched pairs of points have the same label. We say that the request σ_i is served by server s iff σ_i is labeled s in C_i. We say that a server s moves from a point p to a point q iff p is labeled s in configuration C_i and q is labeled s in configuration C_j and $i < j$.

Definition 5 *A k-server algorithm $\mathcal{A}$ is lazy, iff for every request sequence $\sigma = \sigma_1, \sigma_2, \cdots, \sigma_{|\sigma|}$. and for every initial configuration C_0 the following holds. Let C_i, $i = 1, 2, \cdots, |\ \sigma\ |$, be the configurations selected by $\mathcal{A}$. Then, for every i, $i \leq i \leq |\ \sigma\ |$, if $\sigma_i \in C_{i-1}$, then $C_i = C_{i-1}$, and if $\sigma_i \notin C_{i-1}$, then there exists $p \in C_{i-1}$ such that $C_i = C_{i-1} - p + \sigma_i$.*

Definition 6 *The cost associated with a k-server algorithm $\mathcal{A}$ given an initial configuration C_0 and a request sequence σ is denoted by,*

$$cost_A(C_0, \sigma) = \sum_{i=1}^{|\sigma|} M(C_i, C_{i-1}).$$

Definition 7 *Where $c \in \Re$, an on-line k-server algorithm $\mathcal{A}$ is said to be α-competitive against an algorithm $\mathcal{B}$ iff for every request sequence σ and for every initial configuration C_0,*

$$cost_{\mathcal{A}}(C_0, \sigma) \leq \alpha \cdot cost_{\mathcal{B}}(C_0, \sigma).$$

The infimum of all such c is called the competitive ratio of $\mathcal{A}$ against $\mathcal{B}$.

Definition 8 *An on-line k-server algorithm $\mathcal{A}$ serving requests in a metric space $\mathcal{M}$ is said to be c-competitive iff for every k-server algorithm $\mathcal{B}$ serving requests in $\mathcal{M}$, $\mathcal{A}$ is c-competitive against $\mathcal{B}$. For a metric space $\mathcal{M}$, an infinite sequence of algorithms $\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_k, \cdots$, where for every k, $k \geq 1$, $\mathcal{A}_k$ is an on-line k-server algorithm serving requests in $\mathcal{M}$, is said to be competitive iff there exists a function $\alpha(k)$ of k alone such that for every k, $k \geq 1$, $\mathcal{A}_k$ is $\alpha(k)$-competitive.*

Note that this definition is similar to the definition in [42] and [43] excluding the additive term that we disallow. Any algorithm that is competitive by this definition is also competitive to [42] and [43].

Definition 9 *At any stage we denote by $w(X)$ the optimal cost of servicing the requests so far and ending up at the configuration X; the function w defined from the set of all configurations to the positive real numbers is called work function.*

Definition 10 *A function w is called convex if for all configurations A, B there exists a bijection $h : A \to B$ such that for all bipartitions of A into X, Y, we have,*

$$w(A) + w(B) \geq w(X \cup h(Y)) + w(h(X) \cup Y) \tag{2.1}$$

2.5 The Work Function Algorithm

Suppose ab denote the distance between points a and b.

Fact 1 *If w is the current work function then the resulting work function w' after servicing a new request r is,*

$$w'(X) = \min_{x \in X}\{w(X - x + r) + rx\}.$$

Notice that when X includes the request r then $w'(X) = w(X)$. Because r belongs to $X - x + r$ we have that $w'(X - x + r) = w(X - x + r)$. Consequently, $w'(X) = \min_{x \in X}\{w'(X - x + r) + rx\}$.

From this we get,

Fact 2 *If w is the current work function and r is the last request then $\forall X$: $w(X) = \min_{x \in X}\{w(X - x + r) + rx\}$.*

From Definitions 3 and 9, we have,

> *Let w be a work function and let the on-line algorithm be in configuration A. Assume that r is requested and let w' be the resulting work function. The work function algorithm moves the server from a point $a \in A$ to r that minimizes $w'(A') + ra$, where $A' = A - a + r$ is the resulting configuration.*

Figure 2.2: Our Work Function Algorithm

Fact 3 *For a work function w and two configurations X, Y,*

$$w(X) \leq w(Y) + D(X, Y).$$

Consider a work function w and the resulting work function w' after a new request r. By Fact 3 we get,

$$w'(X) = \min_{x \in X}\{w(X - x + r) + rx\} \geq w(X).$$

So we have,

Fact 4 *Let w be the current work function and let w' be the resulting work function after request r. Then $\forall X : w'(X) \geq w(X)$.*

We assume that initially both the off-line algorithm and the on-line algorithm start at the same configuration, X_i. Moreover, without loss of generality we may assume that when the game concludes they are at the same configuration X_f, otherwise by repeatedly requesting points in X_f the on-line algorithm can be forced to converge to it without increasing the off-line cost. It follows from these assumptions that the off-line cost is $w_f(X_f) - w_i(X_i)$, where w_i, w_f are the initial and the final work function, respectively.

Notice that since $r \in A'$ we can replace $w'(A')$ with $w(A')$ in the above definition. The cost of the on-line algorithm is ra. Define the off-line pseudocost

to be $w'(A') - w(A)$. By summing over all moves, the total off-line pseudocost is equal to the total off-line cost.

Consider the sum of the off-line pseudocost and the on-line cost,

$$w'(A') - w(A) + ra.$$

By the definition of the work function algorithm, this is equal to $w'(A) - w(A)$. Trivially, this quantity is bounded by the maximum over all possible configurations. So, we have that the off-line pseudocost plus the on-line cost is bounded above by,

$$\max_X \{w'(X) - w(X)\}.$$

We call this quantity the *full cost* of a move. The total full cost is the sum of the full cost of each move. Clearly, by the definition of the competitive ratio we have:

Fact 5 *If the total full cost is bounded above by $\alpha + 1$ times the off-line cost plus a constant then the work function algorithm is α-competitive.*

The advantage of using full cost instead of real cost is that we don't have to deal at all with the position of the algorithm. Instead, we have only to show that a certain inequality holds for the work function, for all sequences of requests.

The following lemma provides a stronger version of the convexity condition.

Lemma 1 *If there exists a bijection h that satisfies the conditions in the above definition then there exists a bijection h^* that satisfies the same conditions and $h^*(x) = x$ for all $x \in A \cap B$.*

Proof. Please refer to Appendix A. □

2.6 The Competitive Analysis

Lemma 2 *Work functions are convex.*

Proof. We use induction on the number of requests:

If X_i is the initial configuration and w_i is the initial work function then for all configurations X: $w(X) = D(X, X_i)$.

So, we have,

$$w(A) + w(B) = D(A, X_i) + D(B, X_i).$$

Fix two minimum matchings $D(A, X_i)$ and $D(B, X_i)$. Each point x_j in X_i is matched to a point a_j in A and b_j in B. Obviously, by mapping a_j to b_j we obtain a bijection that satisfies the requirements of the lemma.

Assume that w is convex. It suffices to show that after the next request r the resulting w' is also convex.

By Fact 1,

$$w'(A) = w(A - a + r) + ra,$$

for some $a \in A$ and,

$$w'(B) = w(B - b + r) + rb,$$

for some $b \in B$.

By induction, w is convex and let h be the bijection from $A - a + r$ to $B - b + r$ in the convexity condition. According to Lemma 1 we may assume that $h(r) = r$.

Consider now the bijection $h' : A \to B$ such that,

$$h'(x) = \begin{cases} h(x) & \text{if } x \neq a \\ b & \text{if } x = a \end{cases}$$

and a bipartition of A into X and Y.

Without lost of generality assume that $a \in X$.

We have,

$$\begin{aligned} w'(A) + w'(B) &= w(A - a + r) + w(B - b + r) + ra + rb \\ &= w(X - a + r \cup Y) + w(B - b + r) + ra + rb \end{aligned}$$

$$\begin{aligned} &\geq w(X-a+r \cup h(Y)) + w(h(X-a+r) \cup Y) + ra + rb \\ &= w(X-a+r \cup h'(Y)) + w(h'(X)-b+r \cup Y) + ra + rb \\ &\geq w'(X \cup h'(Y)) + w'(h'(X) \cup Y) \end{aligned}$$

where the first inequality is based on the convexity of w and the second one on Fact 1. So, w' is convex and the lemma follows. □

Now we use the convexity condition to prove the following two Lemmata. In fact, we use the weaker condition,

$$\forall a \in A - B: \quad w(A) + w(B) \geq \min_{b \in B}\{(A-a+b) + w(B-b+a)\}.$$

We need the following definition first.

Definition 11 *A configuration A is called minimizer of a point a with respect to w, if a / in$A$ and A minimizes the expression $w(X) - \sum_{x \in X} ax$, that is $w(A) - \sum_{x \in A} ax = \min_X\{w(X) - \sum_{x \in X} ax\}$.*

Lemma 3 *Let w be a convex work function. Consider a new request at r and the resulting work function w'. If A is a minimizer of r with respect to w then A is also a minimizer of r with respect to w'.*

Proof. It suffices to show that for all configurations B, $r \notin B$:

$$w'(B) - \sum_{b \in B} rb \geq w'(A) - \sum_{a \in A} ra.$$

Using Fact 1, we get,

$$\min_{b' \in B}\{w(B-b'+r) + rb' - \sum_{b \in B} rb\} \geq \min_{a' \in A}\{w(A-a'+r) + ra' - \sum_{a \in A} ra\}.$$

Or equivalently, for all $b' \in B$,

$$w(B-b'+r) + rb' - \sum_{b \in B} rb \geq \min_{a' \in A}\{w(A-a'+r) + ra' - \sum_{a \in A} ra\}.$$

We add $w(A)$ to both sides,

$$w(B-b'+r)+w(A)+rb'-\sum_{b\in B} rb \geq \min_{a'\in A}\{w(A-a'+r)+w(A)+ra'-\sum_{a\in A} ra\} \quad (2.2)$$

Because A is a minimizer of r with respect to w, we have,

$$\begin{aligned} w(A)-\sum_{a\in A} ra &\leq w(B+a'-b')-\sum_{b\in B+a'-b'} rb \quad \Leftrightarrow \\ w(A)+ra'-\sum_{a\in A} ra &\leq w(B+a'-b')+rb'-\sum_{b\in B} rb. \end{aligned}$$

From this and the convexity condition,

$$w(B-b'+r)+w(A) \geq \min_{a'\in A}\{w(B-b'+a')+w(A-a'+r)\}.$$

we get (2.2).

This completes the proof. □

Lemma 4 *With the hypotheses of Lemma 3 the full cost is achieved on A, that is,*

$$w'(A)-w(A)=\max_X\{w'(X)-w(X)\}.$$

Proof. It suffices to show, for all configurations B, $r\notin B$, then

$$w'(A)+w(B)\geq w'(B)+w(A).$$

Rewriting $w'(A)$ and $w'(B)$ as in Fact 1 we get,

$$\min_{a'\in A}\{w(A-a'+r)+a'r+w(B)\} \geq \min_{b'\in B}\{w(B-b'+r)+b'r+w(A)\}.$$

Or equivalently, for all $a'\in A$,

$$w(A-a'+r)+a'r+w(B) \geq \min_{b'\in B}\{w(B-b'+r)+b'r+w(A)\}. \quad (2.3)$$

From the hypotheses we get,

$$\begin{aligned} w(A)-\sum_{a\in A} ra &\leq w(A-a'+b')-\sum_{a\in A-a'+b'} ra \quad \Leftrightarrow \\ w(A)+rb' &\leq w(A-a'+b')+ra' \end{aligned}$$

Substituting this to (2.3), it becomes:

$$w(A - a' + r) + w(B) \geq \min_{b' \in B}\{w(A - a' + b') + w(B - b' + r)\}$$

which holds because w is convex. □

Lemmata 3 and 4 can be combined into the following lemma.

Lemma 5 *Let w be a convex work function and let w' be the resulting work function after request r. Then any minimizer A of r with respect to w is also a minimizer of r with respect to w' and the full cost of servicing the request r occurs on A.*

For configurations $U = \{u_1, \ldots, u_k\}$ and $B_i = \{b_{i1}, \ldots, b_{ik}\}$, $i = 1, \ldots, k$.

Let

$$\Psi(w, U, B_1, \ldots, B_k) = kw(U) + \sum_{i=1}^{k} \left(w(B_i) - \sum_{j=1}^{k} u_i b_{ij} \right).$$

Let $\Phi(w)$ denote its minimum value over all configurations U and B_i, $i = 1, \ldots, k$.

Lemma 6 *For any work function w, the minimum value of $\Psi(w, U, B_1, \ldots, B_k)$ is achieved for $r = u_i$, for some i, where r is the last request.*

Proof. By Fact 2, for some $i \in 1 \ldots k$:

$$w(U) = w(U - u_i + r) + ru_i.$$

If we substitute this to $\Psi(w, U, B_1, \ldots, B_k)$, using the k triangle inequalities $ru_i - u_i b_{ij} \geq -rb_{ij}$ we get,

$$\Psi(w, U, B_1, \ldots, B_k) \geq \Psi(w, U - u_i + r, B_1, \ldots, B_k)$$

and the lemma follows since,

$$r \in U - u_i + r.$$

□

Theorem 6 *The work function algorithm is* $(2k-1)$*-competitive.*

Proof. Consider a convex work function w and let w' be the resulting work function after request r.

By Lemma 6 the minimum value of $\Psi(w', U, B_1, \ldots, B_k)$ is achieved for $u_i = r$, for some i. Let A be a minimizer of r with respect to w. Then by Lemma 5, A is also a minimizer of r with respect to w' and it is not difficult to see that the minimum value of $\Psi(w', U, B_1, \ldots, B_k)$ is unaffected if we fix $B_i = A$. Fix the remaining points u_j and b_{ij}, where $\Psi(w', U, B_1, \ldots, B_k)$ achieves its minimum. Let $\Psi_{w'}$, Ψ_w denote the values of Ψ on these points with respect to w' and w.

Obviously,

$$\Phi(w') - \Phi(w) \geq \Psi_{w'} - \Psi_w.$$

Using Fact 4, we have,

$$\Psi_{w'} - \Psi_w \geq w'(A) - w(A).$$

Putting these together, we get,

$$\Phi(w') - \Phi(w) \geq w'(A) - w(A).$$

Because A is a minimizer of r with respect to w, according to Lemma 5 the full cost is $w'(A) - w(A)$. Thus, we conclude that the full cost to serve request r is bounded above by $\Phi(w') - \Phi(w)$. Summing over all moves we get that the total full cost is bounded above by $\Phi(w_f) - \Phi(w_i)$, where w_i and w_f are the initial and the final work functions, respectively.

Let X_i and X_f be the initial and final configurations. We have,

$$\Phi(w_f) \leq \Psi(w_f, X_f, X_f, \ldots, X_f) = 2kw_f(X_f) - 2C(X_f) \leq 2kw_f(X_f),$$

where $C(X)$ denotes the sum of all distances between the points of X. Initially, for all configurations X, $w_i(X) = D(X, X_i)$ and it is not difficult to see that $\Phi(X_i) = -2C(X_i)$. Consequently, the full cost is at most $2kw_f(X_f) + 2C(X_i)$.

Because the off-line cost is $w_f(X_f)$, the total full cost is bounded above by $2k$ times the off-line cost plus a constant depending only on the initial configuration. Using Fact 5, we conclude that the work function algorithm is $(2k-1)$-competitive. □

Chapter 3

The weighted k-server problem

3.1 Introduction

In this chapter we deal with a generalization of the k-server problem, in which the servers are unequal. In the *weighted server model* each of the servers is assigned a positive weight. The cost associated with moving a server is the product of the distance traversed and the server weight.

An on-line weighted server algorithms is called competitive if the competitive ratio depends only on the number of servers, and is independent of the server weights.

Weighted servers on the uniform metric space model problems that use memories that have differing read/write costs, e.g. E^2PROM memories that can be read from in microseconds but require many milliseconds for write operations. The problem we define is to determine what computations should be stored in fast write memories and what computations should be stored in slow write memories. We assume that the time to read from the different memories are similar. This is true in practice.

The justification for using different classes of memory is that the VLSI chip area for memory is inversely proportional to the *write time* to that memory.

Typical memories such as RAM, E^2PROM, and ROM differ by orders of magnitude in their area and write time requirements. The read times of the different memories are very similar. For example, modern smartcards have a chip with 10^2 bytes of RAM, 10^3 bytes of E^2PROM, and 10^4 bytes of ROM, with write times of microseconds, milliseconds and infinity, respectively. The chip area used by the different memories is dominated by the RAM requirements, with E^2PROM and ROM both using approximately equal areas.

The points in the metric space correspond to intermediate computations. The different weights represent the different types of memory. These is a server of weight w for every slot of memory that requires w time for a write operation. If a server resides on a point, then that intermediate computation resides in that slot of memory.

The request sequence $\sigma = \sigma_1, \sigma_2, \ldots$, consists of points that represent read access to some intermediate computation. If such an intermediate result is in memory then we access it free of charge. if not, then it has to be recomputated and stored in some class of memory. Serving a request with a server of least weight corresponds to placing that result in fast memory. The weights are normalized to represent the relative costs of recomputing an intermediate value, and writing to the different classes of memory.

It is trivial to obtain a competitive ratio depending on the ratio between the weights, using any competitive k-server algorithm. The natural question relating to this problem is whether the competitive ratio depends only upon k.

Followed the idea of *work function* we defined in Chapter 2, we give a $(2k^k - 1)$-competitive algorithm for every metric space and every set of weights associated with the servers.

This chapter is organized as follows. In Section 2, we discuss related work and particularly, we give Fiat and Ricklin's algorithm in Section 3. In Section 4 we give our weighted k-server algorithm on any metric space and present its

competitive analysis in the subsequent section.

3.2 Related Work

Lee's model [41] is the first piece of work to deal with the weighted server problem with result that depends on the servers' weights. He gave a lower bound of $k \cdot \frac{w_{min}}{w_{avg}}$ and an upper bound for the uniform metric space of $k \cdot \frac{w_{avg}}{w_{min}}$, where w_{min} and w_{avg} are the minimal weight and the average weight, respectively. Fiat and Ricklin [27] improved Lee's work and gave a $2^{2^{O(k)}}$-competitive k-server algorithm for the uniform metric space and any set of weights associated with those k servers. Moreover, they used the $\mathcal{MIN}$ operator [24], [25] to design an $O(k^{O(k)})$-competitive deterministic and an $O(k^3 \cdot \log k)$-competitive randomized k-server algorithms for the case where the weights are either 1 or w. This is the first application of the randomized $\mathcal{MIN}$ operator.

3.3 Fiat and Ricklin's Algorithm

We describe Fiat and Ricklin's algorithm $\mathcal{SAMPLE}_k$ [27], [28] inductively.

For simplicity, we define the following functions, that range over positive integers: $c(i) = 3^{2^{2i}}$ and $f(i) = 5^i \cdot c(1) \cdots c(i-1) \cdot c(i)$. Note that for all $i \geq 1$; we have:

$$c(i+1) \geq 5 \cdot c(i)^2;$$

and

$$f(i+1) \geq 5 \cdot f(i) \cdot c(i+1).$$

Let $w_1 \leq w_2 \leq \cdots \leq w_k$ be the sequence of weights associated with the servers. We transform these weights into *normalized weights* $\bar{w_1}, \bar{w_2}, \cdots, \bar{w_k}$ with the property that $\bar{w_{i+1}}$ is divisible by $2 \cdot (1 + c(i)) \cdot \bar{w_i}$.

Lemma 7 *Any competitive weighted server algorithm for the normalized weights implies a competitive weighted algorithm for the original weights.*

Proof. We stratch Fiat and Ricklin's main idea of proof. The did the transformation inductively, choosing $\bar{w_1} = w_1$ and $\bar{w_{i+1}}$ to be the smallest multiple of $2 \cdot (1 + c(i)) \cdot \bar{w_i}$, greater than w_{i+1}. It is now simple arithmetic to verify that $\frac{\bar{w_i}}{w_i} \leq f(i)$, for all i. Given a α-competitive algorithm for the normalized weights, it implies an $(f(k) \cdot \alpha)$-competitive algorithm for the original weights. This follows because the on-line algorithm cost is smaller with the original weights than with the normalized weights and the adversary with the normalized weights pays almost $f(k)$ times more than the adversary witht he original weights.

This completes the proof. □

For $k = 1$ they ran the greedy 1-server algorithm. They defined a phase of $\mathcal{SAMPLE}_1$ to be $2(c(1) + 1)$ requests. For $k \geq 2$, $\mathcal{SAMPLE}_k$ operates as follows.

The pseudo-code for the Fiat and Ricklin's algorithm appears in Figure 3.2.

Divide the process of the algorithm into phases. Every phase is independent of all previous requests. A phase is divided into $c(k)+1$ *sub-phases. The first sub-phase begins by moving the server of weight* $\bar{w}_k$ *to an arbitrary point. Now, run* $\mathcal{SAMPLE}_{k-1}$ *with the* $k-1$ *lighter servers, until the cost incurred by* $\mathcal{SAMPLE}_{k-1}$ *reaches* $\bar{w}_k$. *For every point* x, $charge(x)$ *is the total number of requests invoked on* x. *Let* P *be the set of all points requested during the execution of* $\mathcal{SAMPLE}_{k-1}$. *If* $\mid P \mid < c(k)$, *let* $S = P$; *otherwise, take* S *to be the set of the* $c(k)$ *points of maximal charge during the execution of* $\mathcal{SAMPLE}_{k-1}$. *The rest of the* $\mathcal{SAMPLE}_k$ *phase consists of* $c(k)$ *sub-phases, each of which starts by moving the server of weight* $\bar{w}_k$ *to an unmarked point in* S, *and marking that point. Next, the sub-phase runs* $\mathcal{SAMPLE}_{k-1}$ *with the* $k-1$ *lighter severs, until the cost incurred by* $\mathcal{SAMPLE}_{k-1}$ *reaches* $\bar{w}_k$. *The number of sub-phases in a phase of* $\mathcal{SAMPLE}_k$ *is always* $c(k)+1$, *even if the number of points in* S *is smaller than* $c(k)$. *If there are no unmarked points in* S, *simply move the server of weight* $\bar{w}_k$ *to an arbitrary point in the metric space.*

Figure 3.1: Fiat and Ricklin's Algorithm

3.4 The Work Function Algorithm

Suppose ab denote the distance between points a and b and a be the weight of the server a.

Fact 6 *If w is the current work function then the resulting work function w' after servicing a new request r is:*

$$w'(X) = \min_{x \in X}\{w(X - x + r) + x \cdot rx\}.$$

Notice that when X includes the request r then $w'(X) = w(X)$. Because r belongs to $X - x + r$ we have that $w'(X - x + r) = w(X - x + r)$. Consequently, $w'(X) = \min_{x \in X}\{w'(X - x + t) + x \cdot rx\}$. From this we get:

Fact 7 *If w is the current work function and r is the last request then $\forall X$:* $w(X) = \min_{x \in X}\{w(X - x + r) + x \cdot rx\}$.

From Definitions 3 and 9 in previous chapter, we have:

Fact 8 *For a work function w and two configurations X, Y:*

$$w(X) \leq w(Y) + D(X, Y).$$

Consider a work function w and the resulting work function w' after a new request r. By Fact 8 we get:

$$w'(X) = \min_{x \in X}\{w(X - x + r) + x \cdot rx\} \geq w(X)$$

So we have:

Fact 9 *Let w be the current work function and let w' be the resulting work function after request r. Then $\forall X : w'(X) \geq w(X)$.*

```
SAMPLE_k(w̄_1, ..., w̄_k)
  begin
    do forever
      call phase_k(w̄_1, ..., w̄_k);
    end
  end

phase_1(w̄_1)
  begin
    for i = 1 to 2(c(1) + 1) do
      wait;
      upon request on a point p;
      move the w̄_1 server to p;
    end
  end

phase_k(w̄_1, ..., w̄_k)
  begin
    move the w̄_k server to an arbitrary point;
    call sub − phase_k(w̄_1, ..., w̄_k);
    for every point x, let charge(x) equal the number of requests
      invoked on x during the sub-phase;
    let P = {p : charge(p) > 0};
    if (| P |< c(k))
    then
      let S = P
    else
      let S equal the set of c(k) points of maximal charge;
    for i = 1 to c(k) do
      choose an arbitrary point, x ∈ S;
      move the w̄_k server to x;
      call sub − phase_k(w̄_1, ..., w̄_k)
      if | S |> 1
      then remove x from S;
    end
  end
```

$sub - phase_k(\bar{w}_1, \cdots, \bar{w}_k)$
begin
run $\mathcal{SAMPLE}_{k-1}(\bar{w}_1, \cdots, \bar{w}_k)$ until the cost
incurred reaches $\bar{w}_k$;
end

Figure 3.2: Fiat and Ricklin's Algorithm

Let w be a work function and let the on-line algorithm be in configuration A. Assume that r is requested and let w' be the resulting work function. The **work function algorithm** *moves the server from a point $a \in A$ to r that minimizes $w'(A') + a \cdot ra$, where $A' = A - a + r$ is the resulting configuration.*

Figure 3.3: Work Function Algorithm

We assume that initially both the off-line algorithm and the on-line algorithm start at the same configuration, X_i. Moreover, without loss of generality we may assume that when the game concludes they are at the same configuration X_f, otherwise by repeatedly requesting points in X_f the on-line algorithm can be forced to converge to it without increasing the off-line cost. It follows from these assumptions that the off-line cost is $w_f(X_f) - w_i(X_i)$, where w_i, w_f are the initial and the final work function, respectively.

Notice that since $r \in A'$, we can replace $w'(A')$ with $w(A')$ in the above definition. The cost of the on-line algorithm is $a \cdot ra$. Define the off-line pseudo-cost to be $w'(A') - w(A)$. By summing over all moves, the total off-line pseudo-cost is equal to the total off-line cost. Consider the sum of the off-line pseudo-cost and the on-line cost:

$$w'(A') - w(A) + a \cdot ra$$

By the definition of the work function algorithm, this is equal to $w'(A) - w(A)$. Trivially, this quantity is bounded by the maximum over all possible configurations. So, we have that the off-line pseudo-cost plus the on-line cost is bounded above by:

$$\max_X \{w'(X) - w(X)\}$$

We call this quantity the *full cost* of a move. The total full cost is the sum of the full cost of each move. Clearly, by the definition of the competitive ratio we have:

Fact 10 *If the total full cost is bounded above by $\alpha + 1$ times the off-line cost plus a constant then the work function algorithm is α-competitive.*

The advantage of using full cost instead of real cost is that we don't have to deal at all with the position of the algorithm. Instead, we have only to show that a certain inequality holds for the work function, for all sequences of requests.

3.5 The Competitive Analysis

Lemma 8 *Work functions are convex.*

Proof. We use induction on the number of requests.

If X_i is the initial configuration and w_i is the initial work function then for all configurations $X : w(X) = D(X, X_i)$. So, we have:

$$w(A) + w(B) = D(A, X_i) + D(B, X_i).$$

Fix $D(A, X_i)$ and $D(B, X_i)$. Each point x_j in X_i is matched to a point a_j in A and b_j in B. Obviously, by mapping a_j to b_j we obtain a bijection that satisfies the requirements of the lemma.

Assume that w is convex. It suffices to show that after the next request r the resulting w' is also convex.

By Fact 6,

$$w'(A) = w(A - a + r) + a \cdot ra,$$

for some $a \in A$ and,

$$w'(B) = w(B - b + r) + b \cdot rb,$$

for some $b \in B$.

By induction, w is convex and let h be the bijection from $A - a + r$ to $B - b + r$ in the convexity condition. According to Lemma 1 we may assume that $h(r) = r$.

Consider now the bijection $h' : A \to B$ such that

$$h'(x) = \begin{cases} h(x) & if\ x \neq a \\ b & if\ x = a \end{cases}$$

and a bipartition of A into X and Y.

Without lost of generality assume that $a \in X$. We have:

$$\begin{aligned} w'(A) + w'(B) &= w(A - a + r) + w(B - b + r) + a \cdot ra + b \cdot rb \\ &= w(X - a + r \cup Y) + w(B - b + r) + a \cdot ra + b \cdot rb \\ &\geq w(X - a + r \cup h(Y)) + w(h(X - a + r) \cup Y) + a \cdot ra + b \cdot rb \\ &= w(X - a + r \cup h'(Y)) + w(h'(X) - b + r \cup Y) + a \cdot ra + b \cdot rb \\ &\geq w'(X \cup h'(Y)) + w'(h'(X) \cup Y) \end{aligned}$$

where the first inequality is based on the convexity of w and the second one on Fact 6. So, w' is convex and the lemma follows. □

Now we use the convexity condition to prove the following two lemmata. In fact, we use the weaker condition:

$$\forall a \in A - B : w(A) + w(B) \geq \min_{b \in B}\{w(A - a + b) + w(B - b + a)\}$$

We need the following definition first.

Definition 12 *A configuration A is called the* **minimizer** *of a point a with respect to w, if $a \notin A$ and A minimizes the expression $w(X) - \sum_{x \in X} x \cdot ax$, i.e., $w(A) - \sum_{x \in A} x \cdot ax = \min_X\{w(X) - \sum_{x \in X} x \cdot ax\}$.*

Lemma 9 *Let w be a convex work function. Consider a new request at r and the resulting work function w'. If A is a minimizer of r with respect to w then A is also a minimizer of r with respect to w'.*

Proof. It suffices to show that for all configurations B, $r \notin B$:

$$w'(B) - \sum_{b \in B} b \cdot rb \geq w'(A) - \sum_{a \in A} a \cdot ra$$

Using Fact 6, we get:

$$\min_{b' \in B}\{w(B - b' + r) + b' \cdot rb' - \sum_{b \in B} b \cdot rb\} \geq \min_{a' \in A}\{w(A - a' + r) + a' \cdot ra' - \sum_{a \in A} a \cdot ra\}$$

Or equivalently, $\forall b' \in B$:

$$w(B - b' + r) + b' \cdot rb' - \sum_{b \in B} b \cdot rb \geq \min_{a' \in A}\{w(A - a' + r) + a' \cdot ra' - \sum_{a \in A} a \cdot ra\}$$

We add $w(A)$ to both sides:

$$w(B-b'+r)+w(A)+b' \cdot rb' - \sum_{b \in B} b \cdot rb \geq \min_{a' \in A}\{w(A-a'+r)+w(A)+a' \cdot ra' - \sum_{a \in A} a \cdot ra\} \tag{3.1}$$

Because A is a minimizer of r with respect to w:

$$\begin{aligned} w(A) - \sum_{a \in A} a \cdot ra &\leq w(B + a' - b') - \sum_{b \in B + a' - b'} b \cdot rb \\ w(A) + a' \cdot ra' - \sum_{a \in A} a \cdot ra &\leq w(B + a' - b') + b' \cdot rb' - \sum_{b \in B} b \cdot rb \end{aligned}$$

From this and the convexity condition:

$$w(B - b' + r) + w(A) \geq \min_{a' \in A}\{w(B - b' + a') + w(A - a' + r)\}$$

we get (3.1).

This completes the proof. □

Lemma 10 *With the hypothesis of Lemma 9 the full cost is achieved on A; i.e.*

$$w'(A) - w(A) = \max_{X}\{w'(X) - w(X)\}$$

Proof. It suffices to show, for all configurations B, $r \notin B$:

$$w'(A) + w(B) \geq w'(B) + w(A)$$

Rewriting $w'(A)$ and $w'(B)$ as in Fact 6 we get:

$$\min_{a' \in A}\{w(A - a' + r) + a' \cdot ra' + w(B)\} \geq \min_{b' \in B}\{w(B - b' + r) + b' \cdot rb' + w(A)\}$$

Or equivalently, $\forall a' \in A$:

$$w(A - a' + r) + a' \cdot ra' + w(B) \geq \min_{b' \in B}\{w(B - b' + r) + b' \cdot rb' + w(A)\} \quad (3.2)$$

From the hypothesis we get:

$$\begin{aligned} w(A) - \sum_{a \in A} a \cdot ra &\leq w(A - a' + b') - \sum_{a \in A - a' + b'} a \cdot ra \\ w(A) + b' \cdot rb' &\leq w(A - a' + b') + a' \cdot ra' \end{aligned}$$

Substituting this to (3.2), it becomes:

$$w(A - a' + r) + w(B) \geq \min_{b' \in B}\{w(A - a' + b') + w(B - b' + r)\}$$

which holds because w is convex. □

Lemmata 9 and 10 can be combined into the following lemma.

Lemma 11 *Let w be a convex work function and let w' be the resulting work function after request r. Then any minimizer A of r with respect to w is also a minimizer of r with respect to w' and the full cost of servicing the request r occurs on A.*

For configurations $U = \{u_1, \ldots, u_k\}$ and $B_i = \{b_{i1}, \ldots, b_{ik}\}$, $i = 1, \ldots, k$, let

$$\Psi(w, U, B_1, \ldots, B_k) = k^k w(U) + \sum_{i=1}^{k} \left(k^{k-1} w(B_i) - \sum_{j=1}^{k} u_i \cdot u_i b_{ij} \right)$$

Let $\Phi(w)$ denote its minimum value over all configurations U and B_i, $i = 1, \ldots, k$.

Lemma 12 *For any work function w, the minimum value of $\Psi(w, U, B_1, \ldots, B_k)$ is achieved for $r = u_i$, for some i, where r is the last request.*

Proof. By Fact 7, for some $i \in \{1 \ldots k\}$:

$$w(U) = w(U - u_i + r) + u_i \cdot r u_i$$

If we substitute this to $\Psi(w, U, B_1, \ldots, B_k)$, using the k triangle inequalities $u_i \cdot (r u_i - u_i b_{ij}) \geq -r b_{ij}$, we get:

$$\Psi(w, U, B_1, \ldots, B_k) \geq \Psi(w, U - u_i + r, B_1, \ldots, B_k)$$

and the lemma follows since $r \in U - u_i + r$. □

Theorem 7 *The work function algorithm is $(2k^k - 1)$-competitive.*

Proof. Consider a convex work function w and let w' be the resulting work function after request r.

By Lemma 12 the minimum value of $\Psi(w', U, B_1, \ldots, B_k)$ is achieved for $u_i = r$, for some i. Let A be a minimizer of r with respect to w. Then by Lemma 11, A is also a minimizer of r with respect to w' and it is not difficult to see that the minimum value of $\Psi(w', U, B_1, \ldots, B_k)$ is unaffected if we fix $B_i = A$. Fix the remaining points u_j and b_{ij}, where $\Psi(w', U, B_1, \ldots, B_k)$ achieves its minimum. Let $\Psi_{w'}$, Ψ_w denote the values of Ψ on these points with respect to w' and w. Obviously,

$$\Phi(w') - \Phi(w) \geq \Psi'_w - \Psi_w.$$

Using Fact 9, we have,

$$\Psi'_w - \Psi_w \geq w'(A) - w(A).$$

Putting these together, we get,

$$\Phi(w') - \Phi(w) \geq w'(A) - w(A).$$

Because A is a minimizer of r with respect to w, according to Lemma 11 the full cost is $w'(A) - w(A)$. Thus, we conclude that the full cost to serve request r is bounded above by $\Phi(w') - \Phi(w)$. Summing over all moves we get that the total full cost is bounded above by $\Phi(w_f) - \Phi(w_i)$, where w_i and w_f are the initial and the final work functions, respectively.

Let X_i and X_f be the initial and final configurations. We have:

$$\Phi(w_f) \leq \Psi(w_f, X_f, X_f, \ldots, X_f) = 2k^k w_f(X_f) - 2C(X_f) \leq 2k^k w_f(X_f)$$

where $C(X)$ denotes the product of the sum of all distances between the points of X and the weight of the server moved to serve the request r. Initially, for all configurations X, $w_i(X) = D(X, X_i)$ and it is not difficult to see that $\Phi(X_i) = -2C(X_i)$. Consequently, the full cost is at most $2k^k w_f(X_f) + 2C(X_i)$. Because the off-line cost is $w_f(X_f)$, the total full cost is bounded above by $2k^k$ times the off-line cost plus a constant depending only on the initial configuration. Using Fact 10, we conclude that the work function algorithm is $(2k^k - 1)$-competitive.

□

Chapter 4

The Influence of Lookahead

4.1 Introduction

In this chapter we study the influence of *lookahead* on on-line algorithms. The only advantage an off-line algorithm has over an on-line one is its ability to see the future. It might be expected that allowing an on-line algorithm accesses to some finite requests in advance would result in improving its performance relative to that of the off-line algorithm. It follows from the fact that "experience is the best teacher" in our personal lives.

An important question is, what improvement can be achieved in terms of competitiveness, if an on-line algorithm knows not only the present request to be served, but also some future requests. This issue is fundamental from both the practical and the theoretical points of view. In the k-server systems (e.g. any modern memory system, etc.) some requests usually wait in line to be processed by k-server algorithms. One reason is that requests do not necessarily arrive one after the other, but rather in blocks of possibly variable size. Furthermore, if several processes run on a computer, it is likely that some of them incur page faults which then wait for service. Many memory systems are also equipped with different prefetching mechanisms; i.e. on a request not only the currently

accessed page but also some related pages which are expected to be asked next are demanded to be in fast memory. Thus each request generates a number of additional requests. In fact, some paging algorithms used in practice make use of lookahead [51]. In the theoretical context a natural question is: *Is it worth to know the future?*

Contrary to the expectation of "experience is the best teacher", as far as the competitiveness is concerned, no finite lookahead is sufficient for any improvement in the performance of an on-line algorithm. What is the reason for this discrepancy between daily practice and what should be expected from a measure of success for on-line algorithms? The answer is twofold. First, in real life situations there is usually a correlation between past and future. Second, in practice there is a always a limitation on computational resources so that not all possible pre-computation is available.

This chapter is organized as follows. In Section 2, we discuss related work and discuss the role of lookahead in Section 3. The $\mathcal{LRU}$ algorithm with lookahead is given in Section 4 while its competitive analysis is presented in the subsequent section.

4.2 Related Work

Previous research on lookahead in on-line algorithms has mostly addressed dynamic location problems and on-line graph problems [18], [31], [32], [35]; only very little is known in the area of the k-server problem with lookahead.

Consider the intuitive model of lookahead, which we call *pseudo-lookahead.* Let $l \geq 1$ be an integer. We say that an on-line k-server algorithm has a *pseudo-lookahead*, if it sees the present request to be served and the next l future requests. It is well known that this model cannot improve the competitiveness

of any on-line k-server algorithm. If an on-line k-server algorithm has *pseudo-lookahead*, then an adversary that constructs a request sequence can simply replicate each request l times in order to make the lookahead useless.

Theorem 8 *For every on-line pseudo-lookahead k-server algorithm $\mathcal{A}$ there exists a fully on-line (i.e. lookahead $l = 1$) algorithm $\mathcal{B}$ achieving exactly the same competitive ratio.*

Proof. Let $\mathcal{B}$ respond to any sequence of requests $\sigma = \sigma_1, \sigma_2, \cdots, \sigma_t$ by simulating $\mathcal{A}$'s behavior on $\sigma^l = \underbrace{\sigma_1, \cdots, \sigma_1}_{l \text{ times}}, \cdots, \underbrace{\sigma_2, \cdots, \sigma_2}_{l \text{ times}}, \cdots, \underbrace{\sigma_t, \cdots, \sigma_t}_{l \text{ times}}$. It is straightforward to note that,

$$\frac{C_{\mathcal{A},\sigma}}{C_{\mathcal{OPT},\sigma}} = \frac{C_{\mathcal{A},\sigma^l}}{C_{\mathcal{OPT},\sigma^l}}.$$

This completes the proof. □

It should be noted that this simple argument remains valid when the on-line algorithm is allowed to choose its sequences with the help of a random source (and the cost of serving a sequence is defined as the average cost over the bits generated by that source.)

The only result known on the k-server problem with lookahead has been developed by Young [53]. According to Young, a k-server algorithm is on-line with a resource-bounded lookahead of size l if it sees the present request and the maximal sequence of future requests for which it will incur l misses. Young gave deterministic and randomized on-line k-server algorithms with resource-bounded lookahead l which are $\max(\frac{2k}{l}, 2)$-competitive and $2(ln(\frac{k}{l}) + 1)$-competitive, respectively. However, the model of resource-bounded lookahead is unrealistic in practice.

4.3 The Role of l-lookahead

We now introduce a new model of lookahead which has practical as well as theoretical importance. As we shall see, this model can significantly improve

> *The on-line algorithm sees the present request and a sequence of future requests. This sequence contains l pairwise distinct requests which also differ from the current request. More precisely, when serving request σ_t, the algorithm knows requests $\sigma_{t+1}, \sigma_{t+2}, \cdots, \sigma_{t'}$, where $t' = min\{s > t : \mid \{\sigma_t, \sigma_{t+1}, \cdots, \sigma_s\} \mid = l + 1\}$. The request σ_s, where $s \geq t' + 1$, is not seen by the on-line algorithm at time t.*

Figure 4.1: l-lookahead

the competitiveness of any k-server algorithm. Let $\sigma = \sigma_1, \sigma_2, \cdots, \sigma_m$ be a request sequence of length m, and σ_t denote the request at time t. For a given set S, let $\mid S \mid$ denote the cardinality of S and $l \geq 1$ be an integer.

l-lookahead is motivated by the observation that in request sequences generated by real programs, for example, subsequences of consecutive requests generally contain a number of distinct pages. Furthermore, *l-lookahead* is of interest in the theoretical context when we ask how significant it is to know part of the future. An adversary may replicate requests in the lookahead, but nevertheless it has to reveal some really significant information on future requests.

In the following, we always assume that an on-line algorithm has a *l-lookahead*, where $l \geq 1$. If a request sequence $\sigma = \sigma_1, \sigma_2, \cdots, \sigma_m$ is given, then for all $t \geq 1$, we define a value λ_t.

If

$$\mid \{\sigma_t, \sigma_{t+1}, \cdots, \sigma_m\} \mid < l + 1,$$

then

$$\lambda_t = m;$$

otherwise

$$\lambda_t = \min\{t' > t : \mid \{\sigma_t, \sigma_{t+1}, \cdots, \sigma_{t'}\} \mid = l + 1\}.$$

> *Assume $k \geq 3$ and $l \leq k-2$. If the request point r is currently occupied by any of the k servers, then no server moves; otherwise, serve r using that server that was least recently moved to serve a request in the past.*

Figure 4.2: The $\mathcal{LRU}_l$ Algorithm

The lookahead L_t at time t is defined as:

$$L_t = \{\sigma_s : s = t, t+1, \cdots, \lambda_t\}.$$

We say that a request σ_t is in the lookahead of time t if $\sigma_t \in L_t$.

4.4 The $\mathcal{LRU}$ Algorithm with l-lookahead

Unless otherwise stated, we assume in the following that our algorithm is *lazy*; i.e. it only moves a server on a miss.

4.5 The Competitive Analysis

Let $\sigma = \sigma_1, \sigma_2, \cdots, \sigma_m$ be a request sequence of length m. Without loss of generality, both $\mathcal{LRU}_l$ and $\mathcal{OPT}$ start with an empty configuration and that on the first k misses, they move all of their k servers to serve the requests. Furthermore, we assume that σ contains at least $l+1$ distinct requests, where $l \leq k-2$.

For $t = 1, 2, \cdots, \lambda_1 - 1$, let

$$\mu_t = 1,$$

and for $t = \lambda_1, \lambda_1 + 1, \cdots, m$, let

$$\mu_t = \max\{t' < t : \mid \{\sigma_{t'}, \sigma_{t'+1}, \cdots, \sigma_t\} \mid = l+1\}.$$

Define

$$M_t = \{\sigma_s : s = \mu_t, \mu_t + 1, \cdots, t\}.$$

In other words, for a given time t, the set M_t contains the last $l + 1$ requests.

After a request at time t, where $t = 1, 2, \cdots, m$, we denote the set of requests contained in $\mathcal{LRU}_l$'s configuration by $S_{\mathcal{LRU}_l,t}$ and the set of requests contained in $\mathcal{OPT}$'s configuration by $S_{\mathcal{OPT}_t}$. Note that $S_{\mathcal{LRU}_l,0}$ and $S_{\mathcal{OPT},0}$ are the set of requests in the initial configurations; i.e.

$$S_{\mathcal{LRU}_l,0} = S_{\mathcal{OPT},0} = \emptyset.$$

We assign a cost to each different request, say x. Such cost will be updated if x is requested at time t. Let $\eta_{x,t}$ be the cost of x after time t.

If $x \notin S_{\mathcal{LRU}_l,t}$ or $x \in L_t$, then,

$$\eta_{x,t} = 0.$$

Let $j = \mid S_{\mathcal{LRU}_l,t} - L_t \mid$. Assign a cost from the range $[1, j]$ to each request in $S_{\mathcal{LRU}_l,t} - L_t$ such that for any two requests, $x, y \in S_{\mathcal{LRU}_l,t} - L_t$ and

$$\eta_{x,t} < \eta_{y,t},$$

iff the last request to x occurred earlier than the last request to y.

For $t = 1, 2, \cdots, m$, let

$$S_t = S_{\mathcal{LRU}_l,t} - (M_t \cup L_t \cup S_{\mathcal{OPT},t}).$$

We now define the potential function:

$$\Phi_t = \sum_{x \in S_t} \eta_{s,t}.$$

Intuitively, S_t contains those requests which cause $\mathcal{LRU}_l$ to have a higher cost than $\mathcal{OPT}$. Instead of the requests $x \in S_t$, $\mathcal{OPT}$ can keep those requests which are not contained in $S_{\mathcal{LRU}_l,t}$ but will be requested in the future. The cost

$\eta_{x,t}$ of a request $x \in S_t$ equals the number of misses that $\mathcal{LRU}_l$ must incur before it can move a server at x.

For the analysis of our algorithm, we partition the request sequence σ into a number of phases, from phase 0 to phase p, such that the phase 0 contains at most $l+1$ distinct requests and the phase i, where $i = 1, 2, \cdots, p$, follows:

1. Let t_i^b and t_i^e denote the beginning and the end of the phase i, respectively. Phase i contains exactly $l+1$ distinct requests; i.e.

$$| \{\sigma_{t_i^b}, \sigma_{t_i^b+1}, \cdots, \sigma_{t_i^e}\} | = l+1.$$

2. For all $x \in S_{\mathcal{LRU}_l, t_{i-1}^e} - (L_{t_i^e} \cup S_{\mathcal{OPT}, t_{i-1}^e})$, we have:

$$\eta_{x,t_i^e} \leq k - l - 2.$$

In the following, we use mathematical induction reversely to decompose σ. That is, we partition the request sequence starting at the end of σ. Suppose that we have already constructed phases $i+1, i+2, \cdots, p$. We show how to generate the phase i. Let $t_i^e = t_{i+1}^b - 1$. (We let $t_p^e = m$ at the beginning of the decomposition.) Now we set $t = \mu_{t_i^e}$ and compute $S_{\mathcal{LRU}_l, t-1} - L_t$.

If $S_{\mathcal{LRU}_l, t-1} - L_t \neq \emptyset$, then let y be the most recent request in $S_{\mathcal{LRU}_l, t-1} - L_t$. We consider two cases.

1. If $S_{\mathcal{LRU}_l, t-1} - L_t = \emptyset$ or if $S_{\mathcal{LRU}_l, t-1} - L_t \neq \emptyset$ and $y \in S_{\mathcal{OPT}, t-1}$, then let $t_i^b = t$ and call the phase $i = \sigma_{t_i^b}, \sigma_{t_i^b+1}, \cdots, \sigma_{t_i^e}$ a type 1 phase.

2. If $S_{\mathcal{LRU}_l, t-1} - L_t \neq \emptyset$ and $y \notin S_{\mathcal{OPT}, t-1}$ then let t', where $t' < t$, be the time when $\mathcal{OPT}$ moves the server at y most recently. Let $t_i^b = t'$ and call the phase $i = \sigma_{t_i^b}, \sigma_{t_i^b+1}, \cdots, \sigma_{t_i^e}$ a type 2 phase.

Lemma 13 *Phase 0 contains at most $l+1$ distinct requests.*

Proof. We show that the phase 0 is a type 1 phase. This immediately implies that the phase 0 contains at most $l+1$ requests. If the phase 0 was a type 2 phase, then $\mathcal{OPT}$ had moved a server on the first request σ_1. However, this is impossible because the initial configuration is empty and on the first k misses both $\mathcal{LRU}_l$ and $\mathcal{OPT}$ move the servers to serve those requests.

This completes the proof. □

Lemma 14 *Every phase i, $1 \leq i \leq p$ is found on the above partition.*

Proof. Consider an arbitrary phase i, where $1 \leq i \leq p$. Let $t = \mu_{t_i^e}$. If $S_{\mathcal{LRU}_l,t-1} - L_t \neq \emptyset$, then let y be the most recent request in $S_{\mathcal{LRU}_l,t-1} - L_t$ and t", where $t" < t$, be the time when y was requested most recently. If the phase i is a type 2 phase, then let t', where $t' \leq t-1$, be the time when $\mathcal{OPT}$ moves the server at y most recently. (Since $y \notin S_{\mathcal{OPT},t-1}$, we have $t" < t \leq t-1$.)

We show that phase i contains exactly $l+1$ requests. For a type 1 phase there is nothing to show. Suppose that the phase i is a type 2 phase. Then $t_i^b = t'$. Let $s \in [t', t-1]$ be arbitrary and x be the request at time s. We need to show $x \in L_t$.

Assume $x \notin L_t$. Then by the definition of y, we have $x \notin S_{\mathcal{LRU}_l,t-1}$; i.e. x is evicted by $\mathcal{LRU}_l$ at some time $s' \in [s+1, t-1]$. Since y is not evicted by $\mathcal{LRU}_l$ at time s' and y's most recent request is at time $t" < s$, we must have:

$$\begin{aligned} y &\in L_{s'} \\ &\subseteq \{\sigma_{s'}, \cdots, \sigma_{t-1}, \sigma_t, \cdots, \sigma_{t_i^e}\} \\ &= \{\sigma_{s'}, \cdots, \sigma_{t-1}\} \cup L_t. \end{aligned}$$

But $y \notin \{\sigma_{s'}, \cdots, \sigma_{t-1}\}$ and $y \notin L_t$, by the definition of $t"$ and y. Thus $x \notin L_t$ is impossible. We conclude that the phase i contains exactly $l+1$ distinct requests.

Consider an arbitrary request x such that:

$$x \in S_{\mathcal{LRU}_l,t_{i-1}^e} - (L_{t_i^b} \cup S_{\mathcal{OPT},t_{i-1}^e}).$$

If $\eta_{x,t_i^e} = 0$, then the fact clearly holds. Therefore we assume $\eta_{x,t_i^e} \geq 1$. By our partition, $L_{t_i^b}$ contains all requests which are contained in the phase i. Since $\eta_{x,t_i^e} \geq 1$, we have,

$$x \in S_{\mathcal{LRU}_l,t_i^e} - L_{t_i^e},$$

and hence,

$$\begin{aligned} x &\notin L_{t_i^b} \cup L_{t_i^e} \\ &\supseteq L_s, \end{aligned}$$

for all $s \in [t_i^e, t_i^b]$.

Thus, x is a candidate for eviction by $\mathcal{LRU}_l$ throughout the phase i, but is not evicted. This implies immediately that all requests in the phase i; i.e. all requests in $L_{t_i^b}$, also belong to $\mathcal{LRU}_l, t_i^e$.

Using a very similar analysis we can show that $y \neq x$ and $y \in S_{\mathcal{LRU}_l,t_i^e}$. Hence we have identified $l+2$ requests in $S_{\mathcal{LRU}_l,t_i^e}$ which, at time t_i^e, are requested later than x. At time t_i^e, each of these requests has a cost of 0 or a cost which is greater than that of x. Thus, $\eta_{x,t_i^e} \leq k - l - 2$.

This completes the proof. □

Using the partition of σ generated above, we can evaluate $\mathcal{LRU}_l$'s amortized cost on σ. First we bound the increase in potential $\sum_{t=1}^{m} \Phi_t - \Phi_{t-1}$. Then we estimate $\mathcal{LRU}_l$'s actual cost in each phase of σ.

For $t = 1, 2 \cdots, m$, we define

$$N_t = S_t - S_{t-1},$$

where

$$S_0 = S_{\mathcal{LRU}_l,0} - (M_0 \cup L_0 \cup S_{\mathcal{OPT},0}),$$

and

$$M_0 = L_0 = \emptyset.$$

Lemma 15 *If $x \in N_t$, then, $\eta_{x,t} \leq k - l - 1$.*

Proof. By the definition of N_t, we have,

$$x \in S_{\mathcal{LRU}_l,t} - (M_t \cup L_t \cup S_{\mathcal{OPT},t}).$$

Since $x \notin M_t$, request x is not in the interval $[\mu_t, t]$ and hence,

$$x \in S_{\mathcal{LRU}_l,\mu_t - 1}.$$

We have $x \notin M_t \cup L_t$ which implies $x \notin L_s$, for all s where $\mu_s \leq s \leq t$. Thus, x is a candidate for eviction by $\mathcal{LRU}_l$ throughout the interval $[\mu_t, t]$, but is not evicted. It follows that all requests in M_t must be in $S_{\mathcal{LRU}_l,t}$. Note that M_t contains $l + 1$ requests because $\mathcal{OPT}$ does not move any server before the $(k+1)$-*st* miss. At time t, all requests in M_t have a cost of 0 or a cost which is greater than $w_{s,t}$. Thus, $w_{s,t} \leq k - l - 1$.

This completes the proof. □

Lemma 16 *If $x \in S_{t-1} \cap S_t$, then $\eta_{x,t-1} \geq \eta_{x,t}$. In particular, if $\mathcal{LRU}_l$ incurs a miss at time t, then $\eta_{x,t-1} > \eta_{x,t}$.*

Proof. Note that by the definition of S_{t-1} and S_t, we have,

$$x \in S_{\mathcal{LRU}_l,t-1} - L_{t-1}$$

and

$$x \in S_{\mathcal{LRU}_l,t} - L_t.$$

Hence,

$$\eta_{x,t-1} \geq 1$$

and

$$\eta_{x,t} \geq 1.$$

Let y, where $y \neq x$, be a request such that $\eta_{y,t-1} = 0$ and $\eta_{y,t} > 0$. Then, we have

$$w_{x,t} < w_{y,t}.$$

Otherwise, if $w_{y,t-1} > 0$ and $w_{x,t-1} < w_{y,t-1}$, then, we have

$$w_{y,t} = 0,$$

or

$$w_{x,t} < w_{y,t}.$$

As a result,

$$\eta_{x,t-1} \geq \eta_{x,t}$$

Now suppose that $\mathcal{LRU}_l$ incurs a miss at time t. Then, at t, $\mathcal{LRU}_l$ moves the server at z, where $z \neq x$, whose last request occurred earlier than x's last request.

Hence,

$$1 \leq \eta_{z,t-1} \leq \eta_{x,t-1}.$$

Consequently, x's cost must decrease after the server at z is moved; i.e.

$$\eta_{x,t-1} > \eta_{x,t}.$$

This completes the proof. □

Lemma 15 implies that at any time t where $1 \leq t \leq m$, a request $x \in N_t$ can cause an increase in potential of at most $k - l - 1$.

Thus, for every t, $l \leq t \leq m$, we have:

$$\begin{aligned}
\Phi_t - \Phi_{t-1} &= \sum_{x \in S_t} \eta_{x,t} - \sum_{x \in S_{t-1}} \eta_{x,t-1} \\
&= \sum_{x \in N_t} \eta_{x,t} + \sum_{x \in S_{t-1} \cap S_t} (\eta_{x,t} - \eta_{x,t-1}) - \sum_{x \in S_{t-1} - S_t} \eta_{x,t-1} \\
&= (k - l - 1) \cdot \mid N_t \mid - W_t
\end{aligned}$$

where

$$\begin{aligned} W_t &= \Upsilon_t^1 + \Upsilon_t^2 + \Upsilon_t^3 \\ \Upsilon_t^1 &= \textstyle\sum_{x \in N_t}(k - l - 1 - \eta_{x,t}) \\ \Upsilon_t^2 &= \textstyle\sum_{x \in S_{t-1} \cap S_t}(\eta_{x,t-1} - \eta_{x,t}) \\ \Upsilon_t^3 &= \textstyle\sum_{x \in S_{t-1} - S_t}(\eta_{x,t-1}). \end{aligned}$$

Corollary 1 *For all* $t = 1, 2, \cdots, m$,

$$\begin{aligned} \Upsilon_t^1 &\geq 0 \\ \Upsilon_t^2 &\geq 0 \\ \Upsilon_t^3 &\geq 0. \end{aligned} \tag{4.1}$$

Proof. Clearly, $\Upsilon_t^1 \geq 0$ and $\Upsilon_t^3 \geq 0$. The inequality $\Upsilon_t^2 \geq 0$ follows from Lemma 16.

This completes the proof. □

Next we estimate $\sum_{t=1}^m \mid N_t \mid$ and derive a bound on $\sum_{t=1}^m \Phi_t - \Phi_{t-1}$. To each element $x \in N_t$, we assign the most recent move of the server at x by $\mathcal{OPT}$.

More formally, we let,

$$X = \{(x,t) \in \bigcup_{t=1}^{m} N_t \times [1,m] : x \in N_t\}.$$

We define a function $f : X \to [1,m]$. For $(x,t) \in X$ we define

$$f(x,t) = \max\{s \leq t : \mathcal{OPT} \textit{ moves the server at } x \textit{ at time } s\}.$$

Note that f is well-defined. Now we describe the two properties of the function f.

Property 5 *The function* f *is injective.*

Property 6 *Let* $(x,t) \in X$, $f(x,t) = t'$ *and* $t \in [t_i^b, t_i^e]$, *where* $0 \leq i \leq p$. *If* $i = 0$, *then* $t' \in [t_0^b, t_0^e]$. *If* $i \geq 1$, *then* $t' \in [t_{i-1}^b, t_i^e]$.

These two properties are useful when bounding $\mathcal{LRU}_l$'s actual cost in each phase of σ.

Let $T_{\mathcal{OPT}}$ be the set of all $t \in [1, m]$ such that $\mathcal{OPT}$ moves a server at time t. Let $T^1_{\mathcal{OPT}} = \{f(x,t) : (x,t) \in X\}$. Since f is injective and hence;

$$\sum_{t=1}^{m} \mid N_t \mid = \mid X \mid = \mid T^1_{\mathcal{OPT}} \mid .$$

Thus, by Equation (4.1), we obtain:

$$\sum_{t=1}^{m} \Phi_t - \Phi_{t-1} = (k - l - 1) \mid T^1_{\mathcal{OPT}} \mid - \sum_{t=1}^{m} W_t. \tag{4.2}$$

Now we bound $\mathcal{LRU}_l$'s actual cost in each phase of σ. For $i = 0, 1, \cdots, p$, let $C_{\mathcal{LRU}_l,i}$ be the actual cost $\mathcal{LRU}_l$ incurs in serving the phase i, and let $C_{\mathcal{OPT},i}$ be the cost $\mathcal{OPT}$ incurs in serving the phase i.

Furthermore, let

$$T^2_{\mathcal{OPT}} = T_{\mathcal{OPT}} - T^1_{\mathcal{OPT}}$$

and, for $i = 0, 1, \cdots, p$, let

$$T^2_{\mathcal{OPT},i} = \{t \in T^2_{\mathcal{OPT}} : t^b_i \leq t \leq t^e_i\}.$$

Lemma 17 $C_{\mathcal{LRU}_l,0} = C_{\mathcal{OPT},0}$.

Proof. It follows from the fact that the phase 0 contains at most $l + 1 < k$ distinct requests and that on the first k misses, both $\mathcal{LRU}_l$ and $\mathcal{OPT}$ move their severs to serve those requests.

This completes the proof. □

Suppose $C_{\mathcal{LRU}_i,i} \geq 1$. Let

$$\tilde{C}_i = \mid S_{\mathcal{LRU}_l,t^e_{i-1}} - (L_{t^b_i} \cup S_{\mathcal{OPT},t^e_{i-1}}) \mid .$$

For $x \in N_t$, where $t \in [t^b_i, t^e_i]$, let

$$\Upsilon^1_{x,t} = k - l - 1 - \eta_{x,t}.$$

For $x \in S_{t-1} \cap S_t$, let

$$\Upsilon^2_{x,t} = \eta_{x,t-1} - \eta_{x,t}.$$

For $x \in S_{t-1} - S_t$, let

$$\Upsilon^3_{x,t} = \eta_{x,t-1}.$$

Note that,

$$\begin{aligned} \Upsilon^1_t &= \textstyle\sum_{x \in N_t} \Upsilon^1_{x,t} \\ \Upsilon^2_t &= \textstyle\sum_{x \in S_{t-1} \cap S_t} \Upsilon^2_{x,t} \\ \Upsilon^3_t &= \textstyle\sum_{x \in S_{t-1} - S_t} \Upsilon^3_{x,t}. \end{aligned}$$

Corollary 2

$$\begin{aligned} \Upsilon^1_{x,t} &\geq 0 : \forall x \in N_t \\ \Upsilon^2_{x,t} &\geq 0 : \forall x \in S_{t-1} \cap S_t \\ \Upsilon^3_{x,t} &\geq 1 : \forall x \in S_{t-1} - S_t \end{aligned} \tag{4.3}$$

Proof. The inequality $\Upsilon^1_{x,t} \geq 0$ follows from Lemma 15. Lemma 16 implies $\Upsilon^2_{x,t} \geq 0$. If $x \in S_{t-1} - S_t$, then,

$$x \in S_{\mathcal{LRU}_l, t-1} - L_{t-1}$$

and hence,

$$1 \leq w_{x,t-1} = \Upsilon^3_{x,t}.$$

This completes the proof. □

Lemma 18 $t' \in T^2_{\mathcal{OPT},i-1}$.

Proof. For $t = t^e_{i-1}, t^b_i, t^b_i + 1, \cdots, t^e_i$, let $x \notin S_t$.

Since $x \in S_{\mathcal{LRU}_l, t^e_{i-1}} - (L_{t^b_i} \cup S_{\mathcal{OPT}, t^e_{i-1}})$, we have $x \notin S_{\mathcal{OPT}, t^e_{i-1}}$.

Let $t' = \max\{s \leq t^e_{i-1} : \mathcal{OPT}\ \textit{moves the server at}\ x\ \textit{at time}\ s\}$. By our partition, phase i contains exactly $l+1$ distinct requests and Properties 5 and 6, we can show that $t' \geq t^b_{i-1}$ and $t' \notin T^1_{\mathcal{OPT}}$. Thus, $t' \in T^2_{\mathcal{OPT},i-1}$.

This completes the proof. □

Lemma 19 *$t' \in [t_i^b, t_i^e]$ and $\Upsilon^j \geq 1 : j \in \{1,2,3\}$.*

Proof. Let t_{min} be the smallest $t \in \{t_{i-1}^e, \cdots, t_i^e\}$ such that $s \in S_t$.

If $t_{min} = t_{i-1}^e$, we define $t^\dagger$ to be the time when $\mathcal{LRU}_l$ incurs the first miss during phase i. If $\eta(x, t^\dagger) = 0$, then $x \notin S_{t^\dagger}$. Hence, there must exist a t', where $t_i^b \leq t' \leq t^\dagger$, such that $x \in S_{t'-1} - S_{t'}$. Thus, $\Upsilon_{x,t'}^3 \geq 1$. If $\eta(x, t^\dagger) = 1$, then $x \in S_{t^\dagger - 1} \cap S_{t^\dagger}$. By Lemma 16, we have $\Upsilon_{x,t^\dagger}^2 \geq 1$.

If $t_{min} > t_{i-1}^e$, then $\Upsilon_{x,t_{min}}^1 \geq 1$. Suppose $\eta_{x,t_{min}} = k - l - 1$. By our partition, we have $\eta_{x,t_i^e} \leq k - l - 2$. It is simple to show that there must exist a $t' \in \{t_{min} + 1, \cdots, t_i^e\}$ such that $\Upsilon_{x,t'}^2 \geq 1$ or $\Upsilon_{x,t'}^3 \geq 1$.

This completes the proof. □

Lemma 20

$$\tilde{C}_i \leq \mid T_{\mathcal{OPT},i-1}^2 \mid + \sum_{t=t_i^b}^{t_i^e} W_t.$$

Proof. For each request $x \in S_{\mathcal{LRU}_l, t_{i-1}^e} - (L_i^b \cup S_{\mathcal{OPT}, t_{i-1}^e})$, we claim that either:

1. by Lemma 18, $\exists t' \in T_{\mathcal{OPT},i-1}^2$ such that $\mathcal{OPT}$ moves the server at x at t';

   or

2. by Lemma 19, $\exists t' \in [t_i^b, t_i^e]$ and a $j \in \{1,2,3\}$ such that $\Upsilon_{x,t'}^j \geq 1$.

From Corollary 2, we have,

$$\tilde{C}_i \leq \mid T_{\mathcal{OPT},i-1}^2 \mid + \sum_{t=t_i^b}^{t_i^e} W_t.$$

This completes the proof. □

Lemma 21 *For $i = 1, 2, \cdots, p$,*

$$C_{\mathcal{LRU}_l,i} \leq C_{\mathcal{OPT},i} + \mid T_{\mathcal{OPT},i-1}^2 \mid + \sum_{t=b_i^b}^{t_i^e} W_t.$$

Proof. Consider a fixed $i \in \{1, \cdots, p\}$.

If $C_{\mathcal{LRU}_l,i} = 0$, then the inequality clearly holds because, according to Corollary 1, $W_t \geq 0$ for all $t \in [t_i^b, t_i^e]$.

So suppose $C_{\mathcal{LRU}_l,i} \geq 1$. Let

$$\tilde{C}_i = \mid S_{\mathcal{LRU}_l,t_{i-1}^e} - (L_{t_i^b} \cup S_{\mathcal{OPT},t_{i-1}^e}) \mid .$$

And we have,

$$C_{\mathcal{LRU}_l,i} \leq C_{\mathcal{OPT},i} + \tilde{C}_i.$$

Consider a request $x \in S_{\mathcal{LRU}_l,t_{i-1}^e} - (L_{t_i^b} \cup S_{\mathcal{OPT},t_{i-1}^e})$. By Lemmata 18 and 19, we have

$$C_{\mathcal{LRU}_l,i} \leq C_{\mathcal{OPT},i} + \mid T^2_{\mathcal{OPT},i-1} \mid + \sum_{t=b_i^b}^{t_i^e} W_t.$$

This completes the proof. □

Now we can estimate $\mathcal{LRU}_l$'s amortized cost.

Theorem 9 *$\mathcal{LRU}_l$ is $(k-l)$-competitive.*

Proof. Applying Equations (4.2), Lemmata 17 and 21, we can show:

$$\begin{aligned} C_{\mathcal{LRU}_l,\sigma} + \Phi_m - \Phi_0 &= \sum_{i=0}^{p} C_{\mathcal{LRU}_l,i} + \sum_{t=1}^{m} (\Phi_t - \Phi_{t-1}) \\ &\leq \sum_{i=0}^{p} C_{\mathcal{OPT},i} + \sum_{i=0}^{p-1} \mid T^2_{\mathcal{OPT},i} \mid + \sum_{t=t_1^b}^{m} W_t - \\ &\quad \sum_{t=1}^{m} W_t + (k-l-1) \mid T^1_{\mathcal{OPT}} \mid . \end{aligned}$$

Corollary 1 implies that $W_t \geq 0$ for all $t \in [t_0^b, t_0^e]$.

Hence,

$$\begin{aligned} C_{\mathcal{LRU}_l,\sigma} + \Phi_m - \Phi_0 &\leq C_{\mathcal{OPT},\sigma} + \mid T^2_{\mathcal{OPT}} + (k-l-1) \mid T^1_{\mathcal{OPT}} \\ &\leq (k-l) C_{\mathcal{OPT},\sigma}. \end{aligned}$$

This completes the proof. □

Chapter 5

Space Complexity

5.1 Introduction

This chapter is motivated by the need for fast algorithms for on-line problems. On-line problems arise quite often, and quite naturally in computer science. The dynamic nature of these problems demands that algorithms for these problems make decisions without full knowledge of the impact of these decisions on the future performance of the algorithms. Many real life problems, such as paging and scheduling problems, are on-line problems. The k-server problem [42], [43] is a variation of the generalized formulation of on-line problems. Usually it receives requests from a finite metric space and requires algorithms which need to effectively deal with extremely large number of requests within minuscule time frames. Thus, algorithms for such problem need to provide good solutions in *real time*. We can argue, then, that algorithms that require arbitrarily large unbounded amounts of scratch space; e.g. algorithms that physically store and use information about *all* the past requests to describe how to serve the current request, could not possibly schedule service in real time, thus rendering them impractical for such on-line problems.

From a practical standpoint, an important requirement of an on-line algorithm is that it maintain very little state information (memory) from the past, and that its response to each request be easy to compute. For example, the $\mathcal{LRU}$ caching algorithm must maintain the order of the last access to each line in an associative set, for each associative set. Such state memory is expensive and slow to update in hardware, for associative set size larger that 2, as pointed out by So and Rechtschaffen [50]. For $\mathcal{LRU}$ paging, one would have to maintain the order of the last access to each physical memory frame (since physical memory is fully associative) – this is not feasible. Perhaps an alternative to large state memory is *randomization.* We refer here to algorithms that make probabilistic choices during execution, with their performance studied under worst case inputs. On-line randomized k-server algorithms have attracted scant attention prior to our work [2], [3], [7], [19], [20], [30], [36], [37], [40], [44], [45], [46], [47]. Previous theoretical studies have not touched on this issue of the memory resources required by a deterministic on-line k-server algorithm.

Motivated by the above consideration, we restrict ourselves to the k-server problem on finite metric spaces and focus our attention on the *memoryless* k-server algorithms; i.e. algorithms that do not maintain any book-keeping information about past service, but *only* use the current configuration of servers and the position of the next request to schedule service. Naturally, a memoryless algorithm, if shown to be *strongly competitive*, is a resource optimal, on-line strategy for the problem.

This chapter is organized as follows. In Section 2, we discuss related work. Definitions and notation are stated in Section 3. In Section 4 we give a memoryless 2-server algorithm on a simple, infinite metric space and present the main result with the proof in the subsequent section.

5.2 Related Work

The study of deterministic competitive algorithms has concentrated on competitiveness; i.e. comparing on-line algorithms to optimal off-line algorithms on any sequence of operations. Published algorithms proven to be competitive invariably have pessimal response time; i.e. their worst case operation time is as bad as possible.

The known 2-competitive deterministic algorithms for two servers by Manasse *et al.* [42], [43] and Chrobak and Larmore [14] are very space- and time-consuming. If the given metric space has m points, their the algorithms need $O(m)$ space and $O(m)$ time per each request. Thus on a sequence of n requests in an arbitrary metric space, the space needed is $O(n)$ and the total time is $O(n^2)$. Clearly, no server system can allow such delays in practice.

This has motivated research to find faster competitive algorithms for two servers. Chrobak and Larmore [13], [16] improved those results and presented a 4-competitive 2-server algorithm with time complexity $O(1)$ per each request. This algorithm is also 2-competitive, and thus optimal, when the underlying metric space is a tree.

Recently, Estiville-Castro and Sherk [22] proved that any k-server algorithm on a line segment of length l with $O(\lceil \frac{l}{k} \rceil)$ response time has a competitive ratio of at least $\Omega(\frac{l}{k})$.

5.3 Preliminaries

We consider a 2-server algorithm operating in a simple, infinite metric space: the Euclidean (real) line, $\Re^1$. Define the *signed* Euclidean distance, xy, between any two points, $x, y \in \Re^1$ to be $y - x$. The cost incurred by a 2-server algorithm moving a server from point x to y is the absolute value of xy.

We let $\sigma = \sigma_1, \sigma_2, \cdots, \sigma_n$ denote a finite sequence of requested points. For a

request sequence σ, and for each integer $i > 0$, σ^i denotes the request sequence σ repeated i times. A 2-server algorithm, $\mathcal{A}$, has its servers labeled by s_1 and s_2. At any stage, the servers respectively occupy points x and y with $x \leq y$; we denote this canonically by saying that $\mathcal{A}$ is in the configuration $\{x, y\}$. For any configuration $F = \{x, y\}$, we denote by $\tau(F)$ the quantity $xy \geq 0$. Without loss of generality, we assume that server s_1 never crosses server s_2 at any stage of the algorithm. Also, when there is no confusion, we shall sometimes identify the label of a server with the point in $\Re^1$ that it currently covers.

Suppose that algorithm $\mathcal{A}$ starts at some configuration F and subsequently gets a request sequence σ. Then $\mathcal{A}(F, \sigma)$ is defined as the configuration after $\mathcal{A}$ has served σ; the associated cumulative cost of serving σ is denoted by $C_{\mathcal{A}}(F, \sigma)$. We define $C_{\mathcal{OPT}}(F, \sigma)$ to be the optimal cost of serving the sequence σ starting from the configuration F.

Definition 13 *Let $c \geq 1$. The on-line algorithm $\mathcal{A}$ is said to be α-competitive on $\Re^1$ iff for any starting configuration F, there exists a real constant c such that for any request sequence σ, we have:*

$$C_{\mathcal{X}}(F, \sigma) \leq \alpha \cdot C_{\mathcal{OPT}}(F, \sigma) + c.$$

5.4 The $\mathcal{TWO}$ Algorithm

Intuitively, a deterministic, on-line 2-server algorithm is *memoryless* iff it serves the current request based solely on the position of the request and the current configuration of its two servers. We further restrict our algorithm on $\Re^1$ as follows.

Definition 14 *A memoryless 2-server algorithm $\mathcal{A}$ on the Euclidean space $\Re^1$ is said to be uniform iff there exists real constants β_1 and β_2 such that the following holds. For $x < y$, let $\mathcal{A}$ be in the configuration $\{x, y\}$, and let $r \in \Re^1$ be the*

next requested point which is distinct from x and y. Also, let d_1 be the signed distance moved by the server serving r, let d_2 be the signed distance moved by the other server, and let $\beta_r = \frac{d_2}{d_1}$.

1. *If r lies outside the closed interval $[x, y]$ on $\Re^1$ (i.e. $r < x$ or $r > y$), then $\beta_r = \beta_1$.*

2. *If r lies inside the closed interval $[x, y]$ on $\Re^1$ (i.e. $x < r < y$), then $\beta_r = \beta_2$.*

Note that a uniform strategy does not depend on the location of the points in the current server configuration and the current request, but only on the constants β_1 and β_2. This assumption is quite reasonable, and perhaps desirable: memoryless on-line server algorithms should have succinct descriptions and should adopt strategies unencumbered by any exceptional behavior. The definition of uniformity can be probably generalized to higher dimensional Euclidean metric spaces. To our knowledge, all the memoryless on-line server strategies studied in the literature satisfy Definition 14, when restricted to the space $\Re^1$.

The algorithm is depicted in Figure 5.4.

5.5 Competitive Analysis

Observe that $\mathcal{TWO}$ is a uniform algorithm according to Definition 14, with constants $\beta_1 = 0$ and $\beta_2 = -1$.

In order to bound the amortized cost of our algorithm, we first establish some results that delineate the consequences of 2-competitiveness of a uniform 2-server algorithm. Combining these results we can give a complete proof of our main result.

Our competitiveness results for $\mathcal{TWO}$ are established in the framework of a game between the player, $\mathcal{TWO}$, and an arbitrary adversary, $\mathcal{ADV}$. The

At any stage, let $F = \{x, y\}$ *(with* $x \leq y$*) be a configuration of* $\mathcal{TWO}$ *and let the next point requested be* $r \notin \{x, y\}$*. In response,* $\mathcal{TWO}$ *performs exactly one of the following two actions depending on* r*.*

1. *If* r *is not in the closed interval* $[x, y]$ *on the real line, then move the server nearest to* r *(breaking ties arbitrarily) to serve the request without moving the other server.*

2. *If* r *is in the closed interval* $[x, y]$*, then serve the request with the server closer to* r*, while simultaneously moving the other server towards* r *by the same distance. In other words, the next configuration attained by our algorithm is given by* $\{r, x + y - r\}$ *if* $r - x \leq y - r$ *and is given by* $\{x + y - r, r\}$*, otherwise.*

Figure 5.1: The $\mathcal{TWO}$ Algorithm

adversary has its own two servers labeled as a_1 and a_2; if the adversary's current configuration is $\{x', y'\}$ with $x' \leq y'$, then servers a_1 and a_2 respectively cover the points x' and y'. In general, the game proceeds in rounds; a round begins with the adversary specifying the next request and serving the request with one of its servers. The players completes the round by serving the request with one of its servers. To establish a contradiction to the assumed 2-competitiveness of the player, it suffices to show that from any given starting configuration F, the adversary has a strategy such that on a suitably chosen request sequence σ, the quantity $C_{\mathcal{TWO}}(F, \sigma) - \alpha \cdot C_{\mathcal{ADV}}(F, \sigma)$ is greater than c, where c is the constant depending on F in Definition 13 (with $\alpha = 2$) for the player.

Given a sufficiently small real constant $\epsilon > 0$, we define two configurations F_1 and F_2 to be ϵ-aligned iff F_1 and F_2 have one common point, and the remaining pair of corresponding points are no more than ϵ distance apart. Our first proposition shows that the adversary can always force a α-competitive player $\mathcal{TWO}$ to reach a configuration that is ϵ-aligned with the adversary's configuration. We remark that the claim holds for any metric space, not just $\Re^1$.

Lemma 22 *Suppose that $\mathcal{TWO}$ is a α-competitive 2-server algorithm. Let F be any configuration, let x and y be any two distinct points. Then, for any chosen $\epsilon > 0$, there exists an integer $i \geq 1$ such that the configurations $\{x, y\}$ and $\mathcal{TWO}(F, \{x, y\}^i)$ are ϵ-aligned.*

Proof. Otherwise, we can construct an adversary $\mathcal{ADV}$ that stays in the configuration $\{x, y\}$; each subsequent request will cause $\mathcal{TWO}$ to incur cost greater than ϵ while $\mathcal{ADV}$ incurs no cost. The ratio of $\mathcal{TWO}$'s cost to $\mathcal{ADV}$'s cost thus grows unboundedly, contradicting Definition 13.

This completes the proof. □

Lemma 23 *Let $\alpha > 0$ be a constant and $\mathcal{TWO}$ be a uniform 2-server algorithm over the metric space $\Re^1$. Suppose that there is an adversary $\mathcal{ADV}$ such that for*

any starting configuration $F = \{x, y\}$, there is a constant $\epsilon > 0$ depending on F and a request sequence σ such that the following holds.

1. $\tau(\mathcal{TWO}(F, \sigma)) \geq \tau(F)$.

2. $\mathcal{TWO}(F, \sigma)$ *and* $\mathcal{ADV}(F, \sigma)$ *are* ϵ*-aligned.*

3. $C_{\mathcal{TWO}}(F, \sigma) - \alpha \cdot C_{\mathcal{ADV}}(F, \sigma) > \alpha \cdot \epsilon$.

Then, $\mathcal{TWO}$ is not α-competitive on $\Re^1$.

Proof. Assume that the three conditions in the hypothesis hold, and by way of contradiction, let $\mathcal{TWO}$ satisfy Definition 13 with the constant c. Let $F_0 = F$ and $F_1 = \{x_1, y_1\}$ denote the configuration $\mathcal{TWO}(F_0, \sigma)$. From Condition 1, if $t = \frac{x_1 y_1}{xy}$ then $t \geq 1$. Now, define a similarity transformation Γ on $\Re^1$ given by $\Gamma(z) = t \cdot (z - x) + x_1$. Note that Γ maps the points x and y to the points x_1 and y_1 respectively. Moreover, for any distinct points r, $r' \in \Re^1$, $\Gamma(r)\Gamma(r') = t \cdot rr'$. With a slight abuse of notation, for any sequence σ', we let $\Gamma(\sigma')$ denote the ordered request sequence obtained by applying the transformation Γ to each requested point in σ'.

The adversary $\mathcal{ADV}$ adopts the following strategy. Starting at configuration F_0, it first provides the request sequence $\sigma_0 = \sigma, x_1, y_1$ to $\mathcal{TWO}$. Then, it provides sequences $\sigma_1, \sigma_2, \cdots$, such that $\sigma_i = \Gamma(\sigma_{i-1})$, where $i \geq 1$. The serving policy for $\mathcal{ADV}$ is straightforward. If it uses a server [1] on any request r in σ_0, then for every $i \geq 1$, $\mathcal{ADV}$ uses that same server for the corresponding request, $\Gamma^i(r)$, in the sequence σ_i.

Now, since $\mathcal{TWO}(F_0, \sigma) = \{x_1, y_1\}$, it follows that the player incurs no cost on the last two requests in σ_0, while from Condition 2, we conclude that the adversary can reach configuration $\{x_1, y_2\}$ after processing σ_0, such that

$$C_{\mathcal{ADV}}(F_0, \sigma_0) \leq C_{\mathcal{ADV}}(F_0, \sigma) + \epsilon.$$

[1] It is easy to see that $\mathcal{ADV}$ need only move one server per request.

Therefore, using Condition 3, we have $C_{\mathcal{TWO}}(F_0, \sigma_0) - \alpha \cdot C_{\mathcal{ADV}}(F_0, \sigma_0) = \delta > 0$.

Since $\mathcal{TWO}$ is a uniform algorithm, it follows that the distances moved by servers on each request in $\Gamma(\sigma_0)$ are t times the distances moved by the servers on the corresponding request in σ_0. In general, for $i \geq 0$, let $\sigma^i = \sigma_0, \sigma_1, \cdots, \sigma_i$ denote the cumulative request sequence up to stage i, and let $F_{i+1} = \mathcal{TWO}(F_0, \sigma^i)$, the configuration of $\mathcal{TWO}$ after processing sequence σ^i. From the above discussion, the following equation holds for $i \geq 0$.

$$C_{\mathcal{TWO}}(F_i, \sigma_i) - \alpha \cdot C_{\mathcal{ADV}}(F_i, \sigma_i) = \delta \cdot t^i.$$

From this, we can conclude that for $i \geq 0$, then:

$$C_{\mathcal{TWO}}(F_0, \sigma^i) - \alpha \cdot C_{\mathcal{ADV}}(F_0, \sigma^i) = \delta \cdot \sum_{j=0}^{i} t^j.$$

Since $\delta > 0$ and $t \geq 1$, the right hand side of the above equation grows unboundedly with increasing i and will eventually exceed the constant c. This contradicts the assumption that $\mathcal{TWO}$ is α-competitive.

This completes the proof. □

Without loss of generality, we assume that the player starts from the configuration $\{0, L\}$, for some positive real number L.

Lemma 24 *Let $\Im$ be a 2-competitive algorithm with constants β_1 and β_2. From any starting configuration $\{0, L\}$, if $\Im$ gets a request at some point $r \in \Re^1$, then $\Im$ schedules the server that is closer to r to serve that request at r.*

Proof. Let $\{0, L\}$ be the starting configuration of $\Im$, for some $L > 0$. Since $\Im$'s servers never cross each other, it follows that when $r < 0$ or when $r > L$, the servers closer to r, respectively, s_1 and s_2, serve these requests.

Suppose $r < L - r$; i.e. r is in the open interval $[0, L]$ and is closer to s_1. Proving by contradiction, suppose that $\Im$ schedules the server s_2 to move to r. The adversary chooses the strategy of serving a request with its closer server, and provides the following set of requests.

1. Request r.

2. Choose a positive $\epsilon < \frac{2(L-2r)}{3}$, and request $L + r + \epsilon$.

3. For some finite $i \geq 1$, request the sequence $\underbrace{r, L+r+\epsilon, \cdots, r, L+r+\epsilon}_{i \text{ times}}$ until the player's servers are ϵ-aligned at the configuration $\{r, L+r+\epsilon\}$ by Lemma 22.

Simple calculations show that the player incurs a cost of at least $2L+\epsilon$ while the adversary's cost is exactly $2r + \epsilon$; hence $C_{\Im} - 2C_{\mathcal{ADV}} > 2\epsilon$ on the above sequence. Since the adversary and the player end up ϵ-aligned, it follows that the player's servers must be at least a distance of L apart. This satisfies all three conditions of Lemma 23, and we conclude that $\Im$ is not 2-competitive. Hence, the closer server s_1 must have served the initial request at r.

This completes the proof. □

Lemma 25 *If $\Im$ is 2-competitive, then the constant $\beta_1 = 0$.*

Proof. Let $F = \{0, L\}$ be the initial configuration of $\Im$. Suppose that $\Im$ gets a request at the point $r = 2L$. Arguing by contradiction, when $\beta_1 \neq 0$, $\Im$ moves server s_1 to the point $y = \beta_1 L$, since s_2 must serve that request at r by Lemma 24.

We construct an adversary $\mathcal{ADV}$ that is initially in the configuration $\{0, L\}$. On getting the request at r, $\mathcal{ADV}$ moves to the configuration $\{0, 2L\}$. At this stage, the player's configuration is $\{y, 2L\}$, the absolute distance between s_1 and s_2 is $2L - y$, the player's cost is $L+ \mid y \mid$ and the adversary's cost is exactly L. Now, $\mathcal{ADV}$ chooses a positive constant, ϵ, such that $\epsilon < \frac{3}{10} \cdot \mid y \mid$, and selects one of the following strategies depending on the sign of y.

1. If $y > 0$ then $\mathcal{ADV}$ requests the point $x = L + \frac{y}{2} - \epsilon$ and serves the request with server a_2. Next, $\mathcal{ADV}$ requests the sequence of points $\underbrace{0, x, \cdots, 0, x}_{i \text{ times}}$

for some $i \geq 1$ until the player's servers are ϵ-aligned at the configuration $\{0, x\}$ by Lemma 22.

2. If $y < 0$ then $\mathcal{ADV}$ requests the point $x = L + \frac{y}{2} + \epsilon$ and serves the request with server a_1. Next, $\mathcal{ADV}$ requests the sequence of points $\underbrace{x, 2L \cdots, x, 2L}_{i \text{ times}}$ for some $i \geq 1$ until the player's servers are ϵ-aligned at the configuration $\{x, 2L\}$ by Lemma 22.

When $y > 0$ (respectively, when $y < 0$), it is easy to see that the point x is closer to the player's server s_1 (respectively, s_2.) By Lemma 24, $\Im$ is obliged to move s_1 (respectively, s_2) to serve the request at x when $y > 0$ (respectively, when $y < 0$.) However, the adversary serves x with its server a_2 (respectively, a_1.) In both cases, the final configuration reached by the player's servers is such that the distance between $\Im$'s servers is greater than L. Hence, to apply Lemma 23 and contradict the 2-competitiveness of $\Im$, it only remains to show that $C_{\Im} - 2C_{\mathcal{ADV}} > 2\epsilon$ for the above sequence of requests.

If $y > 0$, careful counting establishes that the net cost to the player before ϵ-alignment at $\{0, x\}$ is at least $4L + \frac{y}{2} - \epsilon$ while the adversary's cost is $2L - \frac{y}{2} + \epsilon$. Hence, $C_{\Im} - 2C_{\mathcal{ADV}} \geq \frac{3y}{2} - 3\epsilon > 2\epsilon$, since $\epsilon < \frac{3y}{10}$.

Likewise, if $y < 0$, then $\Im$'s net cost before ϵ-alignment with $\mathcal{ADV}$ at configuration $\{x, 2L\}$ is at least $4L - \frac{5q}{2} - \epsilon$, while the adversary incurs a cost of $2L + \frac{q}{2} + \epsilon$. Hence, $C_{\Im} - 2C_{\mathcal{ADV}} \geq \frac{-7y}{2} - 3\epsilon > 2\epsilon$, since $\epsilon < \frac{-3y}{10} < \frac{-7y}{10}$.

Applying Lemma 23 yields the desired contradiction that $\Im$ is not 2-competitive. Thus, $\beta_1 = 0$; i.e. the other server does not move at all.

This completes the proof. □

Lemma 26 *If $\Im$ is 2-competitive, then the constant $\beta_2 = -1$.*

Proof. At the outset, we note that Lemma 24 and the fact that servers never cross, together imply that $\beta_2 \geq -1$. Otherwise, in configuration $\{0, L\}$, if the

request is at r such that $\frac{L}{1+|\beta_2|} < r < \frac{L}{2}$, then server s_1 serves, but the servers will cross each other.

For the sake of contradiction, let β_2 be strictly greater than -1. Let $\{0, L\}$ be the initial configuration of $\Im$ and $\mathcal{ADV}$. If $\Im$ gets a request at the point $r > \frac{L}{2}$, it serves r using s_2 (by Lemma 24) but simultaneously moves s_1 to a point $y < L - r$ (since $\beta_2 > -1$). Depending on β_2, $\mathcal{ADV}$ selects between the following two strategies; in both cases, $\mathcal{ADV}$ uses a suitably chosen positive constant $\epsilon < \frac{L}{2}$ whose value is made explicit later in the analysis.

1. If $\beta_2 > 0$, then $\mathcal{ADV}$ chooses a positive constant $\epsilon < \frac{L}{2}$, requests $r = \frac{L}{2} + \epsilon$, and serves r with its farther server a_1. The next request is now placed at the point $x = \frac{3L}{2} + 2\epsilon$, and $\mathcal{ADV}$ serves it with the server a_2. Finally, $\mathcal{ADV}$ requests the sequence $\underbrace{r, x, \cdots, r, x}_{i \text{ times}}$ for some $i \geq 1$ until the player's servers are ϵ-aligned at the configuration $\{r, x\}$ by Lemma 22.

2. If $-1 < \beta_2 \leq 0$, then $\mathcal{ADV}$ chooses another positive constant $\epsilon < \frac{L}{2}$, requests $r = \frac{L}{2} + \epsilon$, and serves r with its farther server a_1, incurring a cost of $\frac{L}{2} + \epsilon$. The next request is to the point $x = 2L + y$, and $\mathcal{ADV}$ serves it with the server a_2. The point, $t = L + y + \epsilon$, is requested next and served by $\mathcal{ADV}$ with its server a_1. Finally, $\mathcal{ADV}$ requests the sequence $\underbrace{x + 2\epsilon, t \cdots, x + 2\epsilon, t}_{i \text{ times}}$ for some $i \geq 1$ until the player's servers are ϵ-aligned at the configuration $\{t, x + 2\epsilon\}$ by Lemma 22. $\mathcal{ADV}$ serves the first request to $x + 2\epsilon$ with its server a_2.

We now analyze each of the above cases and determine appropriate values for the constants chosen above. First, suppose that $\beta_2 > 0$. On the request sequence chosen by $\mathcal{ADV}$, Lemma 25 implies that $\Im$ must serve the second request at x with its server s_2, without moving the server s_1. Hence, the total cost of $\Im$ is at least $2L + \epsilon + 2 \cdot \mid y \mid$ while $\mathcal{ADV}$ incurs a cost of $L + 3\epsilon$. Since $y = \beta_2(\epsilon - \frac{L}{2})$ by uniformity, the adversary's choice of a constant $\epsilon < \frac{\beta_2 L}{7+2\beta_2} < \frac{L}{2}$ forces ϵ to be

less than $\frac{2}{7} \mid y \mid$. This ensures that $C_{\Im} - 2C_{\mathcal{ADV}} > 2\epsilon$; applying Lemma 23 yields the desired contradiction to the 2-competitiveness of $\Im$.

Otherwise, suppose that $-1 < \beta_2 \leq 0$. By Lemmata 24 and 25, $\Im$ uses the server s_2 for providing service for each of the first three requests to r, x and t respectively. Counting net cost, we can see that the cost to $\Im$ will be at least $5L + 2y - \epsilon$ while the adversary incurs cost equal to $2L + 2y + 3\epsilon$. Recall that by uniformity, $y = \beta_2(\epsilon - \frac{L}{2})$. Hence, by choosing a constant $\epsilon < \frac{(1+\beta_2)\cdot L}{9+2\beta_2} < \frac{L}{2}$, the adversary can ensure that $\epsilon < \frac{L-2y}{9}$. Consequently, $C_{\Im} - 2C_{\mathcal{ADV}} > 2\epsilon$, and the contradiction follows from the conditions of Lemma 23.

This completes the proof. □

Theorem 10 *TWO is the only uniform, 2-competitive algorithm with 2 servers on* $\Re^1$.

Proof. Lemmata 25 and 26 together establish the theorem. □

5.6 Remarks

It is not clear how one could extend the uniformity assumption to arbitrary spaces. We believe, therefore, that it is very unlikely that the k-server conjecture will be resolved by a uniform algorithm. In other words, a general k-competitive on-line k-server algorithm will probably need to track at least *some* past history.

Chapter 6

Conclusions

6.1 Summary of Our Results

It is only recently that people have looked at problems from the on-line perspective. The concept itself is attractive since a large number of problems in real life have to dealt with in an on-line fashion. In this thesis we considered the k-server problem and its variants. It is the single problem that has spurred the most interest in this field. Previous attacks on this and other on-line problems, say the metrical task systems [9], [10], involved a potential function, a numerical invariant that enables the inductive proof. Our technique is based on more complex invariants, which provide valuable information about the structure of the reachable work functions. As a result, we prove an upper bound of $2k - 1$ which is the best we can prove at this point, although we conjecture that the true worst case behavior of the work function algorithm is k, like [17]. In the Chapter 3 we deal with a generalization of the k-server problem, in which the servers are unequal. Each of the servers is assigned a positive weight. For any metric space, we extend our Work Function Algorithm to give a nearly optimal upper bound. In order to study the amount of improvement that can be achieved if the future is partially known, we introduce a new model of lookahead

for our k-server algorithm. We show that strong lookahead has practical as well as theoretical importance and significantly improves the competitiveness of our k-server algorithm. This is the first model of lookahead having such properties. Finally, we study a sub-class of memoryless server algorithms with 2 servers on the real line and show that there is a unique algorithm in this sub-class whose competitive factor is best possible on the real line.

6.2 Recent Results

As with most research fields, survival of the field depends on finding interesting problems. Therefore, we present a short survey of results related to the subject matter of this thesis that were obtained recently.

6.2.1 The Adversary Models

One possible drawback of the current focus may be that the models of adversaries that have been considered do not model real life situations with any deal of accuracy. This could be because in many cases in practice some correlation exists in the input data. With this in mind, there have been suggestions to consider other kinds of adversaries; for instance, an adversary who gets to draw requests according to any probability distribution on the inputs, but has to decide beforehand what the distribution is. It could be worthwhile exploring what kinds of adversaries model real life situations. It is also conceivable that the correct set of restrictions on the adversary may be problem-dependent. One problem that has been seen perhaps the most attention of this kind has been the paging problem. The paging problem is a special case of the k-server problem, where all the distances in the metric space are one. This problem was first studied by Sleator and Tarjan in this framework [49]. Recent works by Borodin *et al.* [8]; Irani *et al.* [33] and Karlin *et al.* [39] consider this problem with

restrictions on the adversary's power.

The Sleator and Tarjan's result [49] conflict with practical experience on paging in at least two ways. First, $\mathcal{FIFO}$ and $\mathcal{LRU}$ have the same competitiveness, even though in practice $\mathcal{LRU}$ usually outperforms $\mathcal{FIFO}$. Secondly, $\mathcal{LRU}$ usually incurs much less than k times the optimal number of faults, even though its competitiveness is k. The reason for the practical success of $\mathcal{LRU}$ has long been known: *most programs exhibit locality of reference* [1], [21], [48].

Motivated by these observations, Borodin *et al.* [8] proposed a technique for incorporating locality of reference into the traditional Sleator and Tarjan's framework. Their notion of an *access graph* [1] limits the set of request sequences the adversary is allowed to make.

Theorem 11 ([8]) *On any undirected access graph G, $C_{\mathcal{LRU}}(G,k) \leq C_{\mathcal{FIFO}}(G,k)$, where k stands for the number of pages of fast memory.*

Irani *et al.* [33] extended Borodin *et al.*'s result and gave an algorithm that is strongly competitive on *directed access graphs.*

Based on Karlin *et al.*'s findings, Markov paging offers a better theoretical abstraction for locality of reference in real programs. Unlike those models mentioned above, there is no adversary generating the reference string, much as in a real program.. Further, certain simple properties of real programs, such as the fact that a data-dependent loop typically gets executed many times before exiting, can be modeled well. Therefore, they gave an on-line paging algorithm that is efficiently computable and achieves a fault rate within a constant times the best possible, on every Markov chain.

[1] An access graph for a program is a graph that has a vertex for each page that the program can reference. Locality of reference is imposed by the edge relation - the pages that can be referenced after a page p are just neighbors of p in the graph or p itself.

6.2.2 On-line Performance-Improvement Algorithms

For many non-trivial on-line problems, even the best possible on-line algorithm has a large competitive ratio. Sometimes, lookahead does not always help improve their competitiveness. Thus a new challenge in the on-line problems is to design an algorithm that on each application not only is competitive, but also acquires useful information that helps improve future competitiveness.

That is, if the algorithm obtains some partial information about the input (e.g. future page requests, or obstacles that lie ahead in the scene), it may able to improve its competitiveness. For instance such partial information may be available when the algorithm is repeatedly applied to the same problem instance, and on each application it can only see a portion of the input. This gives rise to the possibility that on each application, the algorithm can accumulate information, and use it to improve its competitiveness on future applications. Blum and Chalasani [5], [6] consider the design of on-line algorithms whose performance provably converges quickly to that of the optimal off-line algorithm upon repeated application to a problem. In particular, they developed such algorithms for some natural problems in robot navigation and paging.

Suppose a program repeatedly generates the *same* sequence of page requests. This may happen for instance when the program is inside a loop. A natural question to ask is: *Can one design a paging algorithm whose performance improves as the request sequence is repeated more times?* This is non-trivial in many real computer systems, where the pager is sleepy; i.e. it only "wakes up" when there is a page fault, then decides which page to evict from fast memory, then goes back to sleep. Thus, one each iteration the pager only knows the requests that resulted in page faults, so it only has partial information about the request sequence. There is thus the possibility that the pager can accumulated information about the request sequence and improve its performance as the request sequence is repeated more times.

For the special case of this problem whose k pages are in fast memory (or cache) and there are only $k+1$ pages total, they gave an algorithm for a sleepy pager which has the following behavior when the same page request sequence σ is repeated. (They assumed that on each iteration the pager knows the time relative to the start of the request sequence.)

Theorem 12 ([6]) *For every t, the average t-iteration competitive ratio is at most $42 \cdot \frac{k}{t}\lceil \lg t \rceil$ when $t \leq k$, and at most $42 \cdot \lceil \lg k \rceil$ when $t > k$.*

Appendix A

Proof of Lemma 1

Lemma 1 *If there exists a bijection h that satisfies the conditions in the above definition then there exists a bijection h^* that satisfies the same conditions and $h^*(x) = x$ for all $x \in A \cap B$.*

Proof. Let h be a bijection from A to B that satisfies the conditions of the definition above and it maps the maximum number of elements in $A \cap B$ to themselves. Assume that for some $a \in A \cap B$ we have $h(a) \neq a$. Define a bijection h' that agrees with h everywhere except that,

$$h'(a) = a \qquad h'(h^{-1}(a)) = h(a).$$

Consider now a bipartition of A into X and Y and assume without lost of generality that $h^{-1}(a) \in X$. If $a \in X$ then $h(X) = h'(X)$ and $h(Y) = h'(Y)$ and Inequality (2.1) holds. Otherwise, if $a \notin X$ then let $X' = X + a$ and $Y' = Y - a$ and we have,

$$\begin{aligned} w(A) + w(B) &\geq w(X' \cup h(Y')) + w(h(X') \cup Y') \\ &= w(X \cup h'(Y)) + w(h'(X) \cup Y) \end{aligned}$$

Thus, h' satisfies the convexity condition. Notice h' maps at least one more element in $A \cap B$ to itself than h. This contradicts the assumption that h maps

the maximum number of elements in $A \cap B$ to themselves. So, we can conclude that $h(a) = a$ for all $a \in A \cap B$, proving the lemma. □

Bibliography

[1] L A A Belady. "A study of replacement algorithms for virtual storage computers". *IBM Systems Journal*, 5:78–101, 1966.

[2] S Ben-David, A Borodin, R Karp, G Tardos, and A Widgerson. "On the power of randomization in on-line algorithms". In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pages 379–386, 1990.

[3] S Ben-David, A Borodin, R Karp, G Tardos, and A Widgerson. "On the power of randomization in on-line algorithms". *Algorithmica*, 11:2–14, 1994.

[4] P Berman, H Karloff, and G Tardos. "A competitive 3-server algorithm". In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, 1990.

[5] A Blum and P Chalasani. "Learning switching concept". In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, 1992.

[6] A Blum and P Chalasani. "An on-line algorithm for improving performance in navigation". In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–11, 1993.

[7] A Blum, H Karloff, Y Rabani, and M Saks. "A decomposition theorem and bounds for randomized server problems". In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 197–207, 1992.

[8] A Borodin, S Irani, P Raghavan, and B Schieber. "Competitive paging with locality of reference". In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 249–259, 1991.

[9] A Borodin, N Linial, and M Saks. "An optimal on-line algorithm algorithm for metrical task systems". In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 373–382, 1987.

[10] A Borodin, N Linial, and M Saks. "An optimal on-line algorithm algorithm for metrical task systems". *Journal of the ACM*, 39:745–763, 1991.

[11] M Chrobak, H Karloff, T Payne, and S Vishwanathan. "New results on server problems". In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, 1990.

[12] M Chrobak, H Karloff, T Payne, and S Vishwanathan. "New results on server problems". *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991.

[13] M Chrobak and L L Larmore. "On fast algorithm for two servers". In *Proceedings of the Mathematical Foundations of Computer Science*, 1990.

[14] M Chrobak and L L Larmore. "A new approach to the server problem". *SIAM Journal of Discrete Mathematics*, 00:0–0, 1991.

[15] M Chrobak and L L Larmore. "An optimal on-line algorithm for the server problem on trees". *SIAM Journal on Computing*, 20:144–148, 1991.

[16] M Chrobak and L L Larmore. "On fast algorithms for two servers". *Journal of Algorithms*, 12:607–614, 1991.

[17] M Chrobak and L L Larmore. "The server problem and on-line games". In *Proceedings of the DIMACS Workshop on On-line Algorithms*, pages 11–64, 1991.

[18] F K Chung, R Graham, and M E Saks. "A dynamic location problem for graphs". *Combinatorica*, 9:111–131, 1989.

[19] D Coppersmith, P G Doyle, P Raghavan, and M Snir. "Random walks on weighted graphs and applications to on-line algorithms". In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pages 369–378, 1990.

[20] D Coppersmith, P G Doyle, P Raghavan, and M Snir. "Random walks on weighted graphs and applications to on-line algorithms". *Journal of the ACM*, 40:421–453, 1993.

[21] P J Denning. "Working sets past and present". *IEEE Transactions on Software Engineering*, 6:64–84, 1980.

[22] V Estivill-Castro and M Sherk. "Competitiveness and response time in on-line algorithms". In *Proceedings of the 2nd International Symposium on Algorithms*, pages 284–293, 1991.

[23] A Fiat, R M Karp, M Luby, L A McGeoch, D D Sleator, and N E Young. "Competitive paging algorithms". *Journal of Algorithms*, 12:685–699, 1991.

[24] A Fiat, Y Rabani, and Y Ravid. "Competitive k-server algorithms". In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 454–463, 1990.

[25] A Fiat, Y Rabani, and Y Ravid. "Competitive k-server algorithms". *Journal of Computer and System Sciences*, 48:410–428, 1994.

[26] A Fiat, Y Rabani, Y Ravid, and B Schieber. "A deterministic $O(k^3)$-competitive k-server algorithm for the circle". *Algorithmica*, 11:572–578, 1994.

[27] A Fiat and M Ricklin. "Competitive algorithms for the weighted server problem". In *Proceedings of the 3rd Israeli Symposium on Theory and Computing Systems*, pages 294–303, 1993.

[28] A Fiat and M Ricklin. "Competitive algorithms for the weighted server problem". *Theoretical Computer Science*, 130:85–89, 1994.

[29] R L Graham. "Bounds for certain multiprocessing anomalies". *Bell System Technical Journal*, 45:1563–1581, 1966.

[30] E Grove. "The harmonic on-line k-server algorithm is competitive". In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 260–266, 1991.

[31] M M Halldorsson and M Szegedy. "Lower bounds for on-line graph coloring". In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, 1992.

[32] S Irani. "Coloring inductive graphs on-line". In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 470–479, 1990.

[33] S Irani, A R Karlin, and S Phillips. "Strongly competitive algorithms for paging with locality of reference". In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, 1992.

[34] S Irani and R Rubinfeld. "A competitive 2-server algorithm". *Information Processing Letters*, 39:85–91, 1991.

[35] M Y Kao and S R Tate. "On-line matching with blocked input". *Information Processing Letters*, 38:113–116, 1991.

[36] A R Karlin, M S Manasse, L A McGeoch, and S Owicki. "Competitive randomized algorithms for non-uniform problems". In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 301–309, 1990.

[37] A R Karlin, M S Manasse, L A McGeoch, and S Owicki. "Competitive randomized algorithms for non-uniform problems". *Algorithmica*, 11:542–571, 1994.

[38] A R Karlin, M S Manasse, L Rudolph, and D D Sleator. "Competitive snoopy caching". In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 244–254, 1986.

[39] A R Karlin, S Phillips, and P Raghavan. "Markov paging". In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 208–217, 1992.

[40] R M Karp. "A randomized $(n+1)k$-competitive algorithm on the graph". Personal communication, 1989.

[41] N Lee. The k-server problem with distinguished servers. Master's thesis, University of California, Berkeley, 1991.

[42] M S Manasse, L A McGeoch, and D D Sleator. "Competitive algorithms for on-line problems". In *Proceedings of the 20th ACM Symposium on Theory of Computing*, pages 322–333, 1988.

[43] M S Manasse, L A McGeoch, and D D Sleator. "Competitive algorithms for server problems". *Journal of Algorithms*, 11:208–230, 1990.

[44] L A McGeoch and D D Sleator. "A strongly competitive randomized paging algorithm". *Algorithmica*, 6:816–825, 1991.

[45] P Raghavan. "Lecture notes on randomized algorithms". Technical Report RC15840, IBM, Yorkdown Heights, 1990.

[46] P Raghavan and M Snir. "Memory versus randomization in on-line algorithms". In *Proceedings of the 16th International Colloquium on Automata, Programming and Languages*, pages 687–703, 1989.

[47] P Raghavan and M Snir. "Memory versus randomization in on-line algorithms". Technical report, IBM, Yorkdown Heights, 1990.

[48] G S Shedler and C Tung. "Locality in page reference strings". *SIAM Journal on Computing*, 1:218–241, 1972.

[49] D D Sleator and R E Tarjan. "Amortized efficiency of list update and paging rules". *Communications of the ACM*, 28:202–208, 1985.

[50] K So and R N Rechtschaffen. "Cache operations by MRU change". *IEEE Transactions on Computer System*, 37:700–709, 1988.

[51] J R Spirn. *Program Behavior: Models and Measurements.* Elsevier: New York, 1977.

[52] G Turpin. Recent work on the server problem. Master's thesis, University of Toronto, 1989.

[53] N Young. "On-line caching as caches size varies". In *Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms*, 1991.