On-Line Algorithms and the k-Server Conjecture

Elias Koutsoupias
Department of Computer Science and Engineering
University of California, San Diego

June 1994

Abstract

The Work Function Algorithm, a natural algorithm for the k-server problem, is shown to have competitive ratio at most $2k - 1$ for all metric spaces. It is also shown that the k-server conjecture, which states that there is an on-line algorithm for the k-server problem with competitive ratio k, holds for all metric spaces with $k + 2$ points.

Furthermore, two refinements of competitive analysis are proposed and studied: diffuse adversaries and comparative analysis. They address successfully some of the drawbacks of competitive analysis.

Chapter 1

Introduction

1.1 On-Line Algorithms

The development of classical optimization theory has been based on the assumption that all input data are available to the algorithm. For many real life problems, however, input data become available piece by piece as the computation progresses.

Consider for example the paging problem, one of the classical problems in operating system design: The memory of a typical computer system consists of the fast memory divided into a small number of equal size parts called pages, and the secondary memory, which is much larger but slower. The central processing unit can access only the fast memory, which usually contains copies of certain pages of the secondary memory. Since only data in the fast memory can be accessed, a decision policy is needed to determine which set of pages to maintain in the fast memory each time. In particular, a *page replacement policy* is needed to decide which page to evict from the fast memory to make room for a requested page to be moved in. From the point of view of conventional optimization theory, the input to an algorithm for the paging problem is a sequence of page requests and the goal is to minimize the number of page faults—the number of pages removed from the fast memory. A simple greedy algorithm is optimal for this problem: Postpone the next page fault as long as possible by removing the page that is the last from the pages in fast memory to be requested again. The problem with this algorithm is that it bases its decision on the sequence of future page requests. In reality, the sequence of future page requests is not known and it cannot be computed from the past. Therefore, any algorithm for the paging problem has to base its decisions only on the already revealed sequence of page requests.

The paging problem is a characteristic *on-line problem*: An optimization problem where not all relevant input data are available but are revealed as the computation progresses. In general, an on-line problem is a tuple (I, O, c), where I is a (possibly infinite) set of *input symbols*, O is a set of *output symbols* and $c : \{I^n \times O^{n+1} : n \geq 0\} \mapsto \Re^+$ is a *cost function*. An *on-line algorithm* A for a (I, O, c) problem is a function $A : I^* \mapsto O$. Given the sequence $x = (x_1, x_2, \ldots, x_n)$ of input symbols, A produces a sequence $y = (y_0, y_1, y_2, \ldots, y_n)$ of output symbols, with

$$y_j = A(x_1, x_2, \ldots, x_j),$$

that is, A produces an output symbol for each new input symbol. The cost of A on input x, $\mathrm{cost}_{\mathrm{A}}(x)$, is just $c(x, y)$. For the paging problem with a fast memory of k pages, I is the set of all pages and O is the set of all collections of k pages; that is, y_j is the set of pages in the fast memory in step j. The cost function c is the cost of moving pages in the fast memory as indicated by Y: In step j the cost is proportional to the symmetric difference of y_{j-1} and y_j provided that y_j contains the last requested page x_j; this is the number of pages we have to move in order to change the state of the fast memory from y_{j-1} to y_j (for a typical paging algorithm it is one). Since the page x_j must be moved into the fast memory, we want y_j to contain x_j and this is accomplished by setting the cost of step j arbitrarily high when this is not the case. The cost $c(x, y)$ is the sum of the costs for all steps. Notice that there is an output y_0 that embodies the initial situation; in the paging problem it is the initial set of pages in the fast memory.

For each on-line problem there is an associated *off-line* problem, where the output y_j of an off-line algorithm B can now depend on "future" inputs:

$$y_j = B(x_1, x_2, \ldots, x_n).$$

The optimal (off-line) cost, denoted by $\mathrm{opt}(x)$, is the minimum cost[1] achieved by an off-line algorithm on input x.

[1]At this point, we can avoid a diversion into a fruitless discussion of computability of the optimal solution by assuming that there always exists a computable optimal output, which is the case for all known on-line problems.

In classical combinatorial optimization, there is always an algorithm that can compute an optimal solution for all inputs and the challenge is to do it efficiently. In contrast, in on-line problems the scarce resource is *information*, not computational power. As a result, the study of on-line algorithms so far has primarily focused on the quality of the solution and not on the efficiency of the algorithm. On the other hand, for a fixed input there is an especially tailored on-line algorithm that produces an optimal output. This suggests that there is no 'best' on-line algorithm—one that performs on all inputs as well as any other on-line algorithm. Nevertheless, the development of an algorithmic theory for on-line problems can only be based on a concrete measure of the quality of on-line algorithms.

A novel measure proposed by Sleator and Tarjan [35] is the *competitive ratio* of an algorithm: The worst case ratio of the cost achieved by the algorithm divided by the optimal cost. In particular, the competitive ratio of an algorithm A is

$$R(A) = \max_{x} \frac{cost_A(x)}{\mathrm{opt}(x)} \tag{1.1}$$

where x ranges over all inputs. We usually allow a constant independent of the input x to be added to the cost in order to remove the dependency on the initial output symbol. An algorithm is called *R-competitive* or simply *competitive* if it has a finite competitive ratio R. The competitive ratio of an on-line problem is the best (infimum) competitive ratio achieved by an on-line algorithm. Finally, the study of on-line problems using as a measure the competitive ratio is termed *competitive analysis*.

Competitive analysis has been applied successfully to many natural on-line problems. Furthermore, it served as a unifying measure for studying general properties of on-line problems. General on-line problems, which contain many natural problems as special cases, have been identified and studied. One of them, the *metrical task systems* proposed in [7], generalizes all data management and memory management on-line problems such as the paging problem. A task system services a sequence of arriving input symbols, called *tasks*; the cost of servicing a task depends on the internal state of the system and the type of the task. A metrical task system is a task system with the property that the cost of changing internal states satisfies the triangle inequality: changing from state A to state C costs no more than changing from A to B and from B to C. In the case of the paging problem, the internal states of the system are all possible sets of pages in the fast memory; it is easy to see that the cost of changing internal state satisfies the triangle inequality. A metrical task system is characterized by a metric space (a set of internal states with the cost of changing states) and a set of possible tasks with their cost. When all tasks are allowed, the competitive ratio is $2n-1$, where n is the number of internal states [7, 8].

1.2 The k-Server Problem

An important special case of metrical task systems is the k-server problem, a generalization of the paging problem [28, 29]. The k-server conjecture which states that there is an algorithm for the k-server problem with competitive ratio k has been the most outstanding open problem in the study of on-line algorithms. This conjecture is the main subject of the next chapter.

The k-server problem is defined on a metric space $\mathcal{M}$, which is a (possibly infinite) set of points with a distance function d (nonnegative real function) that satisfies the triangle inequality. In particular, for all points x, y, and z:

$$\begin{aligned} d(x,x) &= 0 \\ d(x,y) &= d(y,x) \\ d(x,y) &\leq d(x,z)+d(z,y) \end{aligned}$$

On the points of the metric space $\mathcal{M}$, k servers reside that can move from point to point. A possible position of the k servers is called a *configuration*; that is, a configuration is a multiset of k points of $\mathcal{M}$. We use capital letters for configurations; we also use $D(X,Y)$ for the minimum distance to move the servers from configuration X to configuration Y. We always assume that the k servers are initially at a fixed configuration A_0. For a set X and a point a we use $X+a$ for $X \cup \{a\}$ and $X-a$ for $X-\{a\}$. We also use $C(X)$ for the sum of all distances of points in X.

A *request sequence* ρ is simply a sequence of points of the metric space $\mathcal{M}$ to be serviced by the k servers; servicing a request entails moving some server to the point of request. In particular, if $\rho = r_1 r_2 \ldots r_n$ is a request sequence, then the k servers service ρ by passing through configurations A_0, A_1, A_2, ..., A_n with $r_j \in A_j$. At step j, the cost of servicing request r_j is merely the cost of moving the k servers from A_{j-1} to A_j; this is equal to

$D(A_{j-1}, A_j)$. The cost, $\mathrm{cost}(\rho)$, of servicing ρ is the sum of the cost for all steps.

Since an on-line algorithm cannot base its decisions on future requests, A_j can depend only on A_0 and the subsequence of requests $r_1 r_2 \ldots r_j$. On the other hand, an off-line algorithm knows the whole request sequence in advance and consequently in this case A_j depends on A_0 and $r_1 r_2 \ldots r_n$. If $\mathrm{opt}(\rho)$ denotes the optimal off-line cost then the competitive ratio of an on-line algorithm is simply

$$\max_{\rho} \frac{\mathrm{cost}(\rho) + c}{\mathrm{opt}(\rho)}$$

where the constant c may depend on the initial position A_0 but not on the request sequence ρ.

The paging problem is the special case of the k-server problem when all distances between different points (pages) are the same. The reason is that the cost for bringing a page into the fast memory is the same for all pages.

In metric spaces $\mathcal{M}$ with k or fewer points an on-line algorithm can initially cover all points with its servers; it never again moves them and therefore, its competitive ratio is 1. The problem becomes interesting for metric spaces with at least $k+1$ points. In [28], it was shown that no on-line algorithm can have competitive ratio less that k and the following conjecture was posed:

Conjecture 1 (The k-Server Conjecture) *For any metric space there is an on-line algorithm with competitive ratio k.*

It was also showed that the conjecture holds for two special cases: when $k = 2$ and when the metric space has exactly $k + 1$ points. The special case of paging had already been shown k-competitive in [35]. The k-server conjecture attracted a lot of interest because of its simplicity, its elegance and its importance in the study of on-line problems.

For some time, it was open whether *any finite* competitive ratio at all is possible. It was shown in [19] that indeed there is an algorithm with finite competitive ratio for all metric spaces. Unfortunately, the competitive ratio increases exponentially with k. This was improved somewhat in [21], where it was shown that a natural memoryless randomized algorithm, the *Harmonic Algorithm*, has a competitive ratio $O(k2^k)$. Using the derandomization technique of [2], this establishes that the deterministic competitive ratio is $O(k^2 4^k)$. The result of [21] was improved in [3] to $O(2^k \log k)$, establishing a deterministic competitive ratio of $O(4^k \log^2 k)$, which was the best known competitive ratio for the general case before this work. Specifically for the 3-server problem, the best result was an 11-competitive algorithm for any metric space [15].

The lack of significant progress towards the k-server conjecture led to the study of special cases of the problem. One of the first results in this area [4], which preceded the work of [19, 21], was a proof that the Harmonic Algorithm for 3 servers is competitive (although with a terribly high competitive ratio). Attacking the problem in special metric spaces led to a k-competitive algorithm for the line [11], which was extended to trees [12]. Finally, an $O(k^3)$ competitive deterministic algorithm for the circle was presented in [20].

One of the problems with competitive algorithms for the k-server problem is that they are not space-efficient. In order to address this problem, [17] considered memoryless randomized algorithms and showed a competitive ratio of k for the class of resistive metric spaces. By derandomization, this results in a $O(k^2)$ deterministic competitive ratio for resistive or approximately resistive metric spaces (one of them is the circle). Especially for the 2-server problem, [23] gave a 10-competitive space-efficient deterministic algorithm and [13] showed that the Harmonic Algorithm is 3-competitive. We should also mention a series [5, 24, 26] of lower bound results for the randomized version of the k-server problem against an *oblivious adversary* and the absence of any interesting upper bound (or even a candidate algorithm).

Although the k-server conjecture remains unsettled, we come very close in proving it in Chapter 2 (Theorem 1) (see also [27]).

1.3 Critique of Competitive Analysis

Although the importance of competitive analysis cannot be reasonably denied (an algorithmic theory of decision-making under uncertainty is of obvious practical relevance and significance), certain aspects of its basic premises, modeling assumptions, and results have been widely criticized with respect to their realism and relation to computational practice. We think that now is a good time to revisit some of the most often-voiced criticisms of competitive analysis and to propose and explore some better-motivated alternatives.

In competitive analysis, the performance of an on-line algorithm is compared against the performance of an all-powerful off-line algorithm, which is called '*adversary*' in the literature. The definition of the competitive ratio is both the weakness and the strength of competitive analysis. It is a strength because the setting is clear, the problems are crisp and sometimes deep, and the results often elegant and striking. But it is a weakness for several reasons. First, in the face of the devastating comparison against an all-powerful off-line algorithm, a wide range of on-line algorithms (good, bad, and mediocre) fare equally badly; the competitive ratio is thus not very informative and it fails to discriminate and to suggest good approaches. Another aspect of the same problem is that, since a worst-case input decides the performance of the algorithm, the optimal algorithms are often unnatural and impractical, and the bounds too pessimistic to be informative in practice. Even enhancing the capabilities of the on-line algorithm in obviously desirable ways (such as a limited lookahead capability) brings no improvement to the ratio. The main argument for competitive analysis over the classical approach is that *the distribution is usually not known*. However, competitive analysis takes this argument way too far: It assumes that *absolutely nothing* is known about the distribution, that *any distribution* of the inputs is in principle possible; the worst-case "distribution" prevailing in competitive analysis is, of course, a worst-case input with probability one. Such complete powerlessness seems unrealistic to both the practitioner (we always know, or can learn, *something* about the distribution of the inputs) and the theoretician of another persuasion (the absence of a prior distribution, or some information about it, seems very unrealistic to a probabilist or mathematical economist).

The paging problem, perhaps the most simple, fundamental, and practically important on-line problem, is a good illustration of all these points. An unreasonably wide range of deterministic algorithms (both the good in practice LRU and the empirically mediocre FIFO) have the same worst-case competitive ratio—k, the amount of available memory.

In traditional competitive analysis, the all-powerful adversary frustrates not only interesting algorithms, but also powerful *information regimes*. The classical example is again from paging: In paging the best competitive ratio of any on-line algorithm is k. But what if we have an on-line algorithm *with a lookahead of ℓ steps*, that is, an algorithm that knows the immediate future? It is easy to see that any such algorithm must fare equally badly as algorithms without lookahead. In proof, consider a worst case request sequence $abdc\cdots$ and take its $(\ell+1)$-*stuttered version*, $a^{\ell+1}b^{\ell+1}d^{\ell+1}c^{\ell+1}\cdots$ Once more, the absolute power of the adversary blurs practically important distinctions. Still, lookahead is obviously a valuable feature of paging algorithms. How can we use competitive analysis to evaluate its power? Notice that this is not a question about the effectiveness of a single algorithm, but about *classes of algorithms*, about the power of *information regimes*—ultimately, about the value of information [31].

Admittedly, there have been several interesting variants of the framework of competitive analysis that were at least partially successful in addressing some of these concerns. Randomized paging algorithms have usually more realistic performance [18, 30, 34]. Some alternative approaches to evaluating on-line algorithms were proposed in [1, 33] for the general case and in [6, 22, 25, 36] specifically for the paging problem.

In Chapter 3, we propose and study two refinements of competitive analysis which seem to go a long way towards addressing the concerns expressed above. Perhaps more importantly, we show that these ideas give rise to interesting algorithmic and analytical problems (which we have only begun to solve in this thesis).

Chapter 2

The k-Server Problem

2.1 Introduction

In this chapter we prove a $2k-1$ upper bound for the competitive ratio of the k-server problem. The algorithm we employ is the *Work Function Algorithm*, a rather natural idea for this problem that was first made explicit in the work of [14] and it has been successfully applied to other problems [9, 10, 16]. In [14], it was shown that the Work Function Algorithm is 2-competitive for $k = 2$. One of the ingredients of our technique is the notion of the *extended cost*, a concept very similar to the *pseudocost* of [14].

Previous attacks on this and other on-line problems 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. There are two invariants that proved crucial: A *quasiconvexity* property of the work function (Lemma 2), and a *duality* condition (Lemma 5). Actually, quasiconvexity is used only in the proof of duality, and the main result (Theorem 1) follows from a potential function and the duality condition.

2.2 The Work Function Algorithm

Consider an optimal off-line algorithm B servicing a request sequence $\rho = \rho_1\rho_2$. After servicing the request sequence ρ_1 the k servers of algorithm B occupy some position X. The cost of servicing ρ can be divided into two parts: the cost of servicing the request sequence ρ_1 starting at the initial configuration and ending up at X and the cost of servicing ρ_2 starting at X. An on-line algorithm A that knows algorithm B cannot know the position X, because X may depend on the future request sequence ρ_2. However, algorithm A can compute the cost of servicing ρ_1 of any possible optimal off-line algorithm. In particular, algorithm A can compute the optimal cost of servicing ρ_1 starting at A_0 and ending up at configuration Y, for every possible configuration Y. This leads to the following definition:

Definition 1 (Work function) *Fix a metric space $\mathcal{M}$ and an initial configuration A_0. For a request sequence ρ define the work function w_ρ from configurations to the positive real numbers: $w_\rho(X)$ is the optimal cost of servicing ρ starting at A_0 and ending up at configuration X.*

We usually omit the subscript ρ from w_ρ. Furthermore, for a work function $w = w_\rho$ we refer to $w' = w_{\rho r}$ as the resulting work function after request r, when ρ and r are understood from the context.

Intuitively, the importance of work functions stems from the almost obvious fact that they encapsulate all the useful information about the past; what an on-line algorithm needs to remember is w_ρ, not ρ, because any other algorithm can be transformed to one with this property without deteriorating its competitiveness.

The initial work function $w_e(X)$ of a configuration X is merely the cost of moving the servers from the initial configuration A_0 to the configuration X: $w_e(X) = D(A_0, X)$.

The value $w_{\rho r}(X)$ for some configuration X can be computed as follows: Clearly, if $r \in X$ then $w_{\rho r}(X) = w_\rho(X)$. Otherwise, if $r \notin X$, some server moved from r to some point $x \in X$ and therefore $w_{\rho r}(X) = w_{\rho r}(X - x + r) + d(r, x) = w_\rho(X - x + r) + d(r, x)$. Combining the two cases, we get:

Fact 1 *Let w be a work function; then the resulting work function w' after request r is*

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

We also get:

Fact 2 *If w is a work function and r is the last request, then for all configurations X*

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

Recall that in the definition of $w(X)$ we require that servers end up at configuration X; this can be done by moving first to configuration Y and then to X. So we have:

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 request r. By Fact 3 we get

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

which translates to:

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

Consider a request sequence ρ and let $w = w_\rho$. Let A be the configuration of some on-line algorithm after servicing ρ. Presumably, the most natural on-line algorithm for the k-server problem is the *Greedy Algorithm*, which moves the closest server to a request, that is, it moves its servers to a new configuration A', with $r \in A'$, that minimizes $D(A, A')$. It is easy to see that the Greedy Algorithm, being too conservative, has no bounded competitive ratio. At the other end of the spectrum lies the *Retrospective Algorithm*: it moves its servers to a configuration A', with $r \in A'$, that minimizes $w'(A')$. The idea is that the off-line algorithm that has its servers at A' seems the best so far. Unfortunately, the Retrospective Algorithm, contrary to the Greedy Algorithm, is too reckless. It appears that a combination of these two algorithms may be a good idea; the Work Function Algorithm combines the virtues of both of them:

Definition 2 (Work Function Algorithm) *Let ρ be a request sequence and let A be the configuration of an on-line algorithm after servicing ρ. The work function algorithm services a new request r by moving its servers to a configuration A', with $r \in A'$, that minimizes $w_{\rho r}(A') + D(A, A')$.*

Notice that since $r \in A'$ we can replace $w'(A')$ with $w(A')$ in the above definition. Moreover, because of the triangle inequality we can assume that $A' = A - a + r$ for some $a \in A$; $A' = A - a + r$ minimizes $w(A') + D(A, A')$. Using this we see that $w'(A) = \min_{x \in A}\{w(A - x + r) + d(x, r)\} = w(A') + d(a, r)$.

The cost of the Work Function Algorithm to service request r is simply $d(a, r)$. We need also to estimate the cost of an optimal off-line algorithm. Instead, we 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, since in the worst case the final configuration of the on-line algorithm is the same with the final configuration of the optimal off-line algorithm; if this is not the case, by extending the request sequence with requests in the final configuration of the off-line algorithm, the off-line cost remains unaffected while the on-line cost increases. Consider now the sum of the off-line pseudocost and the on-line cost:

$$w'(A') - w(A) + d(a, r)$$

which is equal to $w'(A) - w(A)$. This quantity is bounded by its maximum over all possible configurations. Therefore, the off-line pseudocost plus the on-line cost is bounded above by

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

We call this quantity the *extended cost* of a move. The total extended cost is the sum of the extended cost of each move. We say that the extended cost *occurs* on a configuration A when A maximizes the quantity in the extended cost.

Clearly, by the definition of the competitive ratio we have:

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

The extended cost is an overestimation of the actual on-line cost (plus the optimal off-line cost). It was first introduced in [14] in a somehow different form (they called it *on-line pseudocost*). The advantage of using extended cost instead of real cost is that we don't have to deal at all with the configuration of the on-line servers. Instead, in order to prove that the Work

Function Algorithm is competitive we have only to show that a certain inequality holds for all work functions. Its disadvantage, of course, is that it may overestimate the cost of the Work Function Algorithm (although in view of the main result, Theorem 1, of this chapter, the overestimation factor is less than 2).

2.3 Quasiconvexity and Duality

Facts 2 and 3 provide some properties of the work functions. Unfortunately, other functions can satisfy both of them; that is, there are functions that satisfy them and are different from w_ρ for all request sequences ρ (and for all initial configurations A_0). In order to study the behavior of the Work Function Algorithm, it is important to understand the properties of work functions. One very useful property is that all work functions are *quasiconvex*:

Definition 3 *A function w is called quasiconvex if for all configurations A, B there exists a bijection $h : A \to B$ such that for all bipartitions of A into X, Y:*

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

Before we show that all work functions are quasiconvex, we need the following lemma, which provides a stronger form of the quasiconvexity condition by restricting the set of possible bijections.

Lemma 1 *If there exists a bijection h that satisfies the conditions in Definition 3 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 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

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

(h' interchanges the values of h on a and $h^{-1}(a)$).

Consider now a bipartition of A into X and Y and assume (without loss of generality) that $h^{-1}(a) \in X$. If $a \in X$ then $h(X) = h'(X)$ and $h(Y) = h'(Y)$ and (2.1) holds. Otherwise, when $a \notin X$, we reduce the quasiconvexity condition of $X' = X + a$ and $Y' = Y - a$ to the quasiconvexity condition of X and Y as follows:

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

Therefore, h' satisfies the quasiconvexity condition. Because h' maps at least one more element in $A \cap B$ to itself than h, it contradicts the assumption that h maps the maximum number of elements in $A \cap B$ to themselves.

We conclude that $h(a) = a$ for all $a \in A \cap B$, and the lemma holds. □

We are now in a position to show the following important lemma:

Lemma 2 (Quasiconvexity Lemma) *All work functions are quasiconvex.*

Proof. We use induction on the number of requests.

Recall that the initial work function $w_e(X)$ of a configuration X is equal to $D(A_0, X)$, where A_0 is the initial configuration. So we have that

$$w(A) + w(B) = D(A_0, A) + D(A_0, B)$$

Fix two minimum matchings $D(A_0, A)$ and $D(A_0, B)$. Each point x_j in A_0 is matched to some 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.

For the induction step, assume that w is quasiconvex. We want to show that the resulting w' after request r is also quasiconvex.

Fix two configurations A and B. Using Fact 1 to express w' in terms of w we get that $w'(A) = w(A - a + r) + d(r, a)$ for some $a \in A$; similarly $w'(B) = w(B - b + r) + d(r, b)$ for some $b \in B$. The induction hypothesis is that w is quasiconvex, so there exists a bijection h from $A - a + r$ to $B - b + r$ that satisfies the quasiconvexity condition. Furthermore, Lemma 1 allows us to assume that $h(r) = r$.

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

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

We will show that h' satisfies the requirements of the quasiconvexity condition of w'. Consider a bipartition of A into X and Y and without loss of generality assume that $a \in X$. We have:

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

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

Now we use the quasiconvexity condition to prove the next two lemmata. In fact, we use the weaker condition:

$$\begin{aligned}
& \forall a \in A - B : w(A) + w(B) \geq \\
& \quad \min_{b \in B} \{ w(A - a + b) + w(B - b + a) \}
\end{aligned}$$

We need a definition first:

Definition 4 *A configuration A is called minimizer of a point a with respect to w, if $a \notin A$ and A minimizes the expression $w(X) - \sum_{x \in X} d(a, x)$, that is*

$$w(A) - \sum_{x \in A} d(a, x) = \min_X \{ w(X) - \sum_{x \in X} d(a, x) \}$$

Lemma 3 *Let w be a 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, $r \notin B$:

$$w'(B) - \sum_{b \in B} d(r,b) \geq w'(A) - \sum_{a \in A} d(r,a)$$

Using Fact 1, we get

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

or equivalently, for all $b' \in B$:

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

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

$$w(B - b' + r) + w(A) + d(r,b') - \sum_{b \in B} d(r,b) \geq \\ \min_{a' \in A}\{w(A - a' + r) + w(A) + d(r,a') - \sum_{a \in A} d(r,a)\} \tag{2.2}$$

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

$$w(A) - \sum_{a \in A} d(r,a) \leq w(B + a' - b') - \sum_{b \in B + a' - b'} d(r,b)$$

or equivalently

$$w(A) + d(r,a') - \sum_{a \in A} d(r,a) \leq w(B + a' - b') + d(r,b') - \sum_{b \in B} d(r,b)$$

From this and the quasiconvexity condition:

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

we get (2.2). □

The following lemma has the same premises with Lemma 3, but a different conclusion:

Lemma 4 *Let w be a 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 the extended cost occurs at A, that is*

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

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

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

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

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

or equivalently, for all $a' \in A$:

$$w(A - a' + r) + d(r, a') + w(B) \geq \min_{b' \in B} \{w(B - b' + r) + d(r, b') + w(A)\} \tag{2.3}$$

From the hypotheses we get that

$$w(A) - \sum_{a \in A} d(r, a) \leq w(A - a' + b') - \sum_{a \in A - a' + b'} d(r, a)$$

and by simplifying it

$$w(A) + d(r, b') \leq w(A - a' + b') + d(r, a')$$

Substituting this in (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 of the quasiconvexity of w. □

Lemmata 3 and 4 can be combined into the following result which characterizes where the extended cost occurs.

Lemma 5 (Duality Lemma) *Let w be a 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 extended cost of servicing the request r occurs on A.*

We call this the "duality lemma" because it relates a maximum (extended cost) to a minimum (minimizer).

2.4 A Potential for $(2k-1)$-Competitiveness

We are now ready for the last act of the proof, the definition of an appropriate potential. For configurations $U = \{u_1, \ldots, u_k\}$ and $B_i = \{b_{i1}, \ldots, b_{ik}\}$, $i = 1, \ldots, k$, let

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

Let $\Phi(w)$ denote its minimum value over all configurations U and B_i, $i = 1, \ldots, k$; $\Phi(w)$ is called the *potential* of the work function w[1].

The next two lemmata provide some properties of $\Phi(w)$.

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

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

$$w(U) = w(U - u_i + r) + d(r, u_i)$$

If we substitute this to $\Psi(w, U, B_1, \ldots, B_k)$, using the k triangle inequalities $d(r, u_i) - d(u_i, b_{ij}) \geq -d(r, b_{ij})$ 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 $r \in U - u_i + r$. □

The next lemma estimates the potential of the initial work function.

Lemma 7 *For the initial work function $w_e(X) = D(A_0, X)$:*

$$\Phi(w_e) = -2C(A_0)$$

Proof. It is not hard to see that the lemma follows if the minimum value $\Phi(w_e)$ of $\Psi(w, U, B_1, \ldots, B_k)$ is achieved when $U = A_0$ and $B_j = A_0$ for $j = 1, \ldots, k$. Consider a point $u_i \in U$. In the minimum matching $D(A_0, U)$, u_i is matched to some point $a \in A_0$. By using the k triangle inequalities

[1] Our potential differs from what is usually termed as "potential function" in the literature of on-line problems by a constant multiple of the optimal off-line cost.

$d(u_i, b_{ij}) \leq d(a, u_i) + d(a, b_{ij})$ we see that we can replace u_i with a without increasing the value of $\Psi(w, U, B_1, \ldots, B_k)$. Therefore, the minimum $\Phi(w_e)$ of $\Psi(w, U, B_1, \ldots, B_k)$ is achieved for $U = A_0$. Similarly, we can show that $B_i = A_0$ for $i = 1, \ldots, k$ and the lemma follows. $\square$

We are now ready to prove the main result of this chapter:

Theorem 1 *The competitive ratio of the Work Function Algorithm is at most* $(2k-1)$.

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

According to Lemma 6, the minimum value $\Phi(w')$ 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_{jl}, 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. From the definition of $\Phi(w)$ we get that $\Phi(w) \leq \Psi_w$. Obviously then,

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

Consider now the expression $\Psi_{w'} - \Psi_w$. All distances appearing in the definition of $\Psi_{w'}$ appear also in the definition of Ψ_w, because they are defined on the same set of configurations U, B_j, $j = 1, \ldots, k$. Therefore they cancel out. By Fact 4, $w'(U) \geq w(U)$ and $w'(B_j) \geq w(B_j)$, $j = 1, \ldots, k$. From this we get:

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

Putting these together:

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

According to Lemma 5, the extended cost is $w'(A) - w(A)$, because A is a minimizer of r with respect to w. Thus, we conclude that the extended cost to service request r is bounded above by $\Phi(w') - \Phi(w)$. Summing over all moves we get that the total extended cost is bounded above by $\Phi(w_\rho) - \Phi(w_e)$, where w_e and w_ρ are the initial and the final work functions, respectively.

Let A_0 and A_n be the initial and final configurations. We have

$$\begin{aligned}\Phi(w_f) &\leq \Psi(w_f, A_n, A_n, \ldots, A_n)\\ &= 2kw_f(A_n) - 2C(A_n)\\ &\leq 2kw_f(A_n)\end{aligned}$$

The value of $\Phi(w_e)$ is given by Lemma 7, $\Phi(w_e) = -2C(A_0)$. Therefore, the extended cost is at most $2kw_\rho(A_n) + 2C(A_0)$. Because the off-line cost is $w_\rho(A_n)$, the total extended 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. □

2.5 The Case of $k+2$ Points

The proof of Theorem 1 has the following skeleton: Let $\mathcal{L}$ be a set of pairs of hypergraphs and graphs on the points of the metric space; that is, $\mathcal{L}$ contains elements (H, G), where hypergraph H is a multiset of ℓ configurations (or ordered k-tuples); graph G is a collection of edges (unordered pairs of points). In Theorem 1, H is a collection of $2k$ configurations: U (k times) and B_j, $j = 1, \ldots, k$; G contains the edges $[u_j, b_{jl}]$. Consider now the function

$$\Psi(w, H, G) = \sum_{X \in H} w(X) - \sum_{[a,b] \in G} d(a, b) \tag{2.4}$$

and let the potential $\Phi(w)$ denote its minimum value over all pairs $(H, G) \in \mathcal{L}$.

As usual, let w be a work function and let w' be the resulting work function after request r. We want to choose Ψ so that it has the following important property:

> $\Psi(w', H, G)$ achieves its minimum value $\Phi(w')$, when one of the configurations in H is a minimizer of r with respect to w.

If this holds and if furthermore $\Phi(w_e)$ is bounded, then the Work Function Algorithm has competitive ratio $\ell - 1$ by an argument similar to that of Theorem 1. In this section we use the same schema to prove the k-server conjecture when the metric space has exactly $k+2$ points.

Theorem 2 *The k-server conjecture holds for metric spaces with $k+2$ points.*

Proof. Let T be a tree on the points of a metric space $\mathcal{M}$ with $k+2$ points. For an edge X of T let $X^{\mathcal{C}}$ denote the complement of X (set of points not in X). Let H_T be the set of complements of all edges in T—since the metric space has $k+2$ points the complement of an edge is a configuration—and let G_T be the set of all edges (pairs of points) not in T. Define

$$\mathcal{L} = \{(H_T, G_T) : T \text{ is a spanning tree}\}$$

Following the general technique described above, we define $\Psi(w, H_T, G_T)$ as in (2.4):

$$\Psi(w, H_T, G_T) = \sum_{X \in T} w(X^{\mathcal{C}}) - \sum_{[a,b] \in G_T} d(a,b)$$

Since a spanning tree of all points of $\mathcal{M}$ has $k+1$ edges, there are $k+1$ configurations in each H_T. So, in order to prove the theorem, it suffices to show that $\Psi(w', H_T, G_T)$ achieves its minimum value $\Phi(w')$ (over all spanning trees), when one of the configurations in H_T is a minimizer of r with respect to w.

Let V be the set of all points of the metric space $\mathcal{M}$. We add the sum of all distances, $C(V)$, to Ψ:

$$\begin{aligned}
\Psi(w', H_T, G_T) + C(V) &= \sum_{X \in T} w'(X^{\mathcal{C}}) + \sum_{[a,b] \notin G_T} d(a,b) \\
&= \sum_{X \in T} w'(X^{\mathcal{C}}) + \sum_{[a,b] \in T} d(a,b) \\
&= \sum_{X \in T} (w'(X^{\mathcal{C}}) + C(X))
\end{aligned}$$

Clearly, the minimum value of $\Psi(w', H_T, G_T)$ is achieved when T is a minimum spanning tree of the graph with nodes V and weights $w'(X^{\mathcal{C}}) + C(X)$ on each edge X.

Let $A^{\mathcal{C}}$ be a minimizer of r with respect to w; by Lemma 5, it is also a minimizer of r with respect to w'. All we need to show is that some minimum spanning tree (with weights as above) contains the edge A.

For this we will need an important property of minimum spanning trees: for each node a and a minimum weight edge X adjacent from a, there is a minimum spanning tree that contains X [32, Chapter 12]. In particular, edge A has weight

$$w'(A^{\mathcal{C}}) + C(A)$$

We can rewrite it as

$$w'(A^{\mathcal{C}}) - \sum_{a \in A^{\mathcal{C}}} d(r,a) + \sum_{a \in V} d(r,a)$$

Recall that $A^{\mathcal{C}}$ is a minimizer of r with respect to w' and notice that $\sum_{a \in V} d(r,a)$ is independent of A. From these, we can conclude that A is a minimum weight edge adjacent from r. Therefore, there is a minimum spanning tree that contains A; the theorem follows. $\square$

We feel that the technique used in the solution of this important special case may point the way to the ultimate proof of the k-server conjecture. In fact, it was the proof of Theorem 2 that led us to Theorem 1.

Chapter 3

Beyond Competitive Analysis

3.1 Introduction

In this chapter, we propose and study two refinements of competitive analysis. The first refinement, *the diffuse adversary model*, removes the assumption that we know nothing about the distribution—without resorting to the equally unrealistic classical assumption that we know all about it. *We assume that the actual distribution D of the inputs is a member of a known class Δ of possible distributions.* That is, we seek to determine, for a given class of distributions Δ, the performance ratio

$$R(\Delta) = \min_{A} \max_{D \in \Delta} \frac{\mathcal{E}_D(A(x))}{\mathcal{E}_D(\mathrm{opt}(x))} \tag{3.1}$$

That is, the adversary picks a distribution D among those in Δ, so that the expected, under D, performance of the algorithm and the off-line optimum algorithm are as far apart as possible. Notice that, if Δ is the class of all possible distributions, (3.1) reduces to the definition of the traditional competitive ratio, since the worst possible distribution is the one that assigns probability one to the worst-case input, and probability zero everywhere else. Hence the diffuse adversary model is indeed a refinement of competitive analysis.

In the paging problem, for example, the input distribution specifies, for each page a and sequence of page requests ρ, $\mathrm{prob}(a|\rho)$ —the probability that the next page fault is a, given that the sequence so far is ρ. It is unlikely that an operating system knows this distribution precisely. On the other hand, it seems unrealistic to assume that any distribution at all is possible. For example, suppose that the next page request *is not predictable with absolute certainty:* $\mathrm{prob}(a|\rho) \leq \epsilon$, for all a and ρ, where ϵ is a real number between 0 and 1 capturing the inherent uncertainty of the request sequence. This is a simple, natural, and quite well-motivated assumption; call the class of distributions obeying this inequality Δ_ϵ. An immediate question is, what is the resulting competitive ratio $R(\Delta_\epsilon)$?

As it turns out, the answer is quite interesting (Section 2). If k is the storage capacity, the ratio $R(\Delta_\epsilon)$ is shown to coincide with the expected cost of a simple random walk on a directed graph with $O((k+\frac{1}{\epsilon})^{k-1})$ nodes. For $k=2$ this value is easy to estimate: It is between $1+\sqrt{\epsilon}/2$ and $1+2\sqrt{\epsilon}$; for larger values of k we do not have a closed-form solution for the ratio. There are two important byproducts of this analysis: First, extending the work of Chapter 2, we completely characterize the work functions of this special case of the k-server problem. Second, the optimum on-line algorithm is *robust*—that is, the same for all ϵ's—and turns out to be a familiar algorithm that is very good in practice: LRU. It is very interesting that LRU emerges naturally from the analysis of this problem as an optimal algorithm (other algorithms may also be optimal).

The second refinement of competitive analysis that we are proposing is *comparative analysis*. Comparative analysis can be used to evaluate information regimes: Suppose that $\mathcal{A}$ and $\mathcal{B}$ are classes of algorithms—typically but not necessarily $\mathcal{A} \subseteq \mathcal{B}$; that is, $\mathcal{B}$ is usually a broader class of algorithms, a more powerful information regime. The *comparative ratio* $R(\mathcal{A},\mathcal{B})$ is defined as follows:

$$R(\mathcal{A},\mathcal{B}) = \max_{B\in\mathcal{B}} \min_{A\in\mathcal{A}} \max_{x} \frac{A(x)}{B(x)} \tag{3.2}$$

This definition is best understood in terms of a game-theoretic interpretation: $\mathcal{B}$ wants to demonstrate to $\mathcal{A}$ that it is a much more powerful class of algorithms. To this end, $\mathcal{B}$ proposes an algorithm B among its own. In response, $\mathcal{A}$ comes up with an algorithm A. Then $\mathcal{B}$ chooses an input x. Finally, $\mathcal{A}$ pays $\mathcal{B}$ the ratio $\frac{A(x)}{B(x)}$. The larger this ratio, the more powerful $\mathcal{B}$ is in comparison to $\mathcal{A}$. Notice that, if we let $\mathcal{A}$ be the class of on-line algorithms and $\mathcal{B}$ the class of all algorithms—on-line or off-line—then (3.2) becomes the definition of the competitive ratio. Hence, comparative analysis is also a refinement of competitive analysis.

In Section 3 we use comparative analysis to evaluate the power of look-ahead in on-line problems. Specifically, we answer the question of the power of look-ahead for metrical task systems of [7, 8]: If $\mathcal{L}_\ell$ is the class of all algorithms with lookahead ℓ, and $\mathcal{L}_0$ is the class of on-line algorithms, then we show that,

$$R(\mathcal{L}_0, \mathcal{L}_\ell) = 2\ell + 1,$$

(that is, the ratio is at most $2\ell + 1$ for all metrical task systems, and it is exactly $2\ell + 1$ for some), while in the more restricted context of paging

$$R(\mathcal{L}_0, \mathcal{L}_\ell) = \ell + 1.$$

3.2 Diffuse Adversaries

The competitive ratio for a diffuse adversary is given in equation (3.1). Δ is a class of acceptable conditional probability distributions; each $D \in \Delta$ is the distribution of the relevant part of the world conditioned on the currently available information. In the case of the paging problem with a set of pages M, Δ may be any set of functions of the form $D : M^* \times M \mapsto [0, 1]$, where for all $\rho \in M^*$ $\sum_{a \in M} D(a|\rho) = 1$. In the game-theoretic interpretation, as the sequence of requests ρ develops, the adversary chooses the values of $D(a|\rho)$ from those available in Δ to maximize the ratio. Since we deal with deterministic algorithms, the adversary knows precisely the past decisions of A, but the adversary's choices may be severely constrained by Δ.

In this section we shall focus on the class of distributions Δ_ϵ, which contains all distributions $D : M^* \times M \mapsto [0, \epsilon]$—that is to say, all conditional distributions with no value exceeding ϵ.

We will treat the paging problem as a k-server problem with a uniform metric space. We will make use of the convenient assumption that the metric space has an infinite number of points. We will also assume that $1/\epsilon$ is an integer.

Definition 5 *The support $S(w)$ of a work function w is the set of configurations X such that there is no Y with $w(X) = w(Y) + D(X, Y)$. Intuitively, the values of w on its support completely determine w.*

If X is not in the support and $w(X) = w(Y) + D(X, Y)$ then it is better for an off-line algorithm to be at configuration Y than at configuration X, because it can simulate the behavior of an off-line algorithm at configuration X by first moving to it (with no extra cost other than $D(X, Y)$). From now on, we will consider only off-line algorithms that always occupy configurations in the support of the current work function.

The following lemmata, specific for the paging problem and not true in general for the k-server problem, characterize all possible work functions by characterizing the values of each on its support. The first lemma states that all values in the support are the same, and hence what matters is the support itself, not the values of w on it.

Lemma 8 *The support of a work function w contains only configurations on which w achieves its minimum value.*

Proof. Suppose not. Then there is a configuration A in the support such that $w(A) > \min_X w(X)$. Let B be a configuration with $w(B) < w(A)$ and $D(A, B)$ minimum. By the quasiconvexity condition, there is an $a \in A - B$ and $b \in B - A$ such that

$$w(A) + w(B) \geq w(A - a + b) + w(B - b + a).$$

Because $w(A) > w(B$, either $w(A) > w(A - a + b)$ or $w(A) > w(B - b + a)$. In the first case $w(A) = w(A - a + b) + 1$, so A is not in the support. In the second case $B - b + a$ contradicts the choice of B with minimum $D(A, B)$. □

An immediate consequence of Lemma 8 is that the off-line cost is always equal to the minimum value of a work function. The next lemma specifies the structure of the support.

Lemma 9 *For each work function w there is an increasing sequence of sets $P_1 \subset P_2 \subset \ldots \subset P_k$, with $P_1 = \{r\}$, the last request, such that the support of w is precisely*

$$S(w) = \{X : |X \cap P_j| \geq j \text{ for all } j\}$$

Note that the converse, not needed in the sequel, also holds: Any such tower of P_j's defines a work function.

Proof. The proof is by induction on the length of the request sequence. Initially take $P_j = \{a_1, \ldots, a_j\}$, where $\{a_1, \ldots, a_k\}$ is the initial configuration.

Suppose that we have the P_j's for w, and let w' be the resulting work function after request r. Then the corresponding sets for w' are, if $r \in P_t$:

$$\begin{aligned} P'_1 &= \{r\}; \\ P'_i &= P_{i-1} + r, \qquad 2 \leq i \leq t; \\ P'_i &= P_i, \qquad t < i. \end{aligned}$$

If r does not belong to any of the P_t's, then

$$\begin{aligned} P'_1 &= \{r\}; \\ P'_i &= P_i + r, \qquad 1 < i. \end{aligned}$$

The induction step now follows. □

Definition 6 *The signature $P(w) = (P_1, P_2, \ldots, P_k)$ of a work function w is the sequence in Lemma 9, and its type is the k-tuple $p(w) = (|P_1| = 1, |P_2|, \ldots, |P_k|)$. Moreover, for two types $p(w_1) = (p^1_1, p^1_2, \ldots, p^1_k)$ and $p(w_2) = (p^2_1, p^2_2, \ldots, p^2_k)$ we use $p(w_1) \preceq p(w_2)$ iff $p^1_j \leq p^2_j$ for all $j = 1, \ldots, k$.*

Consider a work function w and let w' be the resulting work function after request r. If $r \in P_k(w)$ then the support of w' consists of the configurations in the support of w that contain r. In particular, we have that $S(w') \subset S(w)$ and $p(w') \preceq p(w)$. In contrast, if $r \notin P_k(w)$ then $p(w) \preceq p(w')$. Notice also that in this case we have

$$\min_X w'(X) = \min_X w(X) + 1$$

and for each $X \in S(w')$ there is an $Y \in S(w)$ that differs in one point. That is

$$\max_{X \in S(w')} \min_{Y \in S(w)} D(X, Y) = 1$$

We want to find an optimal on-line algorithm and the worst distribution for this algorithm. So, let A be an on-line algorithm. After servicing a request sequence the servers of A are at a configuration A_ρ and the work function is w_ρ. We want to estimate the *disadvantage* $\psi(w_\rho, A_\rho)$ of the position A_ρ with respect to the work function w_ρ. So, if A is an optimal on-line algorithm with competitive ratio R then the expected future on-line cost in the worst case is approximately R times the expected future off-line cost. Their difference expresses exactly the disadvantage of A_ρ with respect to w_ρ.

Definition 7 *The disadvantage of a configuration A_0 with respect to a work function w is*

$$\psi(w, A_0) = \min_{A} \max_{D \in \Delta_\epsilon} \mathcal{E}_{x \in D}(\mathrm{cost}_A(x, A_0) - R \cdot \mathrm{opt}(x, w))$$

where $\mathrm{cost}_A(x, A_0)$ *is the cost of an on-line algorithm A to service the request sequence x starting at configuration A_0 and* $\mathrm{opt}(x, w)$ *is the related optimal cost.*

The disadvantage is what in literature is called potential function; in fact, it is the minimum potential function. It is clear that the algorithm A that achieves the minimum in $\psi(w, A_0)$ is optimal and the associated distribution $D \in \Delta_\epsilon$ is the worst distribution for A. So, we can use ψ as a guide to determine an optimal algorithm and the worst distribution for it. It seems very difficult to determine ψ, but fortunately, we don't need to do it; it suffices to use some general properties of ψ. The next lemma describes a very useful property of ψ.

Lemma 10 *Let w_1, w_2 be work functions and A_0 be a configuration. We have*

$$\psi(w_1, A_0) \leq \psi(w_2, A_0) + R \cdot \max_{X \in S(w_2)} \min_{Y \in S(w_1)} D(X, Y)$$

Proof. Notice first that the on-line cost $\mathrm{cost}_A(x, A_0)$ in the definition of ψ is independent of the work function; only the off-line cost $\mathrm{opt}(x, w)$ depends on the work function. An off-line algorithm that starts with work function w_1 and has its servers at a configuration Y in the support of w_1 can first move to a configuration X in the support of w_2 and then simulate any off-line algorithm that starts with w_2. The difference in their cost is just the cost of the initial step, which is at most

$$\max_{X \in S(w_2)} \min_{Y \in S(w_1)} D(X, Y)$$

and the lemma follows. □

Notice that when $S(w_1) \subset S(w_2)$ then Lemma 10 gives that $\psi(w_1, A_0) \leq \psi(w_2, A_0)$. For simplicity, we may sometimes treat the inequality in the lemma as a strict inequality.

We are now ready to describe some properties of the optimal on-line algorithm and the worst distribution in Definition 7. The first observation is that if w is a work function with signature $P(w) = (P_1, P_2, \ldots, P_k)$ then an optimal on-line algorithm has all its servers in the set P_k. The reason is that if this is not the case, future distributions that assign non-zero probability to points outside P_k can avoid the points in A_0 (the assumption that there are infinite points is very useful here); so, the servers at these points are useless. Let us call an on-line algorithm that keeps its servers in the current P_k *reasonable*. The next lemma is about the worst distribution against any reasonable algorithm.

Lemma 11 *For any reasonable on-line algorithm A there is a worst distribution that assigns probability zero to the points covered by the servers of A.*

Proof. Let w be a work function with signature $P = (P_1, P_2, \ldots, P_k)$ and let A_0 be the configuration of the on-line servers. Assume that the next request r belongs to A_0 and let w' be the resulting work function. So, the on-line cost to service r is zero. Since A is reasonable the request r belongs to P_k and hence, the off-line cost is zero, i.e. the minimum value of w' is the same with the minimum value of w. So, a request $r \in A_0$ does not affect the on-line or the off-line cost. It does affect the support though: the support of w' is a subset of the support of w, and by Lemma 10 this gives some advantage to the on-line algorithm at A_0. $\square$

From now on, we will consider only distributions that assign probability zero to points occupied by on-line servers. For these distributions the on-line cost at each step is one. Clearly, the worst distribution is the one that keeps the off-line cost small.

Assume for the moment that the on-line algorithm can move to any configuration with no extra cost. This assumption is useful because it removes from consideration the actual configuration of the on-line servers. In particular, it allows us to compare work functions by comparing their types (and not their signatures): Lemma 10 applied in this context gives that the worst distribution should favor a work function w_1 to w_2 if $p(w_2) \preceq p(w_1)$.

Let w be the current work function with signature $P = (P_1, P_2, \ldots, P_k)$. Consider an ordering $a_1, a_2, \ldots$ of the points with respect to P, such that the points in P_1 are before the points in P_2 and so on; points in the same set P_j are ordered arbitrarily; finally the points in P_k come before the points not in P_k. Consider the resulting work functions w_1' and w_2' after requests a_i and

a_j, respectively, with $i < j$ and notice that if $a_j \in P_k$ then $p(w'_2) \preceq p(w'_1)$. This means that the worst distribution prefers to give probability to a_i than to a_j. Similarly, using Lemma 10 we deduce that a request in P_k should be preferred to a request not in P_k.

In summary, the worst distribution assigns probability ϵ to points not covered by on-line servers in the order $a_1, a_2, \ldots$. Clearly, an optimal on-line algorithm should try to prevent this and hence, $A_0 = \{a_1, a_2, \ldots, a_k\}$ is the best configuration for an on-line algorithm—A_0 achieves the minimum value of $\psi(w, A_0)$. Notice that there is a familiar algorithm that achieves exactly this: the LRU algorithm. Algorithm LRU moves the server from a_k to the next request. Moreover, the LRU is always at the best configuration $\{a_1, a_2, \ldots, a_k\}$ at no extra cost other than servicing the requests. This also shows that the assumption above that we charge only cost one to the on-line algorithm at each step while we allow its servers to move to any configuration with no extra cost is a valid assumption for LRU. So, we have

Lemma 12 *Algorithm LRU is optimal against a diffuse adversary.*

Notice that LRU is indeed a refinement of the Work Function Algorithm. We still need to estimate the competitive ratio of LRU. Although LRU does not have to remember all the values of the work function (or equivalently its signature), the analysis of its competitive ratio depends on it. From the discussion above, we need only to estimate the off-line cost, which is the expected cost of a Markov process $M_{k,\epsilon}$.

The states of $M_{k,\epsilon}$ are the types of the work functions $(p_1, p_2, \ldots, p_k)$ with $1 = p_1 < p_2 < \cdots < p_k$. From state $(p_1, p_2, \ldots, p_k)$ which corresponds to a signature $P = (P_1, P_2, \ldots, P_k)$, there are transitions to the following $k-1$ types:

$$
\begin{array}{l}
(1, p_1 + 1, p_3, p_4, \ldots, p_k) \\
(1, p_1 + 1, p_2 + 1, p_4, \ldots, p_k) \\
\quad\vdots \\
(1, p_1 + 1, p_2 + 1, p_3 + 1, \ldots, p_{k-1} + 1)
\end{array}
$$

They correspond to the case of the next request r being in P_t, $t = 2, 3, \ldots, k$. There is also a transition to the type

$$(1, p_2 + 1, p_3 + 1, \ldots, p_k + 1)$$

that corresponds to the case $r \notin P_k$. The cost of each transition is the associated off-line cost. So, all transitions have cost zero, except the last one, which has cost one, because a request r increases the minimum value of the work function if and only if $r \notin P_k$.

Finally, the transition probabilities are determined by the worst distribution discussed above. The total probability of the first t transitions, $t = 1, \ldots, k-1$ is ϵ for each point in P_{t+1} except the k points occupied by the servers of LRU (in case $p_{t+1} < k$ the probability is zero). Therefore, the probability of the first t transitions is $\max\{(p_{t+1} - k)\epsilon, 0\}$. The probability of the last transition is the remaining probability $1 - (p_k - k)\epsilon$, which also shows that there is no need to consider types with p_k greater than $k + 1/\epsilon$ because these types are 'unreachable'. The importance of this fact is that the Markov process $M_{k,\epsilon}$ is finite (it has $O((k+1/\epsilon)^{k-1})$ states), despite the fact that the metric space is infinite.

Let $c(M_{k,\epsilon})$ be the expected cost of each step of the Markov process $M_{k,\epsilon}$, which is also the expected off-line cost. Since the on-line cost of each step is one, we get

Theorem 3 *Algorithm LRU is optimal against a diffuse adversary with competitive ratio*

$$R(k, \epsilon) = 1/c(M_{k,\epsilon})$$

It seems difficult to determine the exact competitive ratio. In fact, the next corollary suggests that it may not be expressible by a simple closed-form expression.

Corollary 1 *The competitive ratio for $k = 2$ is*

$$R(2, \epsilon) = 1 + \frac{1}{1 + (1-\epsilon)\left(1 + (1-2\epsilon)\left(\cdots\left(1 + 2\epsilon(1+\epsilon)\right)\cdots\right)\right)} \tag{3.3}$$

In particular, $1 + \sqrt{\epsilon}/2 \leq R(2, \epsilon) \leq 1 + 2\sqrt{\epsilon}$.

Proof. It is not difficult to see that the Markov process is identical to the following process: in each phase repeatedly choose uniformly a number in $1, \ldots n$, where $n = 1/\epsilon$; a phase ends when we choose a number twice. A phase is a cycle in the Markov chain that starts (and ends) at state with type $(1, 2)$. The cost of the Markov chain is one less than the expected length of a phase (all transitions in the cycle have cost one except the last one). It is not hard now to verify expression (3.3).

Notice that each of the first $\sqrt{n}$ numbers has probability at most $1/\sqrt{n}$ to end the phase. In contrast, each of the next $\sqrt{n}$ numbers has probability at least $1/\sqrt{n}$ to end the phase. Elaborating on this observation we get that $1 + \sqrt{\epsilon}/2 \leq R(2, \epsilon) \leq 1 + 2\sqrt{\epsilon}$. $\Box$

3.3 Comparative Analysis

On-line algorithms deal with the relations between *information regimes*. Formally but briefly, an information regime is the class of all functions from a domain D to a range R which are constant within a fixed partition of D. Refining this partition results in a richer regime. Traditionally, the literature on on-line algorithms has been preoccupied with comparisons between two basic information regimes: The *on-line* and the *off-line* regime (the off-line regime corresponds to the fully refined partition). As we argued in the introduction of this chapter, this has left unexplored several intricate comparisons between other important information regimes.

Comparative analysis is a generalization of competitive analysis allowing comparisons between arbitrary information regimes, via the comparative ratio defined in equation (3.2). Naturally, such comparisons make sense only if the corresponding regimes are rich in algorithms—single algorithms do not lend themselves to useful comparisons.

We apply comparative analysis to the *lookahead problem* in task systems. An on-line algorithm for a metrical task system has lookahead ℓ if it can base its decision not only on the past, but also on the next ℓ tasks. All on-line algorithms with lookahead ℓ comprise the information regime $\mathcal{L}_\ell$. Thus, $\mathcal{L}_0$ is the class of all traditional on-line algorithms.

Metrical task systems are defined on some metric space $\mathcal{M}$; a server resides on some point of the metric space and can move from point to point. Its goal is to process on-line a sequence of tasks $T_1, T_2, \ldots$ The cost $c(T_j, a_j)$ of processing a task T_j is determined by the task T_j and the position a of the server while processing the task. The total cost of processing the sequence is the sum of the distance moved by the server plus the cost of servicing each task T_j, $j = 1, 2, \ldots$

Theorem 4 *For any metrical task system, $R(\mathcal{L}_0, \mathcal{L}_\ell) \leq 2\ell+1$. Furthermore,* *there are metrical task systems for which* $R(\mathcal{L}_0, \mathcal{L}_\ell) = 2\ell + 1$.

Proof. Trivially the theorem holds for $\ell = 0$. Assume that $\ell > 0$ and fix an algorithm B in $\mathcal{L}_\ell$. We shall define an on-line algorithm A without lookahead whose cost on any sequence of tasks is at most $2\ell + 1$ times the cost of B. A knows the position of B ℓ steps ago. In order to process the next task, A moves first to B's last known position, and then processes the task greedily, that is, with the minimum possible cost.

Let $T_1, T_2, \dots$ be a sequence of tasks and let $b_1, b_2, \dots$ be the points where algorithm B processes each task and $a_1, a_2, \dots$ the corresponding points for algorithm A. For simplicity, we define also points $b_j = a_j = a_0$ for negative j's.

Then the cost of algorithm B is

$$\sum_{j=1} (d(b_{j-1}, b_j) + c(T_j, b_j))$$

and the cost of algorithm A is

$$\sum_{j=1} (d(a_{j-1}, b_{j-\ell}) + d(b_{j-\ell}, a_j) + c(T_j, a_j))$$

Recall that in order to process the j-th task, algorithm A moves to B's last known position $b_{j-\ell}$ and then processes the task greedily, that is $d(b_{j-\ell}, a_j) + c(T_j, a_j)$ is the smallest possible. In particular,

$$d(b_{j-\ell}, a_j) + c(T_j, a_j) \leq d(b_{j-\ell}, b_j) + c(T_j, b_j)$$

From this, the fact that costs are nonnegative and the triangle inequality we get

$$\begin{aligned} d(a_{j-1}, b_{j-\ell}) &\leq d(a_{j-1}, b_{j-\ell-1}) + d(b_{j-\ell-1}, b_{j-\ell}) \\ &\leq d(b_{j-1}, b_{j-\ell-1}) + c(T_{j-1}, b_{j-1}) + d(b_{j-\ell-1}, b_{j-\ell}) \end{aligned}$$

Combining these with the triangle inequalities of the form

$$d(b_i, b_{i+2}) \leq d(b_i, b_{i+1}) + d(b_{i+1}, b_{i+2})$$

we get that the cost of algorithm A is at most

$$\begin{aligned}\sum_{j=1} (d(b_{j-1}, b_{j-\ell-1}) + c(T_{j-1}, b_{j-1}) + d(b_{j-\ell-1}, b_{j-\ell}) + d(b_{j-\ell}, b_j) + c(T_j, b_j)) &\leq \\ \sum_{j=1} ((2\ell+1) d(b_{j-1}, b_j) + 2c(T_j, b_j)) &\leq \\ (2\ell+1) \sum_{j=1} (d(b_{j-1}, b_j) + c(T_j, b_j)) &\end{aligned}$$

The last expression is $(2\ell+1)$ times the cost of algorithm B.

For the converse, observe that when $c(T_j, x) = 0$, for all x, and when all triangle inequalities above hold as equalities then a comparative ratio of $2\ell+1$ can be achieved. □

Of course, for certain tasks systems the comparative ratio may be less that $2\ell+1$. For the paging problem it is $\ell+1$.

Theorem 5 *For the paging problem*

$$R(\mathcal{L}_0, \mathcal{L}_\ell) = \min\{k, \ell+1\}$$

Proof. Let $n = \min\{k-1, \ell\}$ and let B be an algorithm for the paging problem in the class $\mathcal{L}_\ell$, that is, with lookahead ℓ. Without loss of generality we assume that B moves its servers only to service requests. Consider the following on-line algorithm A:

> In order to service a request r, A moves a server that has not been moved in the last n times such that the resulting configuration is as close as possible to the last known configuration of B.

Fix a worst request sequence ρ and let $A_0, A_1, \ldots$ and $A_0 = B_0, B_1, \ldots$ be the configurations of A and B in order to service ρ. Without loss of generality, we assume that A moves a server in each step. By definition, A services the t-th request by moving a server not in B_{t-n} (unless $A_{t-1} = B_{t-n}$).

We will first show by induction that A_t and B_t, $t = 0, 1, \ldots$, differ in at most n points, that is $|B_t - A_t| \leq n$. This is obviously true for $t \leq n$. Assume that it holds for $t-1$. If $A_{t-1} = B_{t-n}$ then clearly the statement holds, because in each step the difference can increase by at most one. Otherwise,

A services the t-th request by moving the server from some point x, $x \notin B_{t-n}$. Observe that A_t can differ from B_t in more than n points only if $x \in A_{t-1} \cap B_{t-1}$. However, x can belong to $B_{t-1} - B_{t-n}$ only if x was requested at least once in the steps $t-n+1, t-n+2, \ldots, t-1$, because B moves servers only to service requests. Therefore, A moved a server at x in the last n steps and it cannot move it again. Hence, A_t and B_t cannot differ in more than n points.

The theorem now follows from the observation that for every $n+1$ consecutive moves of A there is a move of some server of B. The reason is this: if B stays in the same configuration then A will converge to the same configuration in at most n moves (recall that A moves a different server each time). $\square$

Bibliography

[1] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, January 1994.

[2] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, January 1994.

[3] Y. Bartal and E. Grove. The Harmonic k-server algorithm is competitive. *Personal Communication.*

[4] P. Berman, H. J. Karloff, and G. Tardos. A competitive three-server algorithm. *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 280–90, 1990.

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

[6] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *Proceedings 23rd Annual ACM Symposium on Theory of Computing*, pages 249–59, 1991.

[7] A. Borodin, N. Linial, and M. E. Saks. An optimal online algorithm for metrical task systems. *Proceedings 19th Annual ACM Symposium on Theory of Computing*, pages 373–82, 1987.

[8] A. Borodin, N. Linial, and M. E. Saks. An optimal online algorithm for metrical task systems. *Journal of the Association for Computing Machinery*, 39(4):745–63, October 1992.

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

[10] W. R. Burley. *On-line algorithms for generalized task systems.* PhD thesis, University of California, San Diego, 1994.

[11] M. Chrobak, H. J. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4:172–81, 1991.

[12] M. Chrobak and L. L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM Journal on Computing*, 20(1):144–8, February 1991.

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

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

[15] M. Chrobak and L. L. Larmore. Generosity helps or an 11-competitive algorithm for three servers. *Journal of Algorithms*, 16(2):234–63, March 1994.

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

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

[18] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–99, December 1991.

[19] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. *Proceedings. 31st Annual Symposium on Foundations of Computer Science*, pages 454–63 vol.2, 1990.

[20] A. Fiat, Y. Rabani, Y. Ravid, and B. Schieber. A deterministic $O(k^3)$-competitive k-server algorithm for the circle. *Algorithmica*, 11(6):572–78, June 1994.

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

[22] S. Irani, A. R. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 228–36, 1992.

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

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

[25] A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov paging. *Proceedings 33rd Annual Symposium on Foundations of Computer Science*, pages 208–17, 1992.

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

[27] E. Koutsoupias and C. H. Papadimitriou. On the k-server conjecture. *Proceedings 26th Annual ACM Symposium on Theory of Computing*, pages 507–11, 1994.

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

[29] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–30, June 1990.

[30] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–25, 1991.

[31] C. H. Papadimitriou. On the value of information. *World Congress of Economics, Moscow*, August 1992.

[32] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization : algorithms and complexity.* Prentice Hall, 1982.

[33] P. Raghavan. A statistical adversary for on-line algorithms. *On-line algorithms: proceedings of a DIMACS workshop. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:79–83, 1992.

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

[35] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.

[36] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–41, June 1994.