On Online Computation

Sandy Irani * Anna R. Karlin †

Abstract

This chapter presents an introduction to the competitive analysis of online algorithms. In an online problem, data is supplied to the algorithm incrementally, one unit at a time. The online algorithm must also produce the output incrementally: after seeing t units of input, it must output the t^{th} unit of output. Since decisions about the output are made with incomplete knowledge about the entire input, an online algorithm often can not produce an optimal solution. Such an algorithm can only hope to approximate the performance of the optimal algorithm that sees all the inputs in advance. An online algorithm is said to be competitive if its performance is close to that of the optimal offline algorithm on each input.

We introduce the basic principles underlying the design and analysis of online algorithms and illustrate these ideas with results for three different problems: paging, metrical task systems and the k-server problem. We then give detailed descriptions of two recent results. The first is the analysis of the work-function algorithm for the k-server problem ([KP94b]), and the second is the exponential function technique for virtual circuit routing ([AAF+93]). Finally, we discuss new directions in the analysis of online algorithms intended to overcome some of the shortcomings of traditional competitive analysis.

1 Introduction

Suppose you are about to go skiing for the first time in your life. Naturally, you ask yourself whether to rent skis or to buy them. Renting skis costs, say, \$30, whereas buying skis costs, say \$300. If you knew how many times you would go skiing in the future (ignoring complicating factors such as inflation, and changing models of skis), then your choice would be clear. If you knew you would go at least 10 times, you would be financially better off by buying skis right from the beginning, whereas if you knew you would go less than 10 times, you would be better off renting skis every time.

Alas, the future is unclear, and you must make a decision nonetheless. In this sense, you are faced with an *online problem*, and must deal with events (in this case ski trips) as they arrive

*Department of Computer Science, University of California, Irvine, CA

†Department of Computer Science, University of Washington Seattle, WA.

without knowledge (or with only limited knowledge) of future events. An *online algorithm* is an algorithm which must solve an online problem.

Online algorithms are approximation algorithms of a certain sort: they attempt to approximate the performance of the optimal offline algorithm OPT for the problem. OPT is the algorithm that knows the future in its entirety, and deals with events as they arrive at minimum total cost.

For the ski-rental problem, we observe that if an adversary determines exactly at what point you will never go skiing again, it can ensure that you pay at least twice what you could have paid, regardless of your strategy. Indeed, suppose your algorithm is to continue renting until you have skied j times and then to buy skis. The adversary will then simply wait until you buy skis, and then make sure you never ski again. Since you pay $30j + 300$, and you could have paid $\min(30j, 300)$, you pay at least twice the optimal offline cost.

Furthermore, there is a trivial online algorithm that is guaranteed to achieve this bound: rent until you've skied 10 times, then buy. In so doing, if you ski j times, where $j \leq 10$, you pay $30j$, exactly the optimal offline cost. If $j > 10$, you pay $2 \cdot 300$, twice the optimal offline cost.

The principle underlying this algorithm is often referred to as *the ski principle*. Although it may seem like a good idea to take the action which is cheapest in the short term, online algorithms must hedge their bets. Indeed, it may be that by taking an expensive action at some point, the algorithm pays less in future events. In essence, the ski principle says that once the algorithm has incurred enough total cost by executing a number of cheap actions, the algorithm can afford to take a more expensive action, which can be amortized against the collection of cheap actions.

Online problems are pervasive in computer science, in such diverse areas as scheduling, routing, memory management, dynamic data structures, dynamic graph algorithms, and robot motion planning. This chapter is a survey of techniques used in competitive algorithms which we illustrate using problems from several applications. The chapter is organized as follows. In Section 2, we introduce the basic definitions of competitive analysis and several motivating example problems. In Section 3, we present the theoretical underpinnings of deterministic competitive algorithms. In section 4, we present the theory underlying randomized competitive algorithms. In Sections 5 and 6, we give detailed proofs of two recent exciting results in online algorithms. Section 5 presents the proof of the recent tour-de-force of competitive algorithms, the proof that the work-function algorithm is $2k - 1$ competitive for the k-server problem. Section 6 presents the exponential function technique for virtual circuit routing, alias online multicommodity flow, and gives an historical overview of the area. In Section 7 we discuss some of the shortcomings of competitive analysis and recent work which addresses some of these problems. Finally, in section 8, we give some open problems and directions for future research.

2 Three Examples of Competitive Analysis

We will be using three examples to illustrate the various notions that arise in online algorithms. These are: the paging problem, the k-server problem and metrical task systems. Each problem is a special case of the next.

2.1 Paging

The paging problem is the most fundamental and practically important online problem in computer science. Consider a two-level store consisting of a fast memory (the *cache*) that can hold k pages, and a slow memory that can store n pages. The n pages in slow memory represent the virtual memory pages. A *paging algorithm* is presented with a sequence of requests to virtual memory pages. If the page requested is in fast memory (a *hit*), no cost is incurred; but if not (a *fault*), the algorithm must bring it into the fast memory at unit cost. The algorithm must decide which of the k pages currently in fast memory to evict in order to make room for the newly requested page.

If we knew the future, the decision would be clear. It has long been known that the optimal offline algorithm, called MIN, is the algorithm which always evicts the page whose next request is furthest in the future. Unfortunately, in practice paging decisions are made without knowledge of future requests. A paging algorithm is *on-line* if it makes the decision of which page to evict without knowing which pages will be requested in the future. We will discuss one commonly used online paging algorithm called *Least-Recently-Used* (LRU). On a fault, LRU evicts the page in memory whose most recent access was earliest.

How do we evaluate the performance of such an online paging algorithm?

A traditional worst-case analysis of paging is completely uninformative, since any paging algorithm can be made to fault on every single request by an adversary which always requests the most recently discarded page. Hence, from the worst-case point of view, all online paging algorithms are equivalent. However, in practice some algorithms perform much better than others and a theoretical analysis should reflect this difference.

Alternatively, one could employ average-case analysis. The problem here is that one must postulate a statistical model for the input. It is difficult to come up with a fixed probability distribution that captures realistic instances, since the patterns of access tend to change dynamically with time and across applications. Nonetheless, several of the early analyses of paging algorithms were done assuming such fixed probability distributions.

Motivated by these observations, Sleator and Tarjan proposed the idea of *competitive* *analysis*.[1] In competitive analysis, we compare the performance of the online algorithm against the performance of the optimal offline algorithm on every input and consider the worst-case ratio. Let $\mathtt{cost}_A(\sigma)$ be the cost incurred by an online algorithm A on the input sequence σ. In the case of paging, σ is a sequence of page requests, and $\mathtt{cost}_A(\sigma)$ is the number of page faults incurred by algorithm A on the sequence σ. Let OPT be the optimal offline algorithm, and let $\mathtt{cost}_{OPT}(\sigma)$ be the cost incurred by the optimal offline algorithm on input σ.

We say that the online algorithm A is c-competitive if there exists a constant b such that on every request sequence σ,

$$\mathtt{cost}_A(\sigma) \leq c \cdot \mathtt{cost}_{OPT}(\sigma) + b.$$

In some sense $\mathtt{cost}_{OPT}(\sigma)$ measures the inherent difficulty of σ, and we only ask an online algorithm to perform well relative to the difficulty of the input. The *competitive ratio* of the algorithm A, denoted c_A is the infimum over c such that A is c-competitive. An algorithm is said to be *strongly competitive* if it achieves the best possible competitive ratio for a problem. The competitive ratio has become the standard measuring stick for online algorithms in recent years and has been used by the vast majority of recent work in on-line algorithms.

Notice that we are not placing any computational restrictions on the algorithm – we are simply measuring, from an information theoretic point of view, what kind of solution quality can be obtained given the fact that the decisions have to be made with partial information. How the competitive ratio changes as various computational constraints are imposed is an interesting and largely unexplored issue.

2.2 The k-server problem

In the k-server problem, introduced by Manasse, McGeoch and Sleator [MMS90], we must choose how k mobile servers will serve each of a sequence of requests. More specifically, we have k servers each of which occupies a single point in a fixed metric space M. The metric space is known to the online algorithm in advance. Repeatedly a *request*, a point $x \in M$, appears. Each request must be *served* before the next request appears. To serve the request x, the algorithm must move a server to x unless it already has a server at x. Whenever the algorithm moves a server from point a to point b, it incurs a *cost* of $d_{a,b}$, the distance between a and b in the metric space. The goal is to minimize the total distance moved by all the servers.

The k-server problem encompasses many interesting problems as special cases, for example paging and motion planning for 2-headed disks. To see that the paging problem is a special case of the k-server problem, consider the k-server problem on a *uniform* metric space where the distance between each distinct pair of points is 1. Each point in the space corresponds to a virtual page, and each of the k servers corresponds to one of the k cache locations. The k

[1]Competitive analysis was implicit in early work on bin-packing in the sixties.

points occupied by servers correspond to the k pages in the cache. When a request is made for a page which already resides in the cache, the cost to serve it is 0 since it is covered by a server already. When a request is made for a page p not in the cache, some server moves a distance one from say q in order to serve the request. This corresponds to the page p replacing q in the cache.

If all the requests are known in advance but the requests must still be served in their order they appear in the sequence (i.e. the problem is offline), there is an algorithm to determine the optimal strategy which takes time $O(kn^2)$ where n is the number of requests in the sequence [CKPV91]. Thus, the difficulty of the problem arises solely from its on-line nature.

2.3 Metrical Task Systems

Metrical task system, introduced by Borodin, Linial and Saks [BLS87], are a general model for describing a wide class of problems in which a sequence of tasks must be performed. A *task system* is a triple $(S, [[d]], \mathcal{T})$, where S is a finite set of states and $[[d]]$ is a matrix representing distances between each pair of states. (The distance between any two states i and j is denoted $d_{i,j}$.) The distances obey the triangle inequality and the distance from a state to itself is 0. $\mathcal{T}$ is a set of tasks. A task $T \in \mathcal{T}$ is a non-negative vector indicating the cost of processing that task in each of the states; $T(i)$ is the cost of processing the task T in state i. The input is a sequence of tasks from $\mathcal{T}$: $\sigma = T^1, T^2, \ldots, T^N$, where $T^i \in \mathcal{T}$ for all i. The objective of the algorithm is to determine a state in which to process each task, balancing the cost of moving with the cost of processing the tasks. The algorithm produces a schedule $\pi : \{1, \ldots, N\} \to S$, where $\pi(j)$ is the state in which T^j is processed. If the algorithm is on-line, then the choice of $\pi(j)$ depends only on $T^1, \ldots, T^j$. The cost of a schedule π on σ is the sum of the cost of moving from state to state (the *moving costs*) and the cost of processing the tasks (the *stationary costs*):

$$\texttt{cost}(\pi, \sigma) = \sum_{i=1}^{N} d_{\pi(i-1),\pi(i)} + \sum_{i=1}^{N} T^i(\pi(i)).$$

$\pi(0)$ is some fixed initial state. We denote by $A(\sigma)$ the schedule produced by algorithm A on input σ. The *cost* of algorithm A on σ, denoted $\texttt{cost}_A(\sigma)$, is $\texttt{cost}(A(\sigma), \sigma)$. The cost of the optimal off-line algorithm for the sequence π is

$$\texttt{cost}_{OPT}(\sigma) = \min_{\pi} \texttt{cost}(\pi, \sigma).$$

In most of the interesting examples of task systems, the distances are symmetric (i.e. $d_{i,j} = d_{j,i}$ for all i and j). In this case, the task system is a *metrical task system*. Many online problems in the literature can be described as metrical task systems, including paging and the k-server problem. For example, to express the k-server problem as a metrical task system, we choose the set of states S to be the set of multisets of k points in the metric space in which the servers move. The distance between two states is the minimum total distance k servers would have to move if the were originally occupying one k-set and had to move to the other. This is just the minimum matching between the two k-sets. $\mathcal{T}$ consists of a collection of task vectors, one for each point p in the metric space, where

$$T_p(S) = \begin{cases} 0, & \text{if } p \in S \\ \infty. & \text{if } p \notin S \end{cases}$$

Thus, in order to satisfy the request T_p without incurring a cost of ∞, the algorithm must be in a configuration such that at least one server is occupying the point p.

The k-server problem is not the only problem that can be expressed as a metrical task system. The model is powerful enough to completely formulate a large class of the on-line problems of interest. Examples include: paging, layered graph traversal, operations on dynamic data structures, and maintaining data in a multiprocessing environment ([BFR92, BS89, KMRS88, MMS90, PY91, DS85, ST85, Wes92]).

3 Theoretical Underpinnings: Deterministic Algorithms

3.1 Lower Bounds

A standard technique in proving lower bounds is to employ an adversary which plays against an arbitrary algorithm and concocts an input which forces it to incur a high cost. In proving a lower bound for the best possible competitive ratio an algorithm can achieve on a problem, an adversary has two tasks. The adversary generates an input sequence for A and then performs the sequence of operations himself. The adversary's goal is to find a sequence which forces the algorithm to incur a high cost which he can service at a low cost. Such an adversary is commonly referred to as a *cruel adversary.* A cruel adversary is not a rigorous concept, but roughly speaking, such an adversary chooses the next task so that the online algorithm incurs a non-negative cost while the optimal offline algorithm incurs no extra cost whenever possible. In the setting of metrical task systems, if the algorithm is in state j, then the adversary would choose the next task so that $T(j) = c > 0$ and $T(i) = 0$ for all $i \neq j$.

We illustrate the concept of a cruel adversary with a very simple example.

Theorem 1 *[ST85] If A is any deterministic online paging algorithm, then $c_A \geq k$.*

Proof. We assume that A and OPT both start with the same set of pages in the fast memory. The adversary restricts its request sequence to a set of $k+1$ pages: the k pages initially residing in fast memory and one other page. The adversary always requests the page that is outside of A's fast memory. This process can be continued for an arbitrary number of requests, resulting in an arbitrarily long sequence σ on which A faults on every request.

We must now show that $\texttt{cost}_{OPT}(\sigma) \leq \lceil |\sigma|/k \rceil$. At each fault, the adversary adopts the following strategy: evict the page whose first request occurs farthest in the future. Suppose a page x is evicted by OPT. The next fault occurs the next time x is requested. The adversary is guaranteed that all the other pages in the adversary's fast memory will be requested before x is requested again. There will be at least $k-1$ pages requested between any two faults, so the adversary faults at most on every k^{th} request. ∎

Note that by requesting only $k+1$ distinct pages, the adversary ensures that the offline cost is minimal.

It is worth remarking that this proof was made particularly simple by the fact that determining the optimal offline cost was straightforward. For many online problems, this is not the case. Indeed, many of the online problems that have been studied are NP-complete. Fortunately, it is sufficient to provide an upper bound for the adversary's cost in proving a lower bound on the competitive ratio.

A useful technique for finding an upper bound on the optimal cost is averaging over a collection of algorithms. That is, we find a collection of algorithms $A_1, \ldots, A_m$, such that when a cruel adversary constructs a sequence σ designed to hurt any online algorithm A, the cumulative cost of the m algorithms on sequence σ is at most c times the cost of A on σ, i.e.

$$\sum_{1 \leq i \leq m} A_i[\sigma] \leq cA[\sigma].$$

Since there is an algorithm in the set $A_1, \ldots A_m$ whose cost is at most the average, such a result implies a lower bound of m/c on the competitive ratio.

This is a remarkably powerful technique as illustrated by the following example which gives a lower bound for an online algorithm for the k-server problem in *any* metric space.

Theorem 2 *[MMS90] Let A be any online algorithm for the k-server problem. Then the competitiveness of A is at least k.*

Proof. The adversary chooses $k+1$ points: the k points initially occupied by the servers, which we denote by $\{p_1, \ldots, p_k\}$, and one other point, p_{k+1}. The adversary will only request points from that set. Let A be a deterministic algorithm for the k server problem. On each request, the adversary requests the point where A does not have a server. The adversary continues this process for N requests, where N can be arbitrarily large. Let $\sigma = p_{i_1}, p_{i_2}, \ldots, p_{i_N}$ be the resulting sequence. The cost of A on the sequence is

$$\texttt{cost}_A(\sigma) = \sum_{j=1}^{N-1} d_{p_{i_j}, p_{i_{j+1}}}.$$

We will upper bound the cost of OPT by showing k algorithms $A_1, \ldots, A_k$ whose total cost on σ is equal to A's cost on σ. On the initial request to p_{k+1}, A_i moves the server from point

p_i to point p_{k+1}. From this point on, the set of algorithms maintain the invariant that each algorithm has a server on the point which was last requested. In addition, no two algorithms have the same point vacant.

When p_{i_j} is requested, exactly one algorithm must move a server. That algorithm moves a server from $p_{i_{j-1}}$ to p_{i_j}, and the invariants are maintained. The total cost incurred by all k algorithms is:

$$\sum_{l=1}^{k} \texttt{cost}_{A_l}(\sigma) = \sum_{j=1}^{k} d_{p_j,p_{k+1}} + \sum_{j=1}^{N-1} d_{p_{i_j},p_{i_{j+1}}} = \sum_{j=1}^{k} d_{p_j,p_{k+1}} + \texttt{cost}_A(\sigma).$$

The constant $\sum_{j=1}^{k} d_{p_j,p_{k+1}}$ which we will denote by c accounts for the first move of each algorithm. We have that

$$k\texttt{cost}_{OPT}(\sigma) \leq k \min_l\{\texttt{cost}_{A_l}(\sigma)\} \leq \sum_{l=1}^{k} \texttt{cost}_{A_l}(\sigma).$$

Thus,

$$k\texttt{cost}_{OPT}(\sigma) \leq \texttt{cost}_A(\sigma) + c.$$

Since the sequence can be arbitrarily costly, and hence the $\texttt{cost}_A(\sigma)$ can be made arbitrarily large, the constant c is negligible.

■

3.2 Design Principles

The design of online algorithms is still something of an art. There are, however, a few recurring themes which have been useful in a variety of settings. These are

- greedy algorithms,
- balancing costs,
- forcing the offline cost to increase, and
- combining online algorithms.

We consider each of these in turn.

3.2.1 Greedy Algorithms

In a somewhat vacuous sense, all online algorithms are greedy, since an online algorithm is defined by a function f of the current request, the current state and the history. Essentially the algorithm evaluates the function for each possible way of serving the current request, and chooses the action which minimizes the value of the function. Nonetheless, there are some natural greedy-type algorithms that work well in certain situations.

The first is the *local greedy* algorithm. This algorithm chooses an action which minimizes the cost to serve the current request. This algorithm performs abysmally in some situations. For example, for the k-server problem, it has no bounded competitive ratio. On the other hand, there are online problems for which it is optimal, including some load balancing problems and constructing online Steiner trees. In addition, there are weighted local greedy algorithms. One such example is the exponential function technique used for virtual circuit routing which is discussed in detail in Section 6.

The second is the *retrospective* algorithm. This algorithm keeps track of history and what states OPT would be in if the sequence seen so far (including the current request) were the entire sequence. The online algorithm then moves to the closest one of these states. This algorithm is 2-competitive for ski-rental, but has no bounded competitive ratio for the k-server problem. A good illustration of a retrospective algorithm was developed independently in [KP93] and [KMV94] for online minimum weight bipartite matching.

Third is the celebrated *work function* algorithm (WFA)[BLS87, CL92, KP94b]. This is a combination of the first two types of greedy algorithms. It works as follows. Let $opt_t(s)$ be the optimal cost of serving the first t requests and ending up in state s. If the WFA is in state s_t after serving the t^{th} request, then it serves the $(t+1)st$ request in state s_{t+1} where s_{t+1} is the state minimizing $opt_{t+1}(s_{t+1}) + d_{s_t,s_{t+1}}$. This algorithm is known to have optimal competitiveness or near optimal competitiveness for several online problems including the k-server problem. In section 5, we give the proof that the work function algorithm is $(2k-1)$-competitive for the k-server problem.

There are also situations in which the WFA is not competitive. Burley shows that the WFA is not competitive for the problem of Layered Graph Traversals (see [Bur93]). Interestingly, he shows that for this problem, a slightly altered version of the algorithm is competitive, where the state for the $t+1^{st}$ request is chosen to minimize $\alpha \cdot opt_{t+1}(s_{t+1}) + d_{s_t,s_{t+1}}$, for some $\alpha > 1$. It is a major open problem to characterize online problems for which each of these types of algorithms is competitive.

3.2.2 Balancing Costs

The principle of balancing costs has already been discussed in the introduction under the name of the "ski principle". Suppose the online algorithm has two choices for how to service a request. If the future is one way, then one of the choices will ultimately be cheaper, whereas if the future is another way, the other choice will ultimately be cheaper. Balancing costs, says that an online algorithm should roughly try to incur the same cost on each of the possible futures.

Another example of an algorithm that employs this idea is a k-server algorithm called $BALANCE$, due to Manasse, McGeoch and Sleator [MMS90], which is optimal for metric spaces consisting of $k+1$ points. $BALANCE$ maintains the cumulative distance D_i traveled by the i^{th} server. Let d_i denote the distance from server i to the next request. $BALANCE$ sends the server to next request which minimizes $D_i + d_i$.

Clearly $BALANCE$ is attempting to keep the total distance moved by the various servers more or less equal. By doing so, it guards against futures in which different servers are active. Additional intuition for this algorithm comes from considering why the local greedy algorithm fails. By simply moving the server that is closest to the request, the online algorithm may ping-pong a server a between two points alternately requested, simply because moving a far away server b in is locally expensive. The $BALANCE$ algorithm does not permit this to happen because as soon as server a ping-pongs for a total cost greater than that of moving b to the request, $BALANCE$ moves b.

We prove that $BALANCE$ has optimal competitiveness for metric spaces with $k+1$ points.

Theorem 3 *[MMS90] Let c_{BAL} be the competitive ratio of $BALANCE$ on a metric space with $k+1$ points. Then, $c_{BAL} \leq k$.*

Proof. At any point in time, there is exactly one point which is not occupied by one of $BALANCE$'s servers. This point is called the *hole*. We can assume that the request sequence always requests $BALANCE$'s hole. A request to another point will cost $BALANCE$ nothing and can only serve to increase OPT's cost.

The points in the space are numbered $\{1, \ldots, k+1\}$. Let s_i denote the configuration which has a server on every point except i. Let $opt_t(s_i)$ be the optimal way of serving the first t requests and ending up in configuration s_i. Note that the cost of moving from s_i to s_j is $d_{i,j}$ and, thus, $opt_t(s_i) \leq opt_t(s_j) + d_{i,j}$.

Let D_i^t denote the distance traversed after the first t requests by $BALANCE$'s server which occupies point i. If BALANCE does not have a server on point i, then D_i^t is undefined. Let h^t denote the location of $BALANCE$'s hole after t requests. The $t+1^{st}$ request is to point h^t. We make the following claim:

Claim : For $i \neq h^t$, $D_i^t \leq opt_t(s_i)$.

We first prove the theorem modulo the claim. The total work done by $BALANCE$ through time t is

$$\sum_{i \neq h^t} D_i^t \quad \leq \quad \sum_{i \neq h^t} opt_t(s_i)$$

$$\leq \quad k[\min_{1\leq i\leq k+1} opt_t(s_i)] + k \max_{i\neq j} d_{i,j}$$

$$\leq \quad k[\text{ cost of } OPT] + b$$

where b depends only on the distances in the metric space and not on the length of the sequence.

We prove the claim by induction on t. The claim is clearly true for $t = 0$. Now consider a request to h^t at time $t + 1$. Let min be the point which minimizes $\min_{i\neq h^t}\{D_i^t + d_{i,h^t}\}$. $BALANCE$ sends the server on min to h^t. Thus, $h^{t+1} \leftarrow min$ and $D_{h^t}^{t+1} \leftarrow D_{min}^t + d_{min,h^t}$. Furthermore $opt_{t+1}(s_{h^t}) \leftarrow \min_{i\neq h^t} opt_t(s_i) + d_{i,h^t}$. Since by the inductive hypothesis, for $i \neq h^t$, $opt_t(s_i) \geq D_i^t$, we get that $opt_{t+1}(s_{h^t}) \geq D_{h^t}^{t+1}$. The rest of the D's and values of $opt_{t+1}(s_j)$, for $j \neq h^t$ remain unchanged. ■

The balancing principle in its simplest form does not always work however. In particular, if there are more than $k + 1$ points in the metric space, then $BALANCE$ is not competitive, even when there are only two servers. Irani and Rubinfeld show in [IR91] that a simple variant of $BALANCE$ which minimizes $D_i + 2d_i$ is 10-competitive for two servers. Chrobak and Larmore show in [CL91a] that this variant is no better than 6-competitive and Kleinberg shows in [Kle94] that any balance-like algorithm for two servers which sends the server which minimizes $D_i + f(d_i)$ for any function f can be no better than 3.82-competitive.

3.2.3 Forcing Cruel Adversary Cost to Increase

An online algorithm can choose an action to take in order to make sure that it will "hasten" the time at which the offline cost will increase.

An example of this paradigm is a general class of paging algorithms called *marking* algorithms. A marking algorithm proceeds in phases. At the beginning of a phase all the nodes are unmarked. Whenever a page is requested, it is marked. On a fault, the marking algorithm evicts an unmarked page (chosen by a rule specified by the algorithm), and brings in the requested page. A phase ends just before the first fault after every page in the fast memory is marked (equivalently, a phase ends just before a request to the $k + 1^{st}$ distinct page requested in the phase). At this point all the nodes become unmarked and a new phase begins.

Note that the phases are completely determined by the sequence and not by the choice of which unmarked page the algorithm evicts. The intuition behind the marking algorithms is that an adversary can force any deterministic online paging algorithm to fault on every request. Given this fact, the algorithm can only hope to pick pages to evict so that if the adversary always picks a page outside the algorithm's fast memory, his cost will also increase. This idea is made more explicit in the proof below.

Theorem 4 *[KMRS88] Any marking algorithm is k-competitive.*

Proof. The proof is based on the simple observation that the optimal algorithm must incur at least one fault in a phase. To see this, divide the sequence in segments that start on the second request of a phase and end with the first request of the next phase. The claim is that any algorithm must fault at least once in a segment. At the beginning of a segment, the algorithm has the most recent request in its fast memory (this is just the first request of the phase). If it does not fault during the remainder of the phase, then it must have all k pages requested during the phase residing its fast memory. The first request of the next phase (the last request of the segment) is, by definition, to a page not in this set. Thus, the optimal algorithm must fault.

Meanwhile, the marking algorithm will incur only k faults in a phase. This follows from the fact that there are exactly k distinct pages requested in one phase. Furthermore, once a page has been requested, it becomes marked and will not be evicted for the remainder of the phase. Thus, a marking algorithm will never fault more than once on a given page during a phase.

■

Another way to view this principle is that, in some sense, a marking algorithm uses the recent past to predict the future. All of the marked pages have been requested more recently than any of the unmarked pages. On the assumption that pages that have been recently requested will be more likely to be requested again (i.e. the sequence exhibits locality of reference), a marking algorithm will not evict any marked page. Thus, an online algorithm can take advantage of any local repetitions in the input sequence. An algorithm which can do this will be favored by a competitive analysis for the following reason. The offline algorithm will have a lower cost on sequences the exhibit locality of reference and will tend to have a higher cost on sequences in which many different pages are requested in turn. Since the online algorithm is evaluated in comparison to the offline algorithm, it must fare well on those sequences for which the offline algorithm has a low cost. These are exactly the sequences in which exhibit locality of reference. *Least-Recently-Used* (which is also a marking algorithm) uses the recent history of requests to predict the future even more explicitly.

For another example of this principle, see the Move-To-Front heuristic for maintaining a linked list in [ST85]. A more involved use, in conjunction with the balancing costs principle is the Traversal Algorithm in [BLS87].

3.2.4 Online Choice of Online Algorithms

Let $A_1, \ldots, A_m$ be a set of online algorithms for a problem P with input set I. Assume that P can be represented as a metrical task system. Each A_i has a competitive ratio a with respect to the optimum offline algorithm, but only for a *subset* of the possible inputs such that the union of these subsets covers I. Given this setup, one can construct a generic deterministic (resp. randomized) online algorithm for P which is $a(2em+1)$-competitive (resp. $O(a \log m)$-competitive) over all possible inputs. This is sometimes referred to as the MIN operator on online algorithms.

Fiat *et al.* observed in [FRR90] that the deterministic version of the MIN operator is essentially the solution to the 'm cow paths problem' analyzed in [BYCR93] and [PY91]. In this problem there are m paths of unknown finite length which all share a common starting point. The goal is to reach the end of one of them, travelling at most some constant times the length of the minimum path. The operator, described here for metrical task systems, works as follows. Let s be the starting configuration for all algorithms and $\sigma = \langle T^1, T^2, \ldots, T^N \rangle$ be the sequence of tasks. Let σ_j denote the first j tasks in the sequence σ.

The cost of A_i on the first j requests when starting in configuration s is denoted by $\texttt{cost}_{A_i}(s, \sigma_j)$. Let $\texttt{state}_{A_i}(s, \sigma_j)$ denote the state that A_i is in after serving σ_j. The algorithm $MIN = MIN\{A_0, \ldots, A_{m-1}\}$ is always currently following one of the algorithms. That is to say, after serving σ_j, the algorithm is in state $\texttt{state}_{A_i}(s, \sigma_j)$ for some i. The initial algorithm is chosen arbitrarily. MIN keeps a bound B which is initially set to $m/(m-1)$. Suppose MIN is following A_i when the j^{th} request is received. That is, MIN is in state $\texttt{state}_{A_i}(s, \sigma_{j-1})$. All costs are scaled so that $\min_{1 \leq i \leq m} \texttt{cost}_{A_i}(s, \sigma_j) = 1$. Before serving T^j, MIN checks $\texttt{cost}_{A_i}(s, \sigma_j)$ against B. If $\texttt{cost}_{A_i}(s, \sigma_j) > B$, then MIN goes to the configuration of $A_{(i+1) \pmod m}$ just before serving T^j (i.e., $\texttt{state}_{A_{(i+1) \pmod m}}(s, \sigma_j)$) and starts following A_{i+1}. Each time a switch is performed, $B \leftarrow Bm/(m-1)$. We state here, without proof, the result implied by [BYCR93].

Theorem 5 *Given any m algorithms for a metrical task system, let $MIN = MIN\{A_0, \ldots, A_{m-1}\}$. For any sequence of tasks σ and any initial starting configuration s,*

$$\texttt{cost}_{MIN}(s, \sigma) \leq (2em+1) \cdot \min_{1 \leq i \leq m} \{\texttt{cost}_{A_i}(s, \sigma)\}.$$

The randomized MIN operator was presented and analyzed in [FFK$^+$91], and later improved in [ABM93]. [ABM93] also gives an improved deterministic MIN operator in which an $O(\sum_{i=1}^{m} a_i)$-competitive algorithm is obtained, where each A_i is a_i-competitive for its portion of the inputs.

3.3 Bounding Competitiveness

3.3.1 Potential Functions

The most common technique for proving upper bounds on the competitive ratio is to construct a *potential function.* Suppose that A is an online algorithm. We use a potential function to analyze A as follows. Fix an input sequence σ of m operations and run A and OPT simultaneously on σ. The potential function maps the configurations of A and OPT to a real number Φ. Let Φ_i denote the value of the potential function after OPT and A have served the first i requests of the sequence. Let t_i denote the cost of A on the i^{th} operation, and o_i the cost of OPT on the i^{th} operation. The *amortized* cost of A for the i^{th} operation, denoted a_i, is defined to be $t_i + \Phi_i - \Phi_{i-1}$. An upper bound for the cost of A on the entire sequence is proven by bounding the amortized cost of A for each operation. Suppose we can find a nonnegative potential function whose initial value is 0 such that for every i, $1 \leq i \leq m$,

$$a_i \leq c \cdot o_i.$$

Then

$$\begin{aligned}
\texttt{cost}_A(\sigma) &= \sum_{i=1}^{m} t_i \\
&\leq \Phi_m - \Phi_0 + \sum_{i=1}^{m} t_i \\
&= \sum_{i=1}^{m} (t_i - \Phi_{i-1} + \Phi_i) \\
&= \sum_{i=1}^{m} a_i \leq c \sum_{i=1}^{m} o_i = c \cdot \texttt{cost}_{OPT}(\sigma).
\end{aligned}$$

The use of a potential function to analyze a sequence of operations is called *amortized analysis* [Tar85] because the cost of each operation is amortized (i.e. averaged) across the entire sequence of requests. An individual operation may be costly for A relative to the cost of OPT, however the average cost of an operation over the sequence can be bounded by twice the average optimal cost. Note that the term 'average' refers to the average over the sequence and does not imply any statistical assumptions about the input sequence. Potential functions are often used in online algorithms because typically the input to an online problem is a sequence of operations and the goal is to upper bound the cost of an algorithm over the entire sequence.

The key to making such a proof work lies in the choice of an appropriate potential function. Typically, a potential function will measure the 'distance' from the online algorithm's configuration to OPT's configuration. We illustrate these ideas using a potential function proof that LRU is a strongly competitive paging algorithm.

We define for each page p, a value $a[p]$. $a[p] = 0$ if and only if p is not in the fast memory. If $a[p] > 0$, then $a[p]$ denotes p's place in the LRU queue. Since initially the cache is empty, we start with $a[p] = 0$ for all p. When LRU copies a new page into the fast memory, it decrements $a[p]$ for every p in the fast memory, ejects the page whose a value is 0 and for the new page p', it sets $a[p'] \leftarrow k$. If a page p is requested which is already in LRU's fast memory, then for every p' such that $a[p'] > a[p]$, LRU will decrement $a[p']$. Then it sets $a[p] \leftarrow k$.

Let S be the set of pages that OPT has in its fast memory. The potential function is defined to be

$$\Phi = \sum_{p \in S} (k - a[p]).$$

Consider a single page request. Let $\Delta\texttt{cost}_{LRU}$ denote the cost of LRU and let $\Delta\texttt{cost}_{OPT}$ denote the cost of OPT to satisfy the request. The change in potential is denoted $\Delta\Phi$. Since both fast memories start out empty, $\Phi_0 = 0$. Furthermore $\Phi_i \geq 0$ for all i because $a[P] \leq k$ for all P. Thus, it follows from the following lemma that LRU is k-competitive.

Lemma 6 *[KMRS88] For any request in any sequence of page requests,*

$$\Delta\texttt{cost}_{LRU} + \Delta\Phi \leq k(\Delta\texttt{cost}_{OPT}).$$

Proof. Consider a request for a page p. We divide the request into two steps: in step 1, we let OPT satisfy the request and then in step 2, we let LRU satisfy the request. We prove the lemma for each step separately.

Step 1: Let OPT satisfy the request. If OPT does not fault, $\Delta\Phi = \Delta\texttt{cost}_{OPT} = \Delta\texttt{cost}_{LRU} = 0$. If OPT does fault, the right hand side of the inequality is k. Thus, we have to show that $\Delta\Phi \leq k$. OPT loads a new page p and evicts one page p'. In the worst case, p is not in LRU's fast memory and p' is. In this case $\Delta\Phi = (k - a[p]) - (k - a[p']) \leq k$.

Step 2: Now let LRU satisfy the request. If LRU does not fault, then we have to show that $\Delta\Phi \leq 0$. Let $a[p] = i$. We know that $p \in S$. So Φ decrease by $k - i$ due to the change in $a[p]$. The a value for at most $k - i$ other pages will change and each of these will decrease by at most 1. Thus, $\Delta\Phi \leq 0$.

If LRU does fault, the left hand side of the inequality is 1. Thus, we have to show that $\Delta\Phi \leq -1$. For every page that is both in LRU and OPT's fast memory, the potential function increases by 1 (because the a value is decremented by 1). There are at most $k - 1$ of them because OPT has the newly requested page and LRU does not. The new page p is already in OPT's fast memory, and the a value for this page goes from 0 to k, causing Φ to decrease by k. Thus, $\Delta\Phi \leq (k-1) - k = -1$.

■

3.3.2 Existence and Construction of Potential Functions

Although potential functions are often difficult to find, it is not difficult to show that they always exist. [2] Consider an arbitrary metrical task system and the graph G constructed as follows. There is a vertex corresponding to each pair (s_A, s_{opt}) where s_A is a state of the online

[2] This has been observed independently by several researchers. The presentation here is due to D.D. Sleator.

algorithm A, *including* all its variables, and s_{opt} is a state of OPT. There is an edge e from (s_1, s_2) to (s_3, s_4) if there is a task T such that A services T in state s_1 by moving to s_3, and if OPT *can* service T in state s_2 by moving to s_4.

Suppose that "c" is the competitive ratio we are trying to prove. Then, the edge e is labelled with a cost, denoted $cost(e)$, equal to c times the adversary cost for the task minus the on-line cost, i.e.

$$c(e) = c * (d(s_2, s_4) + T(s_4)) - (d(s_1, s_3) + T(s_3)).$$

Let s_0 be the starting state of the algorithms A and OPT, and let G' be the subgraph of G reachable from (s_0, s_0). If the algorithm A is c-competitive, the graph G' cannot have a negative cycle. If it does, then there is a sequence of requests and a behavior of the adversary such that around this cycle, the adversary cost is less than $1/c$ times the on-line algorithm, which violates c-competitiveness.

It is a standard trick (see [Tar83]) that in a directed graph with no negative cycles, there exists a labeling of the vertices with a potential $\Phi(v)$ such that if the cost of an edge (v, w) is replaced by:

$$newcost(v, w) = cost(v, w) + \Phi(v) - \Phi(w)$$

then all the new costs are non-negative. In fact, it is easy to construct such a potential function. Choose any starting vertex v (from which all vertices can be reached), and choose $\Phi(v) = 0$. Then for each vertex w, the value of $\Phi(w)$ is the length of the shortest path in the graph from v to w, where the length of a path is the sum of the costs of the edges in the path. This Φ is of course the potential function needed to prove the c-competitiveness of the on-line algorithm.

The above formulation allows the use of linear programming to directly derive the minimum value of c. Let c and the $\Phi(v)$'s be the variables of a linear program. Then we wish to Minimize c under the constraint that the *newcost*s are all non-negative. This approach may be impractical, however, since the graph G' may be too large. Nonetheless, variants of this idea have been used to prove bounds for small instances of the k-server problem (see [KMMO90, LR94]).

In general, potential functions have been found by trial and error. In most cases, the potential functions is a linear combinations of certain simple functionals measuring the distance between the online and the offline paper. Bern, Greene and Ragunathan automate part of the task of designing a potential function for problems in online cache sharing by using linear programming to find the optimal coefficients once their choice of components had been determined (see [BGR93]).

4 Theoretical Underpinnings: Randomized Algorithms

A randomized online algorithm is an online algorithm that makes random choices as it responds to requests. Since the cost incurred by a randomized algorithm on a request sequence depends on the results of the coin tosses the algorithm makes, it is a random variable. Therefore, the performance of such an algorithm is assessed by considering its expected cost on a request sequence.

For randomized online algorithms, there is no unique notion of an adversary. The issue which distinguishes different adversaries is the extent to which they know the outcomes of random choices made by the algorithm and how they themselves service the sequence they generate.

We divide adversaries into two types:

- An *oblivious adversary* must choose the entire request sequence σ, without any knowledge either of the outcome of the coin tosses or of the specific actions taken as a result of the coin tosses. However, the oblivious adversary does know the algorithm itself including the probability distribution of actions taken for a given input.
- An *adaptive adversary* is one that chooses each request in the input string σ based on knowledge of all actions taken so far, and of the outcome of all previous coin tosses.

Under the category of adaptive adversaries, one can distinguish between two types of adversaries, depending on how the adversary itself serves the sequence.

- An *adaptive offline* adversary waits until the entire sequence σ has been generated and services it optimally. In other words, the cost incurred by this adversary is the optimal offline cost on the entire sequence σ.
- An *adaptive online* adversary must itself serve the sequence σ that it generates in an *online* fashion.

Competitiveness is then defined as follows: A randomized online algorithm A is *c-competitive against oblivious (resp. adaptive online, adaptive offline) adversaries* if there exists an α such that for all oblivious (resp. adaptive online, adaptive offline) adversaries, ADV,

$$E[\texttt{cost}_A(\sigma) - c \cdot \texttt{cost}_{ADV}(\sigma)] \leq \alpha$$

where σ is the sequence generated by the adversary in its interaction with A and $\texttt{cost}_{ADV}(\sigma)$ is the cost incurred by the adversary. The *competitive ratio of a randomized algorithm against adversaries of type ADV* is

$$c_A^{ADV} \equiv \inf_c \{c | A \text{ is } c\text{-competitive against all adversaries of type } ADV\}$$

Here the expectation is taken over all choices made by the online algorithm. Since the actions of an oblivious adversary can not depend on any of the random choices of the algorithm, its behaviour is deterministic; the sequence σ it generates as well as its cost $\texttt{cost}_{OBL}(\sigma)$ is not a random variable. A randomized algorithm A is c-competitive against all oblivious adversaries if there is a constant b such that for all input sequences σ,

$$E[\texttt{cost}_A(\sigma)] \leq c \cdot \texttt{cost}_{OPT}(\sigma) + b.$$

For the other two adversaries, σ depends on the coin tosses of A. This means that the costs $\texttt{cost}_{ADOFF}(\sigma)$ and $\texttt{cost}_{ADON}(\sigma)$ are themselves random variables, so the expectation calculated above applies to the adversary cost as well.

The least powerful of the adversaries is the oblivious, because an adaptive adversary has the option of deciding the request sequence in advance and answering it optimally. The most powerful of the adaptive adversaries is the adaptive offline since the adaptive offline adversary can always choose to service the sequence online. Let $\mathcal{C}_{OBL}$ (respectively $\mathcal{C}_{ADON}, \mathcal{C}_{ADOFF}$) be the best competitive ratio achievable against any oblivious (respectively adaptive online, adaptive offline) adversary on a problem. That is $\mathcal{C}_{OBL} = \inf_A c_A^{OBL}$, $\mathcal{C}_{ADON} = \inf_A c_A^{ADON}$, and $\mathcal{C}_{ADOFF} = \inf_A c_A^{ADOFF}$. Let $\mathcal{C}_{DET}$ be the best competitive ratio achievable by a deterministic algorithm for the task system. We have that:

$$\mathcal{C}_{OBL} \leq \mathcal{C}_{ADON} \leq \mathcal{C}_{ADOFF} \leq \mathcal{C}_{DET}.$$

The distinction between the three adversaries was made by Raghavan and Snir in [RS89] and by Ben-David, Borodin, Karp, Tardos and Wigderson in [BDBK+90].

4.1 Example: Paging

4.1.1 Memoryless randomized paging

The lower bound of k for the competitive ratio of any deterministic paging algorithm shown in Section 3.1 can easily be extended to apply to randomized algorithms against an adaptive online adversary. Thus, no advantage can be gained in using randomization against an adaptive adversary. It is natural then to ask whether a randomized algorithm fares better against an oblivious adversary. The difference is, in fact, quite dramatic as we shall see later in this section. First, we begin our discussion of randomized paging algorithms with the simplest randomized algorithm: on a fault, evict a random page from the cache. This is a memoryless algorithm: the only information used in making replacement decisions is which pages are currently in the cache.

It turns out that this particular use of randomization does not yield a lower competitive ratio than that achieved by the deterministic algorithms we have discussed, even against an oblivious adversary: this algorithm is k-competitive against any oblivious adversary. The lower bound is a special case of the following theorem proved by Coppersmith, Doyle, Raghavan and Snir[CDRS93].

Theorem 7 *Any memoryless k-server algorithm A on any metric space has competitive ratio at least k against an oblivious adversary.*

4.1.2 The Randomized Marking Algorithm

Although no memoryless randomized algorithm has competitiveness below k against oblivious adversaries, there do exist nonmemoryless randomized algorithms which beat the deterministic bound. Fiat, Karp, Luby, McGeogh and Young [FKL+91] were able to show that a randomized version of the marking algorithm has competitive ratio $2H_k$ against an oblivious adversary, where H_k is the k^{th} harmonic number. The algorithm, called RMA (for Randomized Marking Algorithm), works as follows:

Initially all pages are unmarked.

- If there is a request for a page not in the cache, bring the page in, replacing a random unmarked page and mark the new page. (If all pages were already marked, unmark all of them first.)
- If the request is for a page that is in the cache then mark it.

Theorem 8 *The randomized marking algorithm RMA has competitive ratio $2H_k$ against any oblivious adversary, where H_k is the k^{th} harmonic number.*

Proof. Assume OPT and RMA start with the same cache contents. As before we divide the sequence σ into phases. The i^{th} phase ends immediately before the $k+1^{st}$ distinct page is requested in the phase. We will analyze the cost of both OPT and RMA phase by phase. Note that once a page is marked, it is not evicted from the cache for the remainder of the phase. Therefore, if we denote the set of pages requested in phase i by P_i, then at the end of a phase i, the contents of RMA's cache is exactly P_i. Furthermore, RMA will not fault twice on the same page within a phase. Thus, we need only account for faults incurred on the first request to any given page in a phase.

Let m_i be the number of *new* requests in phase i, (i.e. the number of pages requested in phase i that were not requested in phase $i-1$). A requested in the first phase is also new if it is not one of the pages initially in the cache. Since any new page is not in RMA's cache at the beginning pf a phase, RMA must fault once on every new page requested. Now we must analyze the expected number of faults on requests to *old* pages (i.e., pages which are not new). What is the probability that RMA faults on the j^{th} old page requested? Let's suppose that just before the j^{th} old page is requested, there have been ℓ new pages requested so far in the phase. It is easy to show by induction that at this time there are exactly ℓ pages in P_{i-1} that are not in the cache. Furthermore, these are distributed uniformly at random among the $k-(j-1)$ unmarked pages in P_{i-1}. Since the adversary has fixed the request in advance, the probability that it is not in the cache is exactly $\ell/k-(j-1)$. Since ℓ is always at most m_i, the probability of a fault on the request to the j^{th} old page is at most

$$\frac{m_i}{k-(j-1)}.$$

Therefore, the expected cost of the marking algorithm in the i^{th} phase is

$$\begin{aligned} E[\texttt{cost}_{RMA_i}(\sigma)] &\leq m_i + \sum_{1\leq i\leq k-m_i} \frac{m_i}{k-i+1} \\ &\leq m_i H_k. \end{aligned}$$

Summing up over all phases, we get that the expected cost for the algorithm over the entire sequence is

$$E[\texttt{cost}_{RMA}(\sigma)] \leq H_k \sum_i m_i.$$

For the offline algorithm we claim that

$$\texttt{cost}_{OPT}(\sigma) \geq \sum_i \frac{m_i}{2}.$$

Consider the $(i-1)^{st}$ and i^{th} phases. The number of distinct pages requested in both phases is $k+m_i$. Since OPT has only k pages in the cache at the beginning of the $(i-1)^{st}$ phase, it must incur at least m_i faults during the two phases. Applying this argument to every pair of adjacent phases, we have that $\texttt{cost}_{OPT}(\sigma) \geq \sum_i m_{2i}$ and $\texttt{cost}_{OPT}(\sigma) \geq \sum_i m_{2i+1}$. Therefore OPT has cost at least the average of these, i.e. $\texttt{cost}_{OPT}(\sigma) \geq \sum_i m_i/2$. Thus, $E[\texttt{cost}_{RMA}(\sigma)] \leq 2H_k \texttt{cost}_{OPT}(\sigma)$. ■

A much more complicated algorithm was shown to be H_k competitive by McGeoch and Sleator[MS91]

4.2 Lower Bounds

A common technique for proving lower bounds on the competitive ratio of a randomized online algorithm against oblivious adversaries is to examine the performance of deterministic algorithms on inputs from a given probability distribution $\mathcal{P}$ over request sequences, σ.

Definition 9 *Let $\mathcal{P}$ be a probability distribution on request sequences σ. An algorithm A is* c-competitive against $\mathcal{P}$ *if there exists a constant, b, such that*

$$E_{\mathcal{P}}(\texttt{cost}_A(\sigma)) \leq c \cdot E_{\mathcal{P}}(\texttt{cost}_{OPT}(\sigma)) + b$$

Definition 10 *Let $c_A^{\mathcal{P}}$ denote the infimum over all c such that A is c-competitive against $\mathcal{P}$, and let c_R^{OBL} denote the competitive ratio of a randomized algorithm R against OBL.*

The basic theorem that allows us to prove lower bounds for randomized online algorithms is the following.

Theorem 11

$$\inf_R c_R^{\mathrm{OBL}} = \sup_{\mathcal{P}} \inf_A c_A^{\mathcal{P}}.$$

In other words, the competitive ratio of the best randomized algorithm against oblivious adversaries is equal to the competitive ratio of the best deterministic algorithm, A, on inputs generated from the "worst" probability distribution. The proof of this theorem follows from the minimax theorem of game theory and was first observed by Yao [Yao80] in the context of ordinary complexity theory, and by Borodin, Linial and Saks [BLS87] in the context of online algorithms.

4.2.1 Example: Paging

We now illustrate the use of Theorem 11 for proving lower bounds on c_R^{OBL} by showing a lower bound for randomized paging algorithms against an oblivious adversary. The bound comes close to matching the upper bound shown in Section 4.1.2. The following theorem is due to Fiat, Karp, Luby, McGeoch, Sleator and Young [FKL+91].

Theorem 12 *Let R be any randomized paging algorithm. If the number of pages is greater than or equal to $k+1$, where k is the size of the cache, the competitive ratio of R against any oblivious adversary is greater than or equal to H_k.*

Proof. We will find our lower bound on c_R^{OBL} by exhibiting a probability distribution $\mathcal{P}$ for which $c_A^{\mathcal{P}} \geq H_k$ for all deterministic algorithms A. Let S be a set of $k+1$ pages which include the k pages initially in the cache. Take $\mathcal{P}$ to be the uniform distribution on the $k+1$ pages in S. That is, a sequence σ of m requests is generated by independently selecting each request at random from S. Clearly, the expected performance of any deterministic algorithm on inputs generated from this distribution is

$$E_{\mathcal{P}}[\mathtt{cost}_A(\sigma)] = \frac{m}{k+1} \tag{1}$$

for $|\sigma| = m$. To see this, note that the probability that a given requested page is not in the cache is $\frac{1}{k+1}$. We now need an upper bound on the expected performance of OPT.

Once again, we divide the sequence of page requests into non-overlapping phases such that each phase contains maximal runs of requests to at most k distinct pages. As we have seen, if there are r distinct phases then the optimal algorithm can service the sequence with at most

$r+1$ faults. At the beginning of a phase OPT replaces the one page currently in the cache that will not be requested in the phase. Therefore, if $N(m)$ is the random variable which is the number of phases in a sequence σ of length m (generated from $\mathcal{P}$), then the expected offline cost satisfies

$$E_{\mathcal{P}}(\texttt{cost}_{OPT}) \leq E(N(m)+1). \tag{2}$$

Since the durations of successive phases, are independent, identically distributed random variables, we have by the elementary renewal theorem [3] that

$$\lim_{m\to\infty} \frac{m}{E(N(m))} = E(X_i),$$

where $E(X_i)$ is the expected length of the i^{th} phase.

The expected length of a phase, $E(X_i)$ is easily seen to be $(k+1)H_k$ (this is a so-called "coupon collectors problem".)

Therefore, we have that

$$\begin{aligned}
\lim_{m\to\infty} \frac{E_{\mathcal{P}}(\texttt{cost}_A)}{E_{\mathcal{P}}(\texttt{cost}_{OPT})} &\geq \frac{m/k+1}{m/E(X_i)} \\
&\geq \frac{E(X_i)}{k+1} \\
&= \frac{(k+1)\cdot H_k}{k+1} \\
&= H_k.
\end{aligned}$$

Applying Theorem 11 yields a lower bound of H_k on c_R^{OBL}.

4.3 The relationships between the adversaries

The power of a randomized online algorithm is measured by the competitive ratio it achieves and the strength of the adversary it plays against. The relationships between the three adversaries we have described was studied by Ben-David, Borodin, Karp, Tardos and Wigderson [BDBK$^+$90] in a general model called *request-answer games*. Practically every online problem can be formulated as a request-answer game. We do not define this model here; for further details see [BDBK$^+$90]. For the purposes of the subsequent discussion we present the results of [BDBK$^+$90] for online problems which can be posed as metrical task systems (a more restrictive model than request-answer games).

[3] For a detailed treatment of renewal theory, see for example Feller's book [Fel50] or almost any book on probability theory and stochastic processes.

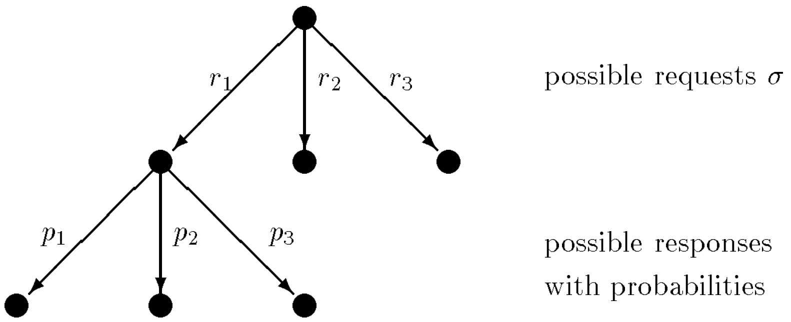

Figure 1: Game tree description of interaction between the randomized online algorithm and adversary

We review the definitions of the three adversaries in a game-theoretic context: the interaction between an adversary and the online algorithm can be viewed as a game. Consider the game tree where the root node has level 0 and represents the starting state of both algorithm and adversary. Nodes at level $2i$ represent possible states of the online algorithm after i requests have been processed. Branches from nodes at an even level to a node at the next level down represent the possible choices of requests the adversary can make. Each such arc is labeled with the name of the request. Branches from nodes at an odd level to a node at the next level down represent the possible ways the online algorithm can serve the request in its current state. Each such arc is labeled with the probability that the online algorithm will take it. Figure 4.3 gives an example of part of such a tree.

The fact that all the adversaries know the online algorithm and the probability distributions it uses means that all adversaries know the entire game tree in advance. Each adversary then consists of a requesting component and a servicing component. The requesting component is a choice of one request r_v from each node v at an even level in the tree. In other words, if the online algorithm reaches v, the next request that adversary gives it is r_v. The requesting component of an adversary can be expressed as a pruned game tree where every node at an even level has exactly one child. The different adversaries are then distinguished as follows:

- For an *oblivious* adversary, for all v at level $2i$ in the tree r_v is the same, so that no matter what the state of the algorithm is, the i^{th} request is the same. In other words, the adversary is oblivious to the random choices (and hence the exact state) of the online algorithm in generating the next request. The oblivious adversary services the resulting sequence optimally.

- *adaptive* adversaries may choose any request from any even-leveled node in the tree. It is adaptive because each request r_v is tailored to the node v itself, i.e. the decision of the next request to give depends on the history and state of the online algorithm.

- An *adaptive online* adversary is further restricted to make a choice as to how it would serve that request at the time it makes the request, i.e. without knowledge of which path in the game tree will be taken from that node (or equivalently without knowledge of future coin tosses of the online algorithm)
- An *adaptive offline* adversary serves the sequence of requests optimally at the end.

4.3.1 Adaptive Offline versus Adaptive Online and Oblivious

We saw that the use of randomization against an oblivious adversary gave us an *exponential* improvement in the competitive ratio for paging, relative to deterministic algorithms. In fact, the performance of a randomized online algorithm depends on the adversary against whom it plays. Nevertheless, there are certain relationships between the performance of algorithms against different adversaries. The first result of Ben-David *et al.* shows that the existence of algorithms, one of which does "well" against any oblivious adversary and one of which does "well" against any adaptive online adversary (not necessarily the same algorithm) implies the existence of an algorithm that does "well" against any adaptive offline adversary.

Theorem 13 *Let G be an online algorithm that has a competitive ratio of α against adaptive online adversaries and suppose that there is existis an online algorithm H which has a competitive ratio of β against oblivious adversaries Then G is $\alpha\beta$-competitive against adaptive offline adversaries.*

This theorem is tight in the sense that there exist G and H as in the theorem such that G is no better than $\alpha\beta$ competitive against adaptive offline adversaries: we discussed a randomized paging algorithm that has competitive ratio H_k against oblivious adversaries. In addition, the randomized paging algorithm which evicts a random page in the cache on a page fault, has competitive ratio k against adaptive online adversaries. Furthermore, the competitive ratio of the random eviction algorithm is no better than kH_k. The above theorem implies that the competitive ratio of this latter algorithm against adaptive offline adversaries is at most kH_k and this is indeed the case. For paging, however, there exist other algorithms (even deterministic) which are k competitive against adaptive offline adversaries.

It is possible to show, though, that for any α and β, there is an online problem such that there is an algorithm which is α-competitive against any adaptive online adversary and there is a β-competitive algorithm against any oblivious adversary but there is no algorithm that has a competitive ratio less than $\alpha\beta$ against every adaptive offline adversary. The proof of this is not presented here, but can be found in [BDBK+90]. One interesting aspect of the proof is that the problem for which the property holds is not a metrical task system. It is not known if the analogous result holds for metrical task systems. In fact, it is not even known if there is a metrical task system for which there exists an algorithm whose competitive ratio against adaptive online adversaries is strictly less than the competitive ratio of the best algorithm against adaptive offline adversaries.

4.3.2 Adaptive Offline versus Deterministic

Since the adaptive offline adversary is the most powerful of the three that we have considered for randomized algorithms, an interesting question is whether online algorithms are aided by randomization at all when pitted against such an adversary. Intuitively, one expects the answer to be no, and the next theorem shows that this is indeed the case.

Theorem 14 *Let G be an α-competitive online algorithm against $ADOFF$. Then there is a deterministic algorithm that has competitive ratio α.*

In general, Theorem 14 is not constructive. Deng and Mahajan show in [DM91] that there are games for which there is a simple and computable c-competitive randomized algorithm against adaptive online adversaries for some c but any deterministic algorithm which achieves any bounded competitive ratio is not computable. However, Ben-David *et al.* in [BDBK+90] show that for a wide class of randomized algorithms, the corresponding deterministic algorithm is indeed computable. Theorems 14 and 13 can be combined to give the following two corollaries:

Corollary 15 *Let G be a randomized algorithm that is α-competitive against any adaptive online adversary. If there is a β-competitive algorithm against an oblivious adversary, then there exists a deterministic $(\alpha \cdot \beta)$-competitive algorithm.*

Corollary 16 *Let G be a randomized algorithm that is α-competitive against any adaptive online adversary. There exists a deterministic (α^2)-competitive algorithm.*

5 The k-server problem revisited

5.1 History

The k-server problem was posed in 1988 by Manasse, McGeoch and Sleator [MMS90]. They proved a lower bound of k on the competitive ratio, and an upper bound of k for the special cases where $k = 2$ and where $n = k + 1$. They conjectured that there is a k-competitive algorithm for any metric space. Since then proving k-server conjecture has been a primary focus for researchers in online algorithms. For some time, it was even unknown whether there was an algorithm that achieves a competitiveness which can be bounded by any function of k on all metric spaces. In response to this, researchers focused on algorithms for specific types of metric spaces (e.g. trees, the circle) [CKPV91, CL91b, CDRS93, Kar, AKPW92]. Finally, in 1990, Fiat, Rabani & Ravid [FRR90] proved that a certain algorithm has competitive ratio which was a function of k only, albeit rather large: $O(k^k)$! This was improved by Grove in

1991 [Gro91] who showed that a simple and natural randomized algorithm has competitive ratio $O(k2^k)$.

In 1994, Koutsoupias & Papadimitriou [KP94b] proved that an algorithm that had long been conjectured [CL92] to be strongly competitive is $(2k-1)$-competitive. This is undoubtedly the most celebrated result in the area of online algorithms to date. In this section, we present the key features of their proof.

5.2 Notation and Properties of Work Functions.

We use the following notation:

- S is the set of points in the underlying metric space.
- S^k is the set of all size k multisets of points in S.
- $\forall X, Y \in S^k$, $D(X, Y)$ is the minimum cost of moving from X to Y (i.e. the minimum matching between the points in X and the points in Y.)
- σ is the request sequence, and σ_t is the t^{th} request in σ ($\sigma_t \in S$).
- $\mathrm{opt}_t(\sigma, X)$ is the minimum cost of serving the first t requests in σ, ending up in state X, assuming that X_0 is the starting configuration.

$\mathrm{opt}_t(\sigma, X)$ can be computed using dynamic programming as follows:

$$\mathrm{opt}_0(\sigma, X) = D(X, X_0).$$

If $\sigma_t \in X$,

$$\mathrm{opt}_t(\sigma, X) = \mathrm{opt}_{t-1}(\sigma, X).$$

If $\sigma_t \notin X$,

$$\mathrm{opt}_t(\sigma, X) = \min_{Y|\sigma_t \in Y} \left[\mathrm{opt}_t(\sigma, Y) + D(X, Y)\right].$$

These two cases can be combined to give

$$\mathrm{opt}_t(\sigma, X) = \min_{Y|\sigma_t \in Y} \left[\mathrm{opt}_{t-1}(\sigma, Y) + D(Y, X)\right].$$

The functions $\mathrm{opt}_t(\sigma, \cdot)$ are typically called *work functions*, and are usually denoted by w_t. Our change of notation is in order to emphasize the fact that they represent the optimal cost of servicing a sequence of requests.

We will need the following easy-to-prove properties of work functions:

1. $\forall X, Y$, $\mathrm{opt}_t(\sigma, X) \leq \mathrm{opt}_t(\sigma, Y) + D(Y, X)$.

2. $\text{opt}_t(\sigma, X) = \min_{x \in X} [\text{opt}_t(\sigma, X - x + \sigma_t) + d(x, \sigma_t)]$, since

$$\begin{aligned}
\text{opt}_t(\sigma, X) &= \min_{Y|\sigma_t \in Y} [\text{opt}_t(\sigma, Y) + D(X, Y)] \\
&= \min_{x \in X} [\text{opt}_t(\sigma, Y) + d(\sigma_t, x) + D(X - x + \sigma_t, Y)] \\
&\geq \min_{x \in X} [\text{opt}_t(\sigma, X - x + \sigma_t) + d(\sigma_t, x)] .
\end{aligned}$$

3. $\text{opt}_t(\sigma, X) \geq \text{opt}_{t-1}(\sigma, X)$.

5.3 The Work Function Algorithm WFA

We review the work function algorithm described earlier. Suppose that after $t - 1$ requests, the algorithm is in state X_{t-1}, and the next request σ_t is r.

The Work Function Algorithm serves the request by moving to the configuration X with $r \in X$ that minimizes

$$\underbrace{\text{opt}_t(\sigma, X)}_{\textit{retrospective}} + \underbrace{D(X, X_{t-1})}_{\textit{local greedy}}$$

This algorithm is a combination of two natural algorithms neither of which is competitive by itself. The second term corresponds to the *local greedy* algorithm: the one which serves the request in the cheapest possible way, ignoring history altogether. The first term corresponds to the *retrospective* algorithm, the one which tries to chase the optimal offline algorithm (by moving into a state which minimizes the OPT cost so far). The problem with the latter is that it may be very expensive to chase OPT; an algorithm which does this may find that this expense was unnecessary only a few requests later. Thus, the second term prevents the algorithm from incurring large local costs in attempting to mimic OPT's behavior.

Using property 3, we can observe that the state X which minimizes the above equation only differs from the algorithm's present configuration X_{t-1} in the location of one server. Thus, WFA serves the request by moving a server at x to r where x minimizes

$$\min_{x \in X} [\text{opt}_t(\sigma, X_{t-1} + r - x) + d(r, x)] = \text{opt}_t(\sigma, X_{t-1})$$

Note that $X_{t-1} + r - x = X_t$, and so we have

$$\min_{x \in X} [\text{opt}_t(\sigma, X_t) + d(r, x)] = \text{opt}_t(\sigma, X_{t-1}) \tag{3}$$

A more general version of this algorithm was initially proposed by Borodin, Linial and Saks [BLS87] in 1987 for metrical task systems. It was subsequently studied extensively by

Chrobak and Larmore [CL92], for the special case of the k-server problem. Their experiments suggested that it is k-competitive which they were able to prove for some special cases. They also introduced some ideas for approaching the problem that ultimately turned out to be important. We now present the 1994 result of Koutsoupias and Papadimitriou [DK93] showing that there exists a constant c, such that for any input σ,

$$WFA(\sigma) \leq (2k-1)\mathrm{OPT}(\sigma) + c. \tag{4}$$

5.4 Proof of $(2k-1)$-Competitiveness

We will be able to assume that the online and offline algorithms start and end in the same configuration. Indeed, suppose that X_f is the final configuration of OPT, and WFA finishes serving the sequence in a different configuration from X_f. Then we can append the sequence σ with requests to points in X_f until WFA moves to X_f. (Any algorithm which doesn't eventually move it's servers to X_f has no bounded competitive ratio.) The cost of WFA on the appended sequence is greater than or equal to the cost of WFA on the original sequence, and the cost of OPT on the appended sequence is equal to the cost of OPT on the original, therefore it is sufficient to prove equation 4 for the appended sequence.

5.4.1 Offline Pseudo-Cost

Define the *offline pseudo-cost* or amortized offline cost, $\widetilde{\mathrm{opt}}_t$ on the t^{th} step to be

$$\widetilde{\mathrm{opt}}_t = \mathrm{opt}_t(\sigma, X_t) - \mathrm{opt}_{t-1}(\sigma, X_{t-1}).$$

This is the amortized offline cost for the t^{th} step, since

$$\sum_t \widetilde{\mathrm{opt}}_t = \mathrm{opt}_f(\sigma, X_f) - \mathrm{opt}_0(\sigma, X_0) = \mathtt{cost}_{OPT}(\sigma).$$

We bound WFA_t, the cost incurred by the work function algorithm on the t^{th} request, plus $\widetilde{\mathrm{opt}}_t$, the amortized offline cost for the t^{th} request.

Suppose that WFA moves from x to r in order to service σ_t. Then

$$\begin{aligned} WFA_t + \widetilde{\mathrm{opt}}_t &= d(x,r) + \mathrm{opt}_t(\sigma, X_t) - \mathrm{opt}_{t-1}(\sigma, X_{t-1}) \\ &= \mathrm{opt}_t(\sigma, X_{t-1}) - \mathrm{opt}_{t-1}(\sigma, X_{t-1}) \\ &\leq \max_X \left[\mathrm{opt}_t(\sigma, X) - \mathrm{opt}_{t-1}(\sigma, X)\right]. \end{aligned}$$

where the second equation follows from equaiont (3).

Let EC_t, the *extended cost* of the t^{th} request, be defined by

$$EC_t = \max_X \left[\text{opt}_t(\sigma, X) - \text{opt}_{t-1}(\sigma, X)\right].$$

We have that

$$WFA_t + \widetilde{\text{opt}}_t \leq EC_t.$$

The proof will proceed by proving an upper bound on the extended cost of the t^{th} request. The advantage of working with extended costs is that it allows us to entirely ignore the real configuration of the online algorithm. We have simply to show that a certain inequality holds for all work functions. Of course, a potential disadvantage of approaching the problem this way is that we may be overestimating the cost incurred by WFA.

5.4.2 Approach

We seek a potential function $\Phi : X \to \mathcal{R}^+$ satisfying the following two conditions:

$$WFA_t + \widetilde{\text{opt}}_t \leq EC_t \leq \Phi_t - \Phi_{t-1}, \tag{5}$$

and

$$\Phi_f - \Phi_0 \leq (k+1)\texttt{cost}_{OPT}(\sigma) + c. \tag{6}$$

The existence of such a potential function implies that

$$\begin{aligned}
\texttt{cost}_{WFA}(\sigma) + \texttt{cost}_{OPT}(\sigma) &= \sum_t (WFA_t + \widetilde{\text{opt}}_t) \\
&\leq \sum_t EC_t = \sum_t \max_X \left[\text{opt}_t(\sigma, X) - \text{opt}_{t-1}(\sigma, X)\right] \\
&\leq \sum_t (\Phi_t - \Phi_{t-1}) \\
&= \Phi_f - \Phi_0 \\
&\leq (k+1)\texttt{cost}_{OPT}(\sigma) + c.
\end{aligned}$$

which would mean that WFA is k-competitive.

5.4.3 The case $n = k+1$

For the special case of $n = k+1$, the approach just outlined works easily. Take

$$\Phi_t = \sum_X \text{opt}_t(\sigma, X).$$

Then

$$\Delta\Phi = \Phi_t - \Phi_{t-1} \geq \sum_X \left[\text{opt}_t(\sigma, X) - \text{opt}_{t-1}(\sigma, X)\right] \geq EC_t,$$

and

$$
\begin{aligned}
\Phi_f - \Phi_0 &\leq \sum_X \mathrm{opt}_f(\sigma, X) \\
&\leq \sum_X \left[\mathrm{opt}_f(\sigma, X_f) + D(X, X_f)\right] \\
&\leq (k+1)\left[\mathrm{opt}_t(\sigma, X_f)\right] + max_{i,j}\left[(k+1)d(i,j)\right] \\
&= (k+1)\mathtt{cost}_{OPT}(\sigma) + c.
\end{aligned}
$$

Thus we have the conditions we need to show that WFA is k-competitive. Unfortunately, this potential function gives a very poor bound on the competitive ratio when $n > k+1$.

5.5 The Duality Lemma

In order to find a potential funciton satisfying conditions 5 and 6, we must get a handle on the properties of the configuration where the extended cost is achieved. Intuitively, we expect the extended cost to be achieved on an X with the following properties.

1. Points in X must be fairly far from r. If they are very close, or even worse $r \in X$, then $\mathrm{opt}_t(\sigma, X)$ will not be significantly larger than $\mathrm{opt}_{t-1}(\sigma, X)$.
2. $\mathrm{opt}_{t-1}(\sigma, X)$ must not be too large. Indeed, let Z be a configuration such that $r \in Z$. We have that $\mathrm{opt}_{t-1}(\sigma, X) \leq \mathrm{opt}_{t-1}(\sigma, Z) + D(X, Z)$. We claim that this inequality must be strict. Otherwise, if $\mathrm{opt}_{t-1}(\sigma, X) = \mathrm{opt}_{t-1}(\sigma, Z) + D(X, Z)$, then $\mathrm{opt}_t(\sigma, X) = \min_{Y: r \in Y}(\mathrm{opt}_{t-1}(\sigma, Y) + D(X, Y)) \leq \mathrm{opt}_{t-1}(\sigma, Z) + D(X, Z) = \mathrm{opt}_{t-1}(\sigma, X)$. Since work functions cannot decrease, this would imply that $\mathrm{opt}_t(\sigma, X) = \mathrm{opt}_{t-1}(\sigma, X)$, contradicting the assumption that the extended cost is achieved on X.

These two observations motivate the following definition. A configuration X is said to be a *minimizer of r with respect to $opt_{t-1}(\sigma, \cdot)$* if X minimizes:

$$
\mathrm{opt}_{t-1}(\sigma, X) - \sum_{x \in X} d(r, x).
$$

The first term corresponds to the second property, and the second term corresponds to the first property. The crucial lemma conveying this intuition is the duality lemma, so named because it relates a maximum (the extended cost) to a minimum (a minimizer).

Lemma 17 *Let the configuration X be a minimizer of r with respect to $opt_{t-1}(\sigma, \cdot)$. Then:*

- *The extended cost of serving r at the t^{th} step occurs on X.*

- *If X is a minimizer of r with respect to $opt_{t-1}(\sigma, \cdot)$, then X is also a minimizer of r with respect to $opt_t(\sigma, \cdot)$.*

We defer a discussion of the proof of the duality lemma to section 5.7.

5.6 The potential function

We now have the machinery in place to find a potential function whose change at time t is an upper bound on the extended cost of the t^{th} step i.e., we will use the duality lemma to find a Φ such that $\Phi_t - \Phi_{t-1} \geq EC_t$, and such that $\Phi_f - \Phi_0 \leq (2k)OPT(\sigma) + c$. We use the duality lemma in order to define a potential function satisfying these conditions.

Define $k+1$ configurations $U, B_1, B_2, \ldots, B_k$ as follows:

$$U = (u_1, u_2, \ldots, u_k)$$

and

$$B_i = (b_{i1}, b_{i2}, \ldots, b_{ik}), \ i = 1, \ldots, k.$$

We define the potential function in terms of a function ψ where

$$\psi(\mathrm{opt}_t, U, B_1, B_2, \ldots, B_k) = k\mathrm{opt}_t(\sigma, U) + \sum_{i=1}^{k} \left[\mathrm{opt}_t(\sigma, B_i) - \sum_{j=1}^{k} d(u_i, b_{ij}) \right] \tag{7}$$

The potential function ϕ_t is then defined as the minimum value of ψ over all possible configurations U and $B_1, B_2, \ldots, B_k$.

$$\phi_t = \min_{U, B_1, \ldots, B_k} \psi(\mathrm{opt}_t, U, B_1, \ldots, B_k) \tag{8}$$

The following lemma shows that we can assume that the U for which ϕ_t is minimized contains the most recent request $\sigma_t = r$, and that one of the configurations B_i for which ϕ_t is minimized is a minimizer of $\sigma_t = r$ with respect to opt_{t-1}.

Lemma 18 *Assume ϕ_t occurs at configurations $\tilde{U}, \tilde{B}_1, \ldots, \tilde{B}_k$. Let A be a minimizer of $\sigma_t = r$ with respect to opt_{t-1}. Then $\phi_t \geq \psi_t$ where $\psi_t = \psi(opt_t, \tilde{U} - u_i + r, \tilde{B}_1, \ldots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \ldots, \tilde{B}_k)$.*

Proof.

$$\begin{aligned} \phi_t &= \psi(\mathrm{opt}_t, \tilde{U}, \tilde{B}_1, \ldots, \tilde{B}_k) \\ &= k\mathrm{opt}_t(\sigma, \tilde{U}) + \sum_{l=1}^{k} \left[\mathrm{opt}_t(\sigma, \tilde{B}_l) - \sum_{j=1}^{k} d(u_l, \tilde{b}_{lj}) \right] \end{aligned} \tag{9}$$

Since $\mathrm{opt}_t(\sigma, \tilde{U}) = \min_{u \in \tilde{U}} \left[\mathrm{opt}_t(\sigma, \tilde{U} - u + r) + d(u, r) \right]$ by property 2, there is a point, call it u_i, where such a minimum occurs. Substituting $\mathrm{opt}_t(\sigma, \tilde{U}) = \mathrm{opt}_t(\sigma, \tilde{U} - u_i + r) + d(u_i, r)$ into

Equation 9 and applying the triangle inequalities $d(u_i, r) - d(r, b_{ij}) \geq -d(r, b_{ij})$ for $j = 1, \ldots, k$, yields the fact that

$$
\begin{aligned}
\phi_t &= \psi(\mathrm{opt}_t, \tilde{U} - u_i + r, \tilde{B}_1, \ldots, \tilde{B}_k) \\
&\geq \psi(\mathrm{opt}_t, \tilde{U} - u_i + r, \tilde{B}_1, \ldots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \ldots, \tilde{B}_k).
\end{aligned}
$$

The last inequality comes from the fact that A is a minimizer of r with respect to opt_t and so $\mathrm{opt}_t(\sigma, \tilde{B}_i) - \sum_{j=1}^{k} d(r, \tilde{b}_{ij})$ is minimized when $B_i = A$. ∎

Lemma 19 *The change in the potential function at time t is an upper bound on the extended cost at time t, i.e.*

$$
EC_t \leq \phi_t - \phi_{t-1}
$$

Proof. Suppose that ϕ_t occurs at configurations $\tilde{U}, \ldots, \tilde{B}_k$, Let A a minimizer of r with respect to opt_{t-1}. Then $\phi_t \geq \psi_t$ by Lemma 18. Consequently, $\phi_t - \phi_{t-1} \geq \psi_t - \psi_{t-1}$. Therefore,

$$
\begin{aligned}
\phi_t - \phi_{t-1} &\geq \psi_t - \psi_{t-1} \\
&\geq k\left[\mathrm{opt}_t(\sigma, \tilde{U} - u_i + r) - \mathrm{opt}_{t-1}(\sigma, \tilde{U} - u_i + r)\right] \\
&\quad + \sum_{l=1,\ l \neq i}^{k} \left[\mathrm{opt}_t(\sigma, \tilde{B}_l) - \mathrm{opt}_{t-1}(\sigma, \tilde{B}_l)\right] \\
&\quad + \mathrm{opt}_t(\sigma, A) - \mathrm{opt}_{t-1}(\sigma, A) \\
&\geq \mathrm{opt}_t(\sigma, A) - \mathrm{opt}_{t-1}(\sigma, A) \\
&= EC_t
\end{aligned}
$$

The next to the last step follows from the fact that work functions are nondecreasing with time, while the last step follows from the the first part of the Duality Lemma (Lemma 17). ∎

Putting everything together, we have

Theorem 20 *WFA is $(2k-1)$-competitive.*

Proof.

$$
WFA(\sigma) + \mathrm{opt}(\sigma) = \sum_t (WFA_t + \widetilde{opt}_t) \leq \sum_t EC_t \leq \sum_t (\phi_t - \phi_{t-1}) = \phi_f - \phi_o.
$$

The second inequality follows from Lemma 19. It can be easily shown that $\phi_o = c$ for some constant c. Note that

$$
\phi_f \leq \psi(\mathrm{opt}_f, X_f, \ldots, X_f)
$$

$$\begin{aligned}
&= 2k\text{opt}_f(\sigma, X_f) - \sum_{i=1}^{k}\sum_{j=1}^{k} d(x_i^f, x_j^f) \\
&\leq 2k\text{opt}_f(\sigma, X_f) \\
&\leq 2k\text{opt}(\sigma)
\end{aligned}$$

Combining the above we get $\phi_f - \phi_o \leq 2k\text{opt}(\sigma) + c$. Solving for $WFA(\sigma)$ shows that $WFA(\sigma) \leq (2k-1)\text{opt}(\sigma) + c$. Therefore WFA is $(2k-1)$-competitive.

5.7 Quasi-convexity and the Duality Lemma

It is not difficult to show by working backwards from the statement of the duality lemma that the following property of work functions is sufficient to yield a proof. We omit the details.

Lemma 21 *For any two configurations A, B, with $a \in A$,*

$$opt_t(\sigma, A) - opt_t(\sigma, B) \geq \min_{b \in B}\left[opt_t(\sigma, A - a + b) + opt_t(\sigma, B - b + a)\right] \tag{10}$$

In order to prove lemma 21, a more general fact is proven: that the functions $\text{opt}_t(\sigma, \cdot)$ are *quasi-convex*. The property of quasi-convexity is defined as follows:

Definition 22 *The functions $opt_t(\sigma, \cdot)$ are* quasi-convex *if and only if $\forall A, B$ configurations, there exists a bijective map*

$$h : A \to B$$

such that for every bipartition of A into X and Y, i.e.*, $A = X \cup Y$ and $X \cap Y = \phi$, we have*

$$opt_t(\sigma, A) + opt_t(\sigma, B) \geq opt_t(\sigma, X \cup h(Y)) + opt_t(\sigma, h(X) \cup Y)$$

and furthermore the bijection h preserves the intersection of A and B, i.e. $h(r) = r$, $\forall r \in A \cap B$.

This quasi-convexity property implies the desired previous lemma, since we can take the specific bipartition where $X = \{a\}$ and $Y = A - \{a\}$.

We outline the proof of the following lemma.

Lemma 23 *$opt_t(\sigma, \cdot)$ is quasi-convex.*

Proof. The proof by induction on the length of the sequence. The base case is trivial. Suppose that work functions opt_{t-1} are quasiconvex. By definition, we have

$$\text{opt}_t(\sigma, A) = \min_{a \in A}(\text{opt}_{t-1}(\sigma, A - a + r) + d(a, r)) \tag{11}$$

$$\text{opt}_t(\sigma, B) = \min_{b \in B}(\text{opt}_{t-1}(\sigma, B - b + r) + d(b, r)) \tag{12}$$

Choose the specific a, b that minimize the above expressions, and set $A' \equiv A - a + r$, $B' \equiv B - b + r$.

Suppose that X', Y' and $\{a\}$ are a tripartition of A. Apply the induction hypothesis to A' and B' with the partition $X = X' \cup \{r\}$ and $Y = Y'$. Then there is a bijection $h : A' \to B'$ that maps r to itself, with the property that

$$\mathrm{opt}_{t-1}(\sigma, A') + \mathrm{opt}_{t-1}(\sigma, B') \geq \mathrm{opt}_{t-1}(\sigma, h(X') \cup \{r\} \cup Y) + \mathrm{opt}_{t-1}(\sigma, X' \cup \{r\} \cup h(Y))$$

Extend the bijection $h : a \mapsto b$; then add $d(a, r) + d(b, r)$ to both sides of the above equation to get

$$\mathrm{opt}_t(\sigma, A) + \mathrm{opt}_t(\sigma, B) \geq \mathrm{opt}_t(\sigma, h(X') \cup \{b\} \cup Y) + \mathrm{opt}_t(\sigma, X' \cup \{a\} \cup h(Y)),$$

i.e.

$$\mathrm{opt}_t(\sigma, A) + \mathrm{opt}_t(\sigma, B) \geq \mathrm{opt}_t(\sigma, h(X) \cup Y) + \mathrm{opt}_t(\sigma, X \cup h(Y)),$$

as desired. ∎

6 Online Load balancing and Virtual Circuit Routing

Consider the following generic load balancing problem: there is some fixed set of **servers** and a sequence of **tasks** which arrive through time. Each task, upon its arrival, is to be assigned to a subset of the servers who will perform the task. When a server is used to perform a task, the load on that server increases while the task is being performed. The amount of this increase may depend both on the task and the server. An **offline** algorithm knows in advance the sequence of tasks that will arrive into the system. An **online** algorithm learns about a task only at the moment of its arrival. Depending on the model, servers can represent various devices: processors, communication links, input/output ports, etc. A task can be a job to be executed, or a message to be sent or a commodity to be shipped.

There are two natural optimization problems that arise from this scenario. The first is simply to minimize the maximum load on any server at any point in time. The second is to assume that each server has a fixed capacity. Tasks will either be accepted or rejected by the system and all accepted tasks must be assigned in a manner which respects the capacity constraints of each server. The goal is to complete the set of tasks with the most value (i.e. maximize *throughput*). The value of completing a given task must be specified by the model.

There are static and dynamic versions of load balancing. In the static version of the problem, tasks only arrive. Each task must be assigned (or rejected) upon its arrival and after all the tasks have arrived, the value or cost achieved by the algorithm can be determined. A more difficult but more realistic setting incorporates a notion of time in which tasks are arriving and terminating. As each task arrives, it must be assigned to a set of servers which perform the task. Upon completion of a task, it leaves the system. The static version can be viewed as a special case of the dynamic problem in which there are only tasks of infinite duration. The typical interpretation of the static version is that the load of a job is actually the time it takes to complete it. Thus the final load on any server is the time it takes for one server to complete all the tasks assigned to it.

There are two natural notions of preemption. In the first, an algorithm may reassign a task on a different set of servers some time after it has already been assigned. Clearly, there must be some cost associated with such a change. Otherwise, the online algorithm can continually reassign jobs to match the optimal assignment on the sequence seen so far. Typically, the goal is to devise the best algorithm given some number of allowed reschedulings. The second notion of preemption is suited only for the problem of maximizing throughput. Here a task may be aborted in in the middle of execution, say to make room for a more valuable task. No credit is accrued for uncompleted tasks.

Now what remains to be specified for each model is what a task looks like – that is, which subsets of servers can perform it and at what costs. Below is a description of some of the models that have been studied in the recent literature in online algorithms:

Multiprocessors: There are m servers. Each task is described by a duration d and a vector $(p_1, \ldots, p_m)$. Each task is assigned to exactly one server. If the task is assigned to server i then, the load of server i increases by p_i for the duration of the task. Four variations of this model have been studied:

- Uniform: For all i, j, $p_i = p_j$.
- Related: Each server has a speed s_i and $p_i/p_j = s_j/s_i$ for all i, j.
- Subset: Each task can be performed by any server from a given subset. That is, $p_i = c$ or $p_i = \infty$ where c depends only on the task.
- Unrelated: Each p_i can be any non-negative number.

Virtual Circuit Routing: Each server represents an edge in a network. A task specifies two nodes s and t in the network, a duration and bandwidth request p. The algorithm chooses a path from s to t. The load on each of the edges in the path increases by p for the duration of the task. (In this context, the load is often referred to as the *bandwidth*).

In sections 6.1 and 6.2 below, we present two algorithms: one for load balancing on unrelated machines and the other for virtual circuit routing. Both results are due to Aspnes, Azar, Fiat, Plotkin and Waarts ([AAF+93]). The algorithms are non-preemptive and seek to minimize the maximum load. The algorithms are competitive for the static version (i.e. jobs only arrive into the system). The behavior and analysis of the two algorithms are very similar in that they both employ a function which is the sum of a set of numbers that are exponential in the current load on each processor/edge. The algorithm chooses the assignment so as to minimize this function. Interestingly, this function has played a role in other settings outside of online algorithms. In particular, it is often used to guide the search for a good sample point in derandomizing algorithms (see [LN93, LV91, AS92].

Another feature which both algorithms share is that they are based upon an algorithm which knows the optimal cost for the sequence in advance. We will call this optimal cost Λ. The algorithm must simply make sure that its own cost never exceeds $c\Lambda$ in order to get a competitive ratio of c. The algorithm assigns incoming tasks in a manner that guarantees that if its own cost exceeds $c\Lambda$, then the optimal cost must exceed Λ. A simple doubling trick suffices to remove the assumption that the algorithm knows Λ.

6.1 Load Balancing on Unrelated Machines

Let $\sigma = \langle p(1), \ldots, p(k) \rangle$ denote a sequence of jobs which the algorithm receives. Let S_i denote the set of jobs allocated to processor i at the end of the sequence. It will be convenient to keep track of the current load on processor i after the first j jobs have arrived:

$$l_i(j) = \sum_{1 \leq r \leq j : r \in S_i} p_i(r).$$

The cost to the algorithm is the maximum total load assigned to any processor:

$$\max_{1 \leq i \leq m} l_i(k).$$

We start with the assumption that the algorithm U knows a Λ such that for the sequence σ of jobs to be assigned, $\texttt{cost}_{OPT}(\sigma) \leq \Lambda$. We will show that U has a maximum load of $O(\Lambda \log n)$ for that sequence. Let $\tilde{x}$ denote the normalization of x by Λ (i.e., $\tilde{x} = x/\Lambda$). Suppose the algorithm is about to assign the next job $p(j)$. U assigns the new job so as to make $\sum_{i=1}^{n} a^{\tilde{l}_i(j)}$ as small as possible for some fixed a. Let β be some number which is $O(\log n)$. The exact constant in the O as well as a will be chosen later. Upon receiving $p(j)$, U assigns $p(j)$ to the processor i that minimizes

$$a^{\tilde{l}_i(j-1)+\tilde{p}_i(j)} - a^{\tilde{l}_i(j-1)}.$$

If $l_i(j-1) + p_i(j) \geq \beta\Lambda$, then U *fails.*

Lemma 24 *If $\texttt{cost}(\sigma) \leq \Lambda$, then β and a can be chosen so that U never fails. Thus the competitive ratio is $O(\log n)$.*

We define a potential function:

$$\Phi(j) = \sum_{i=1}^{n} a^{\tilde{l}_i(j)}(\gamma - \tilde{l}_i^*(j)).$$

The constant $\gamma > 1$ will also be chosen later.

We will prove that after each job is assigned, the potential function does not increase. But first, let's see how this fact yields the lemma. Since $\Phi(0) = \gamma n$ and $l_i^*(j) \leq \Lambda$, we know that at any point j,

$$\begin{aligned} \gamma n &= \Phi(0) \geq \Phi(j) \\ &= \sum_{i=1}^{n} a^{\tilde{l}_i(j)}(\gamma - \tilde{l}_i^*(j)) \\ &\geq \sum_{i=1}^{n} a^{\tilde{l}_i(j)}(\gamma - 1) \end{aligned}$$

Taking the log of both sides, we get that

$$\begin{aligned} \texttt{cost}_U &\leq \max_i l_i(k) \\ &\leq \Lambda \log_a \left(\frac{\gamma n}{\gamma - 1}\right) = O(\Lambda \log n). \end{aligned}$$

Now to prove that $\Phi(j) - \Phi(j-1) \leq 0$, suppose that OPT assigns $p(j)$ to i and U assigns $p(j)$ to i'.

$$\begin{aligned} \Phi(j) - \Phi(j-1) &= (\gamma - \tilde{l}_{i'}^*(j-1))(a^{\tilde{l}_{i'}(j-1)+\tilde{p}_{i'}(j)} - a^{\tilde{l}_{i'}(j-1)}) - a^{\tilde{l}_i(j)}\tilde{p}_i(j) \\ &\leq \gamma(a^{\tilde{l}_{i'}(j-1)+\tilde{p}_{i'}(j)} - a^{\tilde{l}_{i'}(j-1)}) - a^{\tilde{l}_i(j-1)}\tilde{p}_i(j) \\ &\leq \gamma(a^{\tilde{l}_i(j-1)+\tilde{p}_i(j)} - a^{\tilde{l}_i(j-1)}) - a^{\tilde{l}_i(j-1)}\tilde{p}_i(j) \\ &= a^{\tilde{l}_i(j-1)}(\gamma(a^{\tilde{p}_i(j)} - 1) - \tilde{p}_i(j)) \end{aligned}$$

The first inequality comes from the fact that $\tilde{l}_i(j) \geq \tilde{l}_i(j-1)$. The second inequality comes from the fact that i' minimizes the expression $(a^{\tilde{l}_{i'}(j-1)+\tilde{p}_{i'}(j)} - a^{\tilde{l}_{i'}(j-1)})$.

Since OPT assigns $p(j)$ to i, we know that $0 \leq p_i(j) \leq \Lambda$ and $0 \leq \tilde{p}_i(j) \leq 1$. Thus, we want to choose a and γ so that for any $x \in [0,1]$, $\gamma(a^x - 1) \leq x$. This is true for $a = 1 + 1/\gamma$. γ is then chosen to minimize $\beta = \log_a\left(\frac{\gamma n}{\gamma-1}\right)$. ■

We can obtain an algorithm U' which does not need to know Λ in advance by sacrificing only an extra factor of 4 in the competitive ratio. The algorithm always has a current guess for the optimal cost which it calls Λ. The algorithm initially assigns $\Lambda = \min_i p_i(1)$. The algorithm makes assignments according to U. Whenever the algorithm fails, Λ is doubled and the algorithm starts again, assigning according to U and ignoring all tasks assigned previous to the last doubling.

6.2 Online Virtual Circuit Routing

The circuit routing problem is a generalization of the load balancing problem on unrelated machines. Each edge can be thought of as a processor. In this case, the load of a task is the bandwidth required in the connection. A subset of the edges can perform a given task if the edges form a path between the two endpoints specified in the request. Using a path for a connection increases the load on each edge by the bandwidth requirement of the request. The goal is to minimize the maximum load on any edge at any point in time. We will show an algorithm which is $O(\log n)$-competitive. The bound on the competitive ratio is tight to within a constant factor.

We are given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. Each edge e has a non-negative capacity u_e. The request sequence σ is a sequence of k requests

$$\langle [s(1), t(1), p(1)], [s(2), t(2), p(2)], \ldots, [s(k), t(k), p(k)] \rangle.$$

$s(i)$ and $t(i)$ denote the endpoints of the i^{th} request and $p(i)$ denotes the bandwidth requirement of the request. The request is satisfied by a path P_i from $s(i)$ to $t(i)$. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ denote the set of paths the online algorithm (which we call *Route*) chooses to satisfy the k requests in the sequence. The *relative load* in edge e after j requests have arrived is:

$$l_e(j) = \sum_{i: e \in P_i,\ i \leq j} p(i)/u_e.$$

The cost of *Route* on the sequence is then:

$$\texttt{cost}_{Route}(\sigma) = \max_{1 \leq i \leq m} l_i(k).$$

Similarly, define $\mathcal{P}^* = \{P_1^*, P_2^*, \ldots, P_k^*\}$ to be the set of paths the optimal offline algorithm chooses to satisfy the first k requests. $l_e^*(j)$ and $\texttt{cost}(\sigma)$ are defined analogously for the optimal algorithm.

The algorithm *Route* seeks to minimize $\sum_{e \in E} a^{\tilde{l}_e(j)}$. Again, we assume that we are given Λ, such that $\texttt{cost}(\sigma) \leq \Lambda$. Upon receiving a request (s, t, p), the algorithm *Route* assigns a cost c_e to each edge where

$$c_e = a^{\tilde{l}_e + \tilde{p}/u_e} - a^{\tilde{l}_e}.$$

With these costs assigned to each edge, *Route* picks the path P from s to t of minimum cost. If $\exists e \in P$ such that $l_e + p/u_e > \beta\Lambda$, then *Route* fails. The proof of the following lemma is virtually identical to Lemma 24 for scheduling on unrelated machines.

Lemma 25 *If* $\texttt{cost}(\sigma) \leq \Lambda$, *then* β *and* a *can be chosen so that* U *never fails. Thus the competitive ratio is* $O(\log n)$.

The assumption that *Route* knows Λ can be taken care of in the same manner as U in the previous section.

6.3 Recent Results

6.3.1 Minimizing Maximum Bandwidth

Graham initiated the study of competitive online load balancing in [Gra66] with the proof that the greedy algorithm is 2-competitive for the static problem on uniform machines. Interestingly, although the first algorithm for this problem happened to be online, this version of load balancing is studied in the context of approximation algorithms for NP-hard offline scheduling problems. (See Chapter 1 for a survey.) Recently, the problem reemerged in the context of online algorithms when the bound on the competitive ratio was improved to a constant less than 2 in [BFKV92, KPT94]. The improved bound uses a different algorithm, since the greedy algorithm can do no better than 2. Aspnes *et al.* in [AAF$^+$93] give an asymptotically optimal $O(\log n)$-competitive algorithm for unrelated machines, where n is the number of machines in the system (described in section 6.1). This is generalized (as shown in Section 6.2) for routing virtual circuits in a network to obtain a competitive ratio which is logarithmic in the number of edges in the network. In [ANR92], Azar, Naor and Rom show that the greedy algorithm is $O(\log n)$-comptitive for the subset model even though it has an $\Omega(n)$ competitive ratio in the unrelated machines case shown by Burley.

For the dynamic version, Azar *et al.* in [AzK93] give a constant competitive algorithm for related machines, even when the durations of the tasks are unknown at the time they must assigned. This can not be further generalized due to the result of Azar, Broder and Karlin who show that a lower bound of $\Omega(\sqrt{n})$ for the competitive ratio of any online load balancer for the subset model in [ABK92]. When the durations of the jobs are known, this can be vastly improved to $O(\log nT)$ even on unrelated machines as shown in [AzK93]. (T is the ratio of maximum to minimum job length in the sequence.)

All of the algorithms mentioned up to this point are non-preemptive: once the job has been assigned, it remains assigned to those servers until it terminates. The ability to reassign jobs can be of great advantage to an algorithm, even if it is not permitted to reassign all the jobs. Phillips and Westbrook in [PW93] consider the subset model of [ABK92] where task lengths are unknown and some preemptions may be performed. They showed an $O(\log n/\rho)$-competitive algorithm that uses ρm reassignments on a sequence of m tasks. This compares well with the $\Omega(\sqrt{n})$ lower bound for an online algorithm with no reassignments.

We note here that we use the term "load balancing " to refer to problems where a job must be scheduled upon its arrival into the system. We use the term "scheduling" to refer to problems where the jobs can be held in a buffer until they can be executed. Typically, however, the time to execute a job is not known until it terminates. Online scheduling with the goal of minimizing makespan has been extensively studied (see [DJ81, Jaf80, SWW91, FST94, FKST93]).

6.3.2 Maximizing Throughput

For the problem of maximizing throughput, models differ as to how a value is assigned to each task. In the most general version, each task arrives with a parameter specifying its value. The algorithm's gain is the sum of the value of all successfully completed jobs. In some applications, however, better bounds can be achieved when the value of a job is a function of its other parameters, for example, duration or load times duration. In the circuit routing problem, the value may also be a function of the distance between the two end nodes in the network. These measures attempt to capture bandwidth utilization.

Baruah *et al.* examine a processor scheduling problem in [BKM+91] where the goal is to maximize throughput. They consider a real-time system with one processor which can perform only one task at a time (i.e. the load of every job is 1 and the capacity of the server is 1). However, in their model a task does not necessarily have to be accepted or rejected upon its arrival into the system. Rather, each task has a deadline, and the algorithm accrues the value of all jobs completed before their deadline. The algorithm may choose to abort a currently running job in favor of a job with more value. In this case, the aborted job, if completed, must be restarted from the beginning. They give a constant competitive algorithm when the value of a job is proportional to its duration.

The remainder of the work on throughput maximization is for routing virtual circuits in networks. This area has become known as *call control.* Call control is similar to the multicommodity flow problem discussed in Chapter 5. The main difference between the two are that multicommodity flow is an offline problem where all the jobs arrive at time 0 and last forever. In addition, in multicommodity flow, teh scheduler is allowed to schedule fractions of jobs and receive the corresponding fraction of credit.

For the call control problem, Awerbuch, Azar and Plotkin in [AAP93] show a deterministic, non-preemptive $O(\log \mu)$-competitive circuit routing algorithm in this model, where μ is the product of the ratio of the maximum to minimum job duration, the ratio of the maximum to minimum job value and the number of nodes in the network. They also show that this is the best that any deterministic, non-preemptive routing algorithm can achieve. Their algorithm has the drawback that every task is constrained to require bandwidth at most $1/\log \mu$ of the capacity of any edge (a necessary assumption for deterministic, non-preemptive algorithms). Canetti and Irani show in [CI] a lower bound of $\Omega(\sqrt{\log \mu / \log \log \mu})$ for this problem which appies even to randomized, preemptive algorithms. Their bound also applies to a variety of fixed capacity load balancing problems in the literature. In [ABFR94], Awerbuch *et al.* remove the constraint on task bandwidth for networks with a tree topology via a randomized, non-preemptive routing algorithm. They obtain an algorithm which is $O(\log L \log T \log n)$-competitive, where n is the number of nodes in the network and L is the ratio of maximum to minimum bandwidth of a request in the sequence. [AGLR94] consider the static version of call control where at most one request can be routed on an edge. They improve upon the bounds in [ABFR94] for trees and also develop algorithms for meshes and hypercubes.

An intersting technique called *classify and randomly select* is employed in [ABFR94]. We describe the technique here with respect to the duration of a request, although it can be used with respect to any parameter of the requests in the sequence. Suppose there is an algorithm $\mathcal{A}$ which is c-competitive when the durations of the requests in the sequence vary only by a factor of 2. We can use $\mathcal{A}$ to obtain an algorithm which is $(c \log T)$-competitive for any request sequence. (Recall that T is the ratio of maximum to minimum job duration). The requests are partitioned into $\log T$ bins according to their duration. The requests in a single bin differ in duration by at most a factor of 2. At the beginning of the sequence, the algorithm selects a bin at random and rejects all requests from other bins. The algorithm then uses $\mathcal{A}$ to decide which requests in the selected bin to execute. The expected total value of all the tasks in the selected bin is at least $(1/\log T)$ times the total value of all the tasks. Since $\mathcal{A}$ is c-competitive for the sequence of tasks in the selected bin, the new algorithm is $c \log T$-competitive on the entire sequence. This technique can be applied to the algorithms of [AGLR94] so that they apply to the dynamic version with only an extra factor of $O(\log T)$ in the competitive ratio.

Preemptive call control algorithms are addressed in [GG92, GGK$^+$93, BNCK$^+$]. Garay and Gopal, who initiate the study of preemptive call control in [GG92], show that if the duration of a call is unknown, then no bounded competitive ratio can be achieved even by preemptive algorithms. Still, constant competitive preemptive algorithms exist for simple networks and measures of call value. Garay *et al.* show in [GGK$^+$93] constant competitive algorithms for several measures of call value on a single link and a line network when at most one call can be accommodated on any link. Bar-Noy *et al.* in [BNCK$^+$] generalize their results by showing a constant competitive algorithm on a single link when the value of a call is the bandwidth times the duration and every call has a bandwidth requirement which is at most a limited fraction of the capacity of the link.

7 Variants of Competitive Analysis

Competitive analysis of online algorithms has provided a meaningful way to do a worst-case analysis of online algorithms without assuming a probability distribution over inputs. This means of evaluation has provided new insight to the design of online algorithms as well as to decision making with incomplete information, in general. Moreover, the problems that have been studied in this context have typically been motivated by applications, and many of the algorithms studied have been implemented and widely used (even long before competitive analysis existed). However, there is valid criticism of the model which points to the fact that results are often unrealistically pessimistic. Since many online algorithms fare equally badly in this model, an algorithm which performs well in practice may have the same competitive ratio as one which performs poorly. While it is true that similar problems plague complexity in general, the disparity between theoretical results and practical observations is more disturbing with competitive analysis. Since many of the problems addressed in online algorithms are very applied, there is a higher standard for an algorithm to be considered "practical". The reason for the discrepancy between theory and practice is that somehow pitting an online algorithm which knowns nothing about the future against an adversary which is unlimited in its computational resources as well as its available information is perhaps an unfair comparison. Various efforts have been made to address these concerns, either by limiting the power of the adversary or providing the online algorithm with more information.

Most of this work has addressed the paging problem where practitioners voice reservations about the competitive analysis, citing its inability to discern between LRU and FIFO (algorithms whose performances differ markedly in practice), and the fact that the theoretical competitiveness of LRU is much larger than observed in practice. In addition, it is not necessarily the case that there is no information about the request sequence available in advance. One would like to use some knowledge of a program's access pattern to improve paging performance on the sequence generated by that program.

A graph theoretic model to desribe locality of reference in request sequences was devised in [BIRS91] and later developed in [IKP92]. Instead of requiring that an online algorithm perform well on all input sequences, they define a restricted set of input sequences and analyze the competitive ratio over the restricted set of sequences. The *access graph* for a program $G = (V, E)$ is a graph in which each node corresponds to one of the pages that a program can reference. The sequence of page references must obey locality constraints imposed by the edges of G: following a request to a page (node) u, the next request must be either to u or to a node v such that (u, v) is in E. The graph is known to the online paging algorithm but the particular path taken by the request sequence is revealed online. The intuition is that the access pattern of a program is governed by the data structures and instruction sequences of a program which can be modelled by a graph. Furthermore, this information is available to the paging algorithm before execution. A simple algorithm is devised in [BIRS91] and shown to be close to the best online algorith for every possible undirected access graph. The algorithm is in fact within a constant factor of the best online algorithm for every access graph as shown in [IKP92].

Karlin, Phillips, and Raghavan take this one step farther in [KPR92] by not only providing the online algorithm with the graph but also probabilities along the edges of the graph: they analyze paging algorithms where the request sequence is generated by a Markov chain. Each node in the Markov chain is a virtual memory page. Since the distribution is completely known to the online algorithm, the goal is simply to minimize the fault rate (expected number of faults per request) instead of the cost in comparison to the offline algorithm. Although the optimal online algorithm (which minimizes the fault rate) is available to the online algorithm in information theoretic terms, it is in general, not efficient to compute. Thus, the goal is to approximate this algorithm. The authors prove that many intuitive algorithms do not have a fault rate that it is within a constant of the best online algorithm. They show a somewhat more complicated algorithm which does achieve this goal. Lund, Phillips and Reingold [CLR94] subsequently presented a simpler algorithm for distributional paging.

Koutsoupias and Papadimitriou also suggest two refinements of competitive analysis in[KP94a]. Although most of their results pertain to paging, their measures could potentially shed light on online computation in many other domains. The first of these is *the diffuse adversary model* where the input is generated by a distribution $\mathcal{D}$. Although the online algorithm does not know the exact distribution, it does know that it is chosen from a class Δ of distributions. The adversary can pick the worst $\mathcal{D} \in \Delta$ for an online algorithm A. Once this choice is made, the cost of A on input σ is compared to the optimal offline cost on σ, where σ is picked according to $\mathcal{D}$. The competitive ratio of A is

$$\max_{\mathcal{D}\in\Delta} \frac{E_{\sigma\in\mathcal{D}}[\mathtt{cost}_A(\sigma)]}{E_{\sigma\in\mathcal{D}}[\mathtt{cost}(\sigma)]}.$$

Naturally, the more restricted Δ is, the more information the online algorithm has. Notice that when Δ is the set of all distributions, this is just the usual competitive ratio.

The second measure called *comparative analysis* compares an algorithm to a less powerful algorithm than the optimal offline algorithm. Consider two classes of algorithms $\mathcal{A}$ and $\mathcal{B}$. Typically, $\mathcal{A} \subseteq \mathcal{B}$. $\mathcal{B}$ may have a more powerful information regime or more computational resources. The comparative ratio of $\mathcal{A}$ and $\mathcal{B}$,

$$c(\mathcal{A}, \mathcal{B}) = \max_{B\in\mathcal{B}} \min_{A\in\mathcal{A}} \max_{\sigma} \frac{\mathtt{cost}_A(\sigma)}{\mathtt{cost}_B(\sigma)}.$$

The comparative ratio is probably best viewed as a two player game. Player $\mathcal{B}$ (typically the adversary) picks an algorithm $B \in \mathcal{B}$. Player $\mathcal{A}$ (typically the online algorithm) picks an algorithm $A \in \mathcal{A}$. Then player B picks the input σ so as to maximize the ratio of A's cost on σ to B's cost on σ. Note that if $\mathcal{B}$ is the set of all algorithms (offline and online), and $\mathcal{A}$ is the set of all online algorithms, then $c(\mathcal{A}, \mathcal{B})$ is simply the best competitive ratio for the given problem.

One feature achieved by comparative analysis is that it allows us to address the anomaly observed in competitive analysis of the k-server problem that an online algorithm does not gain anything with additional lookahead. That is, for any l, a k-server algorithm with lookahead l has the same competitive ratio of an online algorithm with no lookahead. The reason is that an adversary which plays against the algorithm with lookahead l can repeat each request $l+1$ times, removing any advantage of the lookahead and not increasing the offline cost. However, if an online algorithm competes against a class of algorithms with limited lookahead it does gain some advantage with additional lookahead. The authors give tight bounds for $c(\mathcal{L}_0, \mathcal{L}_l)$ for task systems and paging, where $\mathcal{L}_i$ is the class of all online algorithms with lookahead i.

Raghavan introduces the notion of a *statistical adversary* in [Rag91] which he uses to address a problem in portfolio management. The idea behind the statistical adversary is that it is free to pick any input as long as it obeys certain statistical properties. The statistical adversary is a compromise between probabilistic analysis and competitive analysis where an unrestricted adversary generates the input. In the portfolio problem, an investor chooses between investing his money in a risk-free asset and a risky asset. The price of the risk-free asset remains fixed while the price of the risky asset changes from day to day. The adversary which determines the sequence of prices for the risky asset is free to choose any sequence as along as it has a fixed mean and fixed variance and only fluctuates within some predetermined bounds. The statistical adversary has also been by Cooperstock, El-Yaniv and Leighton who study foreign currency exchange in [JC93]. They use competitive analysis with a statistical adversary who is restricted to those sequences for which the offline solution is no more than a fixed value.

8 Conclusions and Directions for Future Research

We have presented the basic principles underlying the design and analysis of competitive online algorithms. It is clear that the performance of online algorithms is of fundamental interest in computer science. Numerous problems in operating systems, architecture, parallel and distributed systems, compilers, graphics, robotics and other areas must be dealt with in an online fashion. Any light shed on the design and analysis of these algorithms by theoreticians is of interest to practitioners.

There have been many online problems studied in this framework. We have presented the best results to date for two problems: the k-server problem, and virtual circuit routing. We have omitted presentations of numerous other interesting results, many of which are referenced in the bibliography.

Many interesting problems in this area remain open. First, there are questions that relate to specific online problems. Some of the major outstanding open problems in online algorithms include showing that the work-function algorithm is k-competitive for the k-server problem, obtaining a strongly competitive algorithm for the k-server problem with excursions, and finding a proof of the dynamic optimality conjecture for splay trees. [4] The list goes on and on.

More general results which apply to broad classes of online algorithms, or other general techniques, would represent a more significant advance in the state of the art. For example, a precise characterization of the class of metrical task systems for which the work-function algorithm (or variants of it) is strongly competitive would be extremely interesting.

[4] See [MMS90] for a definition of the k-server problem with excursions. See [DS85] for a description of splay trees and the dynamic optimality conjecture.

In terms of randomized algorithms, the field is wide open. As in the case of deterministic algorithms, there are nearly no general results, other than those of Borodin *et al.* . Moreover, almost all of the specific results for randomized algorithms against oblivious adversaries apply to metrical task systems with uniform distance matrices. Very little is known about randomized algorithms for problems which can be described as a metrical task system with nonuniform distances.

Finally, variants of competitive analysis have only just begun to be studied. Restrictions on adversary request sequences, such as those implied by statistical adversaries, sequences generated from access graphs and the diffuse adversary model, in addition to other kinds of comparisons between information regimes such as comparative analysis represent important directions for future research.

9 Acknowledgements

We gratefully acknowledge comments, discussions, papers and references given to us by Alan Borodin, Marek Chrobak, Ran El-Yaniv, Amos Fiat, Howard Karloff, Elias Koutsoupias, Prabhakar Raghavan and Danny Sleator. The second author would like to thank the students in her 1994 course on online algorithms at the University of Washington, especially Eric Anderson, Ruth Anderson, Donald Chinn, Marc Fiuczynski, Melanie Fulgham, Tracy Kimbrel, Omid Madani, Erik Selberg, Jayram Thathachar and Elizabeth Walkup.

References

[AAF+93] J. Aspens, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. 25th ACM Symposium on the Theory of Computing*, pages 623–631, 1993.

[AAP93] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *34th IEEE Symposium on Foundations of Computer Science*, 1993. 32–40.

[ABFR94] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. of 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 312–320, 1994.

[ABK92] Y. Azar, A. Broder, and A. Karlin. On-line load balancing. In *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, pages 218–225, 1992. To appear in *Theoretical Compute Science.*

[ABM93] Y. Azar, A. Broder, and M. Manasse. On-line choice of on-line algorithms. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 432–440, 1993.

[AGLR94] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance comp uting and communication. In *Proceedings of the 34th Annual Symposium on the Foundations of Computer Science*, pages 412–423, 1994.

[AKPW92] N. Alon, R. Karp, D. Peleg, and D. West. A graph-theoretic game and its application to the k-server problem. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 7, pages 1–9, 1992.

[ANR92] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. In *Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 203–210, 1992.

[AS92] N. Alon and J. Spencer. *The probabilistic method.* John Wiley and Sons, Inc., 1992.

[AzK93] *Online Load Balancing of Temporary Tasks*, 1993.

[BDBK+90] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Widgerson. On the power of randomization in on-line algorithms. In *Proc. 22nd Symposium on Theory of Algorithms*, pages 379–386, 1990.

[BFKV92] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24th ACM Symposium on Theory of Algorithms*, pages 51–58, 1992. To appear in *Journal of Computer and System Sciences.*

[BFR92] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *Proc. of the 24th Symposium on Theory of Computation*, pages 39–48, 1992.

[BGR93] M. Bern, D. Greene, and A. Ragunathan. On-line algorithms for cache sharing. In *Proceedings of the 25th Annual Symposium on the Theory of Computing*, pages 422–430, 1993.

[BS89] D. L. Black and D. D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.

[BIRS91] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 249–259, 1991. To appear in *Journal of Computer and System Sciences.*

[BKM+91] S. Baruah, G. Koren, B. Mishra, A. Ragunathan, L. Rosier, and D. Shasha. On-line scheduling in the presence of overload. In *Proceedings of the 31st Annual Symposium on the Foundations of Computer Science*, pages 100–110, 1991.

[BLS87] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. In *Proc. 19th Annual ACM Symposium on Theory of Computing*, pages 373–382, 1987.

[BNCK$^+$] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber. Bandwidth allocation with preemption. Manuscript.

[Bur93] W. Burley. Traversing layered graphs using the work function algorithm. Manuscript, 1993.

[BYCR93] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993. Preliminary version in Proc. 1st Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science 318, Springer-Verlag, Berlin, 1988, 176–189. Also Tech. Report CS-87-68, University of Waterloo, Department of Computer Science, October, 1987.

[CDRS93] D. Coppersmith, Peter Dolye, P. Raghavan, and M. Snir. Random walks on wieghted graphs and applications to on-line algorithms. *Journal of the ACM*, 1993.

[CI] R. Canetti and S. Irani. On the power of preemption in randomized scheduling. Submitted for publication.

[CKPV91] M. Chrobak, H. Karloff, T. H. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991. Also in Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 1990, pp. 291-300.

[CL91a] M. Chrobak and L. L. Larmore. On fast algorithms for two servers. *Journal of Algorithms*, 12:607–614, 1991.

[CL91b] M. Chrobak and L. L. Larmore. An optimal online algorithm for k servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991.

[CL92] M. Chrobak and L. L. Larmore. The server problem and on-line games. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 7, pages 11–64, 1992.

[DJ81] E. Davis and J.M. Jaffe. Algorithms for scheduling tasks on unrelated processors. *Journal of the ACM*, 28:712–736, 1981.

[DK93] X. Deng and E. Koutsoupias. Competitive implementations of parallel programs. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 455–461, 1993.

[DM91] X. Deng and S. Mahajan. Randomization vs. computability in online problems. In *Proceedings of the 23rd Annual Symposium on the Theory of Computing*, pages 289–298, 1991.

[DS85] R.E. Tarjan D.D. Sleator. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.

[Fel50] W. Feller. *An introduction to probability theory and its applications.* John Wiley and Sons, 1950.

[FFK+91] A. Fiat, D. Foster, H. Karloff, Y. Rabani, Y. Ravid, and S. Vishwanathan. Competitive algorithms for layered graph traversal. In *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, pages 288–297, 1991.

[FKL+91] A. Fiat, R. Karp, M. Luby, L. A. McGeoch, D. Sleator, and N.E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[FKST93] A. Feldmann, M. Y. Kao, J. Sgall, and S. H. Teng. Optimal online scheduling of parallel jobs with dependencies. In *Proc. 25th ACM Symposium on Theory of Computing*, pages 642–651, 1993.

[FRR90] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. In *Proc. 22nd IEEE Symposium on Foundations of Computer Science*, pages 454–463, 1990.

[FST94] A. Feldmann, J. Sgall, and S.H. Teng. Dynamic scheduling on parallel machines. *Theoret. Comput. Sci.*, 1994. To appear in the special issue on dynamic and online algorithms. Also in 32nd IEEE Symposium on Foundations of Computer Science, 1991, 111–120.

[GG92] J.A. Garay and I.S. Gopal. Call preemption in communication networks. In *Proceedings of INFOCOM 1992*, 1992.

[GGK+93] J. Garay, I.S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *Proc. 2nd Israel Symposium on Theory of Computing and Systems*, pages 285 – 293, 1993.

[Gra66] R. Graham. Bounds for certain multiprocessor anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[Gro91] E. Grove. The harmonic k-server algorithm is competitive. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 260–266, 1991.

[IKP92] S. Irani, A. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. In *3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 228–236, 1992.

[IR91] S. Irani and R. Rubinfeld. A competitive 2-server algorithm. *Information Processing Letters*, 39:85–91, 1991.

[Jaf80] J.M. Jaffe. Efficient scheduling of tasks without full use of processor resources. *Theoretical Computer Science*, 12:1–17, 1980.

[JDUGG] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, R.L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms", In *SIAM Journal on Computing*, 3: 299—325, 1974.

[JC93] T. Leighton J. Cooperstock, R. El-Yaniv. The statistical adversary allows on-line foreign exchange wit. Unpublished manuscript, 1993.

[CLR94] C. Lund, S. Phillips and N. Reingold. IP over connection-oriented networks and distributional paging. In *Proceedings of the 34th Annual Symposium on the Foundations of Computer Science*, pages 424–434, 1994.

[Kar] R. Karp. A $2k$-competitive algorithm for the circle. Unpublished manuscript.

[Kle94] J. Kleinberg. A lower bound for two-server balancing algorithms. *Information Processing Letters*, 1994.

[KMMO90] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Randomized competitive algorithms for non-uniform problems. In *1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 301–309, 1990. To appear in *Algorithmica*.

[KMRS88] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[KMV94] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoret. Comput. Sci.*, 127:255–267, 1994.

[KP93] B. Kalyanasundaram and K. Pruhs. On-line weighted matching. *Journal of Algorithms*, pages 478–488, 1993.

[KP94a] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. In *Proceedings of the 34th Annual Symposium on the Foundations of Computer Science*, 1994.

[KP94b] E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. In *Proc. 25th Symposium on Theory of Computing*, pages 507–511, 1994.

[KPR92] A. Karlin, S. Phillips, and P. Raghavan. Markov paging. In *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, pages 208–217, 1992.

[KPT94] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. In *Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 132–140, 1994.

[LN93] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the 25th Annual Symposium on the Theory of Computing*, pages 448 – 457, 1993.

[LR94] C. Lund and N. Reingold. Linear programs for randomized on-line algorithms. In *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, 1994. 382–391.

[LV91] M. Luby and B. Veličković. On deterministic approximation of dnf. In *Proceedings of the 23rd Annual Symposium on the Theory of Computing*, pages 430–438, 1991.

[MMS90] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990. Also in Proc. 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 322-333.

[MS91] L. McGeoch and D. Sleator. A strongly competitive randomized paging algorithm. *J. Algorithms*, 6:816–825, 1991.

[PY91] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.

[PW93] S. Phillips and J. Westbrook. On-line load balancing and network flow. In *Proc. 25th ACM Symposium on Theory of Computing*, pages 402–411, 1993.

[PY89] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. In *16th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science vol. 372*, pages 610–620. Springer-Verlag, 1989.

[Rag91] P. Raghavan. A statistical adversary for on-line algorithms. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Scie nce*, pages 79–83, 1991.

[RS89] P. Raghavan and M. Snir. Memory versus randomization in online algorithms. In *16th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science vol. 372*, pages 687–703. Springer-Verlag, 1989.

[ST85] D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[SWW91] D.B. Shmoys, J. Wien, and D.P. Williamson. Scheduling parallel machines on-line. In *Proceedings of the 31st Annual Symposium on the Foundations of Computer Science*, pages 131–140, 1991.

[Tar83] R. E. Tarjan. *Data Structures and Network Algorithms.* SIAM, Philadelphia, 1983.

[Tar85] R.E. Tarjan. Amortized computational complexity. *SIAM Journal on Discrete Mathematics*, 6(2), 1985.

[Wes92] J. Westbrook. Randomized algorithms for multiprocessor page migration. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 7, pages 135–150, 1992. To appear in *SIAM Journal on Computing.*

[Yao80] A.C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 12th ACM Symposium on Theory of Computing*, 1980.