Online Algorithms: Beyond Competitive Analysis

A modern approach

Reza Dorrigiv and Alejandro López-Ortiz

1 Introduction

Online computation is a model for formulating decision making under uncertainty. In an online problem the algorithm does not know the entire input from the beginning; the input is revealed in a sequence of steps. An online algorithm should make its decisions based only on the observed past and without any secure knowledge about the forthcoming sequence in the future. The cost and effects of a decision taken cannot be undone.

This introduces a new dimension to computation. Consider for example, the scenario of scheduling customer appointments at a bank branch. Some customers wish to perform a simple transaction and can be served by any bank teller while others, may wish to apply for a mortgage, personal loan or open a retirement fund and can only be served by the employees who have the specific training required for these transactions. If we knew the customer needs ahead of time we could book a full roster of appointments in such a way that no customer is left waiting. However, as we know, in practice customers arrive unannounced at various times during the day which makes the scheduling of appointments substantially more difficult and waiting times inevitable.

The two variants of this problem, namely the offline one in which the customers numbers and needs are known ahead of time, and the online version in which customers arrive unannounced lead to completely different approaches to solving the problem. In the offline case a goal is to fit as many customers as possible in a given time slot thus reducing waste, while in the online case we purposely built a certain amount of slack in the solution so we can handle a sudden customer rush.

This sets out an important and broad classification of computational problems into two classes. One consisting of classical problems for which the entire input to the algorithm is given at the beginning of the computation and another consisting of problems in which the input is revealed over time to the algorithm which must now make irrevocable decisions as it goes along. Algorithms of the latter type are known as *online algorithms.* In contrast algorithms of the former type are known as *offline algorithms.*

Paging is another classic example of an online problem. Computers, as we know, are equipped with various means of storing information ranging from hard drives which are big and slow to random access memory (RAM) and cache (level 1 or level 2) which are comparatively faster but smaller in size. A paging algorithm is responsible for mediating between a slower memory and a faster memory. The faster memory is called the cache. The paging strategy must decide which memory pages to keep in the cache without the benefit of knowing in advance the sequence of upcoming page requests. Every time a page request is issued, if the request is to a page already in the cache, the request is called a *hit* and is handled at no cost. On the other hand, in the event the request is not currently in the cache (namely, when a cache miss occurs) the online algorithm must decide irrevocably which page to evict. The goal of a paging algorithm is to minimize the number of faults (cache misses) over the entire input sequence. Observe that the paging algorithm must produce a partial solution after receiving the i^{th} page request and determine which page to evict, shall the page request be a fault.

We can generalize the key concepts of this example to other problems as follows. Let $\sigma = (\sigma_1, \sigma_2, \ldots)$ be an input sequence. We denote by $\sigma_{1:j} = (\sigma_1, \sigma_2, \ldots, \sigma_j)$ the prefix subsequence of the first j requests in σ. An online algorithm $\mathcal{A}$ for an optimization problem takes as input a sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$. The algorithm $\mathcal{A}$ processes the request sequence in order, from σ_1 onwards and produces a partial solution with cost $\mathcal{A}(\sigma_{1:j})$ after the arrival of the jth request. In general it is assumed that the length of the sequence is unknown beforehand and hence an online algorithm performs the same steps on the common prefix of two otherwise distinct input sequences. More formally, if σ' is a prefix of σ then $\mathcal{A}(\sigma') = \mathcal{A}(\sigma_{1:|\sigma'|})$.

The standard model for analysis of algorithms is *worst-case analysis.* In this model we consider the worst performance of an algorithm over all input instances of the same size. Unfortunately, worst-case analysis of online algorithms generally does not give useful information. This is because online algorithms exhibit exactly the same extremely bad behaviour (the worst possible) on some input sequences and thus we cannot differentiate between their worst-case performance. For example consider an online paging algorithm $\mathcal{A}$. No matter how well designed $\mathcal{A}$ might be, at each step we can request the one page that $\mathcal{A}$ just evicted, resulting in a fault at each step. Originally online algorithms were studied using *average-case* or *distributional analysis.* In this model, a probability distribution is assumed for the input and the expected performance of algorithms on this distribution is computed. The main problem with this approach is that we usually do not know this probability distribution in practice and it might be difficult to characterize it formally. Also this distribution might change over the time for some applications.

Competitive analysis is the standard measure for the analysis of online algorithms. In this model we compare the performance of an online algorithm with an offline optimal algorithm OPT which knows the entire sequence in advance. The main idea is that a competitive algorithm may behave poorly on a sequence, provided it is a difficult one, as reflected by the bad performance of OPT on that sequence. Competitive analysis became popular after two papers by Sleator and Tarjan (1985a,b), although the idea of comparing against an offline optimum is present in Graham (1966) and Johnson (1973, 1974). The term competitive analysis was introduced by Karlin et al. (1988).

The competitive ratio has been a great success, enabling the measurement of the performance of various well known heuristics and fostering the development of the field. Online algorithms are more often than not amenable to analysis under this framework; that is, computing the competitive ratio has proven to be effective—even in cases where the exact shape of the OPT solution is unknown. As well, it has been successfully applied outside the original online paging and list update settings to various other applications such as online geometric searching, k-server problem, scheduling, bin packing and other online approximation to NP-complete problems. Competitive analysis is a relatively simple measure to apply yet powerful enough to quantify, to a large extent, the performance of many online algorithms. The growth of online computation is due in no small part to the effectiveness of this measure. Notwithstanding the wide applicability of competitive analysis, it has been repeatedly observed by various researchers that in certain settings the competitive ratio produces results that are too pessimistic or otherwise found wanting. Indeed, the original paper by Sleator and Tarjan (1985a) discusses the various drawbacks of the competitive ratio in the case of the paging problem and intoduces resource augmentation to address some of the observed drawbacks. In her survey of paging algorithms Irani (1998) says:

> Some of the reservations which have been raised about the competitive analysis of paging are 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 what is observed in practice. Also unsettling is fact that in the standard competitive model, a fixed amount of lookahead does not give an on-line algorithm any advantage. [...] It seems intuitive that in practice, an on-line algorithm would benefit from some limited lookahead, but this is not reflected in the model.

These shortcomings have led to the introduction of several alternative measures for analysis of online algorithms. For the case of paging, Irani (1998) says: *"in fact, a survey of the competitive analysis of paging is in some ways a survey of the refinements of competitive analysis."*

The purpose of this textbook is to serve as a guide to the main alternatives to the competitive ratio which have been presented in the literature. The main measures we will cover are:

- pairwise comparisons
- Max/Max ratio
- random order ratio
- relative order,
- bijective analysis
- parameterized analysis

- smoothed analysis
- diffuse adversary
- other measures

The book will also review the state of the art in terms of what is formally known about the performance of online algorithms for some of the best known applications of online algorithms, namely

- list update
- scheduling.
- paging
- bin packing
- routing

After introducing these online problems we then discuss each of the alternative measures proposed. For each measure we give a sample of the cases where it is most applicable, discuss some of its best features as well as its main drawbacks.

2
Applications

2.1 Self-Organizing Data Structures

The *dictionary problem* or the *search problem* is one of the most fundamental problems in theoretical computer science. In it we have a (dynamic) totally ordered set of elements on which we want to efficiently perform a sequence of searches. More specifically we require a data structure that stores a set of n elements and supports the following three basic operations: $Insert(e)$ inserts the element e into the structure, $Delete(e)$ deletes e from the structure, and $Search(e)$ finds the element e. Various efficient data structures have been proposed for the dictionary problem. For example balanced trees such as AVL-trees (Adel'son-Vel'skii and Landis, 1962) and red-black trees (Bayer, 1972; Guibas and Sedgewick, 1978) support all three operations in $\Theta(\log n)$ time in the worst case which is the best possible in the worst case in the comparison-based model. These data structures are static, i.e., they do not change their state on a search operation. Self-Organizing or self-adjusting data structures have a restructuring or self-organizing rule that changes the state of the structure after each operation. Informally, this rule is designed to discover, in an online manner, the properties of the given input sequence and reorganize the structure as it goes along into a state that presumably favors future operations. Self-organizing data structures have several possible advantages over their static counterparts (Sleator and Tarjan, 1985b):

1. Their asymptotic amortized time is usually not much worse than static structures.
2. Their performance is much better than static structures on skewed input sequences as they adjust according to the input.
3. They do not need to maintain additional information (e.g. balance information for search trees) and therefore they need less space.
4. They usually have simple and easy to implement algorithms for their operations.

Their possible disadvantages are (Sleator and Tarjan, 1985b):

- They need more local adjustments (they reorganize their state even on a search).

- Their worst case performance for an individual (non-amortized) operation can be bad.

Two very popular self-organizing data structures for the dictionary problem are unsorted linear lists and binary search trees (Albers and Westbrook, 1998). The most common self-organizing binary search tree is the *splay tree* as introduced by Sleator and Tarjan (1985b). Several results are known about the performance and structure of splay trees and other self-organizing binary search trees (see e.g. Tarjan, 1985; Cole, 2000; Cole et al., 2000; Elmasry, 2004; Georgakopoulos, 2004; Demaine et al., 2007; Wang et al., 2006).

Using an unsorted list to support dictionary operations is usually called the *list update* or *list accessing* problem. Consider an unsorted list of ℓ items. Let $\mathcal{A}$ be an arbitrary online list update algorithm. The input to $\mathcal{A}$ is a sequence of n requests that must be served in an online manner. To serve a request to an item x, $\mathcal{A}$ linearly searches the list until it finds x. If x is the i^{th} item in the list, $\mathcal{A}$ incurs a cost i to access x. Immediately after this access, $\mathcal{A}$ can move x to any position closer to the front of the list at no extra cost. This is called a *free exchange.* Also $\mathcal{A}$ can exchange any two consecutive items at unit cost. These are called *paid exchanges.* An efficient algorithm can thus use free and paid exchanges to minimize the overall cost of serving a sequence. This is called the *standard cost model* (Albers and Westbrook, 1998). Later in this section we will describe an alternative cost model.

List update algorithms were among the first algorithms studied using competitive analysis. Three well known deterministic online algorithms are Move-To-Front (MTF), Transpose (TR), and Frequency-Count (FC). MTF moves the requested item to the front of the list whereas TR exchanges the requested item with the item that immediately precedes it. FC maintains an access count for each item ensuring that the list always contains items in non-increasing order of frequency count. Sleator and Tarjan showed that MTF is 2-competitive, while TR and FC do not have constant competitive ratios (Sleator and Tarjan, 1985a). Since then, several other deterministic and randomized online algorithms have been studied using competitive analysis (see Irani, 1991; Albers, 1998; Albers et al., 1995, for some representative results).

Albers (1998) introduced the algorithm Timestamp (TS) and showed that it is 2-competitive. After accessing an item a, TS inserts a in front of the first item b that appears before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorder the list. Schulz (1998) introduced an infinite (uncountable) family of list update algorithms called Sort-By-Rank (SBR). All algorithms in this family achieve the optimal competitive ratio 2 and they mediate between MTF and TS. Consider a sequence $\sigma = \sigma_1\sigma_2\cdots\sigma_m$ of length m. For an item a and a time $1 \leq t \leq m$, denote by $w_1(a,t)$ and $w_2(a,t)$ the time of the last and the second last access to a in $\sigma_1\sigma_2\cdots\sigma_t$, respectively. If a has not been accessed so far, set

$w_1(a,t) = 0$ and if a has been accessed at most once, set $w_2(a,t) = 0$. Then we define $s_1(a,t) = t - w_1(a,t)$ and $s_2(a,t) = t - w_2(a,t)$. Note that after each access, MTF and TS reorganize their lists so that the items are in increasing order by s_1 and s_2, respectively[1]. For a parameter $0 \leq \alpha \leq 1$, SBR(α) reorganizes its list after the t^{th} access so that items are sorted by their α-rank function defined as $r_\alpha(a,t) = (1-\alpha) \times s_1(a,t) + \alpha \times s_2(a,t)$.[2] More formally, upon a request for an item a in time t, SBR(α) inserts a just after the last item b in front of a with $r_\alpha(b,t) < r_\alpha(a,t)$. Furthermore, if there is no such item b or this is the first access to a, SBR(α) inserts a at the front of the list. Therefore SBR(0) is equivalent to MTF and SBR(1) is equivalent to TS except for the handling of the first accesses, i.e., they would be equivalent if TS moved an item that has been accessed only once so far to the front of the list.

While list update algorithms with better competitive ratios tend to have better performance in practice, the validity of the cost model has been debated. More precisely, Martínez and Roura (2000) and Munro (2000), independently addressed the drawbacks of the standard cost model. Let $(a_1, a_2, \ldots, a_\ell)$ be the list currently maintained by an algorithm $\mathcal{A}$. Martínez and Roura argued that in a realistic setting a complete rearrangement of all items in the list which precede item a_i would in practice require time proportional to i, while this has cost proportional to i^2 in the standard cost model. Munro provided the example of accessing the last item of the list and then reversing the entire list. The real cost of this operation in an array or a linear link list should be $O(\ell)$, while it costs about $\ell^2/2$ in the standard cost model. As a consequence, their main objection to the standard model is that it prevents online algorithms from using their true power. They instead proposed a new model in which the cost of accessing the i^{th} item of the list plus the cost of reorganizing the first i items is linear in i. We will refer to this model as the *modified cost model.*

Surprisingly, it turns out that the offline optimum benefits substantially more from this realistic adjustment than the online algorithms do. Indeed, under this model, every online algorithm has amortized cost of $\Theta(\ell)$ per access for some arbitrary long sequences, while an optimal offline algorithm incurs a cost of $O(\log \ell)$ per access on every sequence. Hence all online list update algorithm have a competitive ratio of $\Omega(\ell/\log \ell)$. One may be tempted to argue that this is proof that the new model makes the offline optimum too powerful and hence this power should be removed, however this is not correct as in real life online algorithms can rearrange items at the cost indicated. Observe that the ineffectiveness of this power for improving the worst case competitive ratio does not preclude the possibility that under certain realistic input distributions (or other similar assumptions on the input) this power might be of use. Martínez and Roura observed this and posed the question: "an important open question is whether there exist alternative ways

[1] For TS, strictly speaking, this applies only to items that have been accessed at list twice.
[2] Schulz (1998) denoted this by $r_t(a,\alpha)$, here we follow the convention of Dorrigiv et al. (2009).

to define competitiveness such that MTF and other good online algorithms for the list update problem would be competitive, even for the [modified] cost model".

2.2 Paging

The paging problem is one of the most important online problems both in theory and in practice. As discussed before, the competitive analysis of online paging algorithms is not entirely satisfactory. Therefore substantial research has been done to obtain alternative performance measures for paging algorithms. We briefly described the paging problem before. In this problem we have a small fast memory (cache) of size k and a larger slow memory of size M. Each input is a sequence of page requests. For each request, if the requested page is in the cache, a *hit* occurs and the algorithm can serve the request without incurring any cost. Otherwise a *fault* occurs and the algorithm should bring the requested page to the cache. Also if the cache is already full, the algorithm should evict at least one page in order to make room for the new page. The paging algorithms are usually specified by the *eviction algorithm* they use on a fault. The cost of a paging algorithm $\mathcal{A}$ on an input sequence σ is the number of faults it incurs to serve σ.

Due to the importance of this problem, several paging algorithms have been proposed. The most well known paging algorithms are Least-Recently-Used (LRU) and First-In-First-Out (FIFO). When an eviction is required, LRU evicts the page that is least recently used and FIFO evicts the page that was first brought to the cache. Flush-When-Full (FWF) is another algorithm that evicts all the pages that are currently in the cache when a fault occurs and the cache is full. It turns out that there exists a simple and efficient optimal algorithm for the offline version of paging (this is not always the case for online problems, e.g., the list update problem (Borodin and El-Yaniv, 1998)). Longest-Forward-Distance (LFD) evicts the page whose next request is latest, if an eviction is required. (Belady, 1966) proved that LFD is an optimal offline paging algorithm.

Furthermore, paging algorithms are classified by certain common properties. A *lazy* paging algorithm does not evict a page on a hit, and evicts at most one page on a fault. Thus FWF is not lazy while LRU and FIFO are lazy algorithms. A paging algorithm is called *conservative* if it incurs at most k page faults on any page sequence that contains at most k distinct pages. It can be shown (Borodin and El-Yaniv, 1998) that LRU and FIFO are conservative algorithms and FWF is not. Another class of online paging algorithms are *marking* algorithms. A marking algorithm $\mathcal{A}$ works in phases. Each phase starts right after the last request of the previous phase and consists of the maximal sequence of requests that contains at most k distinct pages. All the pages in the cache are unmarked at the beginning of each phase. We mark any page just after the first request to it. When an eviction is necessary, $\mathcal{A}$ should evict an unmarked page. It is easy to show that LRU and FWF are marking algorithms while FIFO is not. It is known that the competitive ratio of any conservative or marking paging algorithm is k and this is the best possible among deterministic online algorithms (Borodin and El-Yaniv, 1998). Therefore LRU, FIFO, and FWF all have competitive ratio k. Unfortunately this is not consistent with our expectations. We will elaborate more on this later in the thesis.

Finally we describe a few other paging algorithms that will be of use. LRU-2 is a paging algorithm proposed by O'Neil et al. (1993) for database disk buffering. On a fault, LRU-2 evicts the page whose second to last request is least recent. If there are pages in the cache that have been requested only once so far, LRU-2 evicts the least recently used among them. O'Neil et al. provided experimental results supporting that LRU-2 performs better than LRU in database systems. Boyar et al. (2006) proved that LRU-2 has competitive ratio $2k$. On a fault (with a full cache), Last-In-First-Out (LIFO) evicts the page that is most recently brought to the cache, and Least-Frequently-Used (LFU) evicts the page that has been requested the least since entering the cache. LFU and LIFO do not have constant competitive ratios (Borodin and El-Yaniv, 1998).

As stated before, various alternatives to the competitive ratio have been proposed in the literature. In this chapter we survey several of those alternatives, highlight their distinctive properties and discuss their benefits and drawbacks.

2.3 Max/Max Ratio

The Max/Max ratio Ben-David and Borodin (1994) tries to be more optimistic by comparing the amortized worst case behaviour of the online algorithm with the amortized worst case behaviour of the optimal offline algorithm. Recall that in competitive analysis we compare the two algorithms on the same sequence. However, this approach is sometimes problematic because the existence of only one bad sequence for an online algorithm can drastically change the result. This measure tries to avoid the situation in which a single unusual sequence leads to a very bad ratio. This becomes more clear with the following example used in Ben-David and Borodin (1994) as a motivation for defining the Max/Max ratio. Consider the problem of buying an insurance policy in an online manner. It is reasonable to pay \$5 a month to insure your car against theft. However, this is not a competitive strategy because the offline adversary can select the scenario in which one will never present a claim to the insurance agent. Conversely if no theft insurance is purchased, then the adversary can select the scenario in which the car is stolen. In the Max/Max ratio, we compare the two algorithms on their respective worst case sequences of the same length.

We formally define this measure for an online minimization problem Π. The definition for maximization problems is similar. Let $\mathcal{A}$ be an algorithm for Π.

Definition 2.1 The amortized cost of $\mathcal{A}$ is defined as $M(A) = \limsup_{n\to\infty} M_n(\mathcal{A})$ where $M_n(\mathcal{A}) = \max_{|\sigma|=n} \mathcal{A}(\sigma)/n$. The *Max/Max ratio* of $\mathcal{A}$ denoted $w_M(\mathcal{A})$ is

$$\limsup_{n\to\infty} \frac{M_n(\mathcal{A})}{M_n(\textsc{opt})} = \frac{M(\mathcal{A})}{M(\textsc{opt})}$$

where $\textsc{opt}$ is an optimal offline algorithm.

Note that we can directly compare two online algorithms $\mathcal{A}$ and $\mathcal{B}$ using this measure because we have $\frac{M(\mathcal{A})}{M(\mathcal{B})} = \frac{w_M(\mathcal{A})}{w_M(\mathcal{B})}$. Also when considering the Max/Max ratio, lookahead can improve the online performance even in cases where the competitive ratio does not improve Ben-David and Borodin (1994). In Chapter **??** we will see that all lazy paging algorithms are equivalent according to the Max/Max ratio.

2.4 Random Order Ratio

The random order ratio Kenyon (1996) is another measure that tries to decrease the dependence on some unusual bad sequences. Let $\mathcal{A}$ be an online algorithm for a minimization problem.

Definition 2.2 The *random order ratio* of $\mathcal{A}$ is defined as

$$RC(\mathcal{A}) = \limsup_{\text{OPT}(\sigma)\to\infty} \frac{E_\pi[\mathcal{A}(\sigma_\pi)]}{\text{OPT}(\sigma)}$$

where π is a permutation of $\{1, 2, \ldots, n\}$, σ_π is the permuted sequence $(\sigma_{\pi_1}, \ldots, \sigma_{\pi_n})$, and the expectation is taken over all permutations of $\{1, 2, \ldots, n\}$.

Therefore this measure assumes that all orderings of an input sequence are equally likely. This is a reasonable assumption for some online problems. Kenyon Kenyon (1996) proves a lower and an upper bound on the random order ratio of the BEST-FIT algorithm for online bin-packing which are better than the corresponding competitive ratio bounds. However, it seems that this measure is difficult to generalize to other online problems.

2.5 Relative Worst Order Ratio

The relative worst order ratio Boyar and Favrholdt (2007); Boyar and Medvedev (2008); Boyar et al. (2007) combines some desirable properties of the Max/Max ratio and the random order ratio. Using this measure we can directly compare two online algorithms. Informally, for a given sequence it considers the worst case ordering of that sequence for each algorithm and compares their behaviours on these orderings. Then it finds among all sequences (not just reorderings) the one that maximizes the worst case performance. Thus this measure can be considered as a modification of the Max/Max ratio in that we consider the worst sequence among those which are permutations of each other instead of considering the worst sequence among all those having the same length as the Max/Max ratio does. It is also related to random order ratio as it considers permutations of a sequence. However instead of taking the expectation of the algorithm's behaviour on all permutations, it considers the permutation with the worst behaviour.

Let $\mathcal{A}$ and $\mathcal{B}$ be online algorithms for an online minimization problem. Denote by σ_π the sequence obtained by applying a permutation π to σ, i.e., $\sigma_\pi = (\sigma_{\pi_1}, \ldots, \sigma_{\pi_n})$. Define $\mathcal{A}_W(\sigma) = \max_\pi \mathcal{A}(\sigma_\pi)$.

Definition 2.3 Boyar et al. (2007) Let $S_1(c)$ and $S_2(c)$ be the statements about algorithms $\mathcal{A}$ and $\mathcal{B}$ defined in the following way.

$S_1(c)$: There exists a constant b such that $\mathcal{A}_W(I) \le c \cdot \mathcal{B}_W(\sigma) + b$ for all σ.

$S_2(c)$: There exists a constant b such that $\mathcal{A}_W(I) \ge c \cdot \mathcal{B}_W(\sigma) - b$ for all σ.

The relative worst order ratio $WR_{\mathcal{A},\mathcal{B}}$ of $\mathcal{A}$ to $\mathcal{B}$ is defined if $S_1(1)$ or $S_2(1)$ holds. In this case $\mathcal{A}$ and $\mathcal{B}$ are said to be comparable. If $S_1(1)$ holds, then $WR_{\mathcal{A},\mathcal{B}} = \sup\{r|S_2(r)\}$, and if $S_2(r)$ holds then $WR_{\mathcal{A},\mathcal{B}} = \inf\{r|S_1(r)\}$.

$WR_{\mathcal{A},\mathcal{B}}$ can be used to compare the qualities of $\mathcal{A}$ and $\mathcal{B}$. If $WR_{\mathcal{A},\mathcal{B}} = 1$ then these two algorithms have the same quality with respect to this measure. The magnitude of difference between $WR_{\mathcal{A},\mathcal{B}}$ and 1 reflects the difference between the behaviour of the two algorithms. For a minimization problem, $\mathcal{A}$ is better than $\mathcal{B}$ with respect to this measure if $WR_{\mathcal{A},\mathcal{B}} < 1$ and vice versa.

The idea behind this measure is that some online algorithms perform well on some types of sequence orderings and other algorithms perform well on some other types of orderings. Therefore certain algorithms that cannot be compared using competitive analysis may be comparable in this measure. Boyar and Favrholdt show that the relative worst order ratio is transitive Boyar and Favrholdt (2007).

Note that we can also compare the online algorithm $\mathcal{A}$ to an optimal offline algorithm OPT. The *worst order ratio* of $\mathcal{A}$ is defined as $WR_{\mathcal{A}} = WR_{\mathcal{A},\text{OPT}}$. For some problems, OPT is the same for all orderings of requests on a given sequence and hence the worst order ratio is the same as the competitive ratio. However for other problems such as fair bin packing the order does matter for OPT.

In Boyar and Medvedev (2008), three online algorithms (FIRST-FIT, BEST-FIT, and WORST-FIT) for two variants of the seat reservation problem Boyar and Larsen (1999) are compared using the relative worst order ratio. All of these three algorithms can be compared in this framework while they have the same performance within the classical competitive analysis framework.

For paging algorithms, LRU is strictly better than FWF with respect to the worst order ratio Boyar et al. (2007), while these two algorithms have the same competitive ratio. Also a new paging algorithm, Retrospective-LRU (RLRU), is proposed and it is shown that RLRU is better than LRU with respect to the relative worst order ratio. This is in contrast to what the competitive ratio of these algorithms predicts. It is also shown that lookahead is helpful when we consider the relative worst order ratio.

2.6 Loose Competitiveness

Loose competitiveness was first proposed in Young (1994) and later modified in Young (2002). We describe it for an online minimization problem. It attempts to obtain a more realistic measure by considering the following two aspects in the analysis of online algorithms. First, in many real-life online problems, we can ignore those sequences on which the online algorithm incurs a cost less than a certain threshold. Second, many online problems have a second resource parameter (e.g. size of the cache, number of servers) and input sequences are independent of these parameters. In contrast, in competitive analysis the adversary can select sequences tailored against those parameters. We clarify this situation by considering the paging problem. In this case the problem parameter is k, the size of the cache. Consider the following lower bound on the competitive ratio of any deterministic paging algorithm.

Theorem 2.4 *Sleator and Tarjan (1985a) The competitive ratio of any deterministic online paging algorithm is at least k.*

This result can be easily proven by considering an adversary that selects only $k+1$ pages and at each time requests a page that is not in the cache. For this to work the adversary needs to know the problem parameter k. However, in practice the competitive ratios of many online paging algorithms have been observed to be constant Young (2002), i.e., independent of k. This can be obtained by applying loose competitiveness.

In loose competitiveness we consider an adversary that is oblivious to the parameter by requiring it to give a sequence that is bad for most values of the parameter rather than just a specific bad value. Let $\mathcal{A}_k(\sigma)$ denote the cost of an algorithm $\mathcal{A}$ on an input sequence σ, when the parameter of the problem is k.

Definition 2.5 Young (2002) An algorithm $\mathcal{A}$ is (ε, δ)-loosely c-competitive if, for any input sequence σ and for any n, at least $(1-\delta)\, n$ of the values $k \in \{1, 2, \ldots, n\}$ satisfy

$$\mathcal{A}_k(\sigma) \leq \max\{c \cdot \textsc{opt}_k(\sigma), \varepsilon\, |\sigma|\}.$$

Therefore we ignore sequences σ which cost less than $\varepsilon\, |\sigma|$. Also we require the algorithm to be good for at least a $(1-\delta)$-fraction of the possible parameters. For each online problem, we can select the appropriate constants ε and δ. The following result shows that by this modification of the competitive analysis, we can obtain paging algorithms with constant performance ratios.

Theorem 2.6 *Young (2002) Every k-competitive paging algorithm is (ε, δ)-loosely c-competitive for any $0 < \varepsilon, \delta < 1$, and $c = (e/\delta)\ln(e/\varepsilon)$, where e is the base of the natural logarithm.*

2.7 Accommodating Ratio and Accommodating Function

These two measures are the same as the competitive ratio except that they restrict the set of legal input sequences. They apply to online problems with limited resources and only consider those sequences in which the optimal solution does not benefit from having more than a certain amount of resources. We use an example to explain this. The *fair bin packing* problem consists of n bins of size k and an input sequence σ of items where the size of each item is an integer between 1 and k. This sequence is given to the algorithm in an online manner and we want to maximize the total number of items in the bins. Also the packing should be fair; we can reject an item only if it cannot fit in any bins when it is given. Although the optimal offline algorithm knows the whole sequence σ in advance, it should *fairly* process the requests in the same order.

Now for the accommodating ratio Boyar and Larsen (1999) we consider those sequences that can be packed in n bins by a fair optimal offline algorithm. In general we only consider those sequences in which the optimal offline algorithm does not benefit from having more resources than those already available. For the accommodating function Boyar et al. (2001, 2003) we only consider those sequences that can be packed in αn bins by a fair optimal offline algorithm $\textsc{opt}$. Boyar et al. Boyar et al. (2001) argue that if we allow large values of α, then we will have sequences that cannot be handled very well by $\textsc{opt}$ and this can be unrealistic for some applications. Thus they consider small values of $\alpha \geq 1$.

More precisely, consider an online minimization problem Π with limited resources. Let $\mathcal{A}(\sigma)$ be the cost of an online algorithm A on an input sequence σ and let $\textsc{opt}(\sigma)$ be the cost of an optimal offline algorithm on σ. Let $\textsc{opt}_m$ be the cost of $\textsc{opt}$ when an amount m of the limited resource is available.

Definition 2.7 Boyar et al. (2003) Let Π be an online problem with a fixed amount n of resources. For any $\alpha > 0$, an input sequence σ is said to be an α-sequence, if $\textsc{opt}_{\alpha n}(\sigma) = \textsc{opt}_{n'}(\sigma)$, for all $n' \geq \alpha n$ (1-sequences are also called accommodating sequences).

Definition 2.8 Algorithm $\mathcal{A}$ is c-competitive on α-sequences, if there exists a constant b, such that $\mathcal{A}(\sigma) \leq c \cdot \textsc{opt}(\sigma) + b$, for any α-sequence σ. The *accommodating function* is defined as

$$\mathsf{A}_{\mathcal{A}}(\alpha) = \inf\{c \,|\, \mathcal{A} \text{ is } c\text{-competitive on } \alpha\text{-sequences}\}$$

The accommodating ratio is the same as the competitive ratio when we restrict the input to accommodating sequences (1-sequences). Therefore the accommodating ratio is usually called the competitive ratio on accommodating sequences. Note that accommodating ratio is equivalent to $\mathsf{A}(1)$. Also the competitive ratio is $\lim_{\alpha\to\infty} \mathsf{A}(\alpha)$. Thus the accommodating function is an extension of both the competitive ratio and the accommodating ratio.

Several papers Boyar and Larsen (1999); Boyar et al. (1999); Bach et al. (2000, 2003); Epstein and Favrholdt (2003); Boyar and Nielsen (1999); Azar et al. (2002) use the accommodating ratio as the measure of quality for online algorithms. Boyar et al. Boyar and Larsen (1999) give lower bounds and upper bounds for the competitive ratio and accommodating ratio of two versions (unit price and proportional price) of the *seat reservation* problem. They prove these results for Best-Fit, First-Fit, and general deterministic fair algorithms. The two measures (the competitive ratio and the accommodating ratio), agree on the proportional price problem but differ on the unit price problem. The bounds are tight for the proportional price problem, but not in the unit price problem. Bach et al. Bach et al. (2000, 2003) give tight bounds for the deterministic fair algorithms for the unit price problem. They also consider randomized algorithms and prove some bounds on them.

Several algorithms for the online fair bin packing problem are analyzed in Boyar et al. (1998). Upper bounds and lower bounds are given for the accommodating ratio for First-Fit, Worst-Fit, and general algorithms. The lower bound for First-fit is improved in Boyar and Nielsen (1999). According to these bounds, First-Fit has strictly better accommodating ratio than Worst-Fit. However if we consider the standard competitive ratio, it can be shown Boyar et al. (1999) that Worst-Fit behaves strictly better than First-Fit. Therefore the competitive ratio and the accommodating ratio can give contradictory results. Epstein and Favrholdt Epstein and Favrholdt (2003) consider a variation of online fair bin packing in which bins can have different sizes and give upper bounds and lower bounds for the accommodating ratio of several online algorithms.

The unrestricted bin-packing problem is the same as the fair bin packing problem except that we do not require the algorithms to be fair. Azar et al. Azar et al. (2002) study this variation of the problem and compare it with fair bin packing using the accommodating ratio. They prove an asymptotically tight bound for the accommodating ratio of First-Fit for the fair bin packing problem. They design an online algorithm called Unfair-First-Fit which has asymptotically better accommodating ratio than First-Fit in the unrestricted bin packing problem. Finally upper bounds on the accommodating ratio of deterministic and randomized algorithms are proven for the unrestricted bin packing problem.

The accommodating function is studied by Boyer et al. for the fair bin packing problem Boyar et al. (2001). They prove lower and upper bounds on the accommodating functions of First-Fit, Worst-Fit, and all deterministic fair algorithms for all $\alpha \geq 1$. A variant of the seat reservation problem in which seat changes are allowed is studied in Boyar et al. (2004). Lower bounds and upper bounds for the competitive ratio, accommodating ratio, and accommodating function are proven for several algorithms. Finally, Boyer et al. Boyar et al. (2003) extend the accommodating function to values of $\alpha < 1$. They study the accommodating function of several algorithms for the seat reservation and unrestricted bin packing problems. For the seat reservation problem, they show that we can separate the performance of three algorithms using the accommodating function at $\alpha = 1/3$, while we cannot do the same using the competitive ratio or the accommodating ratio. They also studied the connection between the accommodating function and the resource augmentation technique Kalyanasundaram and Pruhs (2000).

2.8 Access Graph Model

The access graph model was introduced by Borodin et al. to solve two main problems in competitive analysis of online paging algorithms Borodin et al. (1995). One of these problems is that the practical performance ratio of LRU is much better than its competitive ratio. We have mentioned the second problem before: Although LRU and FIFO have the same competitive ratio, LRU behaves much better than FIFO in practice. One reason that LRU has good experimental behaviour is that in practice page requests show *locality of reference*. Temporal locality means that when a page is requested it is more likely to be requested in the near future. Spatial locality means that when a page is requested it is more likely that a nearby page will be requested in the near future.

In the access graph model we weaken the adversary by restricting the set of legal input sequences. This is done by restricting the set of pages that can be requested after each page. More specifically, we have an access graph $G = (V, E)$ so that each vertex v represents a page p_v and there is an edge from a vertex u to a vertex v if and only if p_v can be requested after p_u. This graph can be directed or undirected depending on the actual problem. Locality of reference can be imposed in this model because when we request a page p we should request p or one of its neighbours in the next step. The competitive ratio is the same as in standard competitive analysis except that we restrict ourselves to the input sequences that conform to the given access graph.

Using this model, several interesting results can be obtained Borodin et al. (1995); Irani et al. (1996); Chrobak and Noga (1999); Fiat and Rosen (1997). For every graph G and every number k of pages in the fast memory, let $c_k(G)$ denote the best competitive ratio that can be achieved by an online paging algorithm. Borodin et al. prove that the value $c_k(G)$ is computable for every finite access graph G Borodin et al. (1995). They also show how to compute the competitive ratio of LRU for every access graph and every k within a factor of two (plus additive constant), and propose a simple algorithm that nearly achieves the best competitive ratio for every access graph. This algorithm, called FAR, evicts, on each fault, the unmarked page in cache whose distance from a marked page is maximum in the access graph. They proved that the competitive ratio of FAR for every undirected access graph and every k is within $O(\log k)$ of the best possible competitive ratio. This was later improved by Irani et al. who showed that the competitive ratio of FAR is $O(c_k(G))$ for any undirected graph G Irani et al. (1996). Experimental results of Borodin et al. (1995) show that some variations of FAR behave better than LRU in practice. It is also known that the competitive ratio of LRU is at least as good as FIFO on every access graph Chrobak and Noga (1999).

Karlin, Phillips, and Raghavan Karlin et al. (2000) extended the access graph model by assigning probabilities to the edges of the graph. In other words, they assume that the request sequences are generated by a Markov chain process on the set of pages. Given such a Markov chain, the objective is to find an efficient online algorithm that minimizes the expected number of faults per request. They provide an efficient algorithm that achieve bounds within a constant of the best online algorithm.

2.9 Diffuse Adversary Model

The diffuse adversary model Koutsoupias and Papadimitriou (2000) tries to refine the competitive ratio by decreasing the power of the adversary. It does this by restricting the set of legal input sequences. Recall that in standard competitive analysis we do not put any restriction on the input sequences and so they can have any distribution. In other words, the online algorithm knows nothing about the distribution of the input sequences. At the other end of the spectrum, in classic average-case analysis of online algorithms, the exact distribution of input sequences is known to the online algorithm. In the diffuse adversary model, the online algorithm does not know the exact distribution, but it knows that it is a member of a class Δ of distributions.

Definition 2.9 Let $\mathcal{A}$ be an online algorithm for a minimization problem and let Δ be a class of distributions for the input sequences. Then $\mathcal{A}$ is c-competitive against Δ, if there exists a constant b, such that

$$E_{\sigma \in D}[\mathcal{A}(\sigma)] \leq c \cdot E_{\sigma \in D}[\textsc{opt}(\sigma)] + b,$$

for every distribution $D \in \Delta$, where $\mathcal{A}(\sigma)$ denotes the cost of $\mathcal{A}$ on the input sequence σ and the expectations are taken over sequences that are picked according to D.

In other words the adversary selects the distribution D in Δ that is the worst distribution for $\mathcal{A}$. If Δ is more restricted then $\mathcal{A}$ knows more about the distribution of input sequences and the power of adversary is more constrained. When Δ contains all possible distributions then competitive analysis against Δ is the same as the standard competitive ratio. Therefore the diffuse adversary model is an extension of the standard competitive analysis. Note that we can also model locality of reference using the diffuse adversary model by considering only those distributions that are consistent with the given access graph. This means that if there is no edge between the vertices corresponding to two pages, the probability that one page is accessed after the other should be zero in our distributions.

This model is applied to paging Koutsoupias and Papadimitriou (2000) by considering a class Δ_ε of distributions and proving that LRU has the best competitive ratio against Δ_ε among all deterministic online algorithms. For any sequence ρ of pages and any page p, let $Pr(p|\rho)$ denote the probability that p is the next page requested provided that the request sequence seen so far is ρ. For any $0 \leq \varepsilon \leq 1$, Δ_ε contains distributions in which $Pr(p|\rho) \leq \varepsilon$ for every page p and every page sequence ρ. Young Young (1998, 2000) computed the actual competitive ratios of both deterministic and randomized algorithms against Δ_ε. He proves that around the threshold $\varepsilon \approx 1/k$, the best competitive ratios against Δ_ε are $\Theta(\ln k)$ for both deterministic and randomized algorithms. The competitive ratios rapidly become constant for values of ε less than the threshold. For $\varepsilon = \omega(1/k)$, i.e., values greater than the threshold, the competitive ratio rapidly tends to $\Theta(k)$ for deterministic algorithms while it remains $\Theta(\ln k)$ for randomized algorithms. He shows that for $\varepsilon \geq 1/k$, FIFO and FWF have competitive ratio k against Δ_ε. Thus in this case they are outperformed by LRU, which has competitive ratio $\Theta(\ln k)$.

Becchetti Becchetti (2004) argues that the class Δ_ε does not reflect locality of reference. He considers other classes of distributions that model locality of reference. He then proves that LRU achieves good competitive ratios against these classes, while the performance of FWF against them is bad.

2.10 Probabilistic Competitiveness

Standard competitive analysis is a worst-case measure. Diffuse adversary, as a model for probabilistic competitive analysis, is based on the average-case analysis. Recall that in average-case analysis of online algorithms, we assume a probability distribution D on sequences and compute the expected performance of algorithms under D. The diffuse adversary model is different from the average-case analysis in two ways:

1. It considers a class Δ of possible distributions, instead of a single distribution D.
2. It normalizes the expected performance of algorithms to the expected performance of OPT.

There is an alternative way to compare the performance of online algorithms with OPT under probabilistic assumptions. This is related to two possibilities for defining the competitiveness against a fixed probability distribution. The diffuse adversary model uses the following definition:

Definition 2.10 Irani (1998) Let D be a probability distribution on request sequences σ. An algorithm $\mathcal{A}$ is c-competitive against D if there exists a constant, b, such that

$$E_D[\mathcal{A}(\sigma)] \leq c \cdot E_D[\textsc{opt}(\sigma)] + b.$$

Equivalently, we can say that in this definition we compute

$$\frac{E_D[\mathcal{A}(\sigma)]}{E_D[\textsc{opt}(\sigma)]}.$$

The alternative is the *instance-wise* definition.

Definition 2.11 Let D be a probability distribution on request sequences σ. The *a*verage-case competitive ratio of an algorithm $\mathcal{A}$ against D is defined as

$$E_D\left[\frac{\mathcal{A}(\sigma)}{\textsc{opt}(\sigma)}\right].$$

Fujiwara and Iwama Fujiwara and Iwama (2005) apply average-case competitive analysis to the *ski-rental* problem. In this problem, a skier should decide between renting and purchasing a pair of skis. The cost of purchasing is s times the cost of renting. The problem is online: the skier does not know how many times she would go skiing. It is well known Karlin (1998) that the optimal deterministic strategy is to buy the skis after $s-1$ rental times. Fujiwara and Iwama (2005) considers the following scenario: At each time, the skier quits skiing with probability $\lambda > 0$. Then the total number of ski trips follows exponential distribution $\lambda e^{-\lambda t}$. Fujiwara and Iwama prove that the algorithm with optimal average-case competitive ratio against this distribution always rents skis if $s \geq 1/\lambda$. Otherwise it buys the skis after $\approx s^2\lambda$ times. This is different from optimal algorithms in competitive analysis and standard average-case analysis frameworks.

Similarly, they propose the following alternative definition for the diffuse adversary model.

Definition 2.12 Let $\mathcal{A}$ be an online algorithm for a minimization problem and let Δ be a class of distributions for the input sequences. Then the average-case competitive ratio of $\mathcal{A}$ against Δ is defined as

$$\max_{D\in\Delta}\left\{E_D\left[\frac{\mathcal{A}(\sigma)}{\textsc{opt}(\sigma)}\right]\right\}.$$

Souza Souza (2006) calls the competitive ratios defined in Definitions 2.9 and 2.12 *the expected competitive ratio* and *the average performance ratio*, respectively. He argues that for some applications the average performance ratio is preferable to the expected competitive ratio. Bin packing algorithms have been extensively studied using the average performance ratio, e.g., see Coffman Jr. et al. (1997) and references therein. This ratio has been applied to some scheduling problems as well Coffman Jr. and Gilbert (1985); Scharbrodt et al. (2006); Souza and Steger (2006). Note that the average performance ratio is called the expected competitive ratio in these papers.

Actually, the average-case competitive ratio was first introduced in the context of competitive searches on the real line for a target of unknown location. The classic problem in this field is the cow path problem. As traditionally described, a cow reaches a fork on the road and recalls that in one and only one of the two paths there is a pasture field. This problem was first analyzed under the competitive ratio and its solution predates the introduction of the competitive ratio in online algorithms. The optimal solution under the standard competitive ratio metric is 9-competitive. However, on the average, the pasture is discovered at a cost of approximately 4.59 times the optimal path to the pasture. Interestingly enough, the strategy resulting in the optimal average cost is different from the optimal one under the competitive ratio framework Gal (1979); López-Ortiz (1996). Formally we have

Definition 2.13 The average competitive ratio, or *average ratio* for short is defined as

$$E_{\forall\ |\sigma|}\left[\frac{\mathcal{A}(\sigma)}{\textsc{opt}(\sigma)}\right].$$

2.11 Smoothed Competitiveness

Some algorithms that have very bad *worst case* performance behave very well in practice. One of the most famous examples is the simplex method. This algorithm has very good performance in practice but it has exponential running time in the worst case. Average-case analysis of algorithms can somehow explain this behaviour but sometimes there is no basis to the assumption that the inputs to an algorithm are random.

Smoothed analysis of algorithms Spielman and Teng (2004) tries to explain this intriguing behavior without any prior assumptions on the distribution of the input instances. In this model, we randomly perturb (smoothen) the input instances according to a probability distribution f and then analyze the behavior of the algorithm on these perturbed (smoothed) instances. For each input instance $\check{I}$ we compute the neighborhood $N(\check{I})$ of $\check{I}$ which contains the set of all perturbed instances that can be obtained from $\check{I}$. Then we compute the expected running time of the algorithm over all perturbed instances in this neighborhood. The smoothed complexity of the algorithm is the maximum of this expected running time over all the input instances. Intuitively, an algorithm with bad worst case performance can have good smoothed performance if its worst case instances are isolated. Spielman and Teng show Spielman and Teng (2004) that the simplex algorithm has polynomial smoothed complexity. Several other results are known about the smoothed complexity of algorithms Banderier et al. (2003); Manthey and Reischuk (2007); Blum and Dunagan (2002); Spielman and Teng (2003).

As stated before, competitive analysis is a rather pessimistic measure and an algorithm can have a very bad competitive ratio only because of a few bad sequences. Therefore the competitive ratio is a reasonable choice for applying smoothed analysis. This was first done by Becchetti et al. Becchetti et al. (2003) who introduced *smoothed competitive analysis*. Informally, smoothed competitive analysis is the same as competitive analysis except that we consider the cost of the algorithm on randomly perturbed adversarial sequences. The smoothed competitive ratio of an online algorithm $\mathcal{A}$ for a minimization problem can be formally defined as follows.

Definition 2.14 Becchetti et al. (2003) The *smoothed competitive ratio* of an algorithm $\mathcal{A}$ is defined as

$$c = \sup_{\check{I}} E_{I \leftarrow N(\check{I})}\left[\frac{\mathcal{A}(I)}{\textsc{opt}(I)}\right],$$

where the supremum is taken over all input instances $\check{I}$, and the expectation is taken over all instances I that are obtainable by smoothening the input instance $\check{I}$ according to f in the neighborhood $N(\check{I})$.

Note that it is also possible to define the smoothed competitive ratio as

$$c = \sup_{\check{I}} \frac{E_{I \leftarrow N(\check{I})}[\mathcal{A}(I)]}{E_{I \leftarrow N(\check{I})}[\text{OPT}(I)]}.$$

In Becchetti et al. (2003), the first definition is used but it is remarked that a similar result can be obtained using the second definition. They use the smoothed competitive ratio to analyze the Multi-Level Feedback(MLF) algorithm for processor scheduling in a time sharing multitasking operating system. This algorithm has very good performance in practice but its competitive ratio is very bad. They obtain strictly better ratios using smoothed competitive analysis than with the competitive ratio.

2.12 Search Ratio

The search ratio belongs to the family of measures in which the offline OPT is weakened. It is defined only for the specific case of geometric searches in an unknown terrain for a target of unknown position. Recall that the competitive ratio compares against an all knowing OPT. Indeed, for geometric searches OPT is simply a shortest path algorithm, while the online search algorithm has intricate methods for searching. The search ratio instead considers the case where OPT knows the terrain but not the position of the target. That is, the search ratio compares two search algorithms, albeit one more powerful than the other. By comparing two instances of like objects the search ratio can be argued to be a more meaningful measure of the quality of an online search algorithm. Koutsoupias et al. show that searching in trees exhibits a large competitive ratio regardless of the algorithm, yet under the search ratio framework certain algorithms are far superior to others Koutsoupias et al. (1996).

2.13 Travel Cost

As stated in Chapter **??**, classical complexity time analysis generally uses an unnormalized time measure even though a normalized measure has been defined and proven fruitful in certain settings. This raises the possibility of using an unnormalized cost measure for online algorithms as well. This measure has been used in online geometric searches, in which the main objective is to minimize the length of the longest search sequence, known as the travel cost of the solution. Formally,

Definition 2.15 The *travel cost* of an online algorithm $\mathcal{A}$ on input σ is given by

$$C(n) = \max_{|\sigma| \leq n} \{A(\sigma)\}$$

For example in the case of an actual search and rescue operation in the high seas minimizing the maximum search time is more relevant than the competitive ratio on any particular point in the search path Felner et al. (2004).

2.14 Acceleration Ratio

Acceleration ratio is a worst case measure similar to the competitive ratio. It is defined for some scheduling problems related to anytime algorithms Zilberstein et al. (1999). In an *anytime* algorithm, the output quality is related to the computation time. The behaviour of an anytime algorithm $\mathcal{A}$ on an instance σ is described by a *performance profile* $Q_{\mathcal{A}}(t)$, where $Q_{\mathcal{A}}(t)$ denotes the quality of the output produced by $\mathcal{A}$ in computation time t. It is usually assumed that $Q_{\mathcal{A}}(t)$ is strictly increasing. There are two kinds of anytime algorithms. In an *interruptible* algorithm you can stop the algorithm and ask the result at any time during the execution of the algorithm. On the other hand, in a *contract* algorithm the computation time (contract time) should be given as part of input before the start of its execution. The performance profile $Q_{\mathcal{A}}(t)$ for a contract algorithm $\mathcal{A}$ is the quality of output produced by $\mathcal{A}$ if the computation time t is given to $\mathcal{A}$ prior to its execution.

There are applications in which we need an interruptible algorithm while all we have is a contract algorithm. This happens for example in real-time situations in which we do not know the interruption time in advance. Suppose that we have a contract algorithm $\mathcal{A}$. We can use $\mathcal{A}$ to construct an interruptible algorithm $\mathcal{B}$ as follows. $\mathcal{B}$ considers an increasing sequence $X = (x_1, x_2, \dots)$ of contract times. It first activates $\mathcal{A}$ with contract time x_1. If $\mathcal{A}$ completes its computation without interruption, $\mathcal{B}$ activates $\mathcal{A}$ with contract time x_2. $\mathcal{B}$ continues activating $\mathcal{A}$ with contract times from the sequence X until an interruption occurs; on an interruption, $\mathcal{B}$ returns the output from the last completed execution of $\mathcal{A}$. Since we only consider strictly increasing performance profiles, X should be strictly increasing as well. Now we are ready to define the acceleration ratio.

Definition 2.16 Zilberstein et al. (1999) Let $\mathcal{A}$ be a contract algorithm and $\mathcal{B}$ be an interruptible algorithm produced by the sequence of contracts $X = (x_1, x_2, \dots)$, then the acceleration ratio of X, $r \geq 1$, is the smallest constant c for which:

$$\forall t \geq \frac{x_1}{c} : \quad Q_{\mathcal{B}}(ct) \geq Q_{\mathcal{A}}(t).$$

Russell and Zilberstein Russell and Zilberstein (1991) give a schedule with acceleration ratio of 4 and Zilberstein et al. Zilberstein et al. (1999) show that this is the best possible. Bernstein et al. extend the definition to the case that we have m processors Bernstein et al. (2002). They also give a schedule of acceleration ratio $\frac{(m+1)^{\frac{m+1}{m}}}{m}$ for this case, which is shown to be optimal in López-Ortiz et al. (2006).

2.15 Full Access Cost Model

Recall that the standard model for the analysis of online paging algorithms considers the number of faults as the cost measure: the cost of an algorithm $\mathcal{A}$ on an input sequence σ, denoted by $\mathcal{A}(\sigma)$, is the number of faults $\mathcal{A}$ incurs to serve σ. This is a fair model because usually in the paging problem the fault cost is much more than the hit cost. It is used frequently due to the above fact and its simplicity. Although this is the most common model, a few other cost models have been proposed. One problem with the standard model is that it does not consider the length of sequences; having 100 faults on a sequence of length 100000 is as bad as having 100 faults on a sequence of length 1000. This problem can be fixed by normalizing the number of faults to the length of the sequence. The *fault rate* of an algorithm $\mathcal{A}$ on an input sequence σ, denoted by $F_{\mathcal{A}}(\sigma)$, is defined as $\mathcal{A}(\sigma)/|\sigma|$, where $|\sigma|$ is the length of σ. As stated before, the fault rate is usually used in systems research for analysis of paging. In Section 2.16 we describe some results using fault rate as the cost model.

Torng Torng (1998) proposes the full access cost model for paging. On a hit, the requested page is in fast memory and can be accessed in one time step. On a fault, the corresponding page should be brought to the fast memory in time p and then be accessed in time 1, where p is a parameter of model that is called the *miss penalty*. Thus the cost of a hit is 1 and the cost of a fault is $p + 1$. Suppose that an algorithm $\mathcal{A}$ incurs f faults on an input sequence σ. Then the full access cost of $\mathcal{A}$ on σ is defined as $\mathcal{A}(\sigma) = (|\sigma| - f) \times 1 + f \times (p + 1) = |\sigma| + fp$. Note that in the standard model the cost of a hit is 0 and the cost of a fault is 1. Also we can consider the standard model as a special case of the full access cost model in which we have $p = \infty$. As stated before this is not an unrealistic assumption as the miss penalty for the paging problem can be very large. For example according to Figure 5.37 on page 441 of Hennessy and Patterson (1996), it ranges between 7000 and 150000 on typical systems.

Torng also considers locality of reference in this model. For a sequence σ and an integer $m > 1$, an m-decomposition $D(\sigma, m)$ is defined as partitioning σ into consecutive phases so that each phase is a maximal subsequence that contains at most m distinct pages. More precisely, each phase starts right after the previous phase and ends just before the $(m+1)^{th}$ distinct page. The last phase may contain less than m distinct pages. The size $|D(\sigma, m)|$ of this m-decomposition is defined as the number of its phases. Also $L(\sigma, m) = |\sigma|/|D(\sigma, m)|$ is the average phase length of $D(\sigma, m)$. Note that the phases in a k-decomposition of σ are the same as phases of marking algorithms and are also called k-chunks in literature. Now a sequence σ exhibits significant locality of reference if its k-chunks are long on average, i.e. $L(\sigma, k) \gg k$.

The competitive ratio of an online paging algorithm $\mathcal{A}$ can be defined in an analogous way for this model. Torng shows that the competitive ratio of any marking algorithm $\mathcal{A}$ is $k(p+1)/(k+p) \approx \min(k, p+1)$ and that this is the best possible competitive ratio for any deterministic online algorithm. He also proves the following result about paging with locality of reference. If we consider only those sequences σ for which $L(\sigma, k) \geq ak$, then the competitive ratio of A is $1 + (k-1)/(ak/p+1) < 1 + p/a$ for any marking algorithm $\mathcal{A}$ and again this is the best possible for any deterministic online algorithm, although he considers FIFO as a marking algorithm which is non-standard. He also gives some results for randomized online algorithms.

This model reflects the influence of lookahead. Torng defines a variation of LRU with lookahead and shows that its competitive ratio is better than LRU in the full access cost model.

2.16 Concave Analysis

In this section we describe a model for paging with locality of reference proposed by Albers et al. Albers et al. (2005). This model is based on the concept of working set by Denning Denning (1968). The *working set* is the small set of pages used by a process at any phase of its execution. Consider a function whose value at n is the size of the working set in a window of size n. Denning Denning (1968) shows that if we assume some local statistical regularities in a request sequence, then this function would be increasing and concave.

The idea behind concave analysis Albers et al. (2005) is that we can say a request sequence has locality of reference if the number of distinct pages in a window of size n is small. Consider a function that represents the maximum (average) number of distinct pages in a window of size n, in a request sequence, in terms of n. Extensive experiments with real data show that this function is nearly concave for practical request sequences Albers et al. (2005). Let f be an increasing concave function. There are two possible ways to model locality of reference. In the *Max-Model* we say that a request sequence is consistent with f if the number of distinct pages in any window of size n is at most $f(n)$, for any $n \in \mathcal{N}$. In the *Average-Model* we consider a request sequence consistent with f if the average number of distinct pages in a window of size n is at most $f(n)$, for any $n \in \mathcal{N}$. Now we can model locality by considering only those request sequences that are consistent with f.

Albers et al. consider a slightly more restrictive class of functions called concave* functions, defined as follows.

Definition 2.17 Albers et al. (2005) A function $f : N \to R_+$ is concave* if

1. $f(1) = 1$ and
2. $\forall n \in \mathbb{N} : f(n+1) - f(n) \geq f(n+2) - f(n+1)$.

In the Max-Model we additionally require that f be surjective on the integers between 1 and its maximum value.

Albers et al. Albers et al. (2005) consider the fault rate for measuring the performance of paging algorithms. They define the fault rate of a paging algorithm with respect to a function as follows.

Definition 2.18 Albers et al. (2005) The fault rate of a paging algorithm $\mathcal{A}$ with respect to a concave* function f is

$$F_{\mathcal{A}}(f) = \inf\{r \mid \exists n \in \mathbb{N} : \forall \sigma, \sigma \textit{ consistent with } f, |\sigma| \geq n : F_{\mathcal{A}}(\sigma) \leq r\}.$$

They provide results for several paging algorithms in both models. For the Max-Model they prove a lower bound on the fault rate of any deterministic online paging algorithm and show that LRU matches this lower bound; therefore LRU is an optimal deterministic paging algorithm in this model. They also prove lower bounds and upper bounds for the fault rate of FIFO and FWF. The results show that these algorithms are not optimal as the lower bound is strictly greater than the fault rate of LRU, although they are very close. Finally they give some bounds on the fault rate of LFD. All these bounds are expressed in term of f^{-1}, the inverse function of f, defined as

$$f^{-1}(m) = \min\{n \in \mathcal{N} | f(n) \geq m\}.$$

In other words, $f^{-1}(m)$ denotes the minimum size of a window that contains at least m distinct pages. For example we have $F_{\text{LRU}}(f) = \frac{k-1}{f^{-1}(k+1)-2}$ and $\frac{k-1/k}{f^{-1}(k+1)-1} \leq F_{\text{LRU}}(f) \leq \frac{k}{f^{-1}(k+1)-1}$. Therefore this model distinguishes between LRU and FIFO, although the gap between their fault rates is very small.

They also prove some bounds for these algorithms in the Average-Model. In this model both LRU and FIFO are optimal and we have $F_{\text{LRU}}(f) = F_{\text{FIFO}}(f) = \frac{f(k+1)-1}{k}$. Finally they compare these bounds to the fault rates obtained in some experiments. The rates are close in the Max-Model but not close in the Average-Model. Note that the Average-Model allows more request sequences and may contain some bad sequences that have a large fault rate but do not occur in practice.

2.17 Lookahead

Intuitively, giving an online algorithm partial information about the future makes it more powerful and therefore we expect that an online algorithm with finite lookahead should have better competitive ratio. Unfortunately in many cases competitive analysis does not support this intuition. For example it is well known that giving finite lookahead l to a paging algorithm does not improve its competitive ratio. This is because the adversary can replicate each request l times, making the lookahead useless for the online algorithm.

There are two types of solutions for this counter-intuitive feature. One of them is to define other measures that reflect the effect of lookahead in improving the performance of online algorithms. Max/Max ratio and relative worst order ratio are two such measures. Another solution is to modify the definition of lookahead, ensuring that it provides some useful information for the online algorithm and the adversary cannot make it useless. We describe three such alternative definitions in this section.

Consider *weak lookahead* as the standard definition of lookahead, i.e., an online algorithm has weak lookahead of size l if it sees the current request together with the next l requests. Young proposes *resource-bounded lookahead* as an alternative for weak lookahead Young (1991). An online paging algorithm has resource-bounded lookahead of size l if it sees the current request together with the maximal sequence of future requests for which it incurs at most l faults. Young describes a deterministic marking algorithm that achieves the competitive ratio of $\max\{2k/l, 2\}$ in the resource-bounded lookahead model. Note that the number of pages seen by the algorithm at each time depends on its behaviour in the past and also it can be very large.

Albers introduces *strong lookahead* as a more practical model of lookahead Albers (1997). An online paging algorithm with strong lookahead of size l sees the current request together with the minimal sequence of future requests that contains l pairwise distinct pages other than the current request. She shows that a variant of LRU achieves competitive ratio $k - l$ in the strong lookahead model with $l \leq k - 2$ and that this is the best possible for any deterministic online paging algorithm in this model.

Natural lookahead is another model of lookahead for the paging problem that is proposed by Breslauer Breslauer (1998). An online paging algorithm $\mathcal{A}$ has natural lookahead l if it sees the current request together with the maximal sequence of future requests that contains at most $l+1$ distinct pages that are not in $\mathcal{A}$'s current cache. He proves that a variation of LRU achieves a competitive ratio $\frac{k+l}{l+1}$ in the natural lookahead model.

Note that these alternative models do not reflect how lookahead works in a CPU pipeline. In the pipeline all one can do is look ahead a constant number of instructions into the future, which does not match any of the definitions above. The effect of lookahead for some other online problems is studied in Chung et al. (1989); Irani (1994); Halldórsson and Szegedy (1992); Kao and Tate (1991).

2.18 Comparative Ratio

The comparative ratio was proposed by Koutsoupias and Papadimitriou Koutsoupias and Papadimitriou (2000) as an extension of the competitive ratio to reflect the influence of lookahead.

Definition 2.19 For two classes of algorithms A and B, where typically $A \subseteq B$, the comparative ratio is defined as follows:

$$R(A,B) = \max_{\mathcal{B}\in B} \min_{\mathcal{A}\in A} \max_{\sigma} \frac{\mathcal{A}(\sigma)}{\mathcal{B}(\sigma)}.$$

They provide the following game-theoretic interpretation of this definition: B wants to prove to $\mathcal{A}$ that it is a more powerful class of algorithms. B selects an algorithm $\mathcal{B}$ from its class. A responds by selecting $\mathcal{A}$, one of its algorithms. Finally B chooses a sequence σ that maximizes the ratio $\frac{\mathcal{A}(\sigma)}{\mathcal{B}(\sigma)}$. The goal of B is to maximize this ratio, while A's objective is to minimize it. The larger the ratio, the more powerful B compared to A. Note that by selecting B as the class of all algorithms and A as the class of all online algorithms, we get the competitive ratio definition. Thus competitive analysis is a special case of the comparative analysis.

Koutsoupias and Papadimitriou Koutsoupias and Papadimitriou (2000) prove that finite lookahead of size l is beneficial for paging algorithms. Let A be the class of online paging algorithms and B be the class of paging algorithms with lookahead l. They show the influence of lookahead by proving $R(A,B) = \min\{l+1, k\}$.

2.19 Adequate Analysis

Panagiotou and Souza propose adequate analysis as a model for explaining the efficiency of LRU in practice Panagiotou and Souza (2006). In their work, they classify request sequences according to some parameters and prove an upper bound on the competitive ratio of LRU as a function of these parameters. Then they argue that, in practice, typical request sequences have parameters that lead to a constant competitive ratio for LRU.

For each request σ_i to a page p in a given sequence σ, let d_i be the number of distinct pages that have been requested since the last request to p (not including p). Let c_j be the number of requests σ_i for which $d_i = j$. We have $\text{LRU}(\sigma) = \sum_{j\geq k} c_j$. They prove that $\textsc{opt}(\sigma) \geq \sum_{j\geq k} \frac{j-k+1}{j} c_l$. An (α,β)-*adversary* can only choose sequences for which

$$\sum_{j=k}^{\alpha k-1} c_j \leq \beta \sum_{j\geq \alpha k} c_j.$$

The competitive ratio of LRU against (α,β)-adversaries is at most

$$2(1+\beta)\left(1+\frac{1}{\alpha-1}\right)+\varepsilon,$$

for some $0 \leq \varepsilon \leq 1$. They argue that sequences in practice have large α and small β, and thus the competitive ratio of LRU on them is small.

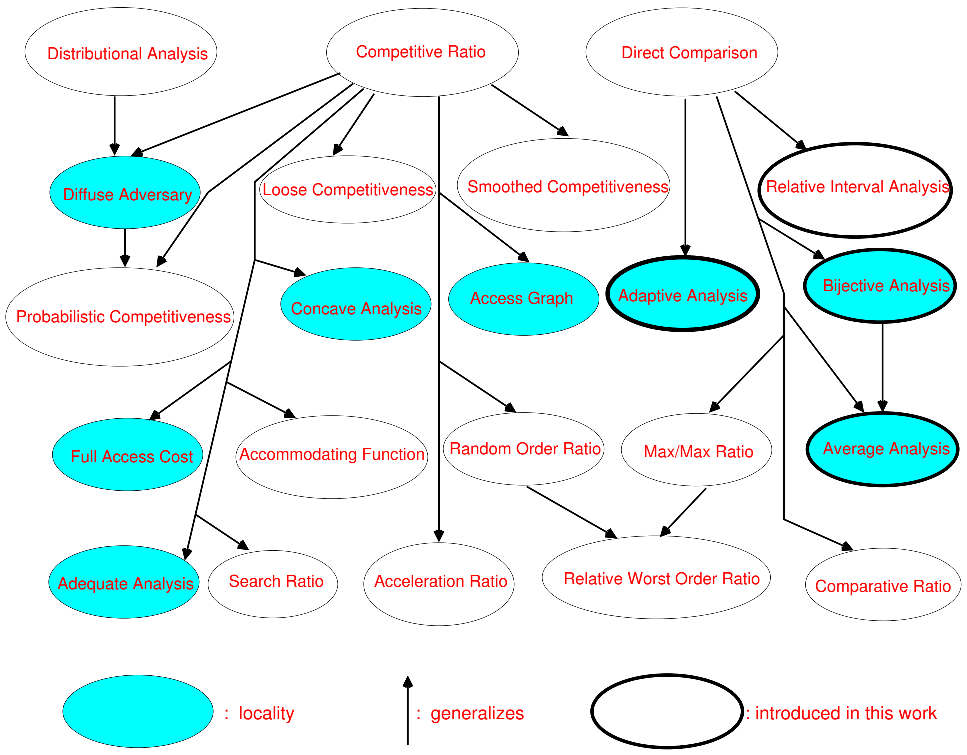

Figure 2.1 Relationships among different alternative measures.

2.20 Conclusion

In this chapter we reviewed various alternative measures for analysis of online algorithms. We also briefly described their weaknesses and strengths, as well as the results that we can get by using them. Figure 2.1 summarizes the relationships among these measures, as well as other alternative measures that we will introduce later in this thesis.

Notes

References

Adel'son-Vel'skii, G. M., and Landis, E. M. 1962. An algorithm for the organization of information. *Soviet Mathematics Doklady*, **3**, 1259–1262.

Albers, S. 1997. On the Influence of Lookahead in Competitive Paging Algorithms. *Algorithmica*, **18**(3), 283–305.

Albers, S. 1998. Improved Randomized On-Line Algorithms for the List Update Problem. *SIAM Journal on Computing*, **27**(3), 682–693.

Albers, S., and Westbrook, J. 1998. Self-organizing data structures. Pages 13–51 of: Fiat, A., and Woeginger, G. J. (eds), *Online Algorithms — The State of the Art*. Lecture Notes in Computer Science, vol. 1442. Springer-Verlag.

Albers, S., von Stengel, B., and Werchner, R. 1995. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, **56**(3), 135–139.

Albers, S., Favrholdt, L. M., and Giel, O. 2005. On Paging with Locality of Reference. *Journal of Computer and System Sciences*, **70**(2), 145–175.

Azar, Y., Boyar, J., Epstein, L., Favrholdt, L. M., Larsen, K. S., and Nielsen, M. N. 2002. Fair versus unrestricted bin packing. *Algorithmica*, **34**(2), 181–196.

Bach, E., Boyar, J., Jiang, T., Larsen, K. S., and Lin, G. 2000. Better bounds on the accommodating ratio for the seat reservation problem. Pages 221–231 of: *Proceedings of the 6th Annual International Computing and Combinatorics Conference (COCOON '00)*.

Bach, E., Boyar, J., Epstein, L., Favrholdt, L. M., Jiang, T., Larsen, K. S., Lin, G., and Stee, R. Van. 2003. Tight Bounds on the competitive ratio on Accommodating sequences for the Seat reservation problem. *Journal of Scheduling*, **6**(2), 131–147.

Banderier, C., Mehlhorn, K., and Beier, R. 2003. Smoothed Analysis of Three Combinatorial Problems. Pages 198–207 of: *Proceedings of the 28th Symposium on Mathematical Foundations of Computer Science (MFCS '03)*.

Bayer, R. 1972. Symmetric Binary B-trees: Data Structure and Maintenance Algorithms. *Acta Informatica*, **1**, 290–306.

Becchetti, L. 2004. Modeling Locality: A Probabilistic Analysis of LRU and FWF. Pages 98–109 of: *Proceedings of the 12th Annual European Symposium on Algorithms (ESA '04)*.

Becchetti, L., Leonardi, S., Marchetti-Spaccamela, A., Schafer, G., and Vredeveld, T. 2003. Average case and smoothed competitive analysis of the multi-level feedback algorithm. Pages 462–471 of: *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03)*.

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

Ben-David, S., and Borodin, A. 1994. A New measure for the study of On-Line Algorithms. *Algorithmica*, **11**(1), 73–91.

Bernstein, D. S., Perkins, T. J., Zilberstein, S., and Finkelstein, L. 2002. Scheduling Contract Algorithms on Multiple Processors. Pages 702–706 of: *Proceedings of the 18th National Conference on Artificial Intelligence.*

Blum, A., and Dunagan, J. 2002. Smoothed Analysis of the Perceptron Algorithm for Linear Programming. Pages 905–914 of: *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA '02).*

Borodin, A., and El-Yaniv, R. 1998. *Online Computation and Competitive Analysis.* Cambridge University Press.

Borodin, A., Irani, S., Raghavan, P., and Schieber, B. 1995. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, **50**(2), 244–258.

Boyar, J., and Favrholdt, L. M. 2007. The relative worst order ratio for online algorithms. *ACM Transactions on Algorithms*, **3**(2).

Boyar, J., and Larsen, K. S. 1999. The Seat Reservation Problem. *Algorithmica*, **25**(4), 403–417.

Boyar, J., and Medvedev, P. 2008. The Relative Worst Order Ratio Applied to Seat Reservation. *ACM Transactions on Algorithms*, **4**(4).

Boyar, J., and Nielsen, M. N. 1999. *An Improved Lower Bound on First-Fit's Accommodating Ratio for the Unit Price Bin Packing Problem.* Tech. rept. PP-1999-11. Department of Mathematics and Computer Science, Odense University.

Boyar, J., Larsen, K. S., and Nielsen, M. N. 1998. *The Accommodating Function – a generalization of the competitive ratio.* Tech. rept. PP-1998-24. Department of Mathematics and Computer Science, Odense University.

Boyar, J., Larsen, K. S., and Nielsen, M. N. 1999. *Separating the Accommodating Ratio from the Competitive Ratio.* Tech. rept. PP-1999-03. Department of Mathematics and Computer Science, Odense University.

Boyar, J., Larsen, K. S., and Nielsen, M. N. 2001. The Accommodating Function: A generalization of the competitive ratio. *SIAM Journal on Computing*, **31**(1), 233–258.

Boyar, J., Favrholdt, L. M., Larsen, K. S., and Nielsen, M. N. 2003. Extending the Accommodating Function. *Acta Informatica*, **40**(1), 3–35.

Boyar, J., Krarup, S., and Nielsen, M. N. 2004. Seat reservation allowing seat changes. *Journal of Algorithms*, **52**(2), 169–192.

Boyar, J., Ehmsen, M. R., and Larsen, K. S. 2006. Theoretical Evidence for the Superiority of LRU-2 over LRU for the Paging Problem. Pages 95–107 of: *Proceedings of the 4th Workshop on Approximation and Online Algorithms (WAOA '06).*

Boyar, J., Favrholdt, L. M., and Larsen, K. S. 2007. The relative worst-order ratio applied to paging. *Journal of Computer and System Sciences*, **73**(5), 818–843.

Breslauer, D. 1998. On competitive on-line paging with lookahead. *Theoretical Computer Science*, **209**(1-2), 365–375.

Chrobak, M., and Noga, J. 1999. LRU is better than FIFO. *Algorithmica*, **23**(2), 180–185.

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

Coffman Jr., E. G., and Gilbert, E. N. 1985. On the Expected Relative Performance of List Scheduling. *Operations Research*, **33**(3), 548–561.

Coffman Jr., E. G., Garey, M. R., and Johnson, D. S. 1997. Approximation algorithms for bin packing: a survey. Pages 46–93 of: Hochbaum, D. S. (ed), *Approximation algorithms for NP-hard problems*. PWS Publishing Company.

Cole, R. 2000. On the Dynamic Finger Conjecture for Splay Trees. Part II: The Proof. *SIAM Journal on Computing*, **30**(1), 44–85.

Cole, R., Mishra, B., Schmidt, J. P., and Siegel, A. 2000. On the Dynamic Finger Conjecture for Splay Trees. Part I: Splay Sorting log n-Block Sequences. *SIAM Journal on Computing*, **30**(1), 1–43.

Demaine, E. D., Harmon, D., Iacono, J., and Patrascu, M. 2007. Dynamic Optimality - Almost. *SIAM Journal on Computing*, **37**(1), 240–251.

Denning, P. J. 1968. The Working Set Model for Program Behaviour. *Communications of the ACM*, **11**(5), 323–333.

Dorrigiv, R., López-Ortiz, A., and Munro, J. I. 2009. An Application of Self-organizing Data Structures to Compression. Pages 137–148 of: *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA '09)*.

Elmasry, A. 2004. On the sequential access theorem and deque conjecture for splay trees. *Theoretical Computer Science*, **314**(3), 459–466.

Epstein, L., and Favrholdt, L. M. 2003. On-line maximizing the number of items packed in variable-sized bins. *Acta Cybernetica*, **16**(1), 57–66.

Felner, A., Stern, R., Ben-Yair, A., Kraus, S., and Netanyahu, N. 2004. PHA*: Finding the Shortest Path with A* in An Unknown Physical Environment. *Journal of Artificial Intelligence Research*, **21**, 631–670.

Fiat, A., and Rosen, Z. 1997. Experimental Studies of Access Graph Based Heuristics: Beating the LRU Standard? Pages 63–72 of: *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*.

Fujiwara, H., and Iwama, K. 2005. Average-Case Competitive Analyses for Ski-Rental Problems. *Algorithmica*, **42**(1), 95–107.

Gal, S. 1979. Search games with mobile and immobile hider. *SIAM Journal of Control and Optimization*, **17**(1), 99–122.

Georgakopoulos, G. F. 2004. Splay trees: a reweighing lemma and a proof of competitiveness vs. dynamic balanced trees. *Journal of Algorithms*, **51**(1), 64–76.

Graham, R. L. 1966. Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal*, **45**, 1563–1581.

Guibas, L. J., and Sedgewick, R. 1978. A dichromatic framework for balanced trees. Pages 8–21 of: *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS '78)*.

Halldórsson, M., and Szegedy, M. 1992. Lower bounds on on-line graph coloring. Pages 211–216 of: *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms (SODA '92)*.

Hennessy, J. L., and Patterson, D. A. 1996. *Computer Architecture: A Quantitative Approach*. Second edn. Morgan Kaufman.

Irani, S. 1991. Two results on the list update problem. *Information Processing Letters*, **38**(6), 301–306.

Irani, S. 1994. Coloring Inductive Graphs On-Line. *Algorithmica*, **11**(1), 53–72.

Irani, S. 1998. Competitive analysis of paging. Pages 52–73 of: Fiat, Amos, and Woeginger, Gerhard J. (eds), *Online Algorithms — The State of the Art*. LNCS, vol. 1442. Springer-Verlag.

Irani, S., Karlin, A. R., and Phillips, S. 1996. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing*, **25**(3), 477–497.

Johnson, D. S. 1973. *Near-optimal bin packing algorithms*. Ph.D. thesis, MIT, Cambridge, MA.

Johnson, D. S. 1974. Fast Algorithms for Bin Packing. *Journal of Computer and System Sciences*, **8**(3), 272–314.

Kalyanasundaram, B., and Pruhs, K. 2000. Speed is as powerful as clairvoyance. *Journal of the ACM*, **47**(4), 617–643.

Kao, M., and Tate, S. R. 1991. Online matching with blocked input. *Information Processing Letters*, **38**(3), 113–116.

Karlin, A. R. 1998. On the Performance of Competitive Algorithms in Practice. Pages 373–384 of: Fiat, Amos, and Woeginger, Gerhard J. (eds), *Online Algorithms — The State of the Art*. LNCS, vol. 1442. Springer-Verlag.

Karlin, A. R., Manasse, M. S., Rudolph, L., and Sleator, D. D. 1988. Competitive Snoopy Caching. *Algorithmica*, **3**(1–4), 77–119.

Karlin, A. R., Phillips, S. J., and Raghavan, P. 2000. Markov Paging. *SIAM Journal on Computing*, **30**(3), 906–922.

Kenyon, C. 1996. Best-Fit Bin-Packing with Random Order. Pages 359–364 of: *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '96)*.

Koutsoupias, E., and Papadimitriou, C. 2000. Beyond competitive analysis. *SIAM Journal on Computing*, **30**(1), 300–317.

Koutsoupias, E., Papadimitriou, C., and Yannakakis, M. 1996. Searching a fixed graph. Pages 280–289 of: *Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP '96)*.

López-Ortiz, A. 1996. *On-line target searching in bounded and unbounded domains*. Ph.D. thesis, University of Waterloo.

López-Ortiz, A., Angelopoulos, S., and Hamel, A. M. 2006. Optimal Scheduling of Contract Algorithms for Anytime Problems. In: *Proceedings of the Twenty First National Conference on Artificial Intelligence*.

Manthey, B., and Reischuk, R. 2007. Smoothed analysis of binary search trees. *Theoretical Computer Science*, **378**(3), 292–315.

Martínez, C., and Roura, S. 2000. On the competitiveness of the move-to-front rule. *Theoretical Computer Science*, **242**(1–2), 313–325.

Munro, J. I. 2000. On the Competitiveness of Linear Search. Pages 338–345 of: *Proceedings of the 8th Annual European Symposium on Algorithms (ESA '00)*. Lecture Notes in Computer Science, vol. 1879.

O'Neil, E. J., O'Neil, P. E., and Weikum, G. 1993. The LRU-K Page Replacement Algorithm For Database Disk Buffering. Pages 297–306 of: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*.

Panagiotou, K., and Souza, A. 2006. On adequate performance measures for paging. Pages 487–496 of: *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*.

Russell, S. J., and Zilberstein, S. 1991. Composing Real-Time Systems. Pages 212–217 of: *Proceedings of the 12th International Conference on Artificial Intelligence*.

Scharbrodt, M., Schickinger, T., and Steger, A. 2006. A New Average Case Analysis for Completion Time Scheduling. *Journal of the ACM*, **53**(1), 121–146.

Schulz, F. 1998. Two New Families of List Update Algorithms. Pages 99–108 of: *Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC '98).*

Sleator, D. D., and Tarjan, R. E. 1985a. Amortized efficiency of list update and paging rules. *Communications of the ACM*, **28**(2), 202–208.

Sleator, D. D., and Tarjan, R. E. 1985b. Self-adjusting Binary Search Trees. *Journal of the ACM*, **32**(3), 652–686.

Souza, A. 2006. *Average Performance Analysis.* Ph.D. thesis, Swiss Federal Institute of Technology Zürich.

Souza, A., and Steger, A. 2006. The Expected Competitive Ratio for Weighted Completion Time Scheduling. *Theory of Computing Systems*, **39**(1), 121–136.

Spielman, D. A., and Teng, S. 2003. Smoothed analysis of termination of linear programming algorithms. *Mathematical Programming*, **97**(1–2), 375–404.

Spielman, D. A., and Teng, S. 2004. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, **51**(3), 385–463.

Tarjan, R. E. 1985. Sequential access in splay trees takes linear time. *Combinatorica*, **5**, 367–378.

Torng, E. 1998. A Unified Analysis of Paging and Caching. *Algorithmica*, **20**(2), 175–200.

Wang, C. C., Derryberry, J., and Sleator, D. D. 2006. O (log log n)-competitive dynamic binary search trees. Pages 374–383 of: *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '06).*

Young, N. E. 1991. *Competitive paging and dual-guided on-line weighted caching and matching algorithms.* Ph.D. thesis, Department of Computer Science, Princeton University.

Young, N. E. 1994. The k-Server Dual and Loose Competitiveness for Paging. *Algorithmica*, **11**(6), 525–541.

Young, N. E. 1998. Bounding the diffuse adversary. Pages 420–425 of: *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA '98).*

Young, N. E. 2000. On-line paging against adversarially biased random inputs. *Journal of Algorithms*, **37**(1), 218–235.

Young, N. E. 2002. On-Line File Caching. *Algorithmica*, **33**(3), 371–383.

Zilberstein, S., Charpillet, F., and Chassaing, P. 1999. Real-Time Problem-Solving with Contract Algorithms. Pages 1008–1015 of: *Proceedings of the 16th International Joint Conference on Artificial Intelligence.*

Author index

Subject index