On-Line Problems with Restricted Input

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science

Lene Monrad Favrholdt
Department of Mathematics and Computer Science
University of Southern Denmark

May 1, 2002

Contents

Chapter 1

Introduction

On-Line Problems. Many real-life problems are on-line, i.e., information is revealed in smaller pieces and it is necessary to take action on each piece of information without knowing the rest. Hence, on-line algorithms differ from off-line algorithms in that they make decisions on an incomplete basis. On-line problems come in two variants, maximization problems where the aim is to maximize benefit and minimization problems where the aim is to minimize cost. Chapter 2 gives a short survey of on-line problems relevant to this thesis.

Measuring On-Line Algorithms. The standard measure for the quality of an on-line algorithm is the competitive ratio, which is, roughly speaking, the ratio of the performance (i.e., the cost/benefit) of the on-line algorithm to an optimal off-line algorithm, i.e., an algorithm that knows the whole input in advance and has all the time it needs to compute an optimal solution. The competitive ratio is defined formally in Section 3.1.

The strength and the weakness of the competitive ratio is that it is a very general measure. Since it is so general and since it is a worst case measure, it cannot be expected to give very detailed information. Indeed, for some problems it fails to reflect reality in that it gives results that are very pessimistic compared to empirical results and/or it does not distinguish algorithms that are known to perform very differently in practice.

This has motivated many researchers to look for more specialized measures. Many of these can be seen as refinements of the competitive ratio. Section 3.3 gives a short survey of such specialized measures. The direction taken in this thesis is to exploit knowledge about the input, since very often it is overly pessimistic to assume that nothing can be predicted about it.

Clearly, if something is known about the input, taking this into account yields a more precise analysis. Furthermore, studying special cases with restricted input can sometimes serve as a stepping stone to the more general (and probably harder) analysis.

On-Line Problems Studied in this Thesis. Chapters 4, 5, 6, and 7 are based on the papers [4, 49, 43, 42, 44] included in Appendix B. The aim has been to give intuition and overview rather than describing all technical details, since these can be found in the papers.

In Chapter 4 we study the paging problem. It is well-known that input sequences to the paging problem exhibit locality of reference, i.e., depending on which pages have been requested lately, the next page requested is likely to belong to a relatively small set of pages. Our way of modeling locality of reference is inspired by Denning's working set model. The model is very simple, and it enables us to use the fault rate as the quality measure. This is a more natural measure than the competitive ratio, and our results seem to be much closer to reality than those obtained for the competitive ratio.

In Chapter 5 we study a version of the edge coloring problem, where a limited number of colors are available, and the aim is to color as many edges as possible. Edges arrive one by one, and each edge must be colored or rejected without knowledge about future edges. This problem has not been studied before. We study the general case as well as the case, where the number of colors available are sufficient for an optimal off-line algorithm to color all edges of the graph.

In Chapter 6 we study dual bin packing in variable sized bins. That is, a fixed number of bins, possibly of unequal sizes, are given, and the goal is to maximize the number of items packed in the bins. We study a very natural class of algorithms, namely those that never reject an item unless it does not fit in any bin. Since, on general sequences, no such algorithm can pack a constant fraction of the items in the worst case, we restrict the input sequences to those that can be packed completely by an optimal off-line algorithm.

Finally, in Chapter 7 we study a simple scheduling problem. The input is a sequence of jobs to be scheduled on two machines with possibly different speeds. The goal is to minimize the time it takes to complete all jobs. We study preemptive scheduling as well as non-preemptive scheduling. In preemptive scheduling it is allowed to break a job in smaller pieces and run the pieces in disjoint time intervals, possibly on different machines. We give optimal algorithms for the special case, where the job sizes are non-increasing.

For the general case, optimal algorithms have already been identified. However, these have not been generalized to optimal algorithms for any number of machines. For non-preemptive scheduling, algorithms with a constant competitive ratio have been devised. For preemptive scheduling, nothing is known so far except for the case of non-decreasing speed ratios.

Danish Summary. A Danish summary can be found in Appendix A.

Notation. For any algorithm A and any input sequence σ, $\mathsf{A}(\sigma)$ denotes the cost/benefit of running A on σ. Similarly, $\mathrm{OPT}(\sigma)$ denotes the optimal solution to σ, i.e., the cost/benefit of running an optimal off-line algorithm on σ.

H_k denotes the kth harmonic number, i.e., $H_k = \sum_{i=1}^{k} \frac{1}{i} \approx \ln(k+1)$.

Chapter 2

On-Line Problems

The on-line problems investigated in this thesis are Paging with Locality of Reference, Edge Coloring with a Fixed Number of Colors, Dual Bin Packing in Variable-Sized Bins, and Scheduling on Two Related Machines to Minimize Makespan, see Chapters 4, 5, 6, and 7. In this section we define these problems and a few other on-line problems that are relevant to this thesis and give a brief description of previous results. For a more thorough description, see [54], where many of these problems are surveyed.

When talking about performance guarantees for algorithms for minimization (maximization) problems, we mean upper (lower) bounds on the competitive ratio. Similarly, impossibility results are lower (upper) bounds on the competitive ratio.

2.1 Paging

In the paging problem, we are dealing with two levels of memory that can store pages of equal sizes. There is a large, slow memory and a smaller, fast memory, often called the cache. The cache can hold k pages. The input is a sequence of requests to pages of the slow memory. Whenever a page is requested that is currently not in the cache, it must be brought from the slow memory to the cache at a cost of 1. This is called a *page fault*. To make room for the new page, a page must be evicted from the cache. The page to be evicted must be chosen without any knowledge of future requests.

Some well-studied paging algorithms are the following.

LRU (Least Recently Used) On a page fault, LRU evicts the page that has not been requested for the longest time.

FIFO (First In First Out) On a page fault, FIFO evicts the page that has been in cache for the longest time.

Marking Algorithms This is a whole class of algorithms that work in phases. Each phase contains requests to exactly k distinct pages, and the first page of a phase is not requested in the previous phase. Thus, phases are maximal in the sense that if a phase is extended to "the right", it will contain more than k distinct pages.

Each time a page is requested, it is marked (unless it is already marked). Only unmarked pages are evicted. At the end of a phase, all pages in the cache are marked. The marks are all erased, and a page can be evicted to make room for the first page requested in the next phase.

A popular randomized marking algorithm is the algorithm Mark_R that chooses the unmarked page to be evicted uniformly at random.

Note that LRU is a marking algorithm and FIFO is not. A rather primitive marking algorithm is the algorithm FWF (Flush When Full). When a phase ends, it simply evicts all pages from the cache.

For the paging problem, the results of competitive analysis are very negative compared to what is observed in practice. Any deterministic on-line paging algorithm has a competitive ratio of at least k [95], the size of the cache, and any randomized algorithm has a competitive ratio of at least H_k [51].

Several deterministic algorithms are k-competitive, i.e., they have optimal competitive ratio. These include FIFO and any marking algorithm [73]. This does not reflect reality well, since empirical results show that LRU is better than FIFO. Moreover, it seems unnatural that an algorithm as primitive as FWF is optimal.

The randomized marking algorithm Mark_R is $2H_k$-competitive [51] (the exact competitive ratio of Mark_R is $2H_k - 1$ [1]). There are more complicated randomized paging algorithms that achieve the optimal competitive ratio of H_k [85, 1].

2.2 The k-Server Problem

The k-server problem is a generalization of the paging problem and was proposed in [83]. A metric space and k servers are given. The servers are placed on points in the metric space. The input is a sequence of requests to points in the metric space. Each request must be served by moving a server to the requested point (unless a server is already placed on that point). Each request must be served without knowledge of future requests. The cost to be minimized is the total distance traveled by the k servers.

The paging problem is a special case of the k-server problem, since it can be modelled by a uniform metric space with one point for each page in the slow memory and one server for each slot in the fast memory. Fetching a page to the fast memory corresponds to putting a server on the corresponding point.

The work function algorithm is $(2k-1)$-competitive in any metric space [78]. Since the k-server problem is a generalization of the paging problem, clearly k is a lower bound on the competitive ratio for the problem. It has been conjectured that the competitive ratio of the problem is k [83].

2.3 Metrical Task Systems

This problem was formulated in [21]. Again, we have a metric space, but only one server. The points of the metric space are called states. Let N denote the number of states. The input is a sequence of tasks. A task is characterized by an N-ary vector giving the cost of servicing the task in each state. Each task must be served without any knowledge of future tasks. For each task, the server is moved to a new state (or stays where it is) at a cost corresponding to the distance between the old and the new state. The task is then processed in the new state at a cost given by the cost vector of the task.

Metrical task systems generalize many on-line problems. To see that the k-server problem is a special case, consider a metric space constructed from a given k-server problem in the following way. The metric space has exactly one point for each subset of the k-server metric space of size k. It can be assumed without loss of generality that the k servers always occupy k distinct points. Thus, the points of the new metric space correspond to the possible placements of the k servers. The distance between two points in the new metric space is the minimum cost of moving the servers from the configuration corresponding to one point to the configuration corresponding to the other point.

The work function algorithm has an optimal competitive ratio of $2N-1$ [21]. Since the problem is very general, there are several important special cases for which this does not yield a good ratio. For instance, for the paging problem it gives a ratio of $2\binom{M}{k}-1$, if M is the size of the slow memory.

2.4 Scheduling

In the basic scheduling problem, m machines/processors are given and the input is a sequence of jobs, each characterized by its size (running time). The goal is to schedule each job on a machine, such that the time it takes to complete all jobs is minimized. This time is called the *makespan*. In the on-line version, each job must be scheduled without any knowledge of future jobs.

This basic problem has many applications and has been studied in several papers [3, 7, 12, 30, 31, 41, 45, 48, 55, 59, 60, 94, 92, 99]. Let m be the number of machines. The algorithm List Scheduling schedules each job on a currently least loaded machine. This algorithm was studied in [60] for the off-line problem, but since it schedules each job without exploiting any knowledge of the future jobs, it also works for the on-line problem. It was shown to have a competitive ratio of $2-\frac{1}{m}$ (though it was not called the competitive ratio). For $m=2$ and $m=3$, this is best possible for deterministic algorithms [48]. For $m \geq 2$, the algorithm *M2* described in [3] is 1.923-competitive. For $m \geq 13$, this is better than $2-\frac{1}{m}$. If $m \geq 64$, the algorithm `MR` presented in [55] is even better. Its competitive ratio tends to $1+\sqrt{(1+\ln 2)/2} < 1.9201$ as m tends to infinity. For $m \geq 4$, the competitive ratio of any deterministic algorithm is at least 1.707 [48], and for $m \geq 80$, it is more than 1.853 [59].

On two machines, the optimal competitive ratio for randomized algorithms is $\frac{4}{3}$ [12].

There are many variations on the basic scheduling problem. For instance, the machines may have different speeds. In this case, List Scheduling is defined such that it schedules each job on a machine where it will finish earliest possible. In the case of identical machines, the two definitions are equivalent. If each machine has a certain speed, independent of the jobs, the machines are said to be *uniformly related*.

For deterministic algorithms, the case of $m=2$ is closed; in this case List Scheduling is optimal. Let q be the speed ratio, i.e., assume that one machine is q times faster than the other. The competitive ratio is $1+\frac{q}{q+1}$ for $q \leq \phi$ and $1+\frac{1}{q}$ for $q \geq \phi$ ($\phi \approx 1.618$ is the golden ratio). Thus, the highest (worst) competitive ratio is ϕ and is attained at $q=\phi$. The performance guarantees as well as the overall impossibility result (the maximum competitive ratio of ϕ) are given in [31], the other impossibility results are given in [45]. The latter paper also shows the following. For $q \geq 2$, the impossibility results are true even for randomized algorithms. For $q<2$, there are randomized algorithms with a better competitive ratio than that of List Scheduling.

		Non-Preemptive	
		Identical	Related
$m = 2$	Deterministic	$C = 1.5$ (LS)	$C = \begin{cases} 1 + \frac{q}{q+1}, & q \le \phi \\ 1 + \frac{1}{q}, & q \ge \phi \end{cases}$ (LS)
	Randomized	$C = 1.333\ldots$	$C \le 1.53$
$m \to \infty$	Deterministic	$C \le 1.920$ (MR)	$C \in O(1)$
$m \ge 80$		$C \ge 1.853$	$C \in O(1)$

	Preemptive	
	Identical	Related
$m = 2$	$C = 1.333\ldots$	$C = 1 + \frac{q}{q^2+q+1} \le 1.333\ldots$
$m > 2$	$C = \frac{m^m}{m^m-(m-1)^m} \to \frac{e}{e-1} \approx 1.582$	

Table 2.1: Known bounds on the competitive ratio C

For general m, there are simple algorithms with constant competitive ratios [7], but the exact overall competitive ratio has not been determined.

It may be allowed to preempt jobs, i.e., it may be allowed to split a job in smaller pieces and run the pieces in disjoint time intervals, possibly on different machines. This variant of scheduling is called *preemptive* scheduling.

The competitive ratio of preemptive scheduling on m *identical* machines is $\frac{m^m}{m^m-(m-1)^m}$. That is, any algorithm, deterministic or randomized, has competitive ratio at least $\frac{m^m}{m^m-(m-1)^m}$ [30, 94], and there exists a deterministic algorithm with this competitive ratio [30, 92]. This ratio tends to $\frac{e}{e-1} \approx 1.582$ as m tends to infinity.

For preemptive scheduling on *two related* machines with speed ratio q, the competitive ratio is $1 + \frac{q}{q^2+q+1}$ [45, 99], for deterministic algorithms as well as randomized. This function attains its maximum value of $\frac{4}{3}$ when $q = 1$.

For the general case of more machines, nothing is known so far. However, [41] gives the optimal competitive ratio as a function of all the speeds in the case where the speed ratios are non-decreasing (that is, if the speeds are $s_1 \le s_2 \le \ldots \le s_m$, then $\frac{s_1}{s_2} \le \frac{s_2}{s_3} \le \ldots \le \frac{s_{m-1}}{s_m}$).

The results are summarized in Table 2.1.

Note that while some of the non-preemptive variants of the scheduling problem described here have randomized algorithms with a better competitive ratio than the optimal competitive ratio for deterministic algorithms, this is not the case for the preemptive variants. Indeed, it has been proven that the competitive ratios of the best deterministic algorithms are also best possible for randomized algorithms. This is rather natural, since the strategy of the best deterministic algorithms is to maintain certain relative levels of the machines (the level of a machine is the time it needs to complete the jobs assigned to it so far). For non-preemptive scheduling randomization helps "spread out" the jobs over the machines. For preemptive scheduling, this can be done more precisely without randomization.

2.5 Bin Packing

The classical bin packing problem is the following. The input is a sequence of items of sizes between 0 and 1. The items must be packed in unit sized bins such that the sum of sizes of the items packed in each bin is at most 1. In the on-line version, each item must be packed without any knowledge of future items. The goal is to pack the items in as few bins as possible. Thus, this problem is a minimization problem.

In [101] a lower bound of 1.5 for any deterministic bin packing algorithm is proven using a sequence with items of three different sizes. In [82] items of 5 different sizes are used to prove a better lower bound of approximately 1.536. [98] gives a tight analysis of the construction from [82] yielding a lower bound of approximately 1.540. In [28] it is argued that these lower bounds are true for randomized algorithms too.

In [91] the best known algorithm, HARMONIC++, is given and proven to be 1.58889-competitive. It is also proven that the algorithm HARMONIC+1 of [90] that was previously thought to be the best algorithm is at best 1.59217-competitive.

Some simpler, classical algorithms are First-Fit and Best-Fit. First-Fit orders the bins according to the order in which they were opened. Each time an item arrives, First-Fit puts it in the first bin in which it fits. If it does not fit in any bin, a new bin is opened. Best-Fit puts the item in the bin in which it leaves the least empty space. If it does not fit in any bin, Best-Fit opens a new bin. The competitive ratio of First-Fit and Best-Fit is 1.7. The upper bound for First-Fit was proven in [57], and the lower bound was shown in [70].

An even simpler algorithm is Next-Fit. Next-Fit packs each item in the last opened bin if it fits there. Otherwise, it opens a new bin and puts the item there. Next-Fit has competitive ratio 2 [68, 69].

First-Fit and Best-Fit belong to the class of algorithms called *Any-Fit* algorithms. An Any-Fit algorithm never packs an item in an empty bin, if the item fits in a non-empty bin. [69] shows that such an algorithm has a competitive ratio between 1.7 and 2. Only a small further restriction is needed to obtain an upper bound matching the lower bound of 1.7. Let the level of a bin denote the total size of the items packed in the bin. An *Almost-Any-Fit* algorithm is an Any-Fit algorithm that never packs an item in a non-empty bin with lowest level, unless there are other bins with the same level, or the bin is the only non-empty bin in which the item fits. Any Almost-Any-Fit algorithm has a competitive ratio of 1.7 [69].

The lower bound for Any-Fit algorithms shows that, to beat First-Fit and Best-Fit, it is sometimes necessary to open a new bin even though the current item fits in an already open bin. On the other hand, for practical applications it may be desirable to sometimes close a bin even though it is not filled completely. A bin is said to be open, if it contains at least one item, and it may still receive more items. A bin is said to be closed, if it contains at least one item, and it will not be considered when packing future items. A bin packing algorithm is said to use *bounded space*, if the maximum number of open bins at any time is bounded by some constant.

[80] proves that the bounded-space algorithm HARMONIC has a competitive ratio that approaches 1.691 as the number of open bins increases, and that this is the best possible competitive ratio of a bounded-space on-line bin packing algorithm.[1]

[1]The precise figure is $h_\infty = \sum_{i=1}^{\infty} \frac{1}{u_i - 1}$, where $u_1 = 2$ and $u_{i+1} = u_i(u_i - 1) + 1$, $i \geq 1$.

2.6 Dual Bin Packing

The dual bin packing problem is a maximization problem. Again, the input is a sequence of items of sizes between 0 and 1. A fixed number of unit sized bins is given, and the aim is to pack as many items in the bins as possible. In [27] this problem is reported to have been named Dual Bin Packing in [81].

A *fair* algorithm for this problem is an algorithm that never rejects an item unless it does not fit in any bin. If the items can be arbitrarily small, no fair algorithm can be competitive [25].

[25] considers the case where all input sequences can be packed completely by an optimal off-line algorithm. In this case, any fair algorithm has a competitive ratio of at least $\frac{1}{2}$, and First-Fit and Best-Fit have a competitive ratio of at least $\frac{5}{8}$ [25]. Furthermore, First-Fit's competitive ratio is at most $\frac{5}{8}$ [8], if the number of bins can be arbitrarily large. [8] also gives a general upper bound of 0.809 for fair deterministic algorithms, when there can be arbitrarily many bins, and an upper bound of $\frac{6}{7}$ for unfair algorithms. Furthermore, an unfair algorithm is devised that has a competitive ratio that tends to $\frac{2}{3}$ when the number of bins goes to infinity.

Note that the name dual bin packing is also sometimes used to refer to bin covering. In this problem, a bin is covered if the items packed in it have a total size of at least 1, and the goal is to cover as many bins as possible.

2.7 Variable-Sized Bin Packing

The classical bin packing problem as well as the dual bin packing problem can be generalized such that there are more than just one bin size. The set of bin sizes is given as a part of the problem.

For the classical bin packing problem with variable-sized bins, there is an unlimited number of bins of each given size. The goal is to minimize the total size of the bins used. For this problem, [34] designs an on-line algorithm Variable Harmonic based on Harmonic. Like Harmonic it uses bounded space. The competitive ratio of this algorithm is the same as the competitive ratio of Harmonic for identical bins. For some combinations of bin sizes, the competitive ratio is even better. If there are only two sizes, 1 and 0.7, the competitive ratio of the problem is at most 1.4, which is smaller than the optimal competitive ratio for identical bins. Hence, in this case, the on-line algorithm "benefits more" from having two sizes of bins to choose from than the off-line algorithm it is measured against.

In Chapter 6 we investigate dual bin packing with variable-sized bins.

2.8 The Seat Reservation Problem

This problem was introduced in [24]. A train has $n \in \mathbb{N}$ seats and travels from station 1 to station $k \in \mathbb{N}$. The input to the problem is a sequence of requests consisting of a start and an end station. Each request must be assigned a seat without any knowledge of the rest of the sequence. Two requests can be assigned the same seat if the start station of one request is the same as or later than the end station of the other request. Requests have to be treated in a fair manner, i.e., if a request can be assigned a seat, it must be assigned a seat. In this case, we say that the request is accepted. Otherwise, it is rejected.

There are two versions of the problem. Either the profit of accepting a request is proportional to the length between its start and end station, or all requests have unit profit. For the proportional price problem, any deterministic on-line algorithm has a competitive ratio proportional to the inverse of the number of stations, even in the case where the input sequences are restricted to those that can be fully accommodated by an optimal off-line algorithm [24]. Thus, depending on the number of stations, the competitive ratio can be arbitrarily bad. For the unit price problem, the situation is the same in the general case, where we have no restriction on the input sequences. For sequences that can be fully accommodated by an optimal off-line algorithm, any deterministic algorithm has a competitive ratio of at least $\frac{1}{2}$ [24], and if the ratio of the number of stations to the number of seats can be arbitrarily large, no deterministic algorithm has a competitive ratio larger than $\frac{1}{2}$ [10].

2.9 Edge Coloring

The classical edge coloring problem is to color the edges of a graph using as few colors as possible, under the constraint that no two adjacent edges may receive the same color. In the on-line version, edges arrive one by one and each edge must be colored before the next edge is seen.

For any graph, let Δ be the maximum vertex degree. In [11] it is shown that the optimal competitive ratio of $2\Delta - 1$ is achieved by the algorithm that numbers the colors and colors each edge with the color of lowest possible number. (This is the algorithm called First-Fit in Chapter 5.)

2.10 Edge Coloring with a Fixed Number of Colors

As far as we know, this variant of the edge coloring problem has not been studied earlier. A limited number of colors are available, and the aim is to color as many edges as possible, again under the constraint that no two adjacent edges may receive the same color. In the on-line version, each edge must be either colored or rejected before the next edge is seen.

This modification of the edge coloring problem is analogous to the modification of the vertex coloring problem made in [24] when defining the seat reservation problem. Assigning seats to requests is equivalent to assigning colors to the vertices of an interval graph.

2.11 Dial a Ride

The dial a ride problem is about transporting objects from one point in a metric space M to another. There is one server available for this. For every pair of points in M, there is a path of a given length. A request consists of a release time, a startpoint, and an endpoint. The release time is the time when the request becomes known to the on-line algorithm. To serve a request, the server must travel from the startpoint to the endpoint. Once an object has been picked up at its startpoint, it cannot be left anywhere else than at its destination point. Thus, if serving one request has been begun, it must be completed before any other request can be served. The server starts at a special point, the origin, and has to end in this point after serving all requests.

The dial a ride problem differs from the other on-line problems described here in that new requests can be released while some of the already released requests have not yet been served and more requests can be released at the same time.

There are several possible object functions for this problem. If the goal is to minimize the total completion time, which is the time when all requests have been served and the server is back in the origin, there exist competitive algorithms [6]. The algorithms IGNORE and REPLAN are both $\frac{5}{2}$-competitive. An algorithm called SMARTSTART is 2-competitive, which is best possible for deterministic algorithms. This algorithm is sometimes "deliberately" idle, i.e., it chooses to do nothing for a while even though there are unserved requests.

If the goal is to minimize the average time from a request is released until it has been served, no on-line algorithm is competitive [62]. This objective is called the *average flow time*.

Chapter 3

Quality Measures

By the quality of an on-line algorithm, we mean the quality of the output of the algorithm. The time complexity of an on-line algorithm is rarely discussed. One might argue that especially for on-line algorithms, time complexity is an important issue. On the other hand, most on-line algorithms studied in the literature are fairly efficient.

When evaluating the quality of algorithms, two main approaches come to mind, worst case and average case analysis. Worst case analysis has the disadvantage that there might be a few rather contrived input sequences giving a performance much worse than the typical sequences. In this sense, average case analysis seems more reasonable. However, this requires a statistical model of the input. Realistic models can be difficult to devise. Furthermore, the analysis tends to be more challenging than worst case analysis.

3.1 Competitive Analysis

The quality measure that has become the standard measure for on-line algorithms is a worst case measure. However, for many problems the worst case performance as an absolute measure does not make sense. For instance, the worst case fault rate of any deterministic paging algorithm is 1. Competitive analysis solves this problem by measuring the performance of the on-line algorithm relative to an optimal off-line algorithm, i.e., an algorithm that knows the whole input sequence from the beginning and has all the time it needs to compute the optimal solution. For many on-line problems, the off-line version is NP-hard. Thus, sometimes, efficient on-line algorithms are measured against an off-line algorithm that cannot even be polynomial, unless NP=P.

The competitive ratio was used already in [95], and in [73] it was named the competitive ratio. The competitive ratio for deterministic algorithms is formally defined in the following way.

Definition 3.1 For any $c \geq 1$, an on-line algorithm A for a minimization problem is *c-competitive*, if there exists a constant b such that

$$\mathsf{A}(\sigma) \;\leq\; c \cdot \mathrm{OPT}(\sigma) + b, \text{ for any input sequence } \sigma,$$

The *competitive ratio* of A is $C = \inf\{c \mid \mathsf{A} \text{ is } c\text{-competitive}\}$.

Definition 3.2 For any $c \leq 1$, an on-line algorithm A for a maximization problem is *c-competitive*, if there exists a constant b such that

$$\mathsf{A}(\sigma) \;\geq\; c \cdot \mathrm{OPT}(\sigma) + b, \text{ for any input sequence } \sigma.$$

The *competitive ratio* of A is $C = \sup\{c \mid \mathsf{A} \text{ is } c\text{-competitive}\}$.

If the inequality holds with $b = 0$, the algorithm is said to be *strictly* c-competitive.

If C is independent of the input sequence, the algorithm is said to be *competitive*.

The *competitive ratio of an on-line problem* is the competitive ratio of the best possible on-line algorithm for the problem. For clarity, this is sometimes referred to as the optimal competitive ratio. Let C be the competitive ratio of some on-line problem. Any algorithm with a competitive ratio $c \in O(C)$ is said to be *strongly competitive*.

Note that some authors define c-competitiveness for maximization problems as $\mathrm{OPT}(\sigma) \leq c \cdot \mathsf{A}(\sigma) + b$, for any input sequence σ. In this way, a good competitive ratio is a low ratio for both maximization and minimization problems. However, the definition chosen here is consistent with most literature on approximation algorithms. To avoid confusion, we will often use the terms performance guarantee and impossibility result instead of the more common terms upper bound and lower bound.

Competitive analysis is often interpreted as a game between the on-line algorithm and an adversary who chooses the input sequence and serves it using an optimal off-line algorithm.

When analyzing randomized algorithms, one must decide on an adversary type. In [15] three types of adversaries for randomized algorithms are defined.

The most commonly used adversary is the *oblivious* adversary. This adversary constructs the input sequence knowing the definition of the algorithm but without knowing the outcome of the random choices made by the algorithm. This adversary is the only adversary considered in this thesis.

A more powerful adversary is the *adaptive on-line* adversary. This adversary may define each request based on the on-line algorithm's answer to all previous requests, but it must serve the request without knowing the random choices made by the on-line algorithm as answer to future requests. This adversary is at least as strong as the oblivious adversary, since it is allowed to define the whole sequence in advance and compute an optimal solution before giving the sequence.

The third adversary, is the *adaptive off-line* adversary. This adversary may define each request based on the on-line algorithm's answer to all previous requests and it serves each request knowing the whole sequence. This adversary is the most powerful of the three. Indeed, against this adversary, no randomized algorithm for a given problem can have a better competitive ratio than the best deterministic algorithm for the problem [15].

When analyzing randomized algorithms, we address the *expected* benefit/cost $E[\mathsf{A}(\sigma)]$ of the algorithm.

Definition 3.3 For any $c \geq 1$, a randomized on-line algorithm A for a minimization problem is *c-competitive*, if there exists a constant b such that

$$E[\mathsf{A}(\sigma)] \;\leq\; c \cdot \mathrm{OPT}(\sigma) + b, \text{ for any input sequence } \sigma.$$

The *competitive ratio* of A is $C = \inf\{c \mid \mathsf{A} \text{ is } c\text{-competitive}\}$.

Definition 3.4 For any $c \leq 1$, an on-line algorithm A for a maximization problem is *c-competitive*, if there exists a constant b such that

$$E[\mathsf{A}(\sigma)] \;\geq\; c \cdot \mathrm{OPT}(\sigma) + b, \text{ for any input sequence } \sigma.$$

The *competitive ratio* of A is $C = \sup\{c \mid \mathsf{A} \text{ is } c\text{-competitive}\}$.

3.2 Limitations of Competitive Analysis

For some on-line problems, competitive analysis yields very pessimistic results. Furthermore, it sometimes fails to distinguish algorithms that are known to perform very differently in practice. Some examples of this were given Chapter 2. Motivated by this, many researchers have proposed refinements to competitive analysis (see the next section).

[14] proves some counterintuitive properties of the competitive ratio. For instance, on-line algorithms for the k-server problem must remember the past to be constant competitive, but knowing a finite part of the future does not help:

- To be constant competitive, any on-line algorithm for the k-server problem must decide how to serve each request based not only on the current request but also on what has happened in the past. Depending on the distances of the metric space, the amount of memory needed can be arbitrarily large.

  This is counterintuitive, since in standard competitive analysis, we do not assume that future requests depend in any way on past requests.

- For the k-server problem, lookahead does not help. That is, knowing the next ℓ requests at each point in time does not improve the competitive ratio, for any finite ℓ.

  This is counterintuitive, since what makes the off-line algorithm so much more powerful than any on-line algorithm is merely the fact that it knows the future.

The paper also gives an example showing that minimizing the amortized cost, i.e., the total cost divided by the number of requests, can be in conflict with minimizing the competitive ratio.

3.3 Refinements of Competitive Analysis

The previous section described some of the drawbacks of competitive analysis. This section describes a number of refinements of competitive analysis. Some are actually not refinements but rather alternatives to competitive analysis. Others are refinements of the problem definition.

Most refinements to competitive analysis fall into one of three categories.

- The set of input sequences is restricted in some way and/or the algorithm is given some information about the input sequence, reflecting that the future is not always completely unpredictable. Examples (that will be defined later in this section) are the accommodating function, access graphs, reasonable load, and lookahead.

  To some extent, the loose competitive ratio also belongs here. Using the loose competitive ratio, sequences with insignificant cost are ignored. Similarly, sequences that are bad only for specific values of the problem parameters are "filtered out".

In a broader sense, restricting the input set can be interpreted as putting a probability distribution on the input set. Examples are the statistical adversary and the diffuse adversary. It can also be argued that the random order competitive ratio belongs here. Random order competitive analysis corresponds to assuming that, for any multiset of requests, any permutation is equally likely.

- The on-line algorithm is given more resources than the off-line algorithm it is measured against (resource augmentation).
- The on-line algorithm is compared to an algorithm (or class of algorithms) less powerful than the optimal off-line algorithm (the comparative ratio).

The rest of this section describes a number of refinements of competitive analysis. Many of these are also described in [65] and [53]. Depending on the problem and the aspects one finds important, different measures may be appropriate.

3.3.1 Resource Augmentation

The idea of resource augmentation is to obtain more optimistic ratios than the standard competitive ratio by measuring the on-line algorithm relative to an optimal off-line algorithm with fewer resources than the on-line algorithm.

The use of resource augmentation in the analysis of on-line algorithms was first introduced in [95], where it is shown, that the competitive ratio of LRU and FIFO is constant, if the on-line algorithm has a cache that is a constant factor larger than that of the off-line algorithm. If h is the size of the off-line cache, the competitive ratio of LRU and FIFO is $\frac{k}{k-h+1}$. Thus, if the off-line cache has size $k(1-\frac{1}{c})$, the competitive ratio of LRU and FIFO is smaller than c.

After some years, the concept of resource augmentation was studied again; [9, 16, 26, 40, 71, 72, 79, 88] study resource augmentation for various scheduling problems. [76] studies resource augmentation for the k-server problem, and [35, 46] study resource augmentation for the bin packing problem. In [88] the concept was named resource augmentation.

In [71], some scheduling problems with one processor are analyzed using resource augmentation. It is assumed that the on-line processor has speed $1+\varepsilon$, $\varepsilon > 0$, whereas the off-line processor has speed 1. The competitive ratio in this case is denoted the *ε-weak* competitive ratio.

The paper considers some preemptive scheduling problems for which no on-line algorithm is competitive. One such problem is the following. A sequence of jobs are to be scheduled on one processor. Each job has a release time and a length. Each job becomes known only at the release time, and its length is unknown until it has been run to completion. The problem is to minimize the average time from a job is released until it has been completed.

Any deterministic on-line algorithm for the problem has a competitive ratio of $\Omega(n^{1/3})$ and any randomized on-line algorithm has a competitive ratio of $\Omega(\log n)$, where n is the number of jobs in the input sequence [87], in other words, no on-line algorithm for this problem is competitive. However, there is a deterministic on-line algorithm, BALANCE, that has an ε-weak competitive ratio of at most $1+\frac{1}{\varepsilon}$. Thus, a constant increase in speed yields a constant competitive ratio. The same can be achieved, giving the on-line algorithm a processor with speed 1 and a processor with speed ε. On the other hand, the algorithm Round Robin has an ε-weak competitive ratio of $\Omega(n^{1-\varepsilon})$.

Thus, the motivation for analyzing the ε-weak competitive ratio is that

— it is a more optimistic measure than the standard competitive ratio

— it tells us how much more performance we get if we increase the speed of the processor

— it helps distinguish different on-line algorithms.

The first item is further elaborated. If $\frac{\mathrm{OPT}_1(\sigma)}{\mathrm{OPT}_{1+\varepsilon}(\sigma)}$ is bounded for all input sequences σ, a bounded ε-weak competitive ratio will imply a bounded competitive ratio, since

$$\frac{\mathrm{OPT}_1(\sigma)}{\mathrm{OPT}_{1+\varepsilon}(\sigma)} \cdot \frac{\mathsf{A}_{1+\varepsilon}(\sigma)}{\mathrm{OPT}_1(\sigma)} = \frac{\mathsf{A}_{1+\varepsilon}(\sigma)}{\mathrm{OPT}_{1+\varepsilon}(\sigma)} \ .$$

Thus, if we consider an input sequence "abnormal", if the optimal off-line performance decreases dramatically, when the speed is decreased slightly, the ε-weak competitive ratio gives us a hint about the competitive ratio on "normal" sequences.

In [16], the upper bound on the ε-weak competitive ratio of BALANCE is improved to $\frac{2}{1+\varepsilon}$. Thus, if BALANCE has a machine that is more than twice as fast as that of the off-line algorithm, BALANCE performs better than the optimal off-line algorithm.

3.3.2 Accommodating Function

Like resource augmentation, the accommodating function applies to any problem with some limited resource. The accommodating function is indeed closely related to resource augmentation. However, whereas resource augmentation is assuming that the on-line algorithm has more resources than the off-line algorithm, the accommodating function is computed by assuming a restricted set of input sequences.

The first step towards defining the accommodating function was taken in [24], where the seat reservation problem is studied. The situation, where the ticket prices are proportional to the distance traveled, as well as the situation where the tickets have a unit price, are studied. For both problems, any deterministic algorithm has a competitive ratio of $\Theta(\frac{1}{k})$, where k is the number of stations. Thus, for a large number of stations, the competitive ratio is very small. However, if the input sequences are restricted to those that can be completely accommodated by an optimal off-line algorithm, any deterministic algorithm for the unit price problem is $\frac{1}{2}$-competitive. Such sequences are called *accommodating* sequences[1]. This restriction on the set of input sequences seems to be a realistic assumption, since it is likely that the management, based on data from earlier years, are able to predict how many cars it will take to accommodate all passengers, if the requests are all known in advance (the seat reservation problem is equivalent to vertex coloring an interval graph, which can be done efficiently). However, this number of cars may not suffice, when the requests are to be served on-line (if the number of stations is large compared to the number of seats, the competitive ratio of any deterministic algorithm is close to $\frac{1}{2}$ [10]). Thus, it seems desirable to have more cars than needed by an optimal off-line algorithm. For other problems it may be more realistic to assume that the resources supplied are not even sufficient for an optimal off-line algorithm. This motivates the definition of α-sequences.

Assume that the amount n of resources are available (n could be the number of seats in the train or the number of bins in the dual bin packing problem). For any $\alpha > 0$, an input sequence is said to be an *α-sequence*, if an optimal off-line algorithm does not benefit from

[1]In [24] the competitive ratio on accommodating sequences was called the accommodating ratio. In later papers this was changed for consistency with common practice in the field

having more than the amount αn of resources. More formally, for any input sequence σ, and any amount m of resources, let $\mathrm{OPT}_m(\sigma)$ denote the benefit/cost of an optimal off-line algorithm on the sequence σ when the amount m of resources is given. An input sequence σ is an α-sequence, if $\mathrm{OPT}_{n'}(\sigma) = \mathrm{OPT}_{\alpha n}(\sigma)$ for any $n' \geq \alpha n$. Thus, accommodating sequences are 1-sequences.

Let A be an on-line algorithm for a maximization problem. The *accommodating function* is defined as

$$\mathcal{A}_{\mathsf{A}}(\alpha) = \sup\{c \mid \mathsf{A} \text{ is } c\text{-competitive on } \alpha\text{-sequences}\}.$$

For minimization problems, the accommodating function is defined analogously:

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

For any "normal" on-line problem, the competitive ratio (with no restriction on the set of input sequences) equals $\lim_{\alpha\to\infty} \mathcal{A}(\alpha)$.

When choosing an on-line algorithm for dual bin packing it can be crucial to know something about the input sequences. If the input sequences are all accommodating, First-Fit is $\frac{5}{8}$-competitive [25], but in the general case, the competitive ratio of First-Fit is $\Theta(s)$, where s is the size of the smallest item in the sequence. An algorithm called Log has a competitive ratio of $\Theta(\frac{1}{\log 1/s})$ in both cases [22]. Thus, if the sequences are known to be accommodating, First-Fit is the best choice, but if the sequences are not likely to be α-sequences for any small α, Log may be the best choice.

For $\alpha < 1$, the accommodating function is closely related to resource augmentation. Assume that the amount n of resources is available. If the input sequences are all α-sequences and $\alpha < 1$, the performance of an optimal off-line algorithm would be the same even if the amount of resources were decreased to αn. This means that any performance guarantee proven in the resource augmentation setting is valid for the accommodating function with $\alpha < 1$. The contrapositive of this observation gives that impossibility results for the accommodating function with $\alpha < 1$ carry over to the resource augmentation setting.

The opposite is not true. In [23] some examples are given where analyzing the accommodating function gives results that are much more optimistic than those obtained with resource augmentation. The competitive ratio of First-Fit for the seat reservation problem does not change significantly when the on-line algorithm is given more seats than the off-line algorithm. Even if the on-line algorithm has $\frac{1}{\alpha}$ times as many seats as the off-line algorithm, the competitive ratio of First-Fit is at most $\frac{1+\alpha}{(\alpha-2/n)(k-1)}$. This fraction tends to 0 as k tends to infinity. On the other hand, the accommodating function of First-Fit is at least $1 - 2^{-\lfloor 1/\alpha \rfloor}$ when $\alpha \leq 1$. Similarly, the competitive ratio of First-Fit for dual bin packing is at least $\frac{3+2\alpha}{8\alpha}$ on α-sequences with $\alpha \leq 1$, but for general sequences, the competitive ratio of First-Fit is at most $\frac{s}{\alpha}$, even if the on-line algorithm has $\frac{1}{\alpha}$ times as many bins as the off-line algorithm. Again, s is the size of the smallest item in the sequence

For those results from resource augmentation that are also valid for the accommodating function, the accommodating function adds an extra, very natural interpretation.

3.3.3 Access Graphs

For the paging problem, the implicit assumption in standard competitive analysis that any sequence of requests may occur is particularly unrealistic. Most programs exhibit *locality of reference*. When a page is referenced, it is more likely to be referenced in the near future (temporal locality), and pages near it in memory are more likely to be referenced in the near future (spatial locality). Locality of reference is the explanation why LRU works well in practice. Indeed, two-level memory is only useful if request sequences are not arbitrary.

In [19] a model of locality of reference is introduced. The structure of the program is represented by a graph that contains a vertex for each page that may be referenced. When a page p is referenced, the next request must be to p or to one of the pages corresponding to the neighbors of p in the graph. Access graphs may be directed or undirected. Here, we only discuss undirected access graphs, since is it sufficient to give a good illustration of the ideas.

In the access graph model, LRU is better than FIFO; on any access graph, the competitive ratio of LRU is at least as good as that of FIFO [33], and there are graphs where the competitive ratio of LRU is much better than that of FIFO. For instance it is not difficult to see that, if the access graph is a line of $k+1$ vertices, the competitive ratio of FIFO is at least $\frac{k+1}{2}$ (in fact this is a lower bound on the competitive ratio of FIFO on any access graph with at least $k+1$ vertices [19]) and the competitive ratio of LRU is 1. In general, if the access graph is a tree, LRU is an optimal deterministic on-line paging algorithm [19].

However, it is clear that there are access graphs for which LRU is not optimal. An example is a ring graph on $k+1$ vertices. [19] gives an algorithm called FAR. Whenever a page must be evicted, FAR chooses a page whose distance in the graph to the page just requested is largest possible. For the ring graph, this is clearly a better strategy than the LRU strategy. [67] shows that FAR is strongly competitive for any access graph.

[50] gives a simple strongly competitive randomized algorithm.

[52] gives paging algorithms that build the access graph on the fly. This means that the access graph need not be known in advance; the access graph may even change dynamically. The algorithms require only $O(k \log n)$ space, where n is the size of the slow memory. A deterministic and a randomized algorithm are given. Both are strongly competitive.

The concept of access graphs has been very successful in the sense that it helps distinguish the performance of different algorithms. However, the results on the actual competitive ratios are sometimes rather difficult to interpret. Some of the results involve finding a vine decomposition (see [18, p. 63]) of the access graph or the maximum number of leaves in any subtree with $k+1$ vertices.

[74] takes the idea of access graphs one step further by introducing probabilities on the edges. Hence, the algorithm knows not only which pages can be requested next, but also the probability of each of these pages to be requested next. In this model it is possible to use the fault rate as the measure instead of measuring the on-line algorithms relative to an off-line algorithm. The paper gives an algorithm that has a fault rate which is within a constant factor of the optimal on-line fault rate.

3.3.4 Loose Competitive Ratio

The loose competitive ratio is defined in [103] for the paging problem, but it should be applicable to other problems as well. In [104] the definition is refined and generalized to file caching. Here we describe the concept as defined in [104] for the special case of paging, since it illustrates the ideas.

A paging algorithm A is (ε, δ)-loosely C-competitive, if for any request sequence σ and any $n \in \mathbb{N}$, at least $(1-\delta)n$ of the cache sizes $k \in \{1, 2, \ldots, n\}$ satisfy

$$\mathsf{A}(\sigma) \leq \max\{C \cdot \mathrm{OPT}(\sigma), \varepsilon|\sigma|\}.$$

Thus, the loose competitive ratio does not consider those sequences that we do not worry about anyway, because they have a low fault rate. Furthermore, sequences that are only bad for a few cache sizes are not considered, since in real life, the sequences are not generated by a cruel adversary that knows the exact hardware configuration.

[104] proves the following result, relevant for many deterministic algorithms. For any $0 < \varepsilon, \delta < 1$, any $\frac{k}{k-h+1}$-competitive algorithm is (ε, δ)-loosely C-competitive, where $C = e\frac{1}{\delta}\lfloor \ln \frac{1}{\varepsilon} \rfloor$. (As in Section 3.3.1 on resource augmentation, h is the size of the off-line cache.) Hence, for constant ε and δ, these algorithms have constant competitive ratios. The result is widely applicable, since FIFO as well as any marking algorithm is $\frac{k}{k-h+1}$-competitive.

The following result is relevant for randomized algorithms. For any $0 < \varepsilon, \delta < 1$, any $O(\ln \frac{k}{k-h+1})$-competitive algorithm is (ε, δ)-loosely C-competitive, where $C \in O(1 + \ln \frac{1}{\delta} + \ln \ln \frac{1}{\varepsilon})$.

[65] poses the following open problem. For many scheduling problems, adversary sequences have been tailormade for the specific number of machines available. (For instance the sequence against List Scheduling on m identical machines is $m(m-1)$ jobs of size 1 followed by one job of size m.) Is it possible to obtain the same impossibility results for the loose competitive ratio?

3.3.5 Statistical Adversary

The statistical adversary introduced in [89] chooses the input sequence such that it is consistent with some statistical assumptions. The idea is to measure the worst case absolute performance. That is, in contrast to competitive analysis, the performance is not measured relative to another algorithm.

As an example, a problem in investment theory is analyzed. The input to the problem is a sequence of stock prices, and it is assumed that the mean and standard deviation are given. The adversary sequence must be chosen among those sequences with the assumed mean and standard deviation. Furthermore, the prices are bounded from above and below.

The statistical adversary is also studied in [32].

Note that in [53] the term statistical adversary is used to denote the stochastic version of the rate ρ adversary defined in [20].

3.3.6 Diffuse Adversary

Assuming that the input sequences are consistent with some specific probability distribution may be as unrealistic as assuming that nothing is known about the input sequences. As a middle ground, [77] proposes to use a whole class of distributions. The algorithm knows the class of distributions, but it does not know which distribution is picked by the adversary. The competitive ratio of an on-line algorithm against this adversary, called the diffuse adversary, is its worst case expected performance ratio, over the distributions in the class. Formally, if Δ is the set of probability distributions, the performance ratio of algorithm A is

$$C(\Delta) = \max_{D \in \Delta} \frac{E_D[\mathsf{A}(\sigma)]}{E_D[\mathrm{OPT}(\sigma)]}.$$

If Δ contains all possible distributions, the ratio becomes equal to the standard competitive ratio; the adversary simply picks a distribution containing only one worst case sequence.

In particular, [77] considers the set of distributions Δ_ε, where, in each step, the probability of any page to be requested next is at most ε, for some $\varepsilon > 0$. They show that LRU is optimal against such an adversary, but they give no closed form for the competitive ratio for $k > 2$.

The class Δ_ε of distributions is further investigated in [105], where the following is proven. The competitive ratio of any deterministic on-line algorithm is

$$C_{\mathrm{det}} \geq \sum_{i=1}^{k-1} \frac{1}{\max\{\varepsilon^{-1} - i, 1\}} \; .$$

An upper bound is given for the class of deterministic lazy marking algorithms. (A lazy algorithm is an algorithm that only evicts a page when it has to. Sometimes such algorithms are called demand paging algorithms.) The bound is

$$C_{\mathrm{m}} \leq 2\sum_{i=1}^{k-1} \frac{1}{\max\{\varepsilon^{-1} - i, 1\}} + 2.$$

For $\varepsilon \leq \frac{1}{k+1}$, the same is true for randomized algorithms.

Note that for $\varepsilon = \frac{1}{n}$,

$$\sum_{i=1}^{k-1} \frac{1}{\max\{\varepsilon^{-1} - i, 1\}} = \begin{cases} H_{n-1} - H_{n-k}, & n \geq k \\ H_{n-1} + k - n, & n \leq k. \end{cases}$$

The upper bound result covers LRU, since it is a lazy marking algorithm, but it does not cover FIFO, since it is not a marking algorithm, and it does not cover FWF, since it is not lazy. Indeed, for $\varepsilon \geq \frac{1}{k+1}$, the competitive ratio of FIFO and FWF is k, just as the standard competitive ratio.

For $\varepsilon \geq \frac{1}{k+1}$, the lower bound for deterministic algorithms is raised to $\sum_{i=1}^{k-1} \frac{1}{\max\{\varepsilon^{-1}-i,1\}} + 1$, and the lower bound for randomized algorithms is H_k, the optimal competitive ratio against the standard oblivious adversary.

Hence, for $\varepsilon \leq \frac{1}{k+1}$, the class of randomized algorithms is no stronger than the class of deterministic algorithms, and for $\varepsilon \geq \frac{1}{k+1}$, the optimal competitive ratio for randomized algorithms is the same as against the standard oblivious adversary. This is perhaps not so surprising. To a great extend, the advantage of randomized algorithms is that the input sequences "look random" from the algorithms perspective — there are no real worst case sequences. Hence, if the input is fairly random (this is the case if ε is small), it seems reasonable that randomized algorithms are not much stronger than deterministic algorithms. On the other hand, if ε is large, the adversary does not differ much from the standard adversary.

3.3.7 Random Order

In [75] the Best-Fit algorithm for bin packing is investigated. Normally, when analyzing bin packing algorithms, the performance is measured using the worst case performance ratio, over all input sequences. This yields the lower bound of 1.7 for both First-Fit and Best-Fit, which is rather pessimistic compared to empirical results on Best-Fit's performance. The lower bound is due to input sequences with items of very special sizes where the items occur in order of non-decreasing size. Most permutations of these input sequences give ratios that are significantly better. This motivates studying the worst case expected performance ratio, over all multisets of items, assuming that any permutation of the items is equally likely.

This expected performance ratio is shown to lie between approximately 1.08 and 1.5 for Best-Fit. Thus, using this performance measure, the performance guarantee for Best-Fit is a little better than the general lower bound on the competitive ratio of any bin packing algorithm.

3.3.8 Reasonable Load

The notion of reasonable load is very similar to the concept of the rate ρ adversary defined in [20] and further investigated in [5]. It also has similarities to the accommodating function.

[62] studies the on-line dial a ride problem with an infinite number of requests. In this case, the total completion time is meaningless. However, if we want to minimize the average flow time, competitive analysis does not yield any information as to which algorithm to choose, since the competitive ratio of any on-line algorithm for this problem is unbounded.

This motivated the authors of [62] to put a restriction on the set of input sequences. Since all requests must be served, it seems reasonable to require that an optimal off-line algorithm is able to do so, i.e., the number of released jobs not yet served does not grow unboundedly. For any $\Delta \in \mathbb{N}$, a request sequence is Δ-*reasonable*, if any sequence of requests released within a time period of length $T \geq \Delta$ can be served in time at most T. A request sequence is *reasonable*, if there exists a Δ, such that the sequence is Δ-reasonable.

On Δ-reasonable request sequences, the algorihm IGNORE, described in detail in [6], yields a maximal flow time of at most 2Δ. On the other hand, there are reasonable request sequences for which the average flow time of the algorithm REPLAN, also described in [6], is unbounded.

Since the off-line version of the dial a ride problem is NP-hard, it seems desirable that the sequences are “more” than reasonable. A sequence is (Δ, ρ)-reasonable, if requests released during a period of time $T \geq \Delta$ can be served in time at most T/ρ. If the problem is solved using a ρ-approximation algorithm, the number of released jobs not yet served will not grow unboundedly, if the sequence is (Δ, ρ)-reasonable, for some bounded Δ.

Note the similarity between (Δ, ρ)-reasonable sequences and $\frac{1}{\rho}$-sequences as defined in Section 3.3.2.

3.3.9 Comparative Ratio

Motivated by the fact that lookahead cannot improve the competitive ratio of an on-line paging algorithm, [77] introduces the comparative ratio. Rather than evaluating the performance of single algorithms, the purpose is to compare the power of classes of algorithms having access to different amounts of information. For instance it is shown that comparing on-line paging algorithms, i.e., algorithms that know only the current page requested, to algorithms that know the current request and the following $\ell - 1$ requests gives a ratio of $\min\{k, \ell\}$.

3.3.10 The Max/Max Ratio

The Max/Max ratio is introduced in [14]. Like the competitive ratio, it is a worst case measure and it measures the on-line performance relative to the optimal off-line performance. However, whereas the competitive ratio measures the performance on each input sequence relative to the performance of an optimal off-line algorithm on that same sequence, the Max/Max ratio compares the performance on each input sequence σ to the worst case performance of an optimal off-line algorithm on all sequences of the same length as σ. More precisely, the Max/Max ratio is the ratio of the worst case cost of the on-line algorithm on sequences of length ℓ to the optimal off-line cost on sequences of length ℓ, as ℓ goes to infinity.

The example problem is the k-server problem with a bounded metric space. For this problem the Max/Max ratio is the worst case amortized cost of the on-line algorithm normalized by the amortized optimal off-line cost. This means that the Max/Max ratio of two on-line algorithms can be compared without referring to an optimal off-line algorithm — the ratio of their Max/Max ratios is simply the ratio of their worst case performances.

The paper gives an on-line memoryless k-server algorithm that, for any k and any bounded metric space G, has a Max/Max ratio of at most $2k$.[2] Moreover, the ratio is within a factor of two of the best possible on-line algorithm. This is in contrast to the competitive ratio which, depending on the distances in the metric space, can be arbitrarily large, for any memoryless algorithm for the k-server problem. It is also shown that lookahead does help. Specifically, it is shown that if the algorithm knows not only the current request but also the following $\ell - 1$ requests, the Max/Max ratio is $\frac{n-1}{\ell}$, if $n - k \leq \ell \leq n - 1$ and n is the number of points in the metric space. If $\ell < n - k$, a lookahead of size ℓ does not improve the Max/Max ratio.

Moreover, it is shown that the best possible Max/Max ratio depends on the metric space. For the uniform metric space with n points, the Max/Max ratio is $\frac{n-1}{n-k}$. Thus, for the uniform metric space with $k+1$ points, the Max/Max ratio is k. On the other hand, there exist metric spaces with arbitrarily many points for which the Max/Max ratio is 1. This is in contrast to the competitive ratio, since any deterministic k-server algorithm has a competitive ratio of at least k in any metric space with more than k points, and it has been conjectured that this is the optimal competitive ratio [83].

Thus, even though the definition of the Max/Max ratio is similar to the definition of the competitive ratio, the Max/Max ratio seems to give more reasonable results for the k-server problem. Unfortunately, for the paging problem, the Max/Max ratio seems to give even less information than the competitive ratio. If there is no restriction on the request sequences, the fault rate of any algorithm (including the optimal off-line algorithm) can be arbitrarily close to one, depending only on the number of distinct pages that can be requested. Thus, considering arbitrarily large numbers of distinct pages, the Max/Max ratio of any algorithm can be arbitrarily close to 1. Moreover, for problems like scheduling to minimize makespan (or the k-server problem with an unbounded metric space) where a bound on the length of the input sequence does not yield a bound on the cost, the Max/Max ratio does not directly apply.

3.3.11 Lookahead

As mentioned in Section 3.2, lookahead cannot improve the competitive ratio of k-server algorithms. This is true for any metric space, and hence it cannot improve the competititive ratio of paging algorithms. As a response to this, some stronger versions of lookahead have been studied.

[2] introduces the notion of *strong lookahead*. Strong lookahead ℓ means that the algorithm knows the minimal prefix of the remaining sequence that contains requests to ℓ distinct pages.

[2]The memoryless algorithm requires some preprocessing consisting in solving an NP-hard problem (the Minmax radius k-clustering problem). However, a 2-approximation algorithm exists [17]. Using this algorithm in the preprocessing step yields a Max/Max ratio of at most $4k$.

A version of LRU, generalized to take advantage of lookahead, is studied. When a page must be evicted, the least recently used page that is not among the pages in the lookahead is chosen. If such a page does not exist, the page that is requested farthest in the future is evicted. With a lookahead of size ℓ, we call this version of the algorithm LRU(ℓ).

The competitive ratio of LRU(ℓ) is $k-\ell$, when $\ell \leq k-2$, and this is best possible among deterministic paging algorithms.

Often, in real applications, requests arrive in blocks. If the sequence is partioned in blocks such that each block is a minimal sequence with ℓ distinct pages, and the algorithm sees one block at a time, LRU(ℓ) is $(k-\ell+1)$-competitive and a variant of $\textsc{Mark}_\text{R}$ is $(2H_{k-\ell})$-competitive, when $\ell \leq k-2$.

3.3.12 Total Access Time

Sometimes the problem description itself can be refined to obtain more realistic results. This is the case for the paging problem. One reason that competitive analysis yields very unrealistic results for the paging problem is that arbitrarily long sequences exist for which the off-line algorithm has no faults at all. Hence, in [97] it is assumed that it takes time 1 to access a page in the fast memory, while fetching a page from the slow memory takes time $p \geq 1$. With this definition of the problem, it is shown that lookahead can give improved competitive ratios.

Since LRU(ℓ) (as defined in Section 3.3.11) is difficult to analyze, a simpler version is studied. Instead of using the full lookahead, only those pages in the lookahead contained in the current marking phase are considered. We call this version LRU(ℓ, k).

The following results are obtained on the refined version of the paging problem.

- Sufficient (finite) lookahead yields algorithms with constant competitive ratios. Specifically, LRU(kp, k) is 2-competitive.
- On sequences with a significant locality of reference, any marking algorithm has a constant competitive ratio. More precisely, for any input sequence, let L be the average length of a phase. If $L \geq ak$, then the competitive ratio of any marking algorithm is less than $1 + \frac{p}{a}$.

  In contrast, LFU(ℓ) (Least Frequently Used) has a competitive ratio of more than p on sequences with much more locality of reference than what is needed for LRU(ℓ) to be 2-competitive.

Chapter 4

Paging with Locality of Reference

The most natural quality measure for paging algorithms seems to be the fault rate, i.e., the number of faults divided by the number of requests. However, if no restriction is put on the set of input sequences, the *worst case* fault rate of any deterministic on-line paging algorithm is 1, since in the worst case, each request is to a page that is currently not in the cache. If the slow memory is much larger than the cache, even the worst case fault rate of any randomized on-line algorithm will be close to 1. Modeling locality of reference is one way of restricting the input sequences. In [4] we study a very simple model of locality of reference.

4.1 The Model

In modeling locality of reference we go back to the working set concept by Denning [37, 38] that is also used in standard text books on operating systems [36, 96] to describe the phenomenon of locality. The set of pages that a process is currently using is called the *working set*. Fixing a point in a request sequence and determining the working set size in a window of size n starting at this point in the sequence, one obtains a function of n whose general behavior is depicted in Figure 4.1. The function is increasing and concave. Denning [37] shows that this is in fact a mathematical consequence of the working set model, assuming statistical regularities locally in a request sequence.

We assume that an application is characterized by a concave function f. The application generates request sequences that are *consistent with f*. We will investigate two models. In the *Max-Model* a request sequence is consistent with f if the maximum number of distinct pages referenced in a window of size n is at most $f(n)$, for any $n \in \mathbb{N}$. In the *Average-Model*

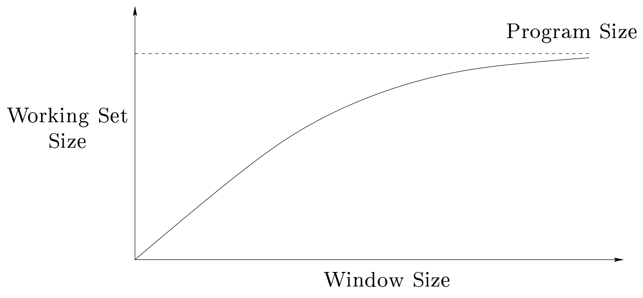
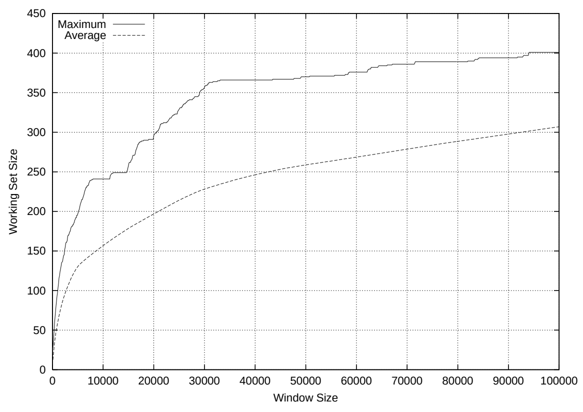
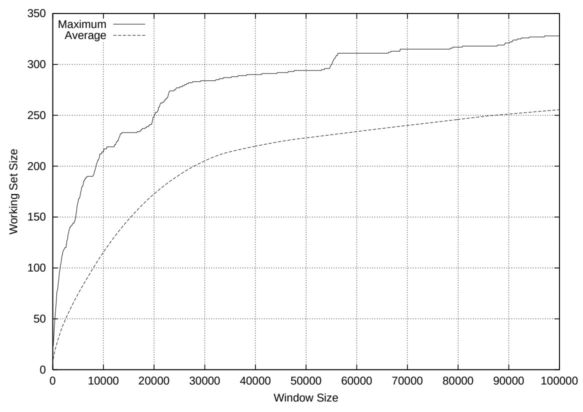


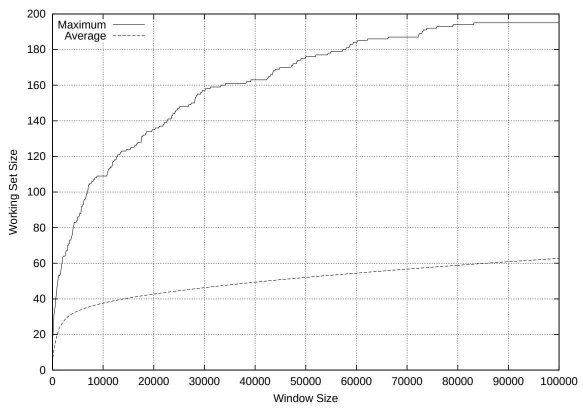
Figure 4.1: Working set size as a function of the window size.



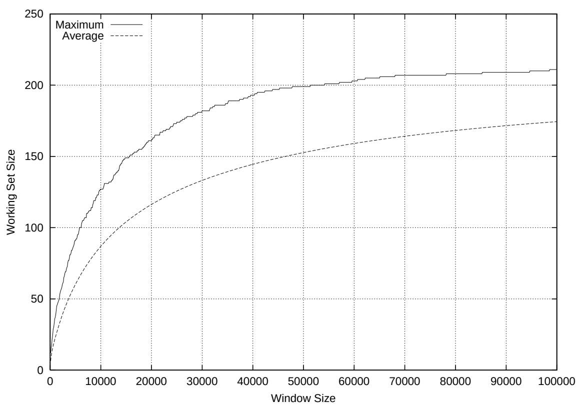
(a) VAX, PASCAL, 500 pages.



(b) VAX, SPIC, 385 pages.



(c) SPARC, GCC, 276 pages.



(d) SPARC, COMPRESS, 229 pages.

Figure 4.2: Maximum and average size of the working set in windows of size up to 100,000 requests. Each diagram's caption gives the architecture, the name of the trace, and the number of distinct pages requested in the entire sequence.

a request sequence is consistent with f if the average number of distinct pages referenced in a window of size n is at most $f(n)$, for any $n \in \mathbb{N}$.

In our model the function f characterizes the maximal/average working set size globally in a request sequence, whereas the original working set model considers working set sizes locally. The Max-Model is closely related to the original working set model. On the other hand, the Average-Model permits a larger class of request sequences. It is interesting if an application changes the working set completely at certain times in a request sequence.

We have performed experiments with traces from standard corpora, analyzing maximum/average working set sizes in windows of size n, see Section 4.6 for details. The result for four of these traces are depicted in Figure 4.2. As illustrated by the figure, the behavior of the working set size proposed by Denning for a single window of increasing size can also be observed *globally*, taking the maximum/average working set size over *all* windows of a request sequence; the curves have an overall concave behavior. We also observe that, for all window sizes, the working set size is very small compared to the window size. This suggests that the model we propose here is indeed reasonable for studying paging algorithms.

Naturally, the functions are not only concave, they are also non-decreasing. Furthermore, since windows of size 1 contain exactly one page, $f(1) = 1$.

If windows of size n contain at most m pages, then a window of size $n + 1$ can contain at most $m + 1$ pages. Thus, in the Max-Model, f is *surjective* on the integers between 1 and its maximum value, i.e., for all natural numbers m between 1 and $\sup\{f(n) \mid n \in \mathbb{N}\}$, there exists an n with $f(n) = m$.

This is all captured in the following definition (the first inequality in 2. says that the function is concave, and the last inequality says that it is non-decreasing).

Definition 4.1 A function $f\colon \mathbb{N} \to \mathbb{R}_+$ is *concave** if

1. $f(1) = 1$
2. $\forall n \in \mathbb{N}$: $f(n + 1) - f(n) \geq f(n + 2) - f(n + 1) \geq 0$.

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

Note that the requirement in the Max-Model that the function be surjective on the integers between 1 and its maximum value implies that $f(n + 1) - f(n) \leq 1$, for all $n \in \mathbb{N}$.

For a given application, a good approximation of f is easy to determine. One only has to scan a sufficiently long request sequence and compute the maximum/average number of pages in windows of size n, just as it was done to obtain the curves in Figure 4.2. Essentially, for each trace, we can use any concave function f that is an upper bound on the observed data points, e.g., we can take the upper convex hull of the points.

For the Max-Model, there might be one small problem; the upper convex hull might not be surjective on the integers between 1 and the maximum value. This can be fixed without changing the upper bound too much. Note that the points of the upper convex hull are points of the original curve. The coordinates of these points are natural numbers and they are connected by straight line segments. The following two steps sketch how to obtain a concave* upper bound from the upper convex hull.

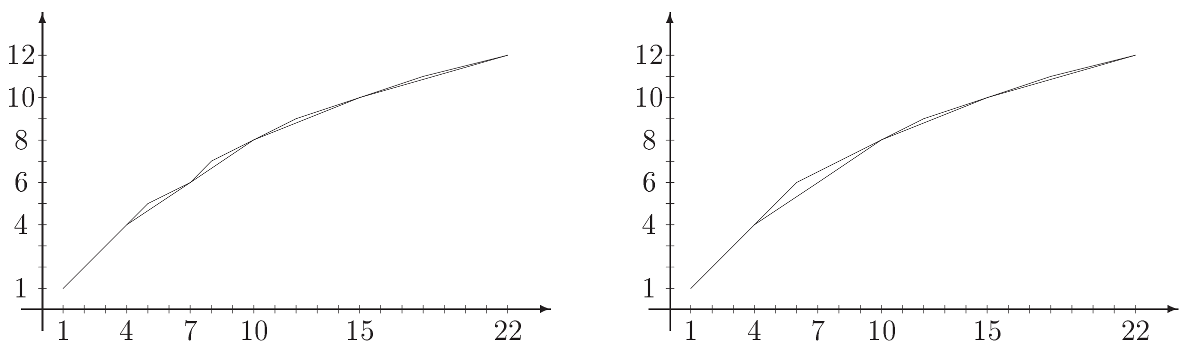
1. For each line segment ℓ with a slope $\frac{a}{b}$, if there is no $m \in \mathbb{N}$ such that $\frac{a}{b} = \frac{1}{m}$, choose $m \in \mathbb{N}$ such that $\frac{1}{m+1} < \frac{a}{b} < \frac{1}{m}$, and replace ℓ by two line segments with slopes $\frac{1}{m+1}$ and $\frac{1}{m}$. Denote the lengths of the projections of these two line segments on the x-axis by x_1 and x_2. These two lengths are the solutions to the linear equation system $x_1(m + 1) + x_2 m = b$ and $x_1 + x_2 = a$.
2. For each line segment that was replaced by two line segments in Step 1, let m be the natural number chosen such that the slope of ℓ lies between $\frac{1}{m+1}$ and $\frac{1}{m}$. If there are other line segments of the upper convex hull with slopes in the same interval, the segments with slope $\frac{1}{m}$ should be moved before all of the segments with slope $\frac{1}{m+1}$.

See Figure 4.3 for an example illustrating the two steps.

In the analysis of the Max-Model, we need a definition of the inverse of a concave* function.

Definition 4.2 For any concave* function f, let $M = \sup\{\lfloor f(n) \rfloor \mid n \in \mathbb{N}\}$. Define $f^{-1}\colon \{m \in \mathbb{N} \mid m \leq M\} \to \mathbb{N}$ by

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



(a) The concave function and the upper bound after Step 1.

(b) The concave function and the upper bound after Step 2.

Figure 4.3: In each subfigure, a concave function and an upper bound on the function is shown.

In words, $f^{-1}(m)$ is the smallest possible size of a window containing m distinct pages.

Both in the Max- and the Average-Model, given a concave* function f, we will analyze the performance of paging algorithms on request sequences that are consistent with f. Practitioners use the *fault rate* to evaluate the performance of paging algorithms. We will use this measure, too.

Definition 4.3 The fault rate of a paging algorithm A on an input sequence σ is

$$F_{\mathsf{A}}(\sigma) = \frac{\mathsf{A}(\sigma)}{|\sigma|} \ .$$

We are interested in the worst case performance on all sequences that are consistent with f.

Definition 4.4 The fault rate of a paging algorithm A with respect to a concave* function f is

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

Throughout the analysis, we assume that the functions considered are concave*. Moreover, we assume that the functions have maximum values of at least $k+1$, since otherwise the fault rate of any reasonable paging algorithm is 0.

4.2 Algorithms

The on-line algorithms considered are all deterministic. They are: LRU, FIFO, and the class of deterministic marking algorithms (see Section 2.1 for the definitions). Furthermore, we study the optimal off-line algorithm LFD. On a fault, LFD evicts the page whose next request is farthest in the future. Since LFD is an off-line algorithm, it cannot be used in practice, but since it is an optimal off-line algorithm [13], it is interesting to analyze it. Note, however, that the fault rate of an on-line algorithm divided by the fault rate of LFD does not necessarily give the competitive ratio of the on-line algorithm on sequences consistent with f.

	Max-Model	Average-Model
Any on-line alg.	$\geq \frac{k-1}{f^{-1}(k+1)-2}$	$\geq \frac{f(k+1)-1}{k}$
LRU	$= \frac{k-1}{f^{-1}(k+1)-2}$	$= \frac{f(k+1)-1}{k}$
FIFO	$\geq \frac{k-1/k}{f^{-1}(k+1)-1}, \quad \leq \frac{k}{f^{-1}(k+1)-1}$	$= \frac{f(k+1)-1}{k}$
Marking	$\leq \frac{k}{f^{-1}(k+1)-1}$	$\leq \frac{4}{3}\frac{f(k)}{k}$
LFD	$\geq \max\limits_{\substack{m\in\mathbb{N}\\ k+m\leq M}} \left\{\frac{m}{f^{-1}(k+m+1)-1}\right\}, \leq 2 \max\limits_{\substack{1\leq m\leq k\\ k+m\leq M}} \left\{\frac{m+1}{f^{-1}(k+m)}\right\}$	$\approx \frac{4M-4k}{4M-k}\frac{f(k+1)}{k+1}$

Table 4.1: Fault rates of the algorithms considered.

4.3 Results

We investigate the Max-Model and the Average-Model in Sections 4.4 and 4.5, respectively. The results are summarized in Table 4.1. Recall that M is the maximum number of distinct pages that can be requested in any sequence consistent with f. In the Average-Model, the exact upper bound on marking algorithms is actually a little smaller than that shown in the table. Similarly, the fault rate of LFD in the Average-Model is slightly larger than that shown in the table.

In the Max-Model, LRU is optimal. FIFO is not quite optimal; the lower bound on the fault rate of FIFO is a little larger than the optimal fault rate for most concave* functions. The upper bound on the fault rate of FIFO (which is almost equal to the lower bound) is equal to the fault rate of the worst possible marking algorithm.

In the Average-Model, both FIFO and LRU are optimal. A worst possible marking algorithm is about a factor of $\frac{4}{3}$ from being optimal. As in the Max-Model, the fault rate of LFD depends on the total number M of pages that may be requested. If M is approximately k, LFD has a fault rate close to 0, as expected. If M is large compared to k, the fault rate is close to $\frac{f(k+1)}{k+1}$. Thus, at first, it might seem that for $M \gg k$, the fault rate of LFD is larger than that of LRU and FIFO, since $\frac{f(k+1)}{k+1} > \frac{f(k+1)-1}{k}$, if $f(k+1) > k+1$. However, for $M \gg k$, the function giving the lower bound for LFD has $f(k+1) \approx k+1$.

Since we consider worst case fault rate, the fault rates predicted by the Max-Model as well as those predicted by the Average-Model are higher than those observed in practice. However, in our experiments, the gap was considerably smaller than the gap between the "theoretical" competitive ratio and the "empirical" competitive ratio.

Our experiments suggest that fault rates predicted by the Max-Model are closer to reality than those predicted by the Average-Model (see Section 4.6). Furthermore, the Max-Model distinguishes LRU and FIFO. On the other hand, only the Average-Model distinguishes FIFO from the class of marking algorithms — in the Max-Model, the fault rate of FIFO cannot be distinguished from primitive algorithms like FWF.

4.4 The Max-Model

In this section we study the Max-Model. Given a concave* function f, $f(n)$ is an upper bound on the maximum number of distinct pages encountered in any n consecutive requests of an input sequence.

The proofs of the results of the Max-Model are all rather simple. The upper bound proofs for the on-line algorithms are similar to the corresponding proofs in competitive analysis. The idea (for LFD too) is to divide the input sequences into phases with a given number of faults or a given number of distinct pages and prove a lower bound on the length of a phase.

The sequences of all lower bound proofs for the on-line algorithms contain exactly $k+1$ distinct pages. All lower bound sequences are constructed in phases, each consisting of blocks of non-decreasing lengths. For all algorithms but FIFO, each block consists of requests to only one page. Since we are dealing with deterministic algorithms, we can choose the page of a block to be the one page that is not in cache at the end of the previous phase.

We will assume that $f(2) = 2$, because if any window of size 2 has requests to less than 2 distinct pages, then the whole sequence has requests to only one page.

In each of the lower bound proofs and in the upper bound proof for LFD, we will need the following simple proposition (Proposition 1 in [4]).

Proposition 4.5 *For any concave* function f, f^{-1} is a strictly increasing function satisfying that, for all* $2 \leq m \leq M-1$,

$$f^{-1}(m+1) - f^{-1}(m) \geq f^{-1}(m) - f^{-1}(m-1).$$

This is where we need that f is surjective on all integers between 1 and M. Consider for instance the linear function $f(n) = \frac{1}{3} + \frac{2}{3}n$. For this function, $f^{-1}(1) = 1$, $f^{-1}(2) = 3$, and $f^{-1}(3) = 4$. Thus, $f^{-1}(3) - f^{-1}(2) < f^{-1}(2) - f^{-1}(1)$.

4.4.1 A Lower Bound for Deterministic Algorithms

To prove the general lower bound of $\frac{k-1}{f^{-1}(k+1)-2}$ on the fault rate of deterministic paging algorithms, we construct an input sequence containing requests to $k+1$ distinct pages p_1, $p_2, \ldots, p_{k+1}$. The sequence is constructed in phases each consisting of $k-1$ blocks. The ith block of a phase consists of $f^{-1}(i+2) - f^{-1}(i+1)$ requests to the page that was not in cache at the end of the previous block. The definition of the block lengths implies that the first i blocks of a phase have a total length of

$$\sum_{j=1}^{i} f^{-1}(i+2) - f^{-1}(i+1) = f^{-1}(i+2) - f^{-1}(2) = f^{-1}(i+2) - 2.$$

In particular, it implies that the length of a phase is $f^{-1}(k+1)-2$. Since the algorithm faults on the first request of each block, this gives the claimed fault rate.

To complete the proof, we must show that the constructed sequence is consistent with f. Thus, for any number n, $1 \leq n \leq k+1$, we must show that any window containing n distinct pages has a length of at least $f^{-1}(n)$. For $1 \leq n \leq 2$, any window with n distinct pages has length at least $f^{-1}(n)$. By Proposition 4.5, the block length is non-decreasing within a phase. Thus, to find a shortest possible window with n distinct pages, $3 \leq n \leq k+1$, we should search at the beginning of phase. More specifically, we consider the first $n-2$ blocks of a phase P, the first request of the $(n-1)$st block of P and the last request before P. This subsequence contains at most n distinct pages, and since the first $n-2$ blocks of a phase have a total length of $f^{-1}(n)-2$, the subsequence has a length of $f^{-1}(n)$. Note that if $n=k+1$, the first $n-1$ blocks of P will constitute all of P.

4.4.2 LRU is Optimal

To prove that LRU is optimal, we partition any input sequence σ consistent with f into phases, such that each phase contains exactly $k-1$ faults, and each phase starts with a fault. Consider an arbitrary phase P. We argue that the subsequence of σ starting at the last request before P and ending at the first request after P (including that request) contains $k+1$ distinct pages. This implies that P has a length of at least $f^{-1}(k+1)-2$, which gives the upper bound.

Let p be the page referenced by the last request before P. Phase P and the first request after P include k page faults. If these page faults are on distinct pages different from p, we are done. If not, then

- one of the k faults is on p, or
- two of the k faults are on the same page.

Note that p is in cache at the beginning of the phase. Thus, if one of the k faults is on p, p is evicted at some point within P. At that point, p is the least recently used page in the cache, which means that k pages different from p are requested within the phase. If the window contains two faults on one page, the same argument applies.

4.4.3 FIFO is Not Quite Optimal

In the proof of the upper bound for LRU, we used the fact that between any *request* to a page p and a *fault* on p there are requests to at least k other pages. This is not necessarily the case for FIFO. However, between any pair of *faults* on a page p, there are faults on at least k other pages. Therefore, when we partition the input sequence into phases, we include k faults in each phase instead of only $k-1$. As for LRU, each phase starts with a fault. Thus, any window containing a whole phase and the first request of the next phase contains $k+1$ faults on $k+1$ distinct pages. Hence, a phase has a length of at least $f^{-1}(k+1)-1$, which gives an upper bound on the fault rate of

$$\frac{k}{f^{-1}(k+1)-1}.$$

To prove an almost matching lower bound, we use a sequence constructed of blocks, phases, and super phases. The sequence contains requests to $k+1$ distinct pages $p_0, p_1, \ldots, p_k$. Each block consists of a number of requests to p_0 followed by one request to another page. The pages p_i, $i \neq 0$, are requested in cyclic order. Each phase consists of $k-1$ blocks.

Assume first that, $f^{-1}(4)-f^{-1}(3) > f^{-1}(3)-f^{-1}(2)$. In this case, the length of the first block of a phase is $f^{-1}(3)-f^{-1}(2)+1 = f^{-1}(3)-1$, and for $2 \leq i \leq k$, the length of the ith block of a phase is $f^{-1}(i+2)-f^{-1}(i+1)$. A super phase consists of k phases. For $k=5$, a super phase might look as illustrated in Figure 4.4.

FIFO faults on each request to a page $p_i \neq p_0$, and each time all k pages p_i, $1 \leq i \leq k$, have been requested, the next request to p_0 is a fault. This gives a total of $(k+1)(k-1) = k^2-1$ faults per super phase.

p_0 p_1	p_0 p_2	p_0 p_0 p_3	p_0 p_0 p_0 p_4
p_0 p_5	p_0 p_1	p_0 p_0 p_2	p_0 p_0 p_0 p_3
p_0 p_4	p_0 p_5	p_0 p_0 p_1	p_0 p_0 p_0 p_2
p_0 p_3	p_0 p_4	p_0 p_0 p_5	p_0 p_0 p_0 p_1
p_0 p_2	p_0 p_3	p_0 p_0 p_4	p_0 p_0 p_0 p_5

Figure 4.4: An example super phase, $k = 5$.

For $2 \leq i \leq k-1$, the first i blocks of a phase have a total length of

$$\begin{aligned} f^{-1}(3) - 1 + \sum_{j=2}^{i} \left(f^{-1}(j+2) - f^{-1}(j+1)\right) &= f^{-1}(3) - 1 + f^{-1}(i+2) - f^{-1}(3) \\ &= f^{-1}(i+2) - 1. \end{aligned}$$

Thus, the length of a phase is $f^{-1}(k+1)-1$, and the length of a super phase is $k(f^{-1}(k+1)-1)$. This gives a fault rate of

$$\frac{k^2-1}{k\left(f^{-1}(k+1)-1\right)} = \frac{k-\frac{1}{k}}{f^{-1}(k+1)-1}.$$

To prove that this is a valid lower bound on the fault rate of FIFO, we must show that the constructed sequence is consistent with f. Since we assume that $f^{-1}(4) - f^{-1}(3) > f^{-1}(3) - f^{-1}(2)$, the second block of a phase is at least as long as the first block of a phase. Thus, by Proposition 4.5, the block lengths are non-decreasing within a phase. Therefore, a shortest possible window containing n distinct pages, $3 \leq n \leq k+1$, can be found by taking the first $n-2$ blocks of a phase and the last request of the previous phase. Such a window has a length of at least $f^{-1}(n) - 1 + 1 = f^{-1}(n)$.

This proves that, if $f^{-1}(4) - f^{-1}(3) > f^{-1}(3) - f^{-1}(2)$, the fault rate of FIFO is at least $\frac{k-1/k}{f^{-1}(k+1)-1}$. This fault rate is larger than that of LRU, if

$$\frac{k-\frac{1}{k}}{f^{-1}(k+1)-1} > \frac{k-1}{f^{-1}(k+1)-2},$$

which is equivalent to $f^{-1}(k+1) > k+2$. Roughly speaking, this will be the case for sequences that exhibit locality of reference within windows of length at least $k+3$.

For completeness, consider also the case $f^{-1}(4) - f^{-1}(3) = f^{-1}(3) - f^{-1}(2)$. In this case, the sequence just described is not consistent with f. Let $s = \min\{i \geq 4 \mid f^{-1}(i+1) - f^{-1}(i) > f^{-1}(i) - f^{-1}(i-1)\}$. For $1 \leq i \leq \min\{s-2, k-1\}$, we let the ith block of a phase have length $f^{-1}(i+2) - f^{-1}(i+1) + 1$. For $\min\{s-2, k-1\} + 1 \leq i \leq k-1$, we let the ith block have length $f^{-1}(i+2) - f^{-1}(i+1)$ as before. This results in a phase length of

$f^{-1}(k+1) - 1 + \min\{s-3, k-2\} = f^{-1}(k+1) + \min\{s-4, k-3\}$ and a fault rate of

$$\frac{k - \frac{1}{k}}{f^{-1}(k+1) + \min\{s-4, k-3\}} \ .$$

Assume that $s \leq k$. Then, the fault rate of FIFO is larger than that of LRU, if

$$\frac{k - \frac{1}{k}}{f^{-1}(k+1) + s - 4} > \frac{k-1}{f^{-1}(k+1) - 2} \ .$$

This is equivalent to $f^{-1}(k+1) > (s-2)k + 2$. If $s \geq k+1$ and $f(3) = 3$, then $f(k+1) = f(2) + \sum_{i=2}^{k}(f(i+1) - f(i)) = 2 + \sum_{i=2}^{k} 1 = k+1$, and the fault rate of any algorithm with respect to f is 1. If $s \geq k+1$ and $f^{-1}(3) > 3$, then the fault rate of FIFO is larger than that of LRU, if

$$\frac{k - \frac{1}{k}}{f^{-1}(k+1) + k - 3} > \frac{k-1}{f^{-1}(k+1) - 2} \ ,$$

which is equivalent to $f^{-1}(k+1) > k^2 - k + 2$. For large values of k, even this amount of locality of reference does not seem unrealistic.

4.4.4 Marking Algorithms

To prove an upper bound on the fault rate of any marking algorithm, we partition the input sequence into phases corresponding to the marking phases. In each phase, exactly k distinct pages are requested, and each page causes at most one fault. When all pages in the cache are marked, a new phase starts when a page not in cache is requested. Thus, the first page of a phase is a page that was not requested in the previous phase. We conclude that a phase has a length of at least $f^{-1}(k+1) - 1$. Since each phase contains at most k faults, the fault rate is at most

$$\frac{k}{f^{-1}(k+1) - 1}.$$

To see that the upper bound is best possible, consider the following class of marking algorithms. On the first fault within a phase, the page that was requested last in the previous phase is evicted. Clearly, this class contains FWF.

The lower bound sequence contains k distinct pages and is constructed in phases, each consisting of k blocks. The ith block of a phase consists of $f^{-1}(i+1) - f^{-1}(i)$ requests to the page that was not in cache at the end of the previous block. Thus, the algorithm faults k times per phase, and the length of a phase is $\sum_{i=1}^{k}(f^{-1}(i+1) - f^{-1}(i)) = f^{-1}(k+1) - 1$.

To see that the sequence is consistent with f, note that the page requested in the second block of a phase is the page that was requested in the last block of the previous phase. Furthermore, by Proposition 4.5, the block lengths are non-decreasing within a phase. Thus, a shortest possible window containing n distinct pages can be found taking the first $n-1$ blocks of a phase (the first block contains only one page) and the first request of the nth block of the phase (the first block of the following phase, if $n = k+1$). This shows that the length of a window containing requests to n distinct pages is at least

$$\sum_{i=1}^{n-1}\left(f^{-1}(i+1) - f^{-1}(i)\right) + 1 = (f^{-1}(n) - 1) + 1 = f^{-1}(n),$$

which proves the consistency with f.

4.4.5 LFD

To prove an upper bound on the fault rate of LFD, we partition any input sequence into phases, such that each phase contains requests to exactly k distinct pages. For each phase i, let m_i be the number of *new* pages, i.e., pages that were not requested in phase $i-1$. Consider the off-line algorithm that evicts only pages that are not requested in the next phase. This is possible, since the cache can hold k pages. This algorithm has at most m_i faults in phase i, which gives an upper bound on the average number of faults per phase of $\overline{m} = \frac{1}{n}\sum_{i=1}^{n} m_i$, where n denotes the number of phases. Since LFD has the best possible fault rate, this is an upper bound on the average number of faults of LFD.

Any two consecutive phases $i-1$ and i have a length of at least $f^{-1}(k+m_i)$. This gives a lower bound on the length of the sequence of approximately $\frac{1}{2}\sum_{i=1}^{n} f^{-1}(k+m_i)$, which is at least $\frac{1}{2}n \cdot f^{-1}\big(\lfloor \frac{1}{n}\sum_{i=1}^{n}(k+m_i)\rfloor\big)$ (by Proposition 2 in the paper). Thus, the average phase length is at least $\frac{1}{2}f^{-1}\big(\lfloor \frac{1}{n}\sum_{i=2}^{n}(k+m_i)\rfloor\big)$. This gives an upper bound on the fault rate of

$$2\,\frac{\overline{m}}{f^{-1}\left(\lfloor \frac{1}{n}\sum_{i=1}^{n}(k+m_i)\rfloor\right)} = 2\,\frac{\overline{m}}{f^{-1}\left(\lfloor k+\overline{m}\rfloor\right)} \le 2 \max_{\substack{1\le m\le k\\ k+m\le M}} \left\{\frac{m+1}{f^{-1}(k+m)}\right\}.$$

We now prove a lower bound that is essentially a factor of two away from the upper bound just proven. Choose an $m \in \mathbb{N}$ such that $\frac{m}{f^{-1}(k+m+1)-1}$ is maximized, and let $N = k+m$. We construct a sequence containing N distinct pages in phases consisting of $N-1$ blocks each. Each block contains requests to only one page, and the N pages are requested in a cyclic order. The page requested in the last block of a phase P_j is not requested in the following phase P_{j+1}. Since this page is in the cache at the end of P_j, at most $k-1$ of the pages requested in P_{j+1} are in cache at the end of P_j. Thus, LFD has at least $N-1-(k-1) = N-k$ faults in each phase. The ith block of a phase has a length of $f^{-1}(i+2) - f^{-1}(i+1)$. Thus, the length of a phase is $f^{-1}(N+1)-1 = f^{-1}(k+m+1)-1$. The argument that the sequence is consistent with f is analogous to that of the proof of the general lower bound for deterministic on-line algorithms. This gives a lower bound of

$$\max_{\substack{m\in\mathbb{N}\\ k+m\le M}} \left\{\frac{m}{f^{-1}(k+m+1)-1}\right\}.$$

4.5 The Average-Model

The proofs of the results of the Average-Model tend to be more complicated than those of the Max-Model. Only the upper bound for LRU is extremely simple to prove.

All bounds on the fault rates of the Average Model are tight in some sense. The general lower bound for deterministic algorithms matches the upper bound on the fault rate of LRU and FIFO with respect to any concave* function. For LFD, there exists a concave* function f such that the fault rate of LFD with respect to this f matches the upper bound on the fault rate of LFD. As to the class of marking algorithms, there is a marking algorithm $\mathcal{M}$ and a concave* function f such that the fault rate of $\mathcal{M}$ with respect to f matches the general upper bound for marking algorithms.

We need some additional notation. For any sequence σ of page requests, $\sigma[i]$ denotes the ith request r in σ as well as the page requested by r, $1 \le i \le |\sigma|$. For $1 \le i \le |\sigma| - \ell + 1$, let $\sigma_\ell[i]$ be the subsequence (window) $\langle \sigma[i], \sigma[i+1], \ldots, \sigma[i+\ell-1]\rangle$. Let $N_\ell(i)$ be the number

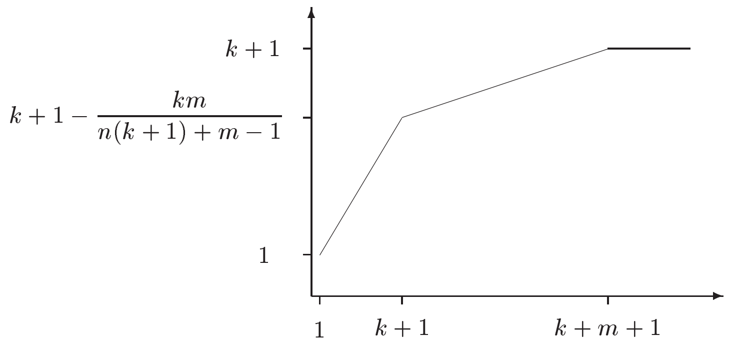


Figure 4.5: A(ℓ), an upper bound on Av(ℓ).

of distinct pages in $\sigma_\ell[i]$, and let $N_\ell = \sum_{i=1}^{|\sigma|-\ell+1} N_\ell(i)$. Let Av($\ell$) be the average number of distinct pages in windows of length ℓ, i.e., $\mathrm{Av}(\ell) = \frac{N_\ell}{|\sigma|-\ell+1}$.

A sequence σ consistent with a given concave* function f has $\mathrm{Av}(\ell) \leq f(\ell)$, $1 \leq \ell \leq |\sigma|$.

4.5.1 A Lower Bound for Deterministic Algorithms

To prove the general lower bound of $\frac{f(k+1)-1}{k}$, we construct an input sequence consisting of requests to $k+1$ distinct pages $p_1, p_2, \ldots, p_{k+1}$. The sequence consists of two parts. For some large integer n, the first part consists of $n(k+1)$ requests that will all make the algorithm fault. To ensure that the sequence is consistent with f, a second part is added. For some integer m dependent on n and f, this part consists of m requests to only one page.

Since the algorithm faults on each of the $n(k+1)$ first requests, the fault rate will be at least $\frac{n(k+1)}{n(k+1)+m}$. For any m such that σ is consistent with f, this fraction yields a valid lower bound on the fault rate of any deterministic algorithm. To find a such m, we should calculate the average function for the sequence, or at least an upper bound on the average function.

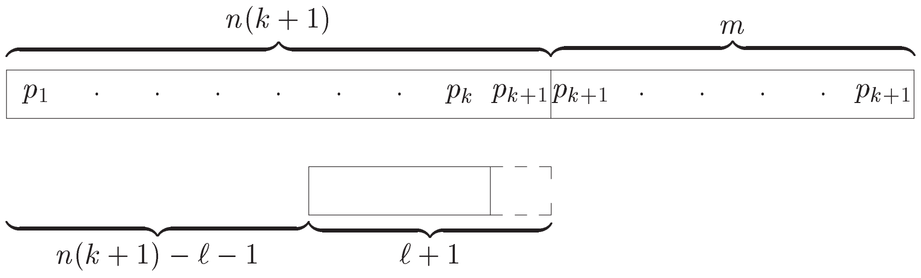
Among all sequences of length $n(k+1)$ containing $k+1$ distinct pages, the sequence $\sigma = \langle p_1, p_2, \ldots, p_{k+1}\rangle^n$ has the highest possible average number of distinct pages, for each possible window length. Thus, the sequence we will investigate is $\sigma = \langle p_1, p_2, \ldots, p_{k+1}\rangle^n \langle p_{k+1}\rangle^m$. The average function Av for this sequence is an upper bound on the average function for any sequence constructed as described.

We will prove that the function A defined below is an upper bound on Av. The function consists of three linear parts (see Figure 4.5):

$$\mathrm{A}(\ell) = \begin{cases} 1 + \left(1 - \frac{m}{n(k+1)+m-1}\right)(\ell - 1), & 1 \leq \ell \leq k+1 \\ k+1 - \frac{km}{n(k+1)+m-1} + \frac{k}{n(k+1)+m-1}\big(\ell - (k+1)\big), & k+1 \leq \ell \leq k+m+1 \\ k+1, & k+m+1 \leq \ell \leq |\sigma| \end{cases}$$

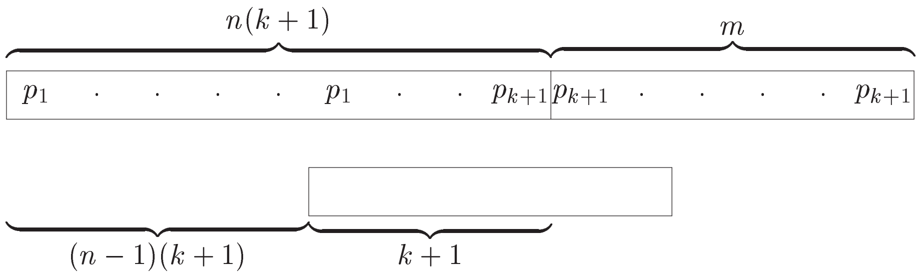
Clearly, $\mathrm{Av}(1) = 1$. Thus, to calculate an upper bound on Av(ℓ) for $1 \leq \ell \leq |\sigma|$, it suffices to calculate an upper bound on $\mathrm{Av}(\ell+1) - \mathrm{Av}(\ell)$ for $1 \leq \ell \leq |\sigma| - 1$.

We first consider small ℓ. Assume $1 \leq \ell \leq k$. The sequence contains $|\sigma| - \ell + 1$ windows of length ℓ. The first $n(k+1) - \ell + 1$ of these windows contain ℓ distinct pages each, the last $m - \ell + 1$ windows contain only one page each, and for each i, $2 \leq i \leq \ell - 1$, there is exactly



$n(k+1)$
m
p_1 · · · · · · · p_k p_{k+1} p_{k+1} · · · · p_{k+1}
$n(k+1)-\ell-1$
$\ell+1$

Figure 4.6: $\ell \leq k$: The windows $\sigma_\ell[1], \ldots, \sigma_\ell[n(k+1)-\ell]$ will each contain one new page, when ℓ is incremented.



$n(k+1)$
m
p_1 · · · · p_1 · · p_{k+1} p_{k+1} · · · · p_{k+1}
$(n-1)(k+1)$
$k+1$

Figure 4.7: $k+1 \leq \ell \leq k+m+1$: The windows $\sigma_\ell[1], \ldots, \sigma_\ell[(n-1)(k+1)+1]$ contain $k+1$ pages each. The windows $\sigma_\ell[n(k+1)], \ldots, \sigma_\ell[n(k+1)-\ell+1]$ contain one page each. For $2 \leq i \leq k$, the window $\sigma_\ell[n(k+1)-i+1]$ contains i pages.

one window containing exactly i distinct pages. When extending the window length from ℓ to $\ell+1$, the first $n(k+1)-\ell$ windows will contain one additional page. The number of distinct pages in each of the rest of the windows will remain unchanged (see Figure 4.6). Thus, if n is much larger than k, $\text{Av}(\ell+1) - \text{Av}(\ell)$ is close to $\frac{n(k+1)}{n(k+1)+m}$. However, we will calculate an exact upper bound on $\text{Av}(\ell+1) - \text{Av}(\ell)$.

For $1 \leq \ell \leq k$, $\text{Av}(\ell+1) - \text{Av}(\ell)$ is a slightly decreasing function of ℓ, since in each step, one window less has its number of distinct pages increased. Thus, for $1 \leq \ell \leq k$,

$$\begin{aligned}\text{Av}(\ell+1) - \text{Av}(\ell) \;&\leq\; \text{Av}(2) - \text{Av}(1) \;=\; \frac{(n(k+1)-1)\cdot 2 + m \cdot 1}{n(k+1)+m-1} - 1 \\ &= 1 - \frac{m}{n(k+1)+m-1}.\end{aligned}$$

For $k+1 \leq \ell \leq k+m$, $\text{Av}(\ell)$ is still an increasing function. This can be seen in the following way. When ℓ is incremented, the number of windows with only one page decreases by one, whereas the number of windows with $k+1$ distinct pages stays the same (and for each i, $2 \leq i \leq k$, the number of windows with exactly i pages remains one). See Figure 4.7. (When the window length reaches $k+1+m$, all windows have $k+1$ distinct pages.)

As ℓ increases, the number of windows of length ℓ decreases. Hence, the drop in the number of windows with requests to only one page means more and more. In other words, between $k+1$ and $k+m+1$, $\mathrm{Av}(\ell)$ grows faster and faster. Thus, the straight line between $(k+1, k+1-\frac{km}{n(k+1)+m-1})$ and $(k+m+1, k+1)$ is an upper bound on Av in this interval. This line has a slope of

$$\frac{k+1-\left(k+1-\frac{km}{n(k+1)+m-1}\right)}{k+m+1-(k+1)} = \frac{k}{n(k+1)+m-1} .$$

What remains to be done is to determine m such that the sequence σ is consistent with a given concave* function f. Since A is an upper bound on Av, σ is consistent with f, if $\mathrm{A}(\ell) \leq f(\ell)$, for all ℓ, $1 \leq \ell \leq |\sigma|$. Since f is concave*, it is sufficient to prove

1. $\mathrm{A}(1) \leq f(1)$
2. $\mathrm{A}(k+1) \leq f(k+1)$
3. $\mathrm{A}(k+m+1) \leq f(k+m+1)$.

1. follows immediately from $\mathrm{A}(1) = 1 = f(1)$.
2. is equivalent to

$$k+1-\frac{km}{n(k+1)+m-1} \leq f(k+1),$$

which in turn is equivalent to

$$m \geq \frac{k+1-f(k+1)}{f(k+1)-1}\big(n(k+1)-1\big).$$

Thus, we let

$$m = \frac{k+1-f(k+1)}{f(k+1)-1}\, n(k+1).$$

If $f(k+1) = k+1$, any deterministic paging algorithm has a fault rate of $1 = \frac{f(k+1)-1}{k}$.

If $f(k+1) < k+1$, m grows linearly with n. Thus, there exists an $n_0 \in \mathbb{N}$ such that, for all $n \geq n_0$, $k+m+1 \geq f^{-1}(k+1)$. Since $\mathrm{A}(k+m+1) = k+1$, this shows that n can be chosen such that 3. is fulfilled.

We obtain a lower bound on the fault rate of any deterministic paging algorithm of

$$\frac{n(k+1)}{n(k+1)+m} = \frac{1}{1+\frac{k+1-f(k+1)}{f(k+1)-1}} = \frac{f(k+1)-1}{k} .$$

4.5.2 Upper Bounds

So far we have focused on windows, e.g., counting the number of distinct pages within windows of a given length or windows containing k faults. In the upper bound proofs of the Average-Model we will instead focus on pages. Assume that the input sequence contains requests to n distinct pages $p_1, p_2, \ldots, p_n$. For $1 \leq i \leq n$, let $w_\ell(i)$ be the number of windows of length ℓ containing a request to p_i. Then, $N_\ell = \sum_{i=1}^{n} w_\ell(i)$. We will say that p_i contributes $w_\ell(i)$ to N_ℓ.

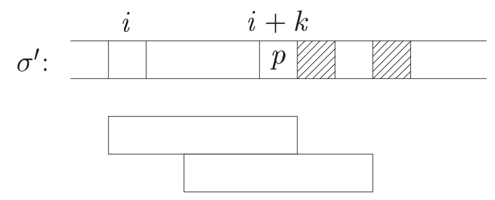


Figure 4.8: For each position j such that $N'_{k+1}(j) < N_{k+1}(j)$, the page requested just after the window $\sigma'_{k+1}[j]$ is different from each page requested inside the window.

4.5.3 LRU and FIFO are Optimal

In the Average Model, both LRU and FIFO are optimal. For LRU, this can be seen in the following way. Whenever a page p is requested, the next k requests cannot incur a fault on p. Thus, each fault on p is contained in $k+1$ windows of length $k+1$ containing no other faults on p. Furthermore, each request to p that does not incur a fault is the first request of a window of length $k+1$ containing no fault on p. This shows that (except for the first and last k requests) each fault contributes $k+1$ to N_{k+1} and each request that does not incur a fault contributes at least 1 to N_{k+1}. This gives approximately

$$N_{k+1} \;\geq\; (k+1)\cdot \mathrm{LRU}(\sigma) + \big(|\sigma| - \mathrm{LRU}(\sigma)\big) \;=\; k\cdot \mathrm{LRU}(\sigma) + |\sigma|,$$

and

$$\mathrm{Av}(k+1) \;\geq\; \frac{k\cdot \mathrm{LRU}(\sigma) + |\sigma|}{|\sigma|} \;=\; k\cdot F_{\mathrm{LRU}}(\sigma) + 1.$$

Since σ is consistent with f,

$$f(k+1) \;\geq\; \mathrm{Av}(k+1) \;\geq\; k\cdot F_{\mathrm{LRU}}(\sigma) + 1.$$

Solving for the fault rate, we obtain an upper on the fault rate of LRU matching the general lower bound.

When it comes to FIFO, we cannot say that there are at least k requests between each pair of requests to a given page p. We can only say that, between two *faults* on p, there are at least k requests. Let the term *free request* denote a request that is not a fault. We must find an alternative way to prove that free requests contribute to N_{k+1}.

Assume that we remove all free requests from the sequence and then we put them back into the sequence one by one. We show that for each request put back into the sequence, N_{k+1} increases by at least one. This is illustrated in Figure 4.8: A request to a page p is inserted just before the $(i+k)$th request of the sequence σ, for some i. The resulting sequence is denoted σ'.

We study windows of length $k+1$ in σ and σ'. For each $j \leq i-1$, $\sigma_{k+1}[j] = \sigma'_{k+1}[j]$ and for each $j \geq i+k$, $\sigma_{k+1}[j] = \sigma'_{k+1}[j+1]$, so we need only consider the windows $\sigma_{k+1}[i], \ldots, \sigma_{k+1}[i+k-1]$ and $\sigma'_{k+1}[i], \ldots, \sigma'_{k+1}[i+k]$. To prove $N'_{k+1} \geq N_{k+1} + 1$, it suffices to prove

$$\sum_{j=i}^{i+k} N'_{k+1}(j) \;\geq\; 1 + \sum_{j=i}^{i+k-1} N_{k+1}(j) \;\;.$$

Let $i \leq j \leq i+k-1$. The window $\sigma'_{k+1}[j]$ contains the requests of $\sigma_k[j]$ and the new request to p. Therefore, $N_{k+1}(j)$ and $N'_{k+1}(j)$ can differ by at most one. Let n be the number of positions j for which $N_{k+1}(j) > N'_{k+1}(j)$. We just need that $N'_{k+1}(i+k) \geq n+1$. If $N_{k+1}(j) > N'_{k+1}(j)$, the last page $\sigma[j+k]$ requested in $\sigma_{k+1}[j]$ contributes to $N_{k+1}(j)$ and p does not contribute to $N'_{k+1}(j)$. This means that $\sigma[j+k]$ is different from each page in $\sigma_k[j]$ and from p. Assume that the windows shown in Figure 4.8 are those windows of σ' containing fewer distinct pages than the corresponding windows in σ. For each such window, the request immediately after the window is different from each request inside the window. Thus, the shaded requests are all to distinct pages different from p. This means that the window $\sigma'_{k+1}[i+k]$ contains at least $n+1$ distinct pages, namely p and those shaded in the figure. This completes the proof that each free request contributes at least one to N_{k+1}.

4.5.4 The Worst Marking Algorithm

We already know that there exists at least one optimal marking algorithm, namely LRU. There exists, however, a marking algorithm $\mathcal{M}$ and a concave* function f such that the fault rate of $\mathcal{M}$ with respect to f is approximately $\frac{4}{3}$ that of LRU. More precisely, the fault rate is

$$F_{\mathcal{M}} = \begin{cases} \dfrac{4k}{3k+2}\dfrac{f(k)}{k}, & k \text{ even} \\ \dfrac{4k}{3k+2-1/k}\dfrac{f(k)}{k}, & k \text{ odd.} \end{cases}$$

As we shall see, this is the worst possible fault rate of any marking algorithm with respect to any concave* function.

Lower bound

Consider the sequence $\textsc{UpDown}_h^n = \langle p_1, p_2, \ldots, p_{h-1}, p_h, p_{h-1}, \ldots, p_2 \rangle^n$, $h, n \in \mathbb{N}$. Such a sequence will also be used for proving the lower bound on the fault rate of LFD. A subsequence $\langle p_1, p_2, \ldots, p_{h-1}, p_h, p_{h-1}, \ldots, p_2 \rangle$ is called a phase. When n goes to infinity the average number of distinct pages in windows of length ℓ goes to

$$\mathrm{Av}_h^\infty(\ell) = \begin{cases} \ell - \dfrac{(\ell-1)^2}{4(h-1)}, & 1 \leq \ell \leq 2h-3,\ \ell \text{ odd,} \\ \ell - \dfrac{(\ell-1)^2-1}{4(h-1)}, & 2 \leq \ell \leq 2h-3,\ \ell \text{ even,} \\ h, & \ell \geq 2h-2. \end{cases}$$

For the windows starting in one of the first $n-1$ phases of $\textsc{UpDown}_h^n$, the average number of distinct pages in a window of length ℓ is $\mathrm{Av}_h^\infty(\ell)$. The sum of the number of distinct pages in all windows of length ℓ contained in the last $\textsc{UpDown}_h^n$ phase is at most $2(h-1)h$. Thus,

$$\begin{aligned} \mathrm{Av}(\ell) &\leq \frac{2(h-1)(n-1)\mathrm{Av}_h^\infty(\ell) + 2(h-1)h}{n \cdot 2(h-1) - \ell + 1} \\ &\leq \frac{2(h-1)(n-1)\mathrm{Av}_h^\infty(\ell) + 2(h-1)h}{2(h-1)(n-1)} \\ &= \mathrm{Av}_h^\infty(\ell) + \frac{h}{n-1}. \end{aligned}$$

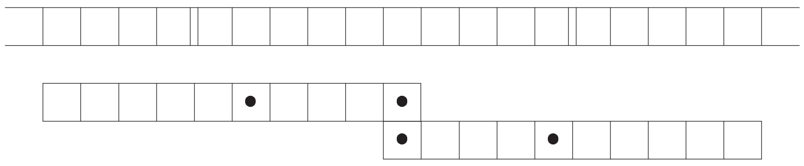


Figure 4.9: A phase containing $k = 10$ requests and the first and last window associated with the phase.

Let $\varepsilon = \frac{h}{n-1}$. Then, it is clear that the function

$$f(\ell) = \begin{cases} 1, & \ell = 1 \\ \mathrm{Av}_h^\infty(\ell) + \varepsilon, & 2 \le \ell \le 2h-3, \\ h, & \ell \ge 2h-2 \end{cases}$$

is an upper bound on $\mathrm{Av}(\ell)$, but this function is not concave*, since $f(2) - f(1) = 2 + \varepsilon - 1 > 1$. Therefore, we use the tighter upper bound

$$f(\ell) = \begin{cases} \min\{\ell, \mathrm{Av}_h^\infty(\ell) + \varepsilon\}, & 1 \le \ell \le 2h-3, \\ h, & \ell \ge 2h-2. \end{cases}$$

Consider now the marking algorithm $\mathcal{M}$ that uses the LIFO rule on the unmarked pages. That is, when a page must be evicted, $\mathcal{M}$ chooses the unmarked page that has been in the cache for the shortest time. $\mathcal{M}$ will fault on each request of $\textsc{UpDown}_{k+1}^n$. Thus,

$$\begin{aligned} F_{\mathcal{M}}(\sigma) &= 1 = \frac{k}{f(k)} \cdot \frac{f(k)}{k} = \frac{k}{\mathrm{Av}_{k+1}^\infty(k) + \frac{k+1}{n-1}} \cdot \frac{f(k)}{k} \\ &= \begin{cases} \dfrac{4k}{3k + 2 - \frac{1}{k} + 4\frac{k+1}{n-1}} \cdot \dfrac{f(k)}{k}, & k \text{ odd}, \\ \dfrac{4k}{3k + 2 + 4\frac{k+1}{n-1}} \cdot \dfrac{f(k)}{k}, & k \text{ even}. \end{cases} \end{aligned}$$

Note that the proof of the lower bound is also valid for FWF.

Upper Bound

The lower bound is best possible as can be seen by the following. Each phase contains requests to exactly k distinct pages. We will count how many windows of length k each of these pages is contained in.

Assume first that k is even. Consider a phase P containing the requests $\sigma[i], \ldots, \sigma[j]$. To ensure that nothing is counted twice, we will consider only the windows $\sigma_\ell[i - \frac{k}{2} + 1], \ldots, \sigma_\ell[j - \frac{k}{2} + 1]$. Figure 4.9 illustrates which windows are considered; the first and the last window are shown. Note that the second and the last request of the phase are contained in $\frac{k}{2} + 1$ of the windows considered. The third and the second to last request of the phase are contained in $\frac{k}{2} + 2$ of the windows considered, and so on. The $(\frac{k}{2} + 1)$st request is contained in k of the pages considered. The first page p requested in the phase is not requested in the previous

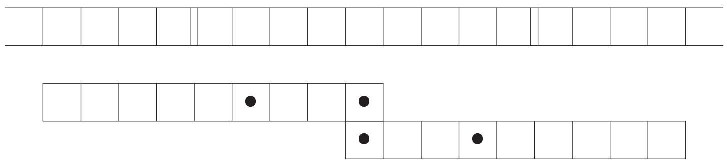


Figure 4.10: A phase containing $k = 9$ requests and the first and last window associated with the phase.

phase, so this page is contained in k windows containing no other requests to p. Thus, the total contribution from a phase is at least

$$2 \cdot \sum_{i=1}^{k/2} \left(\frac{k}{2} + i \right) = \frac{3}{4}k^2 + \frac{1}{2}k.$$

This is at least $\frac{3}{4}k + \frac{1}{2}$ per fault, since each phase contains at most k faults. Thus, $N_{k+1} \geq (\frac{3}{4}k + \frac{1}{2})\mathcal{M}(\sigma)$, and

$$f(k) \;\geq\; \mathrm{Av}(k) \;\geq\; \frac{(\frac{3}{4}k + \frac{1}{2})\mathcal{M}(\sigma)}{|\sigma|} \;=\; \left(\frac{3}{4}k + \frac{1}{2}\right) F_{\mathcal{M}}(\sigma) \;.$$

Hence,

$$F_{\mathcal{M}} \leq \frac{f(k)}{\frac{3}{4}k + \frac{1}{2}} = \frac{4k}{3k+2} \frac{f(k)}{k} \;.$$

Assume now that k is odd (see Figure 4.10). The first request contributes to k windows of length k. The second and the second to last request contribute to $\frac{k-1}{2} + 2$ each. The $\frac{k+1}{2}$th request contributes to k windows. The last request contributes to $\frac{k+1}{2}$ windows. Thus, each phase contributes

$$2k + \frac{k+1}{2} + \sum_{i=2}^{\frac{k-1}{2}} \frac{k-1}{2} + i = \frac{3}{4}k^2 + \frac{1}{2}k - \frac{1}{4}.$$

Doing the same calculations as in the case of k even, we arrive at

$$F_{\mathcal{M}} \;\leq\; \frac{4k}{3k + 2 - \frac{1}{k}} \frac{f(k)}{k} \;.$$

4.5.5 LFD

In this section we prove an upper bound on the fault rate of LFD of approximately $\frac{4M-4k}{4M-k} \frac{f(k+1)}{k+1}$ and give a concave* function with respect to which the fault rate of LFD exactly matches the upper bound.

Upper Bound

Consider any request sequence σ consistent with some concave* function f. Again we will analyze the contribution from faults and free requests to N_{k+1}. Like in the case of LRU and FIFO, no page generates more than one fault within a window of length $k+1$, and hence each fault contributes $k+1$ to N_{k+1}. Determining the contribution from free requests is a little more complicated, and we postpone that a little.

We partition the sequence into phases $P^1, P^2, \ldots, P^n$. The phase P^1 starts with the first request in the sequence and, for $2 \leq i \leq n$, phase P^i starts with the first fault on a page that was evicted in phase P^{i-1}. Thus, within a phase, there is at most one fault on each page, and the k pages that are in fast memory at the beginning of a phase do not generate a fault within the phase. Hence, each phase contains at most $M-k$ faults.

For $1 \leq i \leq n$, let F^i be the number of faults in phase P^i, and let N^i_{k+1} be the contribution to N_{k+1} from requests in P^i. Let W be a lower bound on the contribution to N_{k+1} from free requests within one phase.

Then,

$$\frac{N^i_{k+1}}{F^i} \;\geq\; \frac{(k+1)F^i + W}{F^i} \;\geq\; \frac{(k+1)(M-k)+W}{M-k} \quad .$$

Solving for F^i yields

$$F^i \;\leq\; \frac{M-k}{(k+1)(M-k)+W} \cdot N^i_{k+1}, \text{ and}$$

$$\mathrm{LFD}(\sigma) \;=\; \sum_{i=1}^{n} F^i \;\leq\; \frac{M-k}{(k+1)(M-k)+W} \sum_{i=2}^{n-2} N^i_{k+1} \;=\; \frac{M-k}{(k+1)(M-k)+W} \cdot N_{k+1}.$$

Thus,

$$F_{\mathrm{LFD}}(\sigma) \leq \frac{M-k}{(k+1)(M-k)+W} \cdot \mathrm{Av}(k+1) \leq \frac{M-k}{(k+1)(M-k)+W} \cdot f(k+1) \quad .$$

To finish the proof we must determine a lower bound W on the contribution to N_{k+1} from the free requests of one phase.

First observe that any phase P^i must contain free requests to at least $k-1$ distinct pages. This can be seen in the following way. Let p be the first page requested in phase P^{i+1}, and let s_{i+1} be the index of this request. Then p is evicted at some point during phase P^i. Assume that this happens as a result of a request with index q. Since p is the page to be evicted, the $k-1$ other pages $p_1, \ldots, p_{k-1}$ in the cache are requested at some point between $\sigma[q]$ and $\sigma[s_{i+1}]$. Each of these requests must be free, because otherwise P^i would contain a fault on a page that had been evicted earlier in the phase and this would contradict the definition of a phase.

For $1 \leq j \leq k-1$, let r_j be the first request to p_j after $\sigma[q]$, and let $W(r_j)$ be the number of windows containing r_j, no fault on p_j, and no request to p_j contained in P^{i-1}. Then $W = \sum_{j=1}^{k-1} W(r_j)$ is a lower bound on the contribution to N^i_{k+1}.

It is clear that the first k requests after r_j are not faults on p_j. Thus, when calculating $W(r_j)$, we only need to worry about requests to p_j that are to the left of r_j. Let d_j be the distance between r_j and the last request to p_j to the left of r_j, i.e., if h^r_j is the index of r_j and h^l_j is the index of the last request to p_j before r_j, then $d_j = h^r_j - h^l_j$. $W(r_j) = \min\{k+1, d_j\}$.

Note that $h^l_j \leq q < h^r_j$, and let $d^l_j = q - h^l_j$ and $d^r_j = h^r_j - q$, see Figure 4.11. Then,

$$\sum_{j=1}^{k-1} d_j \;=\; \sum_{j=1}^{k-1} d^l_j + \sum_{j=1}^{k-1} d^r_j \;\geq\; 2\sum_{j=1}^{k-1} j.$$

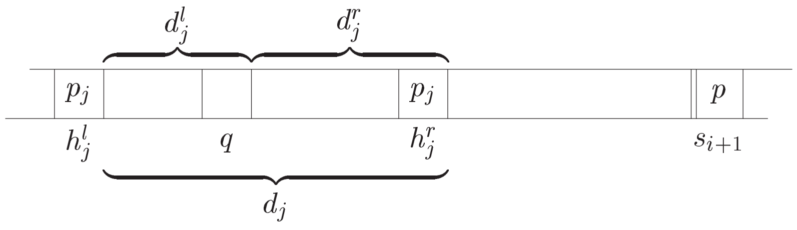


Figure 4.11: $\sigma[h_j^l]$: Last request to p_j before $\sigma[q]$. $\sigma[q]$: Causes p to be evicted. $\sigma[h_j^r]$: First request to p_j after $\sigma[q]$. $\sigma[s_{i+1}]$: First fault on p after $\sigma[q]$ — phase P^{i+1} begins.

Let S be the set of requests such that $d_j \leq k+1$, and let $m = |S|$. Then,

$$\begin{aligned} W &= \sum_{j=1}^{k-1} W(r_j) \;\geq\; (k-1-m)(k+1) + \sum_{r_j \in S} d_j \\ &\geq\; k^2 - 1 - m(k+1) + 2\sum_{j=1}^{m} j \;=\; k^2 - 1 + m^2 - km. \end{aligned}$$

This lower bound on W is minimized, when $m = \frac{k}{2}$, if k is even, and when $m = \frac{k-1}{2}$, if k is odd. Inserting these values, we get

$$W \;\geq\; \begin{cases} \dfrac{3}{4}k^2 - \dfrac{3}{4} \;=\; \dfrac{3}{4}(k-1)(k+1), & k \text{ odd,} \\ \dfrac{3}{4}k^2 - 1 \;=\; \dfrac{3}{4}(k-1)(k+1) - \dfrac{1}{4}, & k \text{ even,} \end{cases}$$

and

$$F_{\mathrm{LFD}}(f) \;\leq\; \begin{cases} \dfrac{4M-4k}{4M-k-3}\,\dfrac{f(k+1)}{k+1}, & k \text{ odd} \\ \dfrac{4M-4k}{4M-k-3-\frac{1}{k+1}}\,\dfrac{f(k+1)}{k+1}, & k \text{ even.} \end{cases}$$

Lower Bound

For some $n \in \mathbb{N}$, consider the sequence $\textsc{UpDown}_M^n$ as defined in Section 4.5.4. This sequence is consistent with the concave* function

$$f(\ell) \;=\; \begin{cases} \min\{\ell,\, \mathrm{Av}_M^\infty(\ell) + \frac{M}{n-1}\}, & 1 \leq \ell \leq 2M-3, \\ M, & \ell \geq 2M-2. \end{cases}$$

For n sufficiently large and k odd,

$$\begin{aligned} f(k+1) &= \mathrm{Av}_M^\infty(\ell) + \frac{M}{n-1} = k+1 - \frac{k^2-1}{4M-4} + \frac{M}{n-1} \\ &= \frac{4(M-1)(k+1)}{4(M-1)} - \frac{(k-1)(k+1)}{4(M-1)} + \frac{M}{n-1} \;=\; \frac{(4M-k-3+\varepsilon)(k+1)}{4(M-1)}\;, \end{aligned}$$

where $\varepsilon = \frac{4(M-1)}{k+1}\frac{M}{n-1}$. Similarly, for k even and n sufficiently large,

$$\begin{aligned} f(k+1) &= k+1-\frac{k^2}{4(M-1)}+\frac{M}{n-1} \\ &= \frac{4(M-1)(k+1)}{4(M-1)}-\frac{(k-1)(k+1)+1}{4(M-1)}+\frac{M}{n-1} \\ &= \frac{(4M-k-3-\frac{1}{k+1}+\varepsilon)(k+1)}{4(M-1)}\ . \end{aligned}$$

It is easily verified that, within the first half of a phase, LFD faults on the first request and the last $M-k-1$ requests. The same is true for the second half of a phase. Hence,

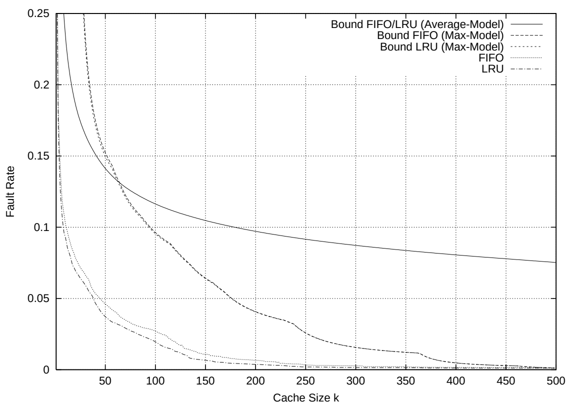
$$\begin{aligned} F_{\mathrm{LFD}}(\textsc{UpDown}^n_M) &= \frac{M-k}{M-1}\frac{f(k+1)}{f(k+1)} \\ &= \begin{cases} \frac{M-k}{M-1}\,\frac{4(M-1)}{4M-k-3+\varepsilon}\,\frac{f(k+1)}{k+1}, & k \text{ odd} \\ \frac{M-k}{M-1}\,\frac{4(M-1)}{4M-k-3-\frac{1}{k+1}+\varepsilon}\,\frac{f(k+1)}{k+1}, & k \text{ even} \end{cases} \\ &= \begin{cases} \frac{4(M-k)}{4M-k-3+\varepsilon}\,\frac{f(k+1)}{k+1}, & k \text{ odd} \\ \frac{4(M-k)}{4M-k-3-\frac{1}{k+1}+\varepsilon}\,\frac{f(k+1)}{k+1}, & k \text{ even.} \end{cases} \end{aligned}$$

4.6 Experiments

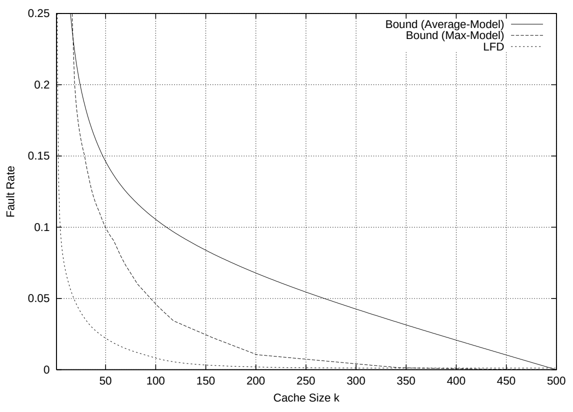
In this section we present some results of our experimental study in which we compared the worst-case fault rates developed in the previous sections to the fault rates observed on real processor traces. We analyzed memory reference traces from the New Mexico State University Trace Base [63] that contains standard benchmarks. We selected traces from VAX and SPARC platforms. More specifically, we chose the ATUM VAX traces and a bundle of SPARC traces that were collected while running the SPEC92 benchmark suite. The sets consist of a collection of 9 respectively 13 memory reference traces from single processes. The request sequences contain both data read/write requests and instruction fetches. The SPARC traces were truncated after 10 million references, whereas the VAX traces vary in length, but are all about 400,000 requests. We worked with a page size of 512 bytes for the VAX architecture and a page size of 2048 bytes for the SPARC architecture.

We first analyzed the maximum and average working set size in windows of up to 100,000 requests. Figure 4.2 in Section 4.3 presents the results for four specific traces, two VAX traces and two SPARC traces.

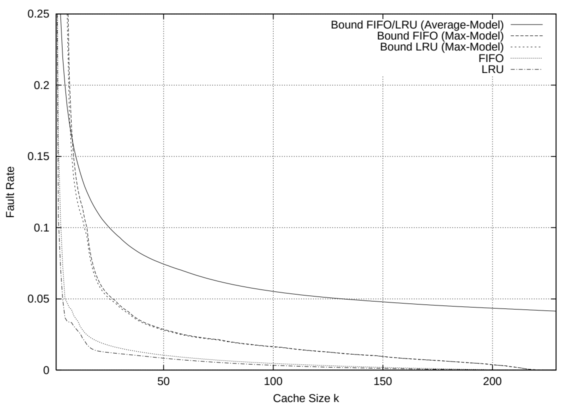
In the second part of the experiments, we evaluated the fault rates of LRU, FIFO, and LFD on the various traces and compared the values to the corresponding bounds we developed for both the Max- and the Average-Model. We performed the comparison for cache sizes ranging from 1 to the maximum working set size. Figure 4.12 presents the results for the VAX Pascal and the SPARC Compress traces. The left part of the figure shows the results for LRU and FIFO. The two lower curves represent the empirical fault rates of LRU and FIFO, while the



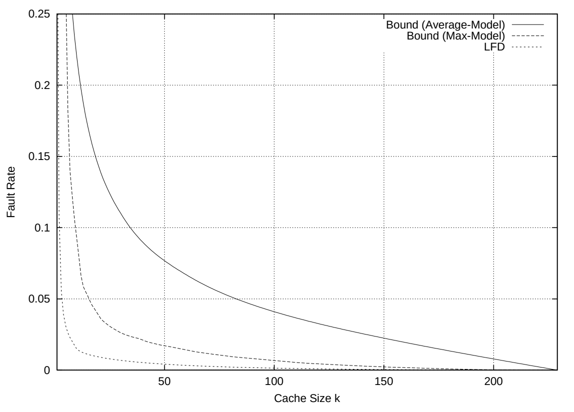
(a) VAX, PASCAL — FIFO and LRU.



(b) VAX, PASCAL — LFD



(c) SPARC, COMPRESS — FIFO and LRU.



(d) SPARC, COMPRESS — LFD.

Figure 4.12: Measured fault rates and upper bounds on the fault rates for FIFO, LRU and LFD. The fast memory size k varies in the range of 1 up to the total number of distinct pages requested in the entire sequence.

two curves in the middle show the corresponding theoretical upper bounds in the Max-Model. The upper curve depicts the bound in the Average-Model. The right part of Figure 4.12 shows the bounds for LFD in the same relative order.

Since the fault rate as defined in 4.4 is a worst-case measure, we cannot expect that the theoretical bounds on the fault rates match the empirical values completely. Nevertheless, the gap is not large and *considerably* smaller than in the case of competitiveness. On real world traces, the "empirical competitiveness" of LRU and FIFO is typically no larger then 4. This was observed in [18, 103] and also shown in our experiments. On the other hand, the competitive ratios from theory are k. Thus, the gap between the theoretical and empirical competitiveness is $k/4$. In our paging model, the gaps are considerably smaller. For the SPARC COMPRESS trace the gap is, expressed as a function linear in k, usually between $k/50$ to $k/30$. For some of the traces we examined, the values were even below $k/1000$.

We also remark that, throughout our experiments, the fault rates predicted in the Max-Model were closer to the empirical fault rates than those of the Average-Model. This corresponds to some extent to the intuition we gained by working on the models. In the Average-Model, the adversary seems to be less limited than in the Max-Model; in the Average-Model, the adversary can give any sequence he likes, and then pad it with a sufficient number of requests to only one page.

Chapter 5

Edge Coloring with a Fixed Number of Colors

In [49] we study the maximization version of edge coloring, i.e., the version where only a limited number of colors are available, and the aim is to color as many edges as possible. This chapter describes the results.

5.1 Algorithms

We mainly consider the class of *fair* algorithms, i.e., algoithms that never reject edges that they are able to color. However, one of the results is valid for any algorithm, fair or not fair, deterministic or randomized. To denote an arbitrary algorithm for edge coloring with a fixed number of colors, we use the term *on-line*^{R}. An algorithm that is fair and might be randomized is called *fair*^{R}. Similarly, we let *fair*^{D} and *on-line*^{D} denote deterministic algorithms that are fair, might not be fair, respectively. We let *off-line* denote an optimal off-line algorithm.

We also consider two specific fair algorithms, First-Fit and Next-Fit. *First-Fit* always uses the lowest numbered color possible. *Next-Fit* uses the colors in a cyclic order. It colors the first edge with the color 1. Whenever it uses a color c, it will color the next edge e with the first color in the sequence $\langle c+1, \ldots, k, 1, \ldots, c-1\rangle$ not used on edges adjacent to e, if any. Intuitively, the Next-Fit strategy is a poor strategy, and as can be seen in the next section, Next-Fit has the worst possible competitive ratio among fair algorithms. Thus, we include it only to prove that the impossibility results for fair algorithms are tight. The First-Fit strategy seems more reasonable, since it tries to save the higher numbered colors until it really needs them. Unfortunately, it turns out that First-Fit is not much better than Next-Fit.

5.2 Results

We have analyzed the competitive ratio of the algorithms and classes of algorithms defined in the previous section. The results are shown in Table 5.1. For general graphs the results are rather pessimistic. No algorithm can be more than $\frac{4}{7}$-competitive and no fair deterministic algorithm can be more than $\frac{1}{2}$-competitive. There is not much room for variation among fair deterministic algorithms, since any fair algorithm is more than $(2\sqrt{3}-3)$-competitive ($2\sqrt{3}-3 \approx 0.4641$). Next-Fit has the worst possible competitive ratio. For most values of k it is close to $2\sqrt{3}-3$, and there are values of k for which it gets arbitrarily close to $2\sqrt{3}-3$.

Comp. Ratio	Fair	Det.	Fair, Det.	Any	Next-Fit	First-Fit
General	$C \geq 2\sqrt{3}-3$ ≈ 0.4641		$C \leq \frac{1}{2}$ $= 0.5$	$C \leq \frac{4}{7}$ ≈ 0.5714	$C = 2\sqrt{3}-3$ ≈ 0.4641	$C \leq \frac{2}{9}(\sqrt{10}-1)$ ≈ 0.4805
k-Colorable	$C \geq \frac{1}{2}$	$C \leq \frac{2}{3}$	$\frac{1}{2} \leq C \leq \frac{2}{3}$		$C = \frac{1}{2}$	$C = \frac{k}{2k-1}$

Table 5.1: Competitive ratios C of the algorithms and classes of algorithms considered

Though, intuitively, First-Fit is a more reasonable algorithm than Next-Fit, we proved that the competitive ratio of First-Fit is at most $\frac{2}{9}(\sqrt{10}-1) \approx 0.4805$, and hence it cannot be much better than Next-Fit.

In the special case where the input graphs are all k-colorable, there might be more variation. The best upper bound we could prove is $\frac{2}{3}$ for deterministic algorithms. The lower bound of $\frac{1}{2}$ for fair algorithms on k-colorable graphs is only a little higher than the lower bound in the general case. Again, Next-Fit is used to prove that the bound is tight. For small values of k, First-Fit is significantly better than Next-Fit, but the difference tends to zero as k increases.

Analyzing the special case of k-colorable graphs is analogous to analyzing the special case for the seat reservation problem, where all request sequences can be accommodated off-line. The difference between accommodating sequences and general sequences is, however, far from as dramatic as for the seat reservation problem. The lower bound for fair algorithms is only raised a little. For small values of k, the competitive ratio of First-Fit is significantly better than that of Next-Fit, for $k = 2$, their respective ratios are $\frac{1}{2}$ and $\frac{2}{3}$, but for large k, the difference is insignificant.

However, analyzing k-colorable graphs has the extra advantage that the analysis of k-colorable graphs in some cases can serve as a stepping stone to the more general analysis with no restrictions on the graphs. This was in particular the case for the lower bounds for fair algorithms.

5.3 Graphs

As described in the previous section, we study the general case as well as the special case where all input graphs are known to be k-colorable. The performance guarantees proven are valid even if we allow multigraphs, i.e., graphs that may have parallel edges, but no loops. The adversary graphs used for proving impossibility results are all simple graphs. Furthermore, the adversary graphs are all bipartite except one that could easily be changed to a bipartite graph. Thus, the impossibility results are all valid, even if the input graphs are known to be simple, bipartite graphs.

5.4 Basics

A *k-coloring* is a coloring using at most k colors. We label the colors $1, 2, \ldots, k$. For any $i, j \in \{1, 2, \ldots, k\}$, we let $C_{i,j}$ denote the subset $\{i, i+1, \ldots, j\}$ of the k colors.

A *bipartite* graph is a graph whose vertex set can be partitioned in two sets X and Y, such that no two vertices within the same set are adjacent. In a *complete* bipartite graph each vertex in X is connected to each vertex in Y.

The *degree* of a vertex x is the number of edges incident to x. The *colored degree* of x is the number of edges incident to x colored by the on-line algorithm under consideration.

An *r-regular* graph is a graph in which all vertices have degree r.

By König's Theorem [100, p. 209], any bipartite graph with maximum degree d is *d-colorable*, i.e., it can be colored using at most d colors.

The following claim is useful when constructing adversary graphs for Next-Fit.

Claim 5.1 *Any coloring in which each color is used on exactly n or $n+1$ edges, for some $n \in \mathbb{N}$, can be produced by Next-Fit, for some ordering of the input sequence. The colors just need to be permuted so that the colors used on $n+1$ edges are the lowest numbered colors.*

5.5 k-Colorable Graphs

We start out investigating the special case, where all input graphs are k-colorable. Proving the performance guarantee for fair algorithms on k-colorable graphs is rather simple and serves as a stepping stone to proving the corresponding guarantee for general graphs. The adversary graphs proving that Next-Fit is worst possible, on k-colorable graphs and in the general case, have the same overall structure. However, in the case of k-colorable graphs, the graphs are constructed such that the vertex degrees are as similar as possible. In the general case, the vertex degrees are determined in a more complicated way.

5.5.1 A Performance Guarantee for Fair Algorithms

For any vertex x, let $d_{\mathrm{c}}(x)$ denote the number of edges incident to x that have been colored by *fair*^{R}. Similarly, let $d_{\mathrm{u}}(x)$ denote the number of edges incident to x that have *not* been colored by *fair*^{R}. We will take the on-line algorithm's perspective and call these edges uncolored edges. To prove that any fair algorithm colors at least half of the edges of any k-colorable graph, we need only two simple observations.

(1) For any vertex x, $d_{\mathrm{c}}(x) + d_{\mathrm{u}}(x) \leq k$, since *off-line* colors all edges incident to x using at most k colors.

(2) For any uncolored edge (x,y), $d_{\mathrm{c}}(x) + d_{\mathrm{c}}(y) \geq k$, since the algorithm is fair and the edge was not colored.

Assume that one unit of some value is put on each edge colored by *fair*^{R}. If the total value can be redistributed to the uncolored edges such that each uncolored edge receives at least one unit, there are at least as many colored as uncolored edges. The redistribution is done in the following way. Each vertex receives half a unit from each colored edge incident to it. In this way, each vertex x receives $\frac{1}{2}d_{\mathrm{c}}(x)$. After that, each vertex splits its value equally among the uncolored edges incident to it. Each uncolored edge (x,y) receives the value

$$\begin{aligned}
\frac{1}{2}\left(\frac{d_{\mathrm{c}}(x)}{d_{\mathrm{u}}(x)} + \frac{d_{\mathrm{c}}(y)}{d_{\mathrm{u}}(y)}\right) &\overset{(1)}{\geq} \frac{1}{2}\left(\frac{d_{\mathrm{c}}(x)}{k - d_{\mathrm{c}}(x)} + \frac{d_{\mathrm{c}}(y)}{k - d_{\mathrm{c}}(y)}\right) \\
&\overset{(2)}{\geq} \frac{1}{2}\left(\frac{d_{\mathrm{c}}(x)}{k - d_{\mathrm{c}}(x)} + \frac{k - d_{\mathrm{c}}(x)}{d_{\mathrm{c}}(x)}\right) \\
&\geq 1.
\end{aligned}$$

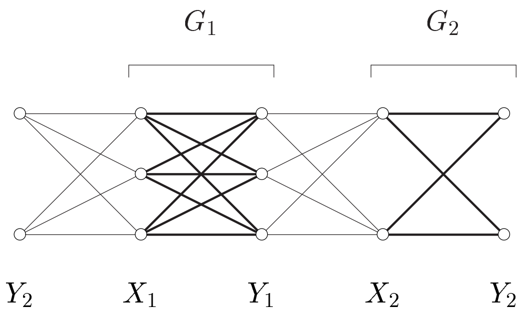


Figure 5.1: The graph G_{NF} when $k = 5$. Note that the two leftmost vertices are the same as the two rightmost vertices.

The last inequality holds, since $x + \frac{1}{x} \geq 2$ for any $x > 0$.

Note that the fairness property is only used to conclude (2). Thus, the performance guarantee is valid for the larger class of algorithms that never reject an edge e, unless it has already colored k edges adjacent to e.

5.5.2 Next-Fit is Worst Possible

When k is even, the competitive ratio of Next-Fit exactly matches the lower bound for fair algorithms. When k is odd, it almost matches the lower bound. This is proven by the following adversary strategy.

The adversary starts out giving the edges of two complete bipartite graphs, $G_1 = (X_1 \cup Y_1, E_1)$ with $|X_1| = |Y_1| = \lceil \frac{k}{2} \rceil$, and $G_2 = (X_2 \cup Y_2, E_2)$ with $|X_2| = |Y_2| = \lfloor \frac{k}{2} \rfloor$.

Consider a coloring where G_1 is colored with $C_{1,\lceil k/2 \rceil}$ and G_2 is colored with $C_{\lceil k/2 \rceil+1,k}$. Each color in $C_{1,\lceil k/2 \rceil}$ is represented at each vertex in G_1 and each color in $C_{\lceil k/2 \rceil+1,k}$ is represented at each vertex in G_2. By Claim 5.1, this coloring can be obtained by Next-Fit.

Next, each vertex in Y_1 is connected to each vertex in X_2 and each vertex in Y_2 is connected to each vertex in X_1, thus creating a "cycle" of complete bipartite graphs, where each bipartite graph shares its left vertices with its left neighbor and its right vertices with its right neighbor. The resulting graph G_{NF} is depicted in Figure 5.1. The new edges between G_1 and G_2 are called E_{12}. After coloring E_1 with $C_{1,\lceil k/2 \rceil}$ and E_2 with $C_{\lceil k/2 \rceil+1,k}$, Next-Fit cannot color any of the edges in E_{12}.

The whole graph is k-regular and bipartite ($X_1 \cup X_2$ forming one set and $Y_1 \cup Y_2$ forming the other). Thus, by König's Theorem, it can be k-colored. Hence, the competitive ratio of Next-Fit on k-colorable graphs is at most

$$\frac{|E_1| + |E_2|}{|E_1| + |E_2| + |E_{12}|} = \frac{\lceil \frac{k}{2} \rceil^2 + \lfloor \frac{k}{2} \rfloor^2}{\lceil \frac{k}{2} \rceil^2 + \lfloor \frac{k}{2} \rfloor^2 + 2\lceil \frac{k}{2} \rceil \lfloor \frac{k}{2} \rfloor} ,$$

which reduces to $\frac{1}{2}$ when k is even, and to $\frac{1}{2} + \frac{1}{2k^2}$ when k is odd.

5.5.3 First-Fit is a Little Better

For any k-colorable graph, let E be the edge set. For any $c \in C_{1,k}$, let E_c denote the set of edges that First-Fit colors with the color c, and let $E_{1,c} = \cup_{i=1}^{c} E_i$. We will prove by induction on c that, for all $c \in C_{1,k}$, $|E_{1,c}| \geq \frac{c}{2k-1}|E|$. Letting $c = k$, this proves the lower bound on the competitive ratio of First-Fit on k-colorable graphs. We only need the following three simple observations.

(1) By the definition of First-Fit, any edge in E_c is adjacent to at least one edge in E_i, $i = 1, \dots, c-1$.

(2) By the definition of First-Fit, any uncolored edge is adjacent to at least one edge of each color.

(3) Since the graph is k-colorable, each vertex has degree at most k. Thus, any edge is adjacent to at most $2(k-1)$ other edges.

For the base case, consider $c = 1$. By (1) and (2), each edge in $E \setminus E_1$ is adjacent to at least one edge in E_1. Thus, by (3), $|E| \leq 2(k-1)|E_1| + |E_1|$, which is equivalent to $|E_1| \geq \frac{1}{2k-1}|E|$.

For the induction step, let $c \in C_{2,k}$. By (1), each edge in E_c is adjacent to at least $c-1$ edges in $E_{1,c-1}$. Thus, each edge in E_c is adjacent to at most $2(k-1)-(c-1) = 2k-c-1$ edges in $E \setminus E_{1,c}$. On the other hand, by (1) and (2), each edge in $E \setminus E_{1,c}$ is adjacent to at least one edge in E_c. Therefore, $|E \setminus E_{1,c-1}| \leq (2k-c-1)|E_c| + |E_c|$, or $|E_c| \geq \frac{1}{2k-c}|E \setminus E_{1,c-1}|$. Thus,

$$\begin{aligned}
|E_{1,c}| &= |E_{1,c-1}| + |E_c| \;\geq\; |E_{1,c-1}| + \frac{|E| - |E_{1,c-1}|}{2k-c} \;=\; \frac{|E| + (2k-c-1)|E_{1,c-1}|}{2k-c} \\
&\geq \frac{|E| + (2k-c-1)\frac{c-1}{2k-1}|E|}{2k-c}, \text{ by the induction hypothesis} \\
&= \frac{|E| - \frac{c-1}{2k-1}|E|}{2k-c} + \frac{c-1}{2k-1}|E| \;=\; \frac{(2k-1)-(c-1)}{(2k-1)(2k-c)}|E| + \frac{c-1}{2k-1}|E| \\
&= \frac{c}{2k-1}|E|.
\end{aligned}$$

To prove that the lower bound is tight, we will construct a graph for which the analysis leading to the bound is tight, i.e., we will construct a graph with the following properties.

(1) Each edge in E_c is adjacent to *exactly* one edge in E_i, $i = 1, \dots, c-1$.

(2) Each uncolored edge is adjacent to *exactly* one edge of each color.

(3) Each vertex has degree k. Thus, each edge is adjacent to *exactly* $2(k-1)$ other edges.

More precisely, we will construct a bipartite k-regular graph G_{FF}, where each edge is adjacent to exactly one edge of each color. Since the graph is bipartite and k-regular, it is k-colorable.

The building blocks of G_{FF} are $\lceil k/2 \rceil$ bipartite biregular graphs $G_1, \dots, G_{\lceil k/2 \rceil}$. Each graph G_i has vertex partition (X_i, Y_i). X_i contains one vertex for each subset of $C_{1,k}$ of size $k+1-i$, and Y_i contains one vertex for each subset of $C_{1,k}$ of size i. The subset of $C_{1,k}$ associated with a vertex x is denoted $C(x)$.

For each vertex $x \in X_i$, there are exactly $k+1-i$ vertices $y \in Y_i$ such that $|C(x) \cap C(y)| = 1$. Similarly, for each vertex $y \in Y_i$, there are exactly i vertices $x \in X_i$ such that $|C(x) \cap C(y)| = 1$. Each vertex $x \in X_i$ is connected to the $k+1-i$ vertices $y \in Y_i$, for which $|C(x) \cap C(y)| = 1$. Thus, each vertex in X_i has degree $k+1-i$ and each vertex in Y_i has degree i. Figure 5.2 shows the graphs G_1 and G_2 when $k = 4$.

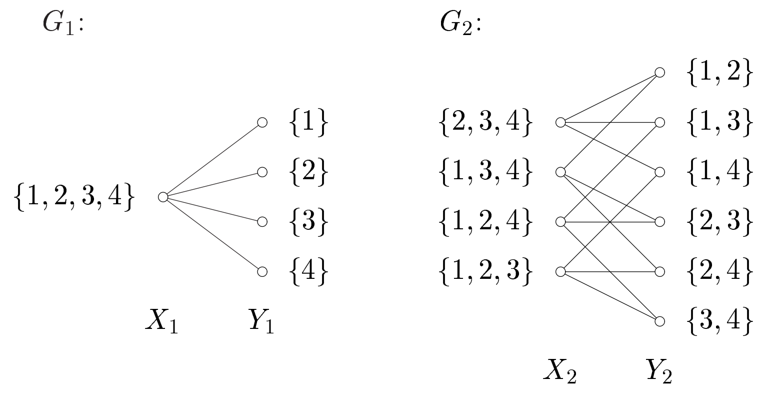


Figure 5.2: The graphs G_1 and G_2 when $k = 4$. Next to each vertex v the color set $C(v)$ is shown.

Consider the coloring of the graphs $G_1, \dots, G_{\lceil k/2 \rceil}$ in which each edge (x, y) is colored with the color in $C(x) \cap C(y)$. An edge (x, y) with the color c is adjacent to one edge of each color in $C(x) \setminus \{c\}$ through the vertex x and one edge of each color in $C(y) \setminus \{c\}$ through the vertex y. Thus, each edge with color c is adjacent to exactly one edge of each color in $C_{1,k} \setminus \{c\}$. Hence, for each G_i, this coloring results if First-Fit is given the edges in order of non-decreasing number.

For each i, $1 \leq i \leq \lceil k/2 \rceil$, the adversary constructs a bipartite graph G_i^{L} consisting of a number of copies of G_i. Let n_i be the number of copies of G_i in G_i^{L}. Then, $n_1 = 1$, and $n_{i+1} = \frac{k-i}{i} n_i$, for $1 \leq i \leq \lceil k/2 \rceil - 1$. For $1 \leq i \leq \lfloor k/2 \rfloor$, the adversary also constructs a graph G_i^{R} isomorphic to G_i^{L}.

Note that, for each pair of vertices $y \in Y_i$ and $x \in X_{i+1}$, $|C(y)| + |C(x)| = k$. For each $y \in Y_i$, there is exactly one vertex $x \in X_{i+1}$ such that $C(x) \cup C(y) = C_{1,k}$. After giving the edges of the graphs $G_1^{\mathrm{L}}, \dots, G_{\lceil k/2 \rceil}^{\mathrm{L}}$ and $G_1^{\mathrm{R}}, \dots, G_{\lfloor k/2 \rfloor}^{\mathrm{R}}$, the adversary connects the k graphs in the following way. Each vertex $y \in Y_i^{\mathrm{L}}$ is connected to $k - i$ vertices $x \in X_{i+1}$, for which $C(y) \cup C(x) = C_{1,k}$. Since $n_{i+1}/n_i = \frac{k-i}{i}$ and $|X_{i+1}|/|Y_i| = \binom{k}{k-i} / \binom{k}{i} = \binom{k}{i} / \binom{k}{i} = 1$, this can be done such that each vertex in X_{i+1}^{L} is connected to i vertices in Y_i^{L}. In this way, each vertex in $X_1^{\mathrm{L}}, \dots, X_{\lceil k/2 \rceil}^{\mathrm{L}}$ and $Y_1^{\mathrm{L}}, \dots, Y_{\lceil k/2 \rceil - 1}^{\mathrm{L}}$ ends up having degree k. The vertices of $G_1^{\mathrm{R}}, \dots, G_{\lfloor k/2 \rfloor}^{\mathrm{R}}$ are connected the same way.

Finally, each vertex in $y^{\mathrm{L}} \in Y_{\lceil k/2 \rceil}^{\mathrm{L}}$ is connected to $\lfloor k/2 \rfloor$ vertices in $y^{\mathrm{L}} \in Y_{\lfloor k/2 \rfloor}^{\mathrm{R}}$, for which $C(y^{\mathrm{L}}) \cup C(y^{\mathrm{R}}) = C_{1,k}$. This is done in a way such that each vertex in $Y_{\lfloor k/2 \rfloor}^{\mathrm{R}}$ is connected to exactly $\lceil k/2 \rceil$ vertices in $Y_{\lceil k/2 \rceil}^{\mathrm{L}}$. If k is even, this is clearly possible, since $|Y_{k/2}^{\mathrm{L}}| = |Y_{k/2}^{\mathrm{R}}|$. If k is odd, it is also possible, since $|Y_{\lceil k/2 \rceil}^{\mathrm{L}}| / |Y_{\lfloor k/2 \rfloor}^{\mathrm{R}}| = \frac{n_{\lceil k/2 \rceil}}{n_{\lfloor k/2 \rfloor}} = \frac{k - \lfloor k/2 \rfloor}{\lfloor k/2 \rfloor} = \frac{\lceil k/2 \rceil}{\lfloor k/2 \rfloor}$. The resulting graph for $k = 4$ is shown in Figure 5.3.

Each of the new edges is adjacent to exactly one edge of each color. Hence, none of these edges are colored by First-Fit.

5.5.4 An Impossibility Result for Deterministic Algorithms

No deterministic algorithm has a competitive ratio of strictly more than $\frac{2}{3}$. To see this, consider the following adversary strategy. The adversary starts out giving the edges of a large $\lceil \frac{k}{2} \rceil$-regular bipartite graph $G = (X \cup Y, E)$. Since the on-line algorithm is deterministic, the

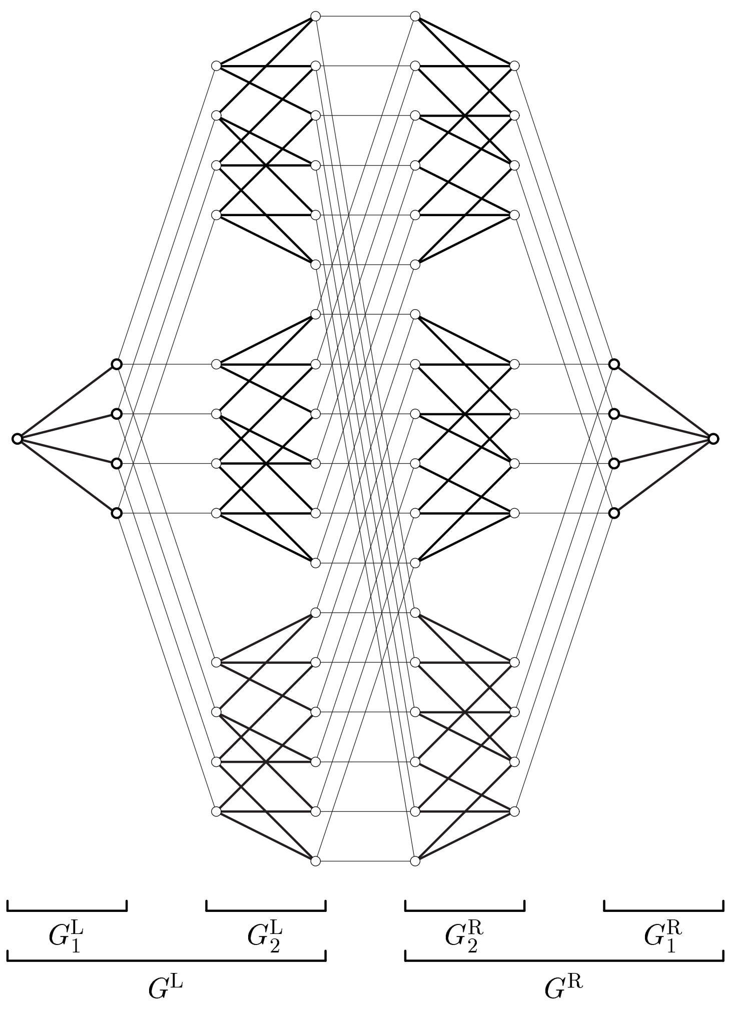


Figure 5.3: The graph G_{FF} when $k = 4$

adversary knows the set of colors represented at each vertex after giving all edges of G. Since *on-line*^{D} uses at most k colors and each vertex has degree $\lceil \frac{k}{2} \rceil$, there are at most $\sum_{i=0}^{\lceil k/2 \rceil} \binom{k}{i}$ different color sets.

For each color set C, the adversary partitions the vertices in X with color set C in sets, such that at most one set has less than k vertices and the rest have exactly k vertices each. The same is done to the vertices in Y. For each color set, at most $2(k-1)$ vertices are not in a set of size k. Thus, if the number of vertices in G is much larger than $2(k-1)$ times the number of color sets, we can ignore these vertices. For each set V of size k, the adversary adds a set U of $\lfloor \frac{k}{2} \rfloor$ new vertices to the graph and connects each vertex in V to each vertex in U. See Figure 5.4. The resulting graph is called G_{Det}. Note that G_{Det} is bipartite and has maximum degree k, and thus is k-colorable.

Let d denote the number of colors represented at each vertex in V, and recall that $d \leq \lceil \frac{k}{2} \rceil$. At most $k - d$ edges incident to each vertex in U can be colored. Hence, the total colored degree of vertices in $U \cup V$ is at most $kd + 2 \cdot \lfloor \frac{k}{2} \rfloor (k - d)$, which reduces to k^2, if k is even, and

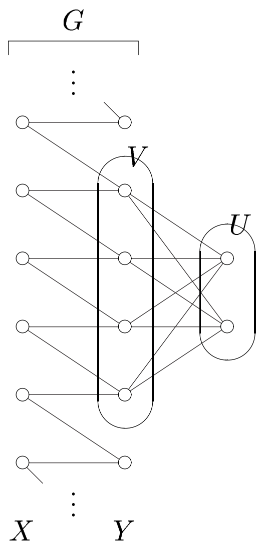


Figure 5.4: A part of the graph G_{Det} when $k = 4$.

to $k^2 - k + d \leq k^2 - k + \lceil \frac{k}{2} \rceil = k^2 - \frac{1}{2}k + \frac{1}{2}$, if k is odd. The total degree of vertices in $U \cup V$ is $k^2 + \lfloor \frac{k}{2} \rfloor k$, which reduces to $\frac{3}{2}k^2$, if k is even, and to $\frac{3}{2}k^2 - \frac{1}{2}k$, if k is odd. Summing the degrees of all vertices in a graph, we get two times the number of edges. Thus, the competitive ratio of *on-line*^{D} is at most

$$C_{\textit{on-line}^D}(k) \leq \begin{cases} \dfrac{k^2}{\frac{3}{2}k^2} = \dfrac{2}{3}, & \text{if } k \text{ is even} \\ \dfrac{k^2 - \frac{1}{2}k - \frac{1}{2}}{\frac{3}{2}k^2 - \frac{1}{2}k} = \dfrac{2}{3} - \dfrac{k-3}{9k^2 - 3k} \leq \dfrac{2}{3}, & \text{if } k \geq 3 \text{ and odd.} \end{cases}$$

5.6 General Graphs

Now we turn to general graphs. That is, there may be some edges that are not colored by *off-line*. We need to distinguish edges that are colored by the on-line algorithm only and edges colored by both the on-line algorithm and *off-line*. Thus, let $d_{\mathrm{d}}(x)$ denote the number of edges incident to x that are "double-colored", i.e., colored by both the on-line algorithm and *off-line*. As before, $d_{\mathrm{c}}(x)$ denotes the number of edges colored by the on-line algorithm and $d_{\mathrm{u}}(x)$ denotes the number of edges colored by *off-line* only. We will not need to consider edges colored by neither algorithm. Note that the double-colored edges are a subset of the colored edges.

5.6.1 A Performance Guarantee for Fair Algorithms

The performance guarantee for fair algorithms is only a little worse than in the special case of k-colorable input graphs. As in the case of k-colorable graphs we need only two simple observations.

(1) For any vertex x, $d_{\mathrm{d}}(x) + d_{\mathrm{u}}(x) \leq k$, since *off-line* colors at most k edges incident to x.

(2) For any uncolored edge (x, y), $d_{\mathrm{c}}(x) + d_{\mathrm{c}}(y) \geq k$, since the algorithm is fair and the edge was not colored.

Observation (2) is the same as in the proof for k-colorable graphs, and (1) is analagous to Observation (1) in the proof for k-colorable graphs.

Our goal is to find a C such that any fair algorithm is C-competitive. Since Next-Fit has a competitive ratio of $\frac{1}{2}$, even in the special case of k-colorable graphs, we know that $0 \leq C \leq \frac{1}{2}$.

As in the proof for k-colorable graphs, we start out putting one unit of some value on each edge colored by *fair*^{R}. If the total value put on colored edges is enough to "pay" the fraction C of a unit for each edge colored by *off-line*, the number of edges colored by *fair*^{R} is at least the fraction C of the number of edges colored by *off-line*. We start out by paying C for each edge colored by both *fair*^{R} and *off-line*. This is done by removing the fraction C of a unit from each of these edges. The remaining value on the colored edges must be distributed to the edges colored by *off-line* only, such that each of these edges receives at least the fraction C of a unit. As for k-colorable graphs, the value on each colored edge is split equally between its endpoints, and each vertex splits its value equally among the uncolored edges incident to it. In this way, each uncolored edge (x, y) receives the value

$$\frac{1}{2}\left(\frac{d_{\mathrm{c}}(x) - Cd_{\mathrm{d}}(x)}{d_{\mathrm{u}}(x)} + \frac{d_{\mathrm{c}}(y) - Cd_{\mathrm{d}}(y)}{d_{\mathrm{u}}(y)}\right) \overset{(1)}{\geq} \frac{1}{2}\left(\frac{d_{\mathrm{c}}(x) - Cd_{\mathrm{d}}(x)}{k - d_{\mathrm{d}}(x)} + \frac{d_{\mathrm{c}}(y) - Cd_{\mathrm{d}}(y)}{k - d_{\mathrm{d}}(y)}\right).$$

By (2), it can be assumed without loss of generality that $d_{\mathrm{c}}(y) \geq \frac{k}{2}$. Thus, $d_{\mathrm{c}}(y) \geq Ck$, and the term $\frac{d_{\mathrm{c}}(y) - Cd_{\mathrm{d}}(y)}{k - d_{\mathrm{d}}(y)}$ is minimized when $d_{\mathrm{d}}(y)$ is minimized, i.e., when $d_{\mathrm{d}}(y) = 0$.

Similarly, if $d_{\mathrm{c}}(x) \geq Ck$, $\frac{d_{\mathrm{c}}(x) - Cd_{\mathrm{d}}(x)}{k - d_{\mathrm{d}}(x)}$ is maximized when $d_{\mathrm{d}}(x) = 0$. If $d_{\mathrm{c}}(x) \leq Ck$, $\frac{d_{\mathrm{c}}(x) - Cd_{\mathrm{d}}(x)}{k - d_{\mathrm{d}}(x)}$ is maximized when $d_{\mathrm{d}}(x)$ is maximized, i.e., when $d_{\mathrm{d}}(x) = d_{\mathrm{c}}(x)$.

Thus, if $d_{\mathrm{c}}(x) \geq Ck$, the uncolored edge (x, y) receives at least

$$\frac{1}{2}\left(\frac{d_{\mathrm{c}}(x)}{k} + \frac{d_{\mathrm{c}}(y)}{k}\right) \overset{(2)}{\geq} \frac{1}{2} \geq C.$$

If $d_{\mathrm{c}}(x) < Ck$, (x, y) receives at least

$$\frac{1}{2}\left(\frac{d_{\mathrm{c}}(x) - Cd_{\mathrm{c}}(x)}{k - d_{\mathrm{c}}(x)} + \frac{d_{\mathrm{c}}(y)}{k}\right) \overset{(2)}{\geq} \frac{1}{2}\left(\frac{(1 - C)d_{\mathrm{c}}(x)}{k - d_{\mathrm{c}}(x)} + \frac{k - d_{\mathrm{c}}(x)}{k}\right),$$

which is greater than or equal to C as long as

$$C \leq \frac{k^2 + (d_{\mathrm{c}}(x))^2 - kd_{\mathrm{c}}(x)}{2k^2 - kd_{\mathrm{c}}(x)} \quad .$$

Hence,

$$C_{fair^R} \geq \min_{d \in C_{1,k}} \left\{\frac{k^2 + d^2 - kd}{2k^2 - kd}\right\} \geq \min_{d \in [1;k]} \left\{\frac{k^2 + d^2 - kd}{2k^2 - kd}\right\} = 2\sqrt{3} - 3 \approx 0.4641.$$

The minimum value of $2\sqrt{3} - 3$ is obtained when $d = (2 - \sqrt{3})k \approx 0.27k$.

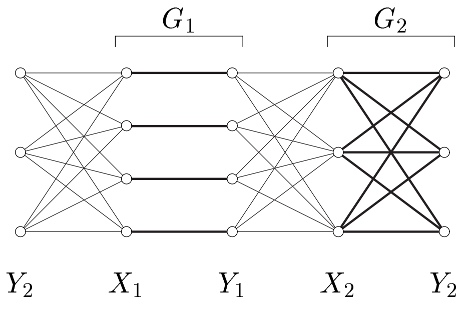


Figure 5.5: The graph G_{NF} when $k = 4$ and $d = 1$, showing that $C_{\mathrm{NF}}(4) \leq \frac{13}{28} \approx 0.4643$.

5.6.2 Next-Fit is Worst Possible

To show that the performance guarantee of the previous section is tight, we describe a family of graphs, for which Next-Fit colors exactly the fraction $\min_{d \in C_{1,k}}\{\frac{k^2+d^2-kd}{2k^2-kd}\}$ of the edges. For each k, the adversary chooses a d close to $(2-\sqrt{3})k$ and constructs a graph G_{NF}, where

(1) For any vertex x, $d_{\mathrm{d}}(x) + d_{\mathrm{u}}(x) = k$.

(2) For any uncolored edge (x, y), $d_{\mathrm{c}}(x) = d$ and $d_{\mathrm{c}}(y) = k - d$.

Consider the two bipartite graphs $G_1 = (X_1 \cup Y_1, E_1)$ and $G_2 = (X_2 \cup Y_2, E_2)$. G_1 is d-regular and has $|X_1| = |Y_1| = k$, and G_2 is complete and has $|X_2| = |Y_2| = k - d$. See Figure 5.5.

The adversary uses k copies of each graph, $G_1^1, \ldots, G_1^k$ and $G_2^1, \ldots, G_2^k$. Consider the coloring where G_1^1 is colored with $C_{1,d}$ and G_2^1 is colored with $C_{d+1,k}$. The coloring of G_1^{i+1} and G_2^{i+1} is obtained from the coloring of G_1^i and G_2^i by shifting the colors once. In this way, each color is used on the same number of edges. Hence, by Claim 5.1, the coloring can be obtained by Next-Fit.

Now, for each i, $1 \leq i \leq k$, each vertex in Y_1^i is connected to each vertex in X_2^i, and each vertex in Y_2^i is connected to each vertex in X_1^i. These new edges are called E_{12}. Next-Fit cannot color any of these edges. However, *off-line* colors all edges of E_1 and E_{12}. Hence, the competitive ratio of Next-Fit is at most

$$\frac{|E_1| + |E_2|}{|E_1| + |E_{12}|} = \frac{kd + (k-d)^2}{kd + 2k(k-d)} = \frac{k^2 - kd + d^2}{2k^2 - kd} \quad .$$

Considering arbitrarily large values of k, this ratio can be arbitrarily close to $2\sqrt{3} - 3$.

5.6.3 First-Fit is Not Much Better

The adversary graph G_{FF} showing that the competitive ratio of First-Fit is at most $\frac{2}{9}(\sqrt{10} - 1) \approx 0.4805$ is inspired by the adversary graph G_{NF} of the previous section. However, there is no ordering of the edges in E_1 and E_2 for which First-Fit will color G_2 with $C_{\lceil \frac{k}{2} \rceil + 1, k}$, if the edges in E_1 and E_2 are given before the edges in E_{12}. Therefore, the graph G_{NF} is extended to contain an extra copy of G_2, G_2'. Each vertex in Y_2 is connected to exactly d vertices in X_2' and vice versa. Now, E_2 denotes the edges in G_2 and G_2' and the edges connecting them. Finally, $2k(k-d)$ new vertices are added, and each vertex in $Y_2 \cup X_2'$ is connected to k of these vertices. Let E_3 denote the set of these extra edges. The graph G_{FF} for $k = 4$ is depicted in Fig. 5.6.

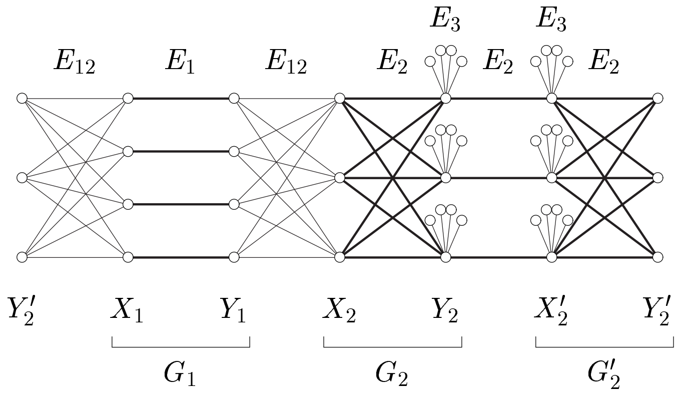


Figure 5.6: The graph G_{FF} when $k = 4$, showing that $C_{\mathrm{FF}}(4) \leq \frac{25}{52} \approx 0.4808$.

If the edges in G_1 and the edges between Y_2 and X_2' are given first (one perfect matching at a time), followed by the edges in G_2 and G_2' (one perfect matching at a time), First-Fit will color E_1 and the edges between Y_2 and X_2' with $C_{1,d}$ and the remaining edges in E_2 with $C_{d+1,k}$. After this, First-Fit will not be able to color any more edges of G_{FF}. On the other hand, it is possible to k-color the set $E_1 \cup E_{12} \cup E_3$ of edges. Thus, the competitive ratio of First-Fit can be no more than

$$\frac{|E_1| + |E_2|}{|E_1| + |E_{12}| + |E_3|} = \frac{kd + 2(k-d)^2 + (k-d)d}{kd + 2k(k-d) + 2k(k-d)} = \frac{2k^2 - 2kd + d^2}{4k^2 - 3kd}.$$

This ratio attains its minimum value of $\frac{2}{9}(\sqrt{10} - 1) \approx 0.4805$, when $d = \frac{1}{3}(4 - \sqrt{10})k$. Thus, for the graph G_{FF}, we choose d to be an integer close to $\frac{1}{3}(\sqrt{10} - 1)k$, and by allowing arbitrarily large values of k, the ratio can be arbitrarily close to $\frac{2}{9}(\sqrt{10} - 1)$.

5.6.4 An Impossibility Result for Fair Deterministic Algorithms

The adversary constructs a simple graph $G = (V_1 \cup V_2, E)$ in two phases. In Phase 1, only vertices in V_1 are connected. In Phase 2, vertices in V_2 are connected to vertices in V_1. Let $|V_1| = |V_2| = n$ for some large integer n.

In Phase 1, the adversary gives an edge between two unconnected vertices $x, y \in V_1$ with a common unused color. Since the edge can be colored, *fair*^{D} will do so. This process is repeated until no two unconnected vertices with a common unused color can be found. At that point Phase 1 ends.

For any vertex x, let $\overline{C}(x)$ denote the set of colors not represented at x. At the end of Phase 1, the following holds true. For each color c and each vertex x such that $c \in \overline{C}(x)$, x is connected to all other vertices y with $c \in \overline{C}(y)$. Since $c \in \overline{C}(x)$, x is connected to at most $k-1$ other vertices. Thus, each of the k colors are missing at at most k vertices: $\sum_{x \in V_1} \overline{C}(x) \leq k^2$.

The edges given in Phase 2 are the edges of a k-regular bipartite graph with V_1 and V_2 forming the two independent sets. Note that, by König's Theorem, such a graph can be k-colored.

In Phase 2, *fair*^{D} colors at most k^2 edges, but *off-line* rejects all edges from Phase 1 and colors all edges from Phase 2, giving a performance ratio of at most

$$\frac{\frac{1}{2}(nk - k^2) + k^2}{nk} = \frac{nk + k^2}{2nk} = \frac{1}{2} + \frac{k}{2n} \ .$$

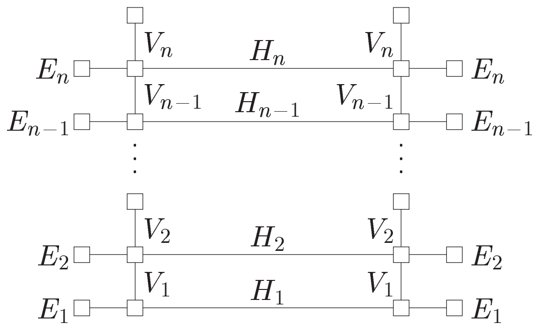


Figure 5.7: Structure of the adversary graph for the general impossibility result.

This shows that, for any constant $\epsilon > 0$, $C_{fair^D} < \frac{1}{2} + \epsilon$.

Note that the adversary graph can easily be modified to be a bipartite graph. Simply replace the vertex set V_1 by two sets X_1 and Y_1, and let the edges of Phase 1 connect vertices in X_1 to vertices in Y_1. At the end of Phase 1, each color is missing at at most $2k-2$ vertices, because, if a color is missing at a vertex in X_1, then it can be missing at at most $k-1$ vertices in Y_1 and vice versa. The vertices of Phase 2 should also be partitioned in two sets X_2 and Y_2. If, for instance, vertices in X_2 are only connected to vertices in X_1, and vertices in Y_2 are only connected to vertices of Y_1, the resulting graph is bipartite.

5.6.5 A General Impossibility Result

We close the chapter with an upper bound of $\frac{4}{7}$ on the competitive ratio of any on-line algorithm for edge coloring. The structure of the adversary graph is depicted in Figure 5.7. If we allow multigraphs, we can think of each box as a single vertex and each line as k parallel edges. Otherwise, we can think of each box as k vertices, and a line between two boxes means that each vertex inside one box is connected to each vertex inside the other box, thus forming a complete bipartite graph. Thus, in this case, each line corresponds to k^2 edges. To make the proof as general as possible we will describe the case of a simple graph.

The edges of the graph are divided into n levels, Level $1, \ldots, n$. The adversary gives the edges, one level at a time, according to the numbering of the levels. Depending on the actions of the on-line algorithm, the adversary might not give all levels of the graph. The edges of Level i are given in three consecutive phases:

1. H_i: Internal (horizontal) edges at Level i. In total k^2 edges.
2. V_i: Internal (vertical) edges between Level i and Level $i+1$. In total $2k^2$ edges.
3. E_i: External edges at Level i. In total $2k^2$ edges.

Vertices that are endpoints of internal edges are called internal vertices.

Note that each internal edge contributes to the degree of two internal vertices, whereas an external edge contributes to the degree of one internal vertex and one external vertex. Since external vertices are no problem — they have a degree of only k — it seems to be better to color external edges than internal edges. In particular, an optimal off-line algorithm colors all external edges and no internal edges. We show that no algorithm that colors at most $\frac{1}{7}$ of the external edges can be better than $\frac{4}{7}$-competitive. However, we also show that no on-line algorithm that colors more than $\frac{1}{7}$ of the external edges can be $\frac{4}{7}$-competitive.

Since the on-line algorithm may be randomized we use random variables to count the number of colored edges. Let X_{H_i} be a random variable counting how many edges *on-line*^{R} will color from the set H_i, and let X_{V_i} and X_{E_i} count the colored edges from V_i and E_i respectively. For $i = 0, \dots, n$, let EXT_i and INT_i be random variables counting the sum of all external and internal edges, respectively, colored by *on-line*^{R} after Level i is given, i.e., $\mathrm{EXT}_i = \sum_{j=1}^{i} X_{\mathrm{E}_j}$ and $\mathrm{INT}_i = \sum_{j=1}^{i}(X_{\mathrm{V}_j} + X_{\mathrm{H}_j})$. Note that $\mathrm{EXT}_0 = \mathrm{INT}_0 = 0$.

Since no algorithm can color more than k edges incident to one vertex, the total colored degree of the internal vertices at the first i levels, $1 \le i \le n$, is at most $2k^2 i$. Each internal edge (excluding V_i) contributes two to this number, and each external edge (including edges in V_i) contributes only one. Thus, the expected number of colored edges on the first i levels is

$$\begin{aligned}
E[\mathrm{INT}_i] + E[\mathrm{EXT}_i] &= (E[\mathrm{INT}_i] - E[X_{\mathrm{V}_i}]) + (E[\mathrm{EXT}_i] + E[X_{\mathrm{V}_i}]) \\
&\le \frac{1}{2}(2k^2 i - E[\mathrm{EXT}_i] - E[X_{\mathrm{V}_i}]) + (E[\mathrm{EXT}_i] + E[X_{\mathrm{V}_i}]) \\
&= k^2 i + \frac{1}{2}(E[\mathrm{EXT}_i] + E[X_{\mathrm{V}_i}]). \qquad (5.1)
\end{aligned}$$

If $E[X_{\mathrm{E}_i}] \le \frac{2}{7}k^2$, for all levels i, $1 \le i \le n$, then $E[\mathrm{EXT}_n] \le \frac{2}{7}k^2 n$. Thus, by (5.1), the expected total number of edges colored by *on-line*^{R} is

$$\begin{aligned}
E[\mathrm{INT}_n] + E[\mathrm{EXT}_n] &\le k^2 n + \frac{1}{2}(E[\mathrm{EXT}_n] + E[X_{\mathrm{V}_n}]) \\
&= k^2 n + \frac{1}{2}E[\mathrm{EXT}_{n-1}] + \frac{1}{2}(E[X_{\mathrm{E}_n}] + E[X_{\mathrm{V}_n}]) \\
&\le k^2 n + \frac{1}{7}k^2(n-1) + \frac{1}{2}2k^2 \\
&= \frac{8}{7}k^2 n + \frac{6}{7}k^2.
\end{aligned}$$

Thus, we get an upper bound on the performance ratio of $\frac{\frac{8}{7}k^2 n + \frac{6}{7}k^2}{2nk^2} = \frac{4}{7} + \frac{3}{7n}$, which can be arbitrarily close to $\frac{4}{7}$, if we allow n to be arbitrarily large.

Otherwise, there exists a level i, $1 \le i \le n$, such that $E[X_{\mathrm{E}_i}] > \frac{2}{7}k^2$. Assume that Level i is the first such level. Thus, $E[\mathrm{EXT}_{i-1}] \le \frac{2}{7}k^2(i-1)$. Furthermore, since the edges in V_{i-1}, H_i, V_i, and E_i all contribute to the degree of the two internal vertices at Level i,

$$E[X_{\mathrm{V}_{i-1}}] + 2E[X_{\mathrm{H}_i}] + E[X_{\mathrm{V}_i}] \le 2k^2 - E[X_{\mathrm{E}_i}] < \frac{12}{7}k^2. \qquad (5.2)$$

If the adversary stops giving edges after Phase 1 of Level i, *off-line* will color $k^2(2i-1)$ edges in total. These are the edges in the sets $\mathrm{E}_1, \mathrm{E}_2, \dots, \mathrm{E}_{i-1}$, and H_i. If the adversary stops giving edges after Phase 2 (or 3) of Level i, *off-line* will color $2k^2 i$ edges. These are the edges in the sets $\mathrm{E}_1, \mathrm{E}_2, \dots, \mathrm{E}_{i-1}$, and V_i. Thus, if the algorithm is $\frac{4}{7}$-competitive, the following two inequalities must hold.

$$E[\mathrm{INT}_{i-1}] + E[\mathrm{EXT}_{i-1}] + E[X_{\mathrm{H}_i}] \ge \frac{4}{7}k^2(2i-1), \text{ and}$$

$$E[\mathrm{INT}_{i-1}] + E[\mathrm{EXT}_{i-1}] + E[X_{\mathrm{H}_i}] + E[X_{\mathrm{V}_i}] \ge \frac{4}{7}k^2 2i.$$

Adding the two inequalities, yields

$$2\big(E[\mathrm{INT}_{i-1}] + E[\mathrm{EXT}_{i-1}]\big) + 2E[X_{\mathrm{H}_i}] + E[X_{\mathrm{V}_i}] \geq \frac{16}{7}k^2 i - \frac{4}{7}k^2.$$

Thus, by (5.1),

$$2k^2(i-1) + E[\mathrm{EXT}_{i-1}] + E[X_{\mathrm{V}_{i-1}}] + 2E[X_{\mathrm{H}_i}] + E[X_{\mathrm{V}_i}] \geq \frac{16}{7}k^2 i - \frac{4}{7}k^2.$$

Now, using (5.2) yields $E[\mathrm{EXT}_{i-1}] > \frac{2}{7}k^2(i-1)$, which is a contradiction. This proves the upper bound of $\frac{4}{7}$.

Chapter 6

Dual Bin Packing in Variable-Sized Bins

In [43] we study a variant of dual bin packing in which the bins may have different sizes. We assume that the input sequences are all accommodating, i.e., for each sequence, all items can be packed in the n available bins by an optimal off-line algorithm. The reason for this restriction is that, for general sequences, no fair on-line algorithm has a constant competitive ratio, even in the case of identical bins [25].

The problem can also be seen as a scheduling problem with n uniformly related machines. Consider a scheduling problem with a deadline and assume that the aim is to schedule as many jobs as possible before this deadline. If an optimal off-line algorithm can schedule all jobs of any input sequence before the deadline, this problem is equivalent to our problem. Our problem can also be seen as a special case of the multiple knapsack problem (see [84, 29]), where all items have unit profit. (This problem was mainly studied in the off-line environment.)

6.1 Algorithms

We study the class of fair algorithms. A *fair* algorithm rejects an item, only if the item does not fit in the empty space left in any bin. Some of the algorithms that are classical for the classical bin packing problem can be adapted to the dual bin packing problem. Such an adaptation was done for identical bins in [25]; the n bins are all considered open from the beginning, and no new bin can be opened. We also use this adaptation.

Some classical fair algorithms are First-Fit, Best-Fit, and Worst-Fit. *First-Fit* is not a single algorithm but a class of algorithms that give an order to the bins. Each item is packed in the first bin (in the ordered set of bins) in which it fits. Among the various versions of First-Fit, two are most natural. *Smallest-Fit* packs each item in the smallest bin it fits in. Similarly, *Largest-Fit* packs each item in the largest bin it fits in. The two other algorithms do not need further adaptation. Thus, *Best-Fit* packs each item in a bin where it leaves the smallest possible empty space, and *Worst-Fit* packs it in a bin where it leaves the largest possible empty space.

We also analyze a class of fair algorithms called *Smallest-Bins-First*. The only thing that characterizes these algorithms — apart from being fair — is that whenever an item is packed in an empty bin, the item fits in no smaller empty bin. Smallest-Fit and Best-Fit belong to this class of algorithms.

6.2 Results

We prove that, on accommodating sequences, the competitive ratio of any fair deterministic algorithm is between $\frac{1}{2}$ and $\frac{2}{3}$. Thus, even though we consider a generalization of dual bin packing in identical bins, the performance guarantee for fair algorithms matches Worst-Fit's performance for identical bins [25]. We give a very simple example showing that both Worst-Fit and Largest-Fit have a competitive ratio of exactly $\frac{1}{2}$ on accommodating sequences.

Smallest-Bins-First algorithms are only a little better; on accommodating sequences, any Smallest-Bins-First algorithm has a competitive ratio of exactly $\frac{n}{2n-1}$, where n is the number of bins. This is in contrast to the case of identical bins, where First-Fit and Best-Fit are $\frac{5}{8}$-competitive.

Finally, any fair randomized algorithm has a competitive ratio of at most $\frac{4}{5}$, even on accommodating sequences.

6.3 A Tight Performance Guarantee

Given any accommodating sequence σ, any fair algorithm A packs at least half of the items in σ. Let A be the set of items accepted by A and let R be the set of items rejected by A. We will prove that $|A| \geq |R|$. The proof is adapted from the proof of a stronger result for identical bins in [25].

Let s be the size of the smallest item in R. From σ we construct a new accommodating sequence σ' in the following way.

- Each item in A of size less than s is removed from σ.
- Each item in A of size $\ell \geq s$ is replaced by $\lfloor \frac{\ell}{s} \rfloor$ items of size s.
- Each item in R of size more than s is replaced by an item of size s.

Clearly, any packing of σ induces a legal packing of σ'. Since all items in σ' have the same size s, packings can only be distinguished by the number of items in each bin. Hence, to calculate an upper bound on $|R|$, we only need to count how many items of size s can be added to the packing of σ' induced by the on-line packing of σ.

Consider the on-line packing of σ. Since A is fair and it rejected an item of size s, the empty space in each bin is less than s. Clearly, removing an item of size less than s increases the empty space in the corresponding bin by less than s. Similarly, replacing an item of size $\ell \geq s$ by $\lfloor \frac{\ell}{s} \rfloor$ items of size s increases the empty space by less than s. Thus, each time an item is removed or replaced, it makes room for at most one extra item of size s. This proves that $|A| \geq |R|$, and hence the algorithm is $\frac{1}{2}$-competitive on accommodating sequences.

The result is tight due to the performance of Worst-Fit and Largest-Fit (see Section 6.5).

6.4 Impossibility Results

The Strict Competitive Ratio

We can easily show that the strict competitive ratio on accommodating sequences is at most $\frac{2}{3}$ for any fair algorithm. Consider for example the following instance with

- 1 bin of size 2

- 1 bin of size 3
- $n-2$ bins of size ε, $0 < \varepsilon < 1$.

The input sequence consists of two or three items that are all too large for the bins of size ε. The first item has size 1.

Assume first that the first item is packed in the bin of size 3. In this case, an item of size 3 arrives next. This item cannot be packed, but clearly both items could be packed, if the first item were packed in the bin of size 2.

If the first item is packed in the bin of size 2, two items of size two will arrive. Only one of these two items can be packed, but the whole sequence could be packed, if the first first item were packed in the bin of size 3.

This gives an upper bound on the strict competitive ratio on accommodating sequences of $\frac{2}{3}$. Furthermore, applying Yao's inequality [102] as described in [18, 65, 66] on these two sequences gives an upper bound of $\frac{4}{5}$ for randomized algorithms. In words Yao's principle says that the competitive ratio of the best randomized algorithm against an oblivious adversary equals the competitive ratio of the best deterministic algorithm on inputs generated from the "worst" probability distribution.

To see that the upper bound of $\frac{4}{5}$ follows from Yao's principle, consider the sequence where the first item of size 1 is followed by one item of size 3 with probability $p_1 = \frac{2}{5}$ and by two items of size 2 with probability $p_2 = \frac{3}{5}$. An algorithm that packs the first item in the bin of size 3 will have an expected performance ratio of at most $p_1 \cdot \frac{1}{2} + p_2 \cdot 1 = \frac{4}{5}$. Similarly, an algorithm that packs the first item in the bin of size 2, will have an expected performance ratio of at most $p_1 \cdot 1 + p_2 \cdot \frac{2}{3} = \frac{4}{5}$. Thus, no deterministic algorithm can have an expected performance ratio larger than $\frac{4}{5}$ on this sequence. This implies an upper bound of $\frac{4}{5}$ on the competitive ratio on accommodating sequences for randomized algorithms.

The Competitive Ratio

We are interested in impossibility results that hold for the competitive ratio in general, and not only for the strict competitive ratio. In Section 6.6, it is shown that any fair algorithm rejects at most $n-1$ items, where n is the number of bins. As long as there is only a constant number of bins, we can view the number of rejected items as just an additive constant, and hence any fair algoirthm has competitive ratio 1. Thus, we need to define arbitrarily long sequences.

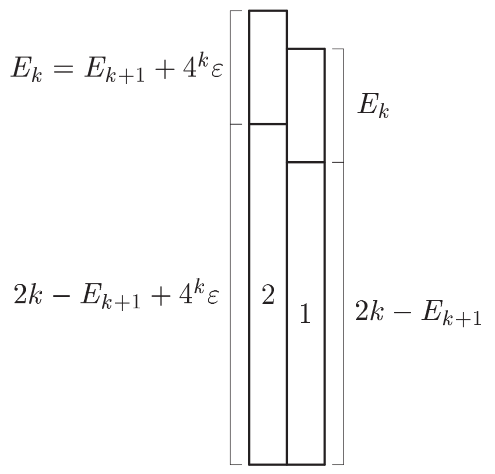
Deterministic Algorithms

We define n bins and an accommodating sequence consisting of $3 \cdot \lfloor \frac{n}{2} \rfloor$ items. Let $\ell = \lfloor \frac{n}{2} \rfloor$. For $k = 1, 2, \ldots, \ell$, we define the pair of bins

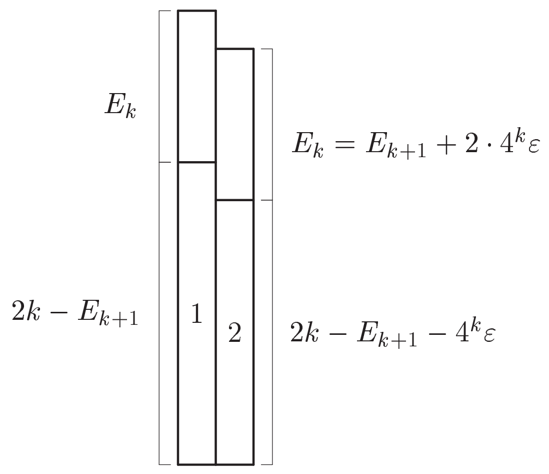
$$B_{2k} \text{ with size } 2k + 2 \cdot 4^k \varepsilon \text{ and } B_{2k-1} \text{ with size } 2k + 4^k \varepsilon,$$

where $\varepsilon \leq \frac{1}{4^n}$ is a positive constant. Thus, $4^\ell \varepsilon \leq 4^{n-1} \varepsilon \leq \frac{1}{4}$. If n is odd, the last bin is of size $\frac{\varepsilon}{2}$ (so that no items are packed in that bin for the sequence we define).

The sequence is defined inductively in Steps $\ell, \ell-1, \ldots, 1$. In Step k, two large items are given and one small item is defined. The small items are all given after Step 1, i.e., after all large items have been given. For each step k, the following will hold.


$E_k = E_{k+1} + 4^k\varepsilon$
E_k
$2k - E_{k+1} + 4^k\varepsilon$
2
1
$2k - E_{k+1}$

(a) If the first large item is put in B_{2k-1}, the next large item has size $2k - E_{k+1} + 4^k\varepsilon$, and $E_k = E_{k+1} + 4^k\varepsilon$.


E_k
$E_k = E_{k+1} + 2 \cdot 4^k\varepsilon$
$2k - E_{k+1}$
1
2
$2k - E_{k+1} - 4^k\varepsilon$

(b) If the first large item is put in B_{2k}, the next large item has size $2k - E_{k+1} - 4^k\varepsilon$, and $E_k = E_{k+1} + 2 \cdot 4^k\varepsilon$.

Figure 6.1: The first large item of Step k has size $2k - E_{k+1}$.

- The on-line algorithm will pack the two large items in B_{2k} and B_{2k-1}, one in each bin.
- After packing the two large items, the empty space in the two bins have the same size denoted E_k. For convenience we define $E_{\ell+1} = 0$.
- The small item will be rejected by the on-line algorithm.

We first present the sequence and then prove that this is indeed the case.

- The first large item given in Step k has size $2k - E_{k+1}$. Thus, the very first item has size 2ℓ and the size of the first large item of each of the following steps depends on the empty space created in the previous step.
- The second large item given in Step k has size $2k - E_{k+1} + 4^k\varepsilon$ or $2k - E_{k+1} - 4^k\varepsilon$ as illustrated in Figure 6.1. Note that $E_k = E_{k+1} + 4^k\varepsilon$ or $E_k = E_{k+1} + 2 \cdot 4^k\varepsilon$.
- The small item defined in Step k has size $S_k = E_k + 4^k\varepsilon$.

Note that if the two large items of Step k are swapped in the on-line packing, the small item fits in B_{2k}. This proves that the sequence is accommodating. Note also that

$$E_{k+1} + 4^k\varepsilon \overset{(1)}{\leq} E_k \overset{(2)}{\leq} E_{k+1} + 2 \cdot 4^k\varepsilon.$$

By (1), $E_{\ell+1} < E_\ell < \ldots < E_1$, and by (2),

$$E_1 \leq E_{\ell+1} + 2\sum_{i=1}^{\ell} 4^i\varepsilon = 0 + 2\frac{4^{\ell+1} - 1}{3}\varepsilon < 4^{\ell+1}\varepsilon \leq 1.$$

This means that both large items given in Step k have a size greater than $2k-1-4^k\varepsilon$. Thus, to prove that none of these two items fit in B_{2k-2}, it suffices to prove $2k-1-4^k\varepsilon \geq 2k-2+2\cdot 4^{k-1}\varepsilon$. This is equivalent to $1 \geq \frac{3}{2}4^k\varepsilon$, which is true since $4^k\varepsilon \leq \frac{1}{4}$.

Finally, by (2), $E_1 < E_k + 4^k\varepsilon = S_k$, $1 \leq k \leq \ell$. Thus, all small items are too large even for the bins B_1 and B_2, and hence they will be rejected.

We conclude that the sequence is accommodating and one out three items is rejected in each step, which proves the bound.

Randomized Algorithms

Since the sequence just described was built step by step depending on the on-line choices, we cannot use it against randomized algorithms. Thus, we describe a simpler sequence proving an upper bound of $\frac{4}{5}$ for randomized algorithms. For simplicity, we describe the proof for deterministic algorithms first. We use

- $\lfloor \frac{n}{2} \rfloor$ bins of size $1+\varepsilon$
- $\lfloor \frac{n}{2} \rfloor$ bins of size $2-\varepsilon$,

where $0 < \varepsilon < \frac{1}{2}$. If n is odd, the last bin has size ε.

The input sequence starts with $\lfloor \frac{n}{2} \rfloor$ items of size 1. Since the algorithm is fair, all $\lfloor \frac{n}{2} \rfloor$ items are accepted. Let x be the number of bins of size $1+\varepsilon$ that receives an item. Since no bin can hold two items, x is the number of empty bins of size $2-\varepsilon$. What happens next depends on the size of x.

If $x \leq \frac{3}{5} \cdot \lfloor \frac{n}{2} \rfloor$, the sequence continues with $\lfloor \frac{n}{2} \rfloor$ items of size $2-\varepsilon$. The on-line algorithm accepts exactly x of these. Clearly, the whole sequence could be packed, but the algorithm packs only the fraction

$$\frac{\lfloor \frac{n}{2} \rfloor + x}{2 \cdot \lfloor \frac{n}{2} \rfloor} \leq \frac{1+\frac{3}{5}}{2} = \frac{4}{5}$$

of the items.

Otherwise, the sequence continues with $\lfloor \frac{n}{2} \rfloor$ items of size $1+\varepsilon$ followed by $\lfloor \frac{n}{2} \rfloor$ items of size $1-\varepsilon$. All items of size $1+\varepsilon$ are accepted. After that all bins contain exactly one item. Items of size $1-\varepsilon$ can only be packed in bins of size $2-\varepsilon$ that contain an item of size 1. Thus, $\lfloor \frac{n}{2} \rfloor - x$ of these items are accepted. Again, the whole sequence could be packed, and the on-line algorithm packs only

$$\frac{3 \cdot \lfloor \frac{n}{2} \rfloor - x}{3 \cdot \lfloor \frac{n}{2} \rfloor} < \frac{3-\frac{3}{5}}{3} = \frac{4}{5}$$

of the items.

Now, to get an upper bound for randomized algorithms, let x denote the expectation of the number of bins of size $1+\varepsilon$ that received an item of size 1. The bound follows by linearity of expectation.

6.5 Worst-Fit and Largest-Fit

In this section we show that, on accommodating sequences, Worst-Fit and Largest-Fit have the worst possible competitive ratio among fair algorithms.

To see this, consider the following set of bins.

- 1 large bin of size n
- $n-1$ small bins of size 1.

The input sequence is given in two steps:

- $n-1$ items of size 1
- $n-1$ items of size $1+\varepsilon$,

where $\varepsilon \leq \frac{1}{n}$ is a positive constant.

Both Worst-Fit and Largest-Fit will pack all items of size 1 in the large bin. After that, all bins have an empty space of size 1, which means that the $n-1$ items of size $1+\varepsilon$ must be rejected. However, the $n-1$ items of size 1 can be packed in the $n-1$ small bins, and the remaining $n-1$ items can be packed in the large bin, since $(n-1)\varepsilon < 1$.

[25] shows that, even in the case of identical bins, the competitive ratio of Worst-Fit is $\frac{1}{2}$.

6.6 Smallest-Bins-First Algorithms

In this section we show that any Smallest-Bins-First algorithm has a competitive ratio of exactly $\frac{n}{2n-1}$ on accommodating sequences.

The Impossibility Result

For the impossibility result, consider the set of n bins b_i, $1 \leq i \leq n$, where b_i has size $1+i\varepsilon$ and $\varepsilon < \frac{1}{n}$ is a positive constant. The sequence is

- 1 item of size $1+(i-1)\varepsilon$, for $i = 1, 2, \ldots, n$
- $n-1$ items of size $\frac{n}{n-1}\varepsilon$.

For each i, $1 \leq i \leq n$, any Smallest-Bins-First algorithm assigns the item of size $1+(i-1)\varepsilon$ to b_i. This leaves an empty space of size ε in each bin. Hence, all items of size $\frac{n}{n-1}\varepsilon$ must be rejected.

An optimal off-line algorithm packs each item of size $1+(i-1)\varepsilon$, $2 \leq i \leq n$, in b_{i-1}. The item of size 1 and the $n-1$ small items can then be packed in b_1.

Thus, the sequence is accommodating, and the algorithms pack only n out of $2n-1$ items.

The Matching Performance Guarantee

For the performance guarantee, we prove an upper bound on the number of rejected items. We use the fact that the total size of the rejected items equals the total empty space in the on-line packing minus the total empty space in an optimal off-line packing, since all items of the sequence can be packed.

For any input sequence, let B be the set of non-empty bins in some optimal off-line packing, let $\overline{B}$ be the set of empty bins, and let $N = |B|$.

If the on-line algorithm does not reject any items, its packing is optimal. Now, assume that at least one item is rejected, and let s be the size of a smallest rejected item. Since the algorithm is fair, the empty space in any bin is less than s. Clearly, the size of a bin is also an upper bound on the empty space in that bin. Thus, the total empty space in the on-line packing is strictly less than $\sum_{b\in B} s + \sum_{b\in \overline{B}} \text{size}(b) = Ns + \sum_{b\in \overline{B}} \text{size}(b)$. Since the total empty space in the off-line packing is at least $\sum_{b\in \overline{B}} \text{size}(b)$, the number of rejected items is strictly less than N, i.e., at most $N-1$. In particular, this means that the number of rejected items is at most $n-1$.

Thus, if there are no empty bins in the on-line packing, the algorithm has packed at least n items and rejected at most $n-1$, yielding a performance ratio of at least $\frac{n}{2n-1}$.

Otherwise, let b be a largest empty bin. Let $I_\leq$ be the set of items no larger than b. Since the algorithm is fair, these items are all accepted. Let $N_\leq$ be the number of non-empty bins no larger than b in some optimal off-line packing. Then, $N_\leq \leq |I_\leq|$, since only the items in $I_\leq$ fit in bins no larger than b.

Let $n_>$ be the number of bins larger than b. These bins are all non-empty in the on-line packing, and by the definition of Smallest-Bins-First algorithms, the first item packed in each of them is larger than b, i.e., not contained in $I_\leq$. Thus, the on-line algorithm accepts at least $|I_\leq|+n_>$ items. Let $N_>$ be the number of non-empty bins larger than b in the optimal off-line packing, and let $N = N_\leq + N_>$ be the total number of non-empty bins in the optimal off-line packing. Then, $|I_\leq| + n_> \geq N_\leq + N_> = N$. Since the number of rejected items is at most $N-1$, this gives a ratio of at least $\frac{N}{2N-1} \geq \frac{n}{2n-1}$.

Chapter 7

Scheduling on Two Related Machines

In [44] and [42] we study scheduling on two uniformly related machines, i.e., one machine is a factor of q faster than the other. Without loss of generality, we assume that the faster machine has speed 1, and the other machine has speed q. Thus, a job of size p can be completed in time p on the fast machine and time qp on the slow machine. We restrict the input sequences to those with *non-increasing job sizes*.

The aim is to minimize the makespan. We determine the optimal competitive ratio as a function of q, $C(q)$. This gives as a by-product the *overall competitive ratio* $\max_{q\geq 1}\{C(q)\}$.

Let M_1 denote the fast machine, and M_q the slow machine. For a given job sequence $J_1, J_2, \ldots, J_\ell$, we let $p_1, p_2, \ldots, p_\ell$ denote the job sizes. The total size of the jobs is denoted by P, i.e., $P = \sum_{i=1}^{\ell} p_i$. The time it takes to complete a job on a given machine is called the *load* of the job on that machine.

For the first k jobs of an input sequence, let OPT_k denote the optimal makespan and let ONL_k denote the makespan of the on-line algorithm under consideration.

7.1 Non-Preemptive Scheduling

Since the analysis of the optimal competitive ratio involves long and tedious proofs, the aim of this section is to give an overview of the analysis and the results. The proofs can be found in the paper in Appendix B.4.

7.1.1 Previous Results

For the off-line problem, the algorithm LPT (Longest Processing Time) has been studied. This algorithm sorts the jobs in non-increasing order and then uses List Scheduling. Since, in this chapter, we assume that the jobs arrive in order of non-increasing size, we obtain the same result using List Scheduling.

For m *identical* machines the competitive ratio of LPT is $\frac{4}{3} - \frac{1}{3m}$ [61]. Thus, on two identical machines, the competitive ratio of LPT is $\frac{7}{6}$, and this is the optimal competitive ratio [93]. These ratios should be compared to $2 - \frac{1}{m}$ and $\frac{3}{2}$ for general sequences. [93] also shows that, for $m = 3$, no deterministic algorithm can have a competitive ratio better than $\frac{1}{6}(1 + \sqrt{37}) \approx 1.18$. Furthermore, the paper gives an $\frac{8}{7}$-competitive randomized algorithm for $m = 2$ and shows that this is best possible.

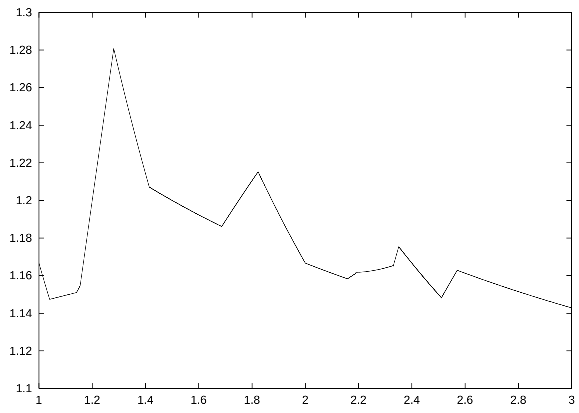


Figure 7.1: The competitive ratio as a function of q

For m *related* machines, the overall competitive ratio of LPT is between 1.52 and $\frac{5}{3}$ [56]. Recall that for general sequences, the competitive ratio is at least 1.853, if $m \geq 80$. In [39], the upper bound is improved to $\frac{19}{12} \approx 1.583$ (unfortunately, the proof does not seem complete). On two related machines, the overall competitive ratio of LPT is at most $\frac{1}{4}(1+\sqrt{17}) \approx 1.28$ [58]. Recall that, for general sequences, the ratio is $\phi \approx 1.618$. In [86], the competitive ratio of LPT for any speed ratio is given. The interval $q \geq 1$ is partitioned in 9 intervals, each with a different function of q for the competitive ratio.

7.1.2 Our Results

We give the optimal competitive ratio as a function of the speed ratio q (see Figure 7.1). The function involves 15 distinct intervals as defined below. In some of those intervals, we give general lower bounds which match the upper bounds in [86]. In those cases, LPT is optimal. In the other intervals, we design new algorithms and prove that they are optimal. Except for the first few jobs, the algorithms all work like LPT.

We show that, in terms of overall competitive ratio, $\frac{1}{4}(1+\sqrt{17})$ is the optimal competitive ratio achieved at $q = \frac{1}{4}(1+\sqrt{17})$ by LPT. Thus, in terms of overall competitive ratio, LPT is optimal, and as in the case of general input sequences, the highest competitive ratio equals the value of q for which it is attained.

The optimal competitive ratio is described by the following function.

$$C(q) = \begin{cases} C_1(q), & 1 \leq q \leq q_1 \approx 1.0401 \\ C_2(q), & q_1 \leq q \leq q_2 \approx 1.1410 \\ C_3(q), & q_2 \leq q \leq \sqrt{\frac{4}{3}} \approx 1.1547 \\ C_4(q), & \sqrt{\frac{4}{3}} \leq q \leq \frac{1}{4}(1+\sqrt{17}) \approx 1.2808 \\ C_5(q), & \frac{1}{4}(1+\sqrt{17}) \leq q \leq \sqrt{2} \approx 1.4142 \\ C_6(q), & \sqrt{2} \leq q \leq \frac{1}{4}(1+\sqrt{33}) \approx 1.6861 \\ C_7(q), & \frac{1}{4}(1+\sqrt{33}) \leq q \leq \frac{1}{2}(1+\sqrt{7}) \approx 1.8229 \end{cases} \qquad C(q) = \begin{cases} C_8(q), & \frac{1}{2}(1+\sqrt{7}) \leq q \leq 2 \\ C_9(q), & 2 \leq q \leq \frac{1}{2}(1+\sqrt{11}) \approx 2.1583 \\ C_{10}(q), & \frac{1}{2}(1+\sqrt{11}) < q \leq q_{10} \approx 2.1956 \\ C_{11}(q), & q_{10} \leq q \leq q_{11} \approx 2.3307 \\ C_{12}(q), & q_{11} \leq q \leq \frac{1}{4}(3+\sqrt{41}) \approx 2.3508 \\ C_{13}(q), & \frac{1}{4}(3+\sqrt{41}) \leq q \leq q_{13} \approx 2.5111 \\ C_{14}(q), & q_{13} \leq q \leq q_{14} \approx 2.5704 \\ C_{15}(q), & q \geq q_{14}, \end{cases}$$

$$C_1(q) = \frac{2}{3} + \frac{1}{2q}, \qquad C_2(q) = 1 + \frac{1}{2}\left(4q^2 + 4q - 1 - \sqrt{(4q^2+4q-1)^2 - 4q^2}\right),$$

$$C_3(q) = \frac{6q+4}{3q+6}, \qquad C_4(q) = q, \qquad C_5(q) = \frac{1}{2} + \frac{1}{q}, \qquad C_6(q) = 1 + \frac{1}{2q+2},$$

$$C_7(q) = \frac{2q+1}{q+2}, \qquad C_8(q) = \frac{2}{3} + \frac{1}{q}, \qquad C_9(q) = 1 + \frac{1}{2q+2}, \quad C_{10}(q) = \frac{3q+2}{2q+3},$$
$$C_{11}(q) = \frac{q^2+3+\sqrt{q^4-6q^2+24q+9}}{6q}, \qquad C_{12}(q) = \frac{q}{2}, \qquad C_{13}(q) = \frac{3}{4} + \frac{1}{q},$$
$$C_{14}(q) = 1 + \frac{q^2+2q-2-\sqrt{q^4+8q+4}}{2q+4}, \qquad C_{15}(q) = 1 + \frac{1}{2q+1},$$

q_1 is the largest real root of $84q^4 - 24q^3 - 80q^2 + 6q + 9$,
q_2 is the largest real root of $27q^4 + 48q^3 - 54q^2 - 48q + 8$,
q_{10} is the smallest real root of $3q^4 - 9q^3 - 8q^2 + 21q + 18$,
q_{11} is the largest real root of $q^3 - 2q - 8$,
q_{13} is the largest real root of $20q^4 - 39q^3 - 46q^2 + 32q + 32$,
q_{14} is the largest real root of $4q^5 + 2q^4 - 24q^3 - 23q^2 + 6q + 8$.

7.1.3 Impossibility Results

The lower bound on the overall competitive ratio is easily proven. Let $q = \frac{1}{4}(1+\sqrt{17})$.

The adversary first gives a job of size $\frac{1}{q}$. If the algorithm assigns this job to the slow machine, it has a competitive ratio of at least q. Thus, assume that it is scheduled on the fast machine. Now, two jobs of size $\frac{1}{2}$ follow. If they are both scheduled on the slow machine, the makespan is q. Otherwise, it is at least $\frac{1}{q} + \frac{1}{2} = q$.

The optimal schedule is obtained by scheduling the first job on the slow machine and the last two jobs on the fast machine, yielding a makespan of 1.

Strictly speaking, this example works only for the strict competitive ratio, but noting that the job sizes could be scaled by any factor, we obtain the impossibility result for the competitive ratio in general.

We now give the sequences proving the impossibility result of each interval. For $i \geq 4$, $C_i(q) \leq q$. Thus, when proving impossibility results for these intervals, we can assume that the first job is scheduled on the fast machine. For intervals 1–3, we need to consider both possibilities.

Interval 1: $\frac{1}{2q},\ \frac{1}{2q},\ \frac{1}{3},\ \frac{1}{3},\ \frac{1}{3}$.

For intervals 2 and 3, let p_1 be the size of the first job. If the first job is put on the slow machine, four more jobs are given. The first of these has size $\frac{3+2q-2q^2}{2q^2+q}p_1$ and the last three all have size $\frac{q+1}{2q+1}p_1$. Otherwise, the following two sequences are used.

Interval 2: $\frac{1}{q} - \frac{2q+1}{q+1}p_5,\ \ \frac{2q+1}{q+1}p_5,\ \ 1-2p_5,\ \ p_5,\ \ p_5,$

$$\text{where } p_5 = \frac{q+1}{2q}\left(4q^2+4q-1-\sqrt{(4q^2+4q-1)^2-4q^2}\right).$$

Interval 3: $-3q^2+4q+4,\ \ q+2,\ \ q+2,\ \ q+2,\ \ 3q^2+q-2,\ \ 3q^2+q-2.$

Intervals 4 and 5: $\frac{1}{q},\ \frac{1}{2},\ \frac{1}{2}$.

Intervals 6 and 9: $2q^2+q-2,\ \ q+2,\ \ q+1,\ \ q+1.$

Interval 7: $q+2,\quad -q^2+2q+2,\quad q^2-1,\quad q^2-1.$

Interval 8: $\frac{1}{q},\ \frac{1}{3},\ \frac{1}{3},\ \frac{1}{3}.$

Interval 10: $2q+3,\quad -q^2+3q+3,\quad q^2-1,\quad q^2-1,\quad q^2-1.$

Interval 11: $4q,\quad 4q^2-3p_5,\quad p_5,\quad p_5,\quad p_5,\quad \text{where } p_5=\frac{1}{3}\left(5q^2-3-\sqrt{q^4-6q^2+24q+9}\right).$

Intervals 12 and 13: $\frac{1}{q},\ \frac{1}{4},\ \frac{1}{4},\ \frac{1}{4},\ \frac{1}{4}.$

Interval 14: $\frac{1}{q},\quad 1-\frac{1}{q}-\frac{q+2}{q+1}p_5,\quad \frac{1}{q}-\frac{q}{q+1}p_5,\quad p_5,\quad p_5,$

$$\text{where } p_5=\frac{q+1}{2q(q+2)}\left(q^2+2q-2-\sqrt{q^4+8q+4}\right).$$

Interval 15:
$q<1+\sqrt{3}$: $2q+1,\quad 2q^2-2q-3,\quad q+1,\quad q+1\quad q+1.$
$q\geq 1+\sqrt{3}$: $2q^2-2q-3,\quad 2q+1,\quad q+1,\quad q+1\quad q+1.$

7.1.4 The New Algorithms

In the intervals where the general lower bound matches the competitive ratio of LPT, clearly LPT is optimal. Those intervals are the following.

- $q=1$ (for $q=1$, the competitive ratio of LPT is $\frac{7}{6}$ [61], and this is optimal [93]).
- $\frac{1}{6}(1+\sqrt{37})\leq q\leq q_9$, where $q_9\approx 2.04$ is the largest real root of $4q^3-4q^2-10q+3$. This is most of interval 4, all of intervals 5–8, and a little of interval 9.
- $q\geq q_{14}\approx 2.57$. This is the last interval (interval 15).

This leaves the following intervals to deal with.

- Intervals 1–4, not including $q=1$ in interval 1, and interval 4 only up to $\frac{1}{6}(1+\sqrt{37})$.
- Intervals 9–14, interval 9 starting only at q_9.

For the first four intervals, we design the algorithm Slow-LPT. Intuitively, the reason why LPT fails in the interval $1<q<\frac{1}{6}(1+\sqrt{37})$ is that the slow machine is not much slower than the faster one. Since the fast machine does not dominate the slow machine so easily, it often makes sense to use the slow machine first, and keep the fast machine free for future jobs.

Since Slow-LPT is optimal in all of interval 4, this gives an alternative optimal algorithm for the interval $\frac{1}{6}(1+\sqrt{37})\leq q\leq\frac{1}{4}(1+\sqrt{17})$.

Algorithm Slow-LPT

```
    Assign $J_1$ to $M_q$. Assign $J_2$ to $M_1$.
    If $q(p_1+p_3)\leq C(q)(p_2+p_3)$, assign $J_3$ to $M_q$, and otherwise to $M_1$.
    Assign the rest of the jobs by the LPT rule.
```

In intervals 9 and 10, 13 and 14, we use the algorithm Balanced-LPT that schedules the second job of the sequence on the slow machine, unless it might break the ratio.

Since Balanced-LPT is optimal in all of interval 9, this gives an alternative optimal algorithm for the interval $2 \leq q \leq q_9$.

Algorithm Balanced-LPT

```
    Assign $J_1$ to $M_1$.
    If $qp_2 > C(q)(p_1 + p_2)$, assign $J_2$ to $M_1$, and otherwise to $M_q$.
    Assign the rest of the jobs by the LPT rule.
```

Finally, for intervals 11 and 12, we introduce the algorithm Opposite-LPT that does the opposite of LPT, unless it might violate the ratio. If $qp_2 < p_1 + p_2$, LPT puts J_2 on M_q, so Opposite-LPT puts J_2 on M_1, unless $p_1 + p_2 > C(q)qp_2$. Similarly, if $qp_2 \geq p_1 + p_2$, Opposite-LPT puts J_2 on M_q, unless $qp_2 > C(q)(p_1 + p_2)$.

Algorithm Opposite-LPT

```
    Assign $J_1$ to $M_1$.
    Assign $J_2$ to $M_1$ if one of the following holds:
        $qp_2 < p_1 + p_2 \leq C(q)\, qp_2$ or $qp_2 > C(q)(p_1 + p_2)$.
    Otherwise, assign $J_2$ to $M_q$.
    Assign the rest of the jobs by the LPT rule.
```

7.1.5 Performance Guarantees

The proofs of the performance guarantees use only a few simple observations.

We assume without loss of generality that OPT $= 1$. Note that $P \leq 1 + \frac{1}{q}$, since the total size of jobs scheduled by OPT is at most 1 on M_1 and $\frac{1}{q}$ on M_q.

We will always assume that the makespan of the on-line algorithm is determined by the last job, J_ℓ, i.e., ONL $>$ ONL$_{\ell-1}$, since if ONL $=$ ONL$_{\ell-1}$, then $\frac{\text{ONL}_{\ell-1}}{\text{OPT}_{\ell-1}} \geq \frac{\text{ONL}}{\text{OPT}}$.

Consider an input sequence $J_1, J_2, \ldots, J_\ell$ and assume that J_ℓ is scheduled according to the LPT rule. Let $P_1^{\ell-1}$ and $P_q^{\ell-1}$ be the total size of jobs assigned to M_1 and M_q, respectively, just before the arrival of J_ℓ. Then, by the assumption that J_ℓ determines the makespan, ONL $= \min\{P_1^{\ell-1} + p_\ell,\, q(P_q^{\ell-1} + p_\ell)\}$.

In [86] it is noted that ONL $\leq 1 + \frac{q}{q+1} p_\ell$. This follows from the following calculations

$$\begin{aligned}
\min\left\{P_1^{\ell-1} + p_\ell,\, q(P_q^{\ell-1} + p_\ell)\right\} &\leq \frac{q}{q+1}\left(P_1^{\ell-1} + p_\ell\right) + \frac{1}{q+1}\, q\left(P_q^{\ell-1} + p_\ell\right) \\
&= \frac{q}{q+1}\left(P_1^{\ell-1} + P_q^{\ell-1} + 2p_\ell\right) = \frac{q}{q+1}\,(P + p_\ell) \\
&\leq 1 + \frac{q}{q+1} p_\ell.
\end{aligned}$$

This implies that, if OPT schedules k jobs on M_1, then $p_\ell \leq \frac{1}{k}$, and ONL $\leq 1 + \frac{q}{k(q+1)}$. Similarly, if OPT schedules k jobs on M_q, then $p_\ell \leq \frac{1}{qk}$, and ONL $\leq 1 + \frac{1}{k(q+1)}$.

This shows that only short sequences can be problematic. Indeed, the impossibility results cannot be obtained with sequences of more than six jobs. This is natural, since the job sizes are non-increasing; for long sequences the last job is small compared to the total job size of the sequence.

Intervals 9 and 10

To give the flavor of how these simple observations are used in the analysis, we give the proof of the performance guarantee for intervals 9 and 10.

Recall that $C_9(q) = 1 + \frac{1}{2q+2}$ and $C_{10}(q) = \frac{3q+2}{2q+3}$, and that intervals 9 and 10 correspond to the interval $2 \le q \le q_{10} \approx 2.20$. In interval 9, $C_{10}(q) \le C_9(q)$, and in interval 10, $C_9(q) \le C_{10}(q)$. Thus, in intervals 9 and 10, $C(q) = \max\{C_9(q), C_{10}(q)\}$.

If OPT runs five jobs on M_1,

$$\text{ONL} \le 1 + \frac{q}{5(q+1)} \le 1 + \frac{1}{2(q+1)} = C_9(q) \le C(q), \text{ since } \frac{q}{5(q+1)} \le \frac{1}{2(q+1)}, \text{ for } q \le \frac{5}{2}.$$

Hence, we assume that OPT runs at most four jobs on M_1. Similarly, we assume that OPT runs at most one job on M_q, since otherwise $\text{ONL} \le 1 + \frac{1}{2(q+1)} = C_9(q)$. Thus, we need only consider sequences of length at most five.

If, in the optimal schedule, no jobs are assigned to M_q, Balanced-LPT will not break the ratio. Hence, we assume that OPT schedules exactly one job on M_q and at most four jobs on M_1.

In intervals 9 and 10, Balanced-LPT always assigns J_2 to M_q, since

$$C(q)\,(p_1 + p_2) \;\ge\; C_9(q)\,(p_1 + p_2) \;\ge\; 2\,C_9(q)\,p_2 \;\ge\; q\,p_2, \;\text{ for } q \;\le\; \frac{1+\sqrt{13}}{2} \approx 2.30.$$

This shows that sequences with at most two jobs cannot break the ratio. It also shows that, if the sequence contains at least three jobs, then $\text{ONL} \le P - p_2$. If OPT does not run J_1 on M_1, $\text{OPT} \ge P - p_2 \ge \text{ONL}$. This leaves only the case, where OPT runs J_1 on M_q and all other jobs on M_1.

Three jobs. Since OPT runs J_1 on M_1, $\text{OPT} \ge qp_1$. By the assumption that the last job determines the on-line makespan, $\text{ONL} \le p_1 + p_3 \le 2p_1 \le q\,p_1$, since ONL runs J_2 on M_q.

Four Jobs. Since OPT runs J_1 on M_q and all other jobs on M_1, $p_1 \le \frac{1}{q}$ and $p_2+p_3+p_4 \le 1$. Combining the latter inequality with $p_4 \le p_3 \le p_2$ yields $p_3 + p_4 \le \frac{2}{3}$. Thus,

$$\text{ONL} \;\le\; p_1 + p_3 + p_4 \;\le\; \frac{1}{q} + \frac{2}{3} \;=\; \frac{2q+3}{3q} \;\le\; \frac{2q+3}{2q+2} \;=\; C_9(q), \quad \text{for } q \ge 2.$$

Five Jobs If Balanced-LPT schedules at least one of the jobs J_3 and J_4 on M_q, $\text{ONL} \le P - (p_2 + p_4) \le 1 + \frac{1}{q} - 2p_5$. Moreover, $\text{ONL} \le 1 + \frac{q}{q+1}\,p_5$. Equating these two upper bounds yields $p_5 = \frac{q+1}{3q^2+2q}$, and hence,

$$\text{ONL} \;\le\; 1 + \frac{q}{q+1}\,\frac{q+1}{3q^2+2q} \;=\; 1 + \frac{1}{3q+2} \;<\; C_9(q).$$

Otherwise, $\text{ONL} \le q\,(p_2+p_5)$. Since OPT runs the last four jobs on M_1, $p_3+p_4+p_5 \le 1-p_2$, implying that $p_5 \le \frac{1}{3}(1 - p_2)$. Thus, $\text{ONL} \le \frac{q}{3}(1 + 2p_2)$. Furthermore, $\text{ONL} \le P - p_2 \le 1 + \frac{1}{q} - p_2$. Equating the two upper bounds gives $p_2 = \frac{-q^2+3q+3}{2q^2+3q}$. Hence,

$$\text{ONL} \;\le\; 1 + \frac{1}{q} + \frac{q^2 - 3q - 3}{2q^2 + 3q} \;=\; \frac{3q+2}{2q+3} \;=\; C_{10}(q).$$

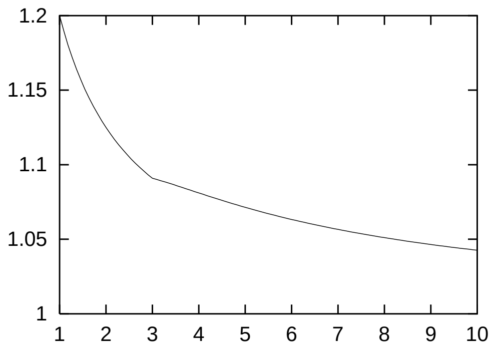


Figure 7.2: The competitive ratio as a function of q

7.2 Preemptive Scheduling

[93] gives the exact competitive ratio of preemptive scheduling of non-increasing sequences on *identical* machines. The ratio tends to $\frac{1}{2}(1+\sqrt{3}) \approx 1.366$ as m tends to infinity. The result is valid for deterministic as well as randomized algorithms.

On general sequences, the competitive ratio for preemptive scheduling on two related machines is $1+\frac{q}{q^2+q+1}$ (see Chapter 2). In this section we prove that, if the job sizes are non-increasing, the competitive ratio is

$$C(q) = \begin{cases} 1+\dfrac{1}{3q+2}, & \text{for } 1 \le q \le 3 \\ 1+\dfrac{q-1}{2q^2+q+1}, & \text{for } q \ge 3, \end{cases}$$

for randomized as well as deterministic algorithms. This result is depicted in Figure 7.2. As for general sequences, the competitive ratio attains its maximum at $q = 1$. The maximum is $\frac{6}{5}$ — a little lower than the maximum of $\frac{4}{3}$ for general sequences.

We design two classes of algorithms, one for $q \le 2$ and one for $q > 2$. The first class of algorithms do not use idle time and resemble previously known algorithms. The second class of algorithms introduce idle time when scheduling the first job. This is in contrast to earlier algorithms. In non-preemptive scheduling idle time is clearly not useful. However, in previous work on preemptive scheduling of general or non-increasing sequences [30, 41, 45, 92, 93, 99], idle time has not been used either. As observed in [30], idle time is never necessary in the case of identical machines. We prove that any optimal algorithm for scheduling non-increasing sequences on two related machines with a speed ratio of more than 2 must introduce idle time when scheduling the first job. It seems reasonable that, for preemptive models where the exact competitive ratio is not yet known, introducing idle time could lead to the design of algorithms with optimal competitive ratio. However, it is not clear how this can be done. Our algorithms introduce idle time only when scheduling the first job (when scheduling later jobs, no additional idle time is introduced). This construction is simple enough to analyze, and leads to algorithms of optimal competitive ratio.

Note that the break point in the competitive ratio is $q = 3$ and not $q = 2$. Even though the algorithms for $q \le 2$ and $2 < q \le 3$ are different, they have the same function as competitive ratio.

7.2.1 Preliminaries

In the proofs of the performance guarantees, and when proving that idle time is needed when $q > 2$, we need the following special case of a result from [64]. For any input sequence, the optimal makespan is $\max\{p_1, \frac{q}{q+1}P\}$. This means that if $P \geq \frac{q+1}{q}p_1$, the optimal makespan is $\frac{q}{q+1}P$.

We let $r_1 = 1 + \frac{1}{3q+2}$ and $r_2 = 1 + \frac{q-1}{2q^2+q+1}$ denote the optimal competitive ratio that we are going to prove for $q \leq 3$ and $q \geq 3$, respectively. In Section 7.2.3, we let r_i denote r_1 or r_2, depending on which range of q is considered.

7.2.2 Algorithms for $q \leq 2$

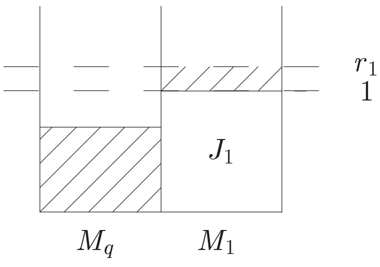
The algorithms for $q \leq 2$ work similarly to the algorithm in [30]. The first job J_1 is scheduled on the fast machine. Without loss of generality we assume that it has size 1.

As long as the total size of jobs does not exceed $1 + \frac{1}{q}$, OPT $= 1$. These jobs are scheduled between time 1 and r_1 on the fast machine first and then from time 0 on the slow machine. We stop when the total size reaches $1 + \frac{1}{q}$ (some job may be partially assigned, denote this job J_p). At this point, the load on the slow machine does not exceed 1. Hence, even if a job was split between the two machines, its two parts do not overlap in time. We have the following situation.

The loads are $r_1 = \dfrac{3q+3}{3q+2}$ (fast)

and $(1 + \dfrac{1}{q} - r_1)q = \dfrac{2q+2}{3q+2}$ (slow).

Note that the ratio of the loads is 3 : 2.



From now on, we keep the ratio of 3 : 2 between the loads, so that the fast machine is always more loaded. The remaining part of J_p (if any) as well as any new arriving job of size p will be split in two pieces of size $\frac{3q}{3q+2}p$ (fast machine) and $\frac{2}{3q+2}p$ (slow machine). The ratio between the extra loads is 3 : 2 as required.

Since the total size of scheduled jobs is at least $1 + \frac{1}{q}$, OPT $= \frac{q}{q+1}P$, and ONL $= \frac{3q}{3q+2}P$. Hence, the competitive ratio of r_1 is kept. To complete the proof, we must prove the following.

(a) The remaining part of J_p is scheduled properly.

(b) Any future job J is scheduled properly.

We prove (a) first. Let p be the size of the remaining part of J_p. The proof is split into two cases.

J_p is the second job in the sequence. Since J_p is scheduled on the fast machine no earlier than time 1, we just need to show that, on the slow machine, it will be completed no later than time 1.

The part of J_p scheduled on M_q adds $\frac{2q}{3q+2}p$ to the load of M_q. Since the size of the second job is at most 1, and $\frac{1}{q}$ of it has already been scheduled, $p \leq 1 - \frac{1}{q}$. Thus, after scheduling all of J_{p}, the load on M_q is

$$\frac{2q+2}{3q+2}+\frac{2q}{3q+2}p \le \frac{2q+2}{3q+2}+\frac{2q}{3q+2}\left(1-\frac{1}{q}\right) = \frac{4q}{3q+2} \le 1, \text{ since } q \le 2.$$

J_{p} arrives as the third job or later. In this case, the second job has size less than $\frac{1}{q}$, and so have later jobs. As in the previous case, an invalid schedule cannot occur, unless J_{p} runs on the slow machine after time 1. Hence, assume that J_{p} runs on M_q after time 1. Since $p < \frac{1}{q}$, this implies that J_{p} is not scheduled on M_1 before time r_1. Thus, it suffices to show that the load on M_q does not exceed r_1.

If the load on M_q exceeds r_1, the total size of the jobs is more than

$$\begin{aligned}\frac{r_1}{q}+\frac{3}{2}r_1 &= \frac{2+3q}{2q}r_1 = \frac{2+3q}{2q}\frac{3q+3}{3q+2} = \frac{3q+3}{2q} = \frac{3}{2}\frac{q+1}{q} = \frac{3}{2}\left(1+\frac{1}{q}\right)\\ &> 1+\frac{1}{q}+\frac{1}{q} > 1+\frac{1}{q}+p,\end{aligned}$$

which is impossible.

Now we prove (b). Let p be the size of J and let P be the total size of previous jobs.

Just before scheduling J, the load on M_1 is $\frac{3q}{3q+2}P$ and the load on M_q is $\frac{2q}{3q+2}P$. Thus, we just need to show that the part of J scheduled on M_q has size at most $\frac{1}{3q+2}P$, i.e., $\frac{2}{3q+2}p \le \frac{1}{3q+2}P$. This is true, since at least two jobs of size at least p have been given before J, and hence $p \le \frac{1}{2}P$.

7.2.3 Algorithms for $q > 2$

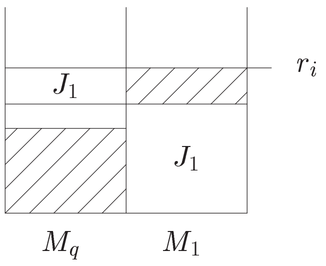
The only real difference between the algorithms for $q > 2$ and those for $q \le 2$ is in the way the first job is scheduled.

Assume without loss of generality that the first job has size 1. We split this job in two pieces of sizes $\frac{q-r_i}{q-1}$ and $\frac{r_i-1}{q-1}$. Since $q > 2$, both fractions are positive. The first piece is scheduled on the fast machine from time 0, and the other is scheduled on the slow machine from time $\frac{q-r_i}{q-1}$ until time $\frac{q-r_i}{q-1}+\frac{r_i-1}{q-1}q = r_i$.

In general, future jobs (or parts of jobs) assigned to the fast machine will be scheduled one after the other without any idle time. Jobs (or parts of jobs) assigned to the slow machine will be scheduled at the first idle time. Once no idle time is left, they will be scheduled after r_i (it might be necessary to split some job and continue it after time r_i).

Similarly to the algorithms for $q \le 2$, as long as the total size is at most $1+\frac{1}{q}$, new jobs are scheduled on the fast machine between time $\frac{q-r_i}{q-1}$ and r_i, and then on the slow machine, starting at time 0.

At the time when the total size of jobs is exactly $1+\frac{1}{q}$, the fast machine is occupied from time 0 until time r_i, since $\frac{r_i-1}{q-1}+r_i \le 1+\frac{1}{q}$, for $q \ge 2$. On the slow machine, there is still idle time, since the total size of jobs is strictly less than $r_i(1+\frac{1}{q})$.



From this time on, OPT $= \frac{q}{q+1}P$. If $q \leq 3$, we will keep the ratio 3 : 2 between the loads of the fast and the slow machines, which gives the desired competitive ratio, just as in the case $q \leq 2$. If $q \geq 3$, we keep the ratio $2 : (1+\frac{1}{q})$, leading to an on-line makespan of $\frac{2q^2}{2q^2+q+1}P$. This gives the desired competitive ratio of

$$\frac{2q^2}{2q^2+q+1} \Big/ \frac{q}{q+1} \;=\; \frac{2q(q+1)}{2q^2+q+1}.$$

We again need to show that the leftover of the job for which the total size reached $1+\frac{1}{q}$ is assigned properly, and that future jobs are assigned properly. As soon as J_1 has been scheduled, the free time slots on the two machines before time r_i are disjoint. Therefore, there is no difference between a complete job and the leftover of a job.

Denote the (leftover of a) job that is being scheduled by J. As earlier, let p denote the size of J and let P denote the total size of earlier jobs.

$2 \leq q \leq 3$: As in the proof of (b) for $q \leq 2$, we just need $p \leq \frac{1}{2}P$. If J is at least the third job in the sequence, we can use the same argument as in the case $q \leq 2$. Otherwise, J is the leftover of a job. Since $\frac{1}{q}$ of this job has already been scheduled, $p \leq 1 - \frac{1}{q}$. For $q \leq 3$, $1 - \frac{1}{q} \leq \frac{1}{2}(1+\frac{1}{q}) \leq \frac{1}{2}P$ holds true.

$q \geq 3$: In this case, the loads are $\frac{2q^2}{2q^2+q+1}P$ (fast) and $\frac{q^2+q}{2q^2+q+1}P$ (slow). Thus, the time interval available for a new job on the slow machine is of length $\frac{q^2-q}{2q^2+q+1}P$, and the load of J on the slow machine is $\frac{q^2+q}{2q^2+q+1}p$. Hence, we just need $p \leq \frac{q-1}{q+1}P$. If J is the third job or more, then $p \leq \frac{1}{2}P$ which is at most $\frac{q-1}{q+1}P$, since $q \geq 3$. Otherwise, as in the case $2 \leq q \leq 3$, $p \leq 1 - \frac{1}{q}$. Since $P \geq 1 + \frac{1}{q}$, we get

$$p \;\leq\; 1 - \frac{1}{q} \;=\; \frac{q-1}{q} \;=\; \frac{q-1}{q+1}\,\frac{q+1}{q} \;\leq\; \frac{q-1}{q+1}P.$$

Idle Time is Necessary

In this section we prove that any optimal algorithm must introduce idle time when scheduling the first job. Assume for the sake of contradiction that an optimal algorithm exists that does not use idle time. Consider such an algorithm and a sequence of two unit jobs.

After the arrival of the first job, OPT = 1. Since $r_i < 2$, the job cannot be scheduled completely on the slow machine, since this would break the ratio, and splitting the job would introduce idle time. Hence, the job must be scheduled completely on the fast machine.

After the arrival of the second job, OPT $= \frac{2q}{q+1}$. Hence the time interval that the algorithm can use on the fast machine is $\frac{2q}{q+1} r_i - 1$. If it uses all of that time interval, it can only schedule the job between time 0 and time 1 on the slow machine. This means that the maximal size that can be scheduled is

$$\frac{2q}{q+1} r_i - 1 + \frac{1}{q}.$$

For $2 < q \leq 3$, this is

$$\frac{3q^2 + q + 2}{3q^2 + 2q},$$

and for $q \geq 3$, it is

$$\frac{2q^3 + q^2 + 1}{2q^3 + q^2 + q}.$$

For $q > 2$, both are less than 1.

7.2.4 General Impossibility Results

To prove that the algorithms of Sections 7.2.2 and 7.2.3 are optimal we use the following simplified version of a lemma in [47].

Consider a sequence of at least two jobs, where $J_{\ell-1}$ and J_ℓ are the last two jobs. The competitive ratio of any preemptive on-line algorithm, deterministic or randomized, is at least

$$\frac{qP}{\mathrm{OPT}_{\ell-1} + q\mathrm{OPT}_\ell}.$$

As in [93], we show that the most difficult cases are sequences of identical jobs.

Consider two sequences consisting of two and three unit size jobs. The optimal makespan is 1 after the first job, $\frac{2q}{q+1}$ after the second job, and $\frac{3q}{q+1}$ after the third job (if it arrives). Thus, the sequence of two jobs gives a lower bound of

$$\frac{2q}{1 + q\frac{2q}{q+1}} = \frac{2q^2 + 2q}{q + 1 + 2q^2} = 1 + \frac{q-1}{2q^2 + q + 1} = r_2,$$

and the sequence of three jobs gives the lower bound

$$\frac{3q}{\frac{2q}{q+1} + \frac{3q^2}{q+1}} = \frac{3(q+1)}{2 + 3q} = 1 + \frac{1}{3q + 2} = r_1.$$

Chapter 8

Conclusion

In this thesis, we have given a survey of measures for the quality of on-line algorithms.

Furthermore, we have studied five on-line problems with restricted input. Below is a summary and a short discussion of the results.

Paging with Locality of Reference. We assume that, for each possible window length ℓ, an upper bound on the maximum/average number of distinct pages within windows of length ℓ is given. This enables us to use the fault rate as the quality measure. We studied LRU, FIFO, the class of deterministic marking algorithms, and the optimal off-line algorithm LFD and proved tight or nearly tight upper and lower bounds on the fault rates. Throughout our experiments, the results of both models were far closer to reality than the results of competitive analysis. The fault rates predicted in the Max-Model were closer to reality than those of the Average-Model, supporting our intuition that in the Max-Model, the adversary is more restricted than in the Average-Model.

Edge Coloring with a Fixed Number of Colors. We first studied the case of k-colorable graphs, i.e., the input graphs can be colored completely with the k colors available. Any fair deterministic algorithm has a competitive ratio between $\frac{1}{2}$ and $\frac{2}{3}$. Next-Fit has a competitive ratio matching the lower bound, and the competitive ratio of First-Fit is $\frac{k}{2k-1}$. Thus, for small k, First-Fit is significantly better than Next-Fit, but for large k, their competitive ratios can hardly be distinguished.

Some of the proofs for k-colorable graphs can be generalized to the case of general graphs, with sligthly different results. Thus, we proved that any fair algorithm has a competitive ratio of at least $2\sqrt{3}-3 \approx 0.4641$, and that this bound is matched by the upper bound for Next-Fit. Though, intuitively, First-Fit is a more reasonable algorithm than Next-Fit, we proved that the competitive ratio of First-Fit is at most $\frac{2}{9}(\sqrt{10}-1) \approx 0.4805$, and hence it cannot be much better than Next-Fit.

Both First-Fit and Next-Fit perform a little worse in the general case than in the case of k-colorable graphs. In neither case did we find an algorithm significantly better than Next-Fit. In the general case, such an algorithm would have to be unfair or randomized, because no fair deterministic algorithm is more than $\frac{1}{2}$-competitive. However, even if we consider unfair and/or randomized algorithms, no algorithm can be more than $\frac{4}{7}$-competitive in the case of general graphs.

Bin Packing in Variable-Sized Bins. When studying bin packing in variable-sized bins, we considered only input sequences that can be packed completely by an optimal off-line algorithm, since for general sequences, no fair algorithm is competitive. The situation for fair algorithms is similar to the situation for fair edge coloring algorithms in the case of k-colorable graphs, with the number n of bins corresponding to k. The competitive ratio of any fair deterministic algorithm is between $\frac{1}{2}$ and $\frac{2}{3}$. The lower bound is tight due to Worst-Fit. A class of algorithms (Smallest-Bins-First) including First-Fit and Best-Fit have competitive ratio $\frac{n}{2n-1}$.

The competitive ratio of Worst-Fit is the same as in the case of identical bins, but the competitive ratio of First-Fit and Best-Fit is worse than for identical bins — in the case of identical bins they have a competitive ratio of at least $\frac{5}{8}$. Thus, in the more general case of variable-sized bins, the variation is much smaller.

An interesting open problem is to find an algorithm with a competitive ratio significantly better than $\frac{1}{2}$ for any number of bins or to show that it does not exist. It could also be interesting to determine whether such an algorithm would have to be unfair.

Scheduling on Two Related Machines. We study the case, where the job sizes are non-decreasing. As expected, this gives a better competitive ratio than in the case of general sequences.

Non-preemptive scheduling: We have determined the ranges of q for which LPT is optimal among deterministic algorithms. For the intervals, where LPT is not optimal, we have devised optimal deterministic algorithms. The range $q \geq 1$ is divided in 15 intervals with different functions describing the competitive ratio, and our proof is divided into cases, mostly covering only two intervals. This does not lend much hope to generalizing our results to the case of more machines. One could hope that there are simpler results for randomized algorithms.

Preemptive Scheduling: We give optimal algorithms, one for the interval $1 \leq q \leq 2$ and one for $q \geq 2$. The competititive ratio consists of two functions, one for the interval $1 \leq q \leq 3$ and one for $q \geq 3$. The algorithms are deterministic, and we prove that no randomized algorithm can have a better competitive ratio.

We prove that for $q > 2$, any optimal on-line algorithm must introduce idle time when scheduling the first job. This is the first on-line scheduling problem, where idle time has been proven to be required. Even though we do not know how to use idle time for other variants of the scheduling problem, the use of idle time might be a step towards optimal algorithms for those variants, where the exact competitive ratio has not yet been determined.

Bibliography

[1] D. Achlioptas, M. Chrobak, and J. Noga. Competitive Analysis of Randomized Paging Algorithms. *Theoretical Computer Science*, 234:203–218, 2000. Also in *ESA 96*, pages 419–430.

[2] S. Albers. On the Influence of Lookahead in Competitive Paging Algorithms. *Algorithmica*, 18:283–305, 1997. Also in *ESA 93*, pages 1–12.

[3] S. Albers. Better Bounds for Online Scheduling. *SIAM Journal on Computing*, 29:459–473, 1999.

[4] S. Albers, L. M. Favrholdt, and O. Giel. On Paging with Locality of Reference. In *34th Annual ACM Symposium on the Theory of Computing (to appear)*, 2002.

[5] M. Andrews, B. Awerbuch, A. Fernández, F. T. Leighton, Z. Liu, and J. M. Kleinberg. Universal-Stability Results and Performance Bounds for Greedy Contention-Resolution Protocols. *Journal of the ACM*, 48(1):39–69, 2001.

[6] N. Ascheuer, S. O. Krumke, and J. Rambau. Online Dial-a-Ride Problems: Minimizing the Completion Time. In *17th International Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 639–650, 2000.

[7] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-Line Load Balancing with Applications to Machine Scheduling and Virtual Circuit routing. *Journal of the ACM*, 44(3):486–504, 1997.

[8] Y. Azar, J. Boyar, L. Epstein, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen. Fair versus Unrestricted Bin Packing. *Algorithmica (to appear)*.

[9] Y. Azar, L. Epstein, and R. van Stee. Resource Augmentation in Load Balancing. In *7th Scandinavian Workshop on Algorithm Theory*, volume 1851 of *Lecture Notes in Computer Science*, pages 189–199, 2000.

[10] E. Bach, J. Boyar, L. Epstein, L. M. Favrholdt, T. Jiang, K. S. Larsen, G.-H. Lin, and R. van Stee. Tight Bounds on the Competitive Ratio on Accommodating Sequences for the Seat Reservation Problem. Journal of Scheduling (to appear).

[11] A. Bar-Noy, R. Motwani, and J. Naor. The Greedy Algorithm is Optimal for On-Line Edge Coloring. *Information Processing Letters*, 44:251–253, 1992.

[12] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New Algorithms for an Ancient Scheduling Problem. *Journal of Computer and System Sciences*, 51(3):359–366, 1995.

[13] L. A. Belady. A Study of Replacement Algorithms for Virtual Storage Computers. *IBM Systems Journal*, 5:78–101, 1966.

[14] S. Ben-David and A. Borodin. A New Measure for the Study of On-Line Algorithms. *Algorithmica*, 11(1):73–91, 1994.

[15] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson. On the Power of Randomization in On-Line Algorithms. *Algorithmica*, 11:2–14, 1994.

[16] P. Berman and C. Coulston. Speed is More Powerful than Clairvoyance. *Nordic Journal of Computing*, 6(2):181, 1999.

[17] M. Bern and D. Eppstein. Approximation Algorithms for Geometric Problems. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 8, pages 296–345. 1997.

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

[19] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive Paging with Locality of Reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995. Also in *STOC 91*, pages 249–259.

[20] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial Queueing Theory. *Journal of the ACM*, 48(1):13–38, 2001. Preliminary version in *STOC 96*, pages 376–385.

[21] A. Borodin, N. Linial, and M. E. Saks. An Optimal On-Line Algorithm for Metrical Task Systems. *Journal of the ACM*, 39:745–763, 92. Also in *STOC 87*, pages 373–382.

[22] J. Boyar, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen. The Competitive Ratio for On-Line Dual Bin Packing with Restricted Input Sequences. *Nordic Journal of Computing*, 8(4):463–472, 2001.

[23] J. Boyar, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen. Extending the Accommodating Function. In *Eighth Annual International Computing and Combinatorics Conference (to appear)*, 2002.

[24] J. Boyar and K. S. Larsen. The Seat Reservation Problem. *Algorithmica*, 25:403–417, 1999.

[25] J. Boyar, K. S. Larsen, and M. N. Nielsen. The Accommodating Function — a Generalization of the Competitive Ratio. *SIAM Journal of Computation*, 31(1):233–258, 2001. Also in *WADS 99*, pages 74–79.

[26] M. Brehop, E. Torng, and P. Uthaisombut. Applying Extra Resource Analysis to Load Balancing. *Journal of Scheduling*, 3:273–288, 2000.

[27] J. L. Bruno and P. J. Downey. Probabilistic Bounds for Dual Bin Packing. *Acta Informatica*, 22:333–345, 1985.

[28] B. Chandra. Does Randomization Help in On-Line Bin Packing. *Information Processing Letters*, 43:15–19, 1992.

[29] C. Chekuri and S. Khanna. A PTAS for the Multiple Knapsack Problem. In *11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 213–222, 2000.

[30] B. Chen, A. van Vliet, and G. J. Woeginger. An Optimal Algorithm for Preemptive On-Line Scheduling. *Operations Research Letters*, 18(3):127–131, 1995.

[31] Y. Cho and S. Sahni. Bounds for List Schedules on Uniform Processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.

[32] A. Chou, J. R. Cooperstock, R. El-Yaniv, M. Klugerman, and F. T. Leighton. The Statistical Adversary Allows Optimal Money-Making Trading Strategies. In *Sixth Symposium on Discrete Algorithms*, pages 467–476, 1995.

[33] M. Chrobak and J. Noga. LRU is Better than FIFO. *Algorithmica*, 23(2):180–185, 1999.

[34] J. Csirik. An On-Line Algorithm for Variable-Sized Bin Packing. *Acta Informatica*, 26:697–709, 1989.

[35] J. Csirik and G. J. Woeginger. Resource Augmentation for Online Bounded Space Bin Packing. In *27th International Colloquium on Automata, Languages and Programming (to appear)*, 2000.

[36] H. M. Deitel. *Operating Systems*. Addison-Wesley, 1990.

[37] P. J. Denning. The Working Set Model of Program Behavior. *Communications of the ACM*, 11:323–333, 1968.

[38] P. J. Denning. Working Sets Past and Present. *IEEE Transactions on Software Engineering*, 6:64–84, 1980.

[39] G. Dobson. Scheduling Independent Tasks on Uniform Processors. *SIAM Journal on Computing*, 13(4):705–716, 1984.

[40] J. Edmonds. Scheduling in the Dark. In *31st Annual ACM Symposium on the Theory of Computing*, pages 179–188, 1999.

[41] L. Epstein. Optimal Preemptive On-Line Scheduling on Uniform Processors with Non-Decreasing Speed Ratios. *Operations Research Letters*, 29(2):93–98, 2001. Also in *STACS 2001*, pages 230–237.

[42] L. Epstein and L. M. Favrholdt. Optimal Preemptive Semi-Online Scheduling to Minimize Makespan on Two Related Machines. *Operations Research Letters (to appear)*.

[43] L. Epstein and L. M. Favrholdt. On-Line Maximizing the Number of Items Packed in Variable-Sized Bins. In *Eighth Annual International Computing and Combinatorics Conference (to appear)*, 2002.

[44] L. Epstein and L. M. Favrholdt. Optimal Non-Preemptive Semi-Online Scheduling to Minimize Makespan on Two Related Machines. In *27th International Symposium on Mathematical Foundations of Computer Science (to appear)*, 2002.

[45] L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger. Randomized Online Scheduling on Two Uniform Machines. *Journal of Scheduling*, 4(2):71–92, 2001.

[46] L. Epstein, S. S. Seiden, and R. van Stee. New Bounds for Variable-Sized and Resource Augmented Online Bin Packing. In *29th International Colloquium on Automata, Languages and Programming (to appear)*, 2002.

[47] L. Epstein and J. Sgall. A Lower Bound for On-Line Scheduling on Uniformly Related Machines. *Operations Research Letters*, 26(1):17–22, 2000.

[48] U. Faigle, W. Kern, and G. Turán. On the Performance of On-Line Algorithms for Partition Problems. *Acta Cybernetica*, 9:107–119, 1989/90.

[49] L. M. Favrholdt and M. N. Nielsen. On-Line Edge-Coloring with a Fixed Number of Colors. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 106–116, 2000.

[50] A. Fiat and A. R. Karlin. Randomized and Multipointer Paging with Locality of Reference. In *27th Annual ACM Symposium on the Theory of Computing*, pages 626–634, 1995.

[51] A. Fiat, M. Karp, M. Luby, A. McGeoch, D. D. Sleator, and N. E. Young. Competitive Paging Algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[52] A. Fiat and M. Mendel. Truly Online Paging with Locality of Reference. In *38th Annual Symposium on Foundations of Computer Science*, pages 326–335, 1997.

[53] A. Fiat and G. J. Woeginger. Competitive Odds and Ends. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[54] A. Fiat and G. J. Woeginger. *Online Algorithms: the State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. 1998.

[55] R. Fleischer and M. Wahl. On-Line Scheduling Revisited. *Journal of Scheduling*, 3(6):343–353, 2000.

[56] D. K. Friesen. Tighter Bounds for LPT Scheduling on Uniform Processors. *SIAM Journal on Computing*, 16(3):554–560, 1987.

[57] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao. Resource Constrained Scheduling as Generalized Bin Packing. *Journal of Combinatorial Theory — Series A*, 21:257–298, 1976.

[58] T. Gonzalez, O. H. Ibarra, and S. Sahni. Bounds for LPT Schedules on Uniform Processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

[59] T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating Adversaries for Request-Answer Games. In *11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 564–565, 2000.

[60] R. L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[61] R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, 17(2), 1969.

[62] D. Hauptmeier, S. O. Krumke, and J. Rambau. The Online Dial-a-Ride Problem under Reasonable Load. *Theoretical Computer Science (to appear)*. Prelimiary version in *CIAC 2000*.

[63] Homepage of New Mexico State University TraceBase (Online). Available: http://tracebase.nmsu.edu/tracebase.html.

[64] E. C. Horvath, S. Lam, and R. Sethi. A Level Algorithm for Preemptive Scheduling. *Journal of the Association for Computing Machinery*, 24(1):32–43, 1977.

[65] S. Irani. Competitive Analysis of Paging. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[66] S. Irani and A. R. Karlin. Online Computation. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 13, pages 521–564. 1997.

[67] S. Irani, A. R. Karlin, and S. Phillips. Strongly Competitive Algorithms for Paging with Locality of Reference. *SIAM Journal on Computing*, 25(3):477–497, 1996. Also in *FOCS 92*, pages 228–236.

[68] D. S. Johnson. *Near-Optimal Bin Packing Algorithms.* PhD thesis, MIT, Cambridge, MA, 1973.

[69] D. S. Johnson. Fast Algorithms for Bin Packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.

[70] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM Journal on Computing*, 3:299–325, 1974.

[71] B. Kalyanasundaram and K. Pruhs. Speed is as Powerful as Clairvoyance. In *36th Annual Symposium on Foundations of Computer Science*, pages 214–221, 1995.

[72] B. Kalyanasundaram and K. Pruhs. Maximizing Job Completions Online. In *European Symposium on Algorithms*, pages 235–246, 1998.

[73] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive Snoopy Caching. *Algorithmica*, 3(1):79–119, 1988.

[74] A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov Paging. In *33rd Annual Symposium on Foundations of Computer Science*, pages 208–217, 1992.

[75] C. Kenyon. Best-Fit Bin-Packing with Random Order. In *7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 359–364, 1996.

[76] E. Koutsoupias. Weak Adversaries for the k-Server Problem. In *40th Annual Symposium on Foundations of Computer Science*, pages 444–449, 1999.

[77] E. Koutsoupias and C. H. Papadimitriou. Beyond Competitive Analysis. In *35th Annual Symposium on Foundations of Computer Science*, pages 394–400, 1994.

[78] E. Koutsoupias and C. H. Papadimitriou. On the k-Server Conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

[79] T. W. Lam and K. K. To. Trade-Offs between Speed and Processor in Hard-Deadline Scheduling. In *10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 1999.

[80] C. Lee and D. Lee. A Simple On-Line Bin Packing Algorithm. *Journal of the ACM*, 32:562–572, 1985.

[81] J. Y. Leung. *Fast Algorithms for Packing Problems*. PhD thesis, Pennsylvania State University, 1977.

[82] F. M. Liang. A Lower Bound for On-Line Bin Packing. *Information Processing Letters*, 10(2), 1980.

[83] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive Algorithms for On-Line Problems. In *20th Annual ACM Symposium on the Theory of Computing*, pages 322–333, 1988.

[84] S. Martello and P. Toth. *Knapsack Problems*. John Wiley and Sons, Chichester, 1990.

[85] L. A. McGeoch and D. D. Sleator. A Strongly Competitive Randomized Paging Algorithm. *Algorithmica*, 6(6):816–825, 1991.

[86] P. Mireault, J. B. Orlin, and R. V. Vohra. A Parametric Worst Case Analysis of the LPT Heuristic for Two Uniform Machines. *Operations Research*, 45:116–125, 1997.

[87] R. Motwani, S. Phillips, and E. Torng. Non-Clairvoyant Scheduling. *Theoretical Computer Science*, 130:17–47, 1994.

[88] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal Time-Critical Scheduling Via Resource Augmentation. *Algorithmica*, 32:163–200, 2002. Also in *STOC 97*.

[89] P. Raghavan. A Statistical Adversary for On-Line Algorithms. In *On-line algorithms: Proceedings of a DIMACS workshop*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 79–83, 1992.

[90] M. B. Richey. Improved Bounds for Harmonic-Based Bin Packing Algorithms. *Discrete Applied Mathematics*, 34:203–227, 1991.

[91] S. S. Seiden. On the Online Bin Packing Problem. In *28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 237–248, 2001.

[92] S. S. Seiden. Preemptive Multiprocessor Scheduling with Rejection. *Theoretical Computer Science*, 262(1–2):437–458, 2001.

[93] S. S. Seiden, J. Sgall, and G. J. Woeginger. Semi-Online Scheduling with Decreasing Job Sizes. *Operations Research Letters*, 27(5):215–221, 2000.

[94] J. Sgall. A Lower Bound for Randomized On-Line Multiprocessor Scheduling. *Information Processing Letters*, 63(1):51–55, 1997.

[95] D. D. Sleator and R. E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202–208, 1985.

[96] A. S. Tanenbaum. *Modern Operating System*. Prentice Hall, 1992.

[97] E. Torng. A Unified Analysis of Paging and Caching. *Algorithmica*, 20:175–200, 1998.

[98] A. van Vliet. An Improved Lower Bound for On-Line Bin Packing Algorithms. *Information Processing Letters*, 43(5):277–284, 1992.

[99] J. Wen and D. Du. Preemptive On-Line Scheduling for Two Uniform Processors. *Operations Research Letters*, 23(3–5):113–116, 1998.

[100] D. B. West. *Introduction to Graph Theory*, page 209. Prentice Hall, 1996.

[101] A. C. Yao. An Improved Lower Bound for On-Line Bin Packing Algorithms. *Journal of the ACM*, 27:277–284, 1980.

[102] A. C. Yao. Towards a Unified Measure of Complexity. In *12th Annual ACM Symposium on the Theory of Computing*, pages 222–227, 1980.

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

[104] N. E. Young. On-Line File Caching. In *Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–86, 1998.

[105] N. E. Young. On-Line Paging against Adversarially Biased Random Inputs. *Journal of Algorithms*, 37:218–235, 2000.

Appendix A

Resumé

Denne PhD-afhandling omhandler on-line algoritmer. En on-line algoritme er en algoritme, der får input i små bidder og må reagere på hver bid uden at vide, hvad der følger efter.

Et kendt eksempel er paging-problemet, hvor man arbejder med to hukommelses-niveauer; der er en stor, langsom hukommelse og en lille, hurtig hukommelse, cache'en. Input til problemet er anmodninger om sider fra den langsomme hukommelse. Hvis den ønskede side ikke allerede er i cache, skal den hentes ind fra den langsomme hukommelse. Samtidig skal en anden side smides ud af cache'en for at gøre plads til den nye. Det tager tid at hente sider fra den langsomme hukommelse, så det ønsker man at gøre så sjældent som muligt. Derfor gælder det om at vælge den side, der skal smides ud, med omhu.

Men hvordan måler man, hvilken strategi der er bedst? Et standardmål for kvaliteten af on-line algoritmer er competitive ratio. Kort fortalt er competitive ratio worst case forholdet mellem on-line algoritmens omkostning og omkostningen af en optimal løsning — d.v.s. den løsning man ville vælge, hvis man kendte hele input-sekvensen fra starten og havde al den tid, man havde brug for, til at finde frem til den allerbedste løsning.

Fordelen og svagheden ved competitive ratio er, at det er et meget generelt mål. Det er en fordel, at det kan anvendes på enhver on-line algoritme, man kan komme i tanker om. Til gengæld giver competitive ratio tit ikke så meget information som mere specialiserede mål. F.eks. giver competitive ratio meget lidt information om forskellige paging-algoritmers kvalitet. Enhver deterministisk paging-algoritme har en competitive ratio, der er mindst lige så stor som størrelsen k af cache'en. Det er et ekstremt pessimistisk resultat sammenlignet med empiriske resultater. Samtidig er der adskillige algoritmer, som alle har competitive ratio k, selvom man har observeret, at der i praksis er meget stor forskel på, hvor godt de fungerer.

Dette har motiveret mange forskere til at finde mere specialicerede kvalitetsmål. Afhandlingen giver en oversigt over resultaterne af disse bestræbelser. Derudover gengives resultater fra fem artikler, som jeg har været medforfatter til. Vores tilgang har været at opnå mere realistiske resultater ved at udnytte viden om input. Tit er det nemlig ikke realistisk at antage, at intet vides om input på forhånd.

Den første artikel handler om paging-problemet. Vi giver en meget simpel model for det fænomen, at input-sekvenser til paging-problemet ofte udviser en bestemt struktur kaldet "locality of reference". Denne model giver os mulighed for at bruge fault rate (hvor tit er vi nødt til at hente en side fra den langsomme hukommelse) som kvalitetsmål. Dette er en mere direkte måde at måle algoritmerne på, og vi opnår resultater, som er langt mere realistiske end dem man opnår, når man analyserer competitive ratio.

Den næste artikel handler om kant-farvning af grafer. Vi går ud fra, at der kun er et begrænset antal farver til rådighed. Målet er at farve så mange kanter i grafen som muligt, under forudsætning af, at to nabokanter aldrig får den samme farve. Kanterne dukker op en efter en, og hver kant skal farves — eller afvises — inden den næste kant afsløres. Vi undersøger det generelle tilfælde såvel som det tilfælde, hvor grafen ville kunne farves med det antal farver, man har til rådighed, hvis man kendte hele grafen fra starten.

I den tredje artikel undersøger vi en variant af bin packing. Et begrænset antal kasser er givet, og input er en sekvens af elementer, som skal pakkes i kasserne. Såvel kasserne som elementerne har en en-dimensionel størrelse. Elementerne ankommer et efter et, og hvert element skal pakkes i en kasse — eller afvises — uden nogen viden om elementerne, som evt. kommer efter. Det gælder om at pakke så mange elementer som muligt uden at overfylde nogen kasse. Vi ser på det tilfælde, hvor kasserne ikke nødvendigvis har samme størrelse. Vi betragter udelukkende sekvenser af elementer, som kan pakkes fuldstændigt i de givne kasser, d.v.s. der er plads til dem alle, hvis de bliver pakket rigtigt. I dette specialtilfælde findes der algoritmer, som altid kan pakke en konstant brøkdel af elementerne. Det er tidligere blevet bevist, at ingen fair algoritme — d.v.s. en algoritme, som aldrig afviser et element, hvis den kan få plads til det i en kasse — kan garantere at pakke nogen bestemt brøkdel af elementerne, medmindre man indfører en begrænsning på mængden af input-sekvenser.

De sidste to artikler handler om planlægningsproblemer. Man har to maskiner eller processorer og et antal jobs, som skal afvikles på de to maskiner, som evt. ikke er lige hurtige. Hvert job har en given størrelse, som svarer til den tid, det tager at afvikle det på en maskine med hastighed 1. Målet er at fordele jobs'ne på de to maskiner, så man tidligst muligt bliver færdig med samtlige jobs. Jobs'ne ankommer et efter et, og for hvert job skal man beslutte, hvilken af de to maskiner, det skal afvikles på, uden at kende fremtidige jobs. Vi antager, at jobs'nes længde er ikke-stigende. Man kan enten antage, at et job kun må køre på den ene maskine, eller at man må splitte jobbet op i mindre dele, som ikke behøver at køre på den samme maskine. I begge tilfælde konstruerer vi algoritmer med optimal competitive ratio for enhver kombination af hastigheder.

Appendix B

Papers

B.1 Paging with Locality of Reference

B.2 Edge Coloring with a Fixed Number of Colors

B.3 On-Line Maximizing the Number of Items Packed in Variable-Sized Bins

B.4 Optimal Non-Preemptive Semi-Online Scheduling on Two Related Machines

B.5 Optimal Preemptive Semi-Online Scheduling on Two Related Machines