Alternative Measures for the Analysis of Online Algorithms

by

Reza Dorrigiv

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2010

© Reza Dorrigiv 2010

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis we introduce and evaluate several new models for the analysis of online algorithms. In an online problem, the algorithm does not know the entire input from the beginning; the input is revealed in a sequence of steps. At each step the algorithm should make its decisions based on the past and without any knowledge about the future. Many important real-life problems such as paging and routing are intrinsically online and thus the design and analysis of online algorithms is one of the main research areas in theoretical computer science.

Competitive analysis is the standard measure for analysis of online algorithms. It has been applied to many online problems in diverse areas ranging from robot navigation, to network routing, to scheduling, to online graph coloring. While in several instances competitive analysis gives satisfactory results, for certain problems it results in unrealistically pessimistic ratios and/or fails to distinguish between algorithms that have vastly differing performance under any practical characterization. Addressing these shortcomings has been the subject of intense research by many of the best minds in the field. In this thesis, building upon recent advances of others we introduce some new models for analysis of online algorithms, namely *Bijective Analysis*, *Average Analysis*, *Parameterized Analysis*, and *Relative Interval Analysis*. We show that they lead to good results when applied to paging and list update algorithms. Paging and list update are two well known online problems. Paging is one of the main examples of poor behavior of competitive analysis. We show that LRU is the unique optimal online paging algorithm according to Average Analysis on sequences with locality of reference. Recall that in practice input sequences for paging have high locality of reference. It has been empirically long established that LRU is the best paging algorithm. Yet, Average Analysis is the first model that gives strict separation of LRU from all other online paging algorithms, thus solving a long standing open problem. We prove a similar result for the optimality of MTF for list update on sequences with locality of reference.

A technique for the analysis of online algorithms has to be effective to be useful in day-to-day analysis of algorithms. While Bijective and Average Analysis succeed at providing fine separation, their application can be, at times, cumbersome. Thus we apply a parameterized or adaptive analysis framework to online algorithms. We show that this framework is effective, can be applied more easily to a larger family of problems and leads to finer analysis than the competitive ratio. The conceptual innovation of parameterizing the performance of an algorithm by something other than the input size was first introduced over three decades ago [124, 125]. By now it has been extensively studied and understood in the context of adaptive analysis (for problems in P) and parameterized algorithms (for NP-hard problems), yet to our knowledge this thesis is the first systematic application of this technique to the study of online algorithms. Interestingly, competitive analysis can be recast as a particular form of parameterized analysis in which the performance of OPT is the parameter. In general, for each problem we can choose the parameter/measure that best reflects the difficulty of the input. We show that in many instances the performance of OPT on a sequence is a coarse approximation of the difficulty or complexity of a given input sequence. Using a finer, more natural measure we can separate paging and list update algorithms which were otherwise indistinguishable under the classical model. This creates a performance hierarchy of algorithms which better reflects the intuitive relative strengths between them. Lastly, we show that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the parameterized case, which matches experimental results. We test list update algorithms in the context of a data compression problem known to have locality of reference. Our experiments show MTF outperforms other list update algorithms

in practice after BWT. This is consistent with the intuition that BWT increases locality of reference.

Acknowledgements

I wish to thank several people who helped me during my PhD studies. First, I would like to express my deep and sincere gratitude to my supervisor, Alejandro (Alex) López-Ortiz, for his support and guidance during all stages of my graduate studies. I was lucky to have such a great supervisor. I have learned many valuable academic and non-academic lessons from him. Among other things, Alex taught me how to do research, how to teach effectively, how to write and review scientific papers, and how to overcome challenges. He was always full of great ideas and generously shared them with me. He constantly encouraged me to expand my knowledge in different areas and work on diverse research problems. His optimism, friendly attitude, and profound knowledge of the field was really helpful. Above all, he was a great friend and I am sure I can always count on his support and guidance. Alex, you made my graduate studies an exciting and joyful experience, thank you!

I would also like to thank my thesis committee members Shai Ben-David, Joseph Cheriyan, Ian Munro, and Robert Tarjan for their insightful comments. In particular, I am deeply thankful to Ian for several fruitful discussions and for being a supportive friend. I wish to thank Spyros Angelopoulos and Martin Ehmsen for their collaboration on some results in this thesis. I would also like to thank Jérémy Barbay, Jonathan Buss, and Jochen Könemann for useful comments on my work.

While working on this thesis, I enjoyed support and encouragement of many good friends. I also benefited from great discussions and collaborations with members of the Algorithms and Complexity group. I am grateful to all my friends and 18 co-authors. In particular, I would like to thank Peyman Afshani, Francisco Claude, Stephane Durocher, Arash Farzan, Robert Fraser, Abbas Heydarnoori, Mehdi Mirzazadeh, Patrick Nicholson, Bashir Sadjad, Alejandro Salinger, and Hamid Zarrabi-Zadeh.

Finally, my profound thanks to my family for their unconditional support. I am deeply grateful to my parents, Ali and Sedigheh, for all their sacrifices. They have always been a constant source of love, support, and encouragement. My father was the greatest teacher in all stages of my life. Unfortunately, he passed away during my PhD studies, but his advice, love, and memory are always with me. I would also like to thank my brother Morteza, and my sisters, Razieh and Mahsa, for their support and encouragement. Last but not least, my special thanks to the love of my life, Somayyeh. Words cannot express my deep appreciation to her. This thesis was not possible without her unconditional love, constant support, and endless encouragement. She has always been there to help me. Thank you Somayyeh, for bringing joy and happiness to my life.

Dedication

To the memory of my father,
Ali,

To my dear mother,
Sedigheh,

and

To my beloved wife,
Somayyeh.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

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

Paging is a classic example of a problem studied in the context of online computation. A paging algorithm must decide which k memory pages to keep in the cache without the benefit of knowing the *sequence* of upcoming page requests. The goal of a paging algorithm is to minimize the number of faults (cache misses) over the entire input sequence. The paging algorithm must produce a partial solution after receiving the i^{th} page request and determine which page to evict, shall the page request be a fault. The performance of the paging algorithm is quantified by the number of faults, which is the *cost* of a particular solution.

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

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

Competitive analysis is the standard measure for the analysis of online algorithms. In this model we compare the performance of an online algorithm with an offline optimal algorithm OPT which knows the entire sequence in advance. The main idea is that a competitive algorithm may behave poorly on a sequence, provided it is a difficult one, as reflected by the bad performance of OPT on that sequence. Competitive analysis became popular after two papers by Sleator and Tarjan [140, 141] in 1985, although its ideas were used by Graham [84] and Johnson [98, 99] before. The term competitive analysis was introduced by Karlin et al. [108]. We formally define competitive analysis in Chapter 2.

The competitive ratio has been a great success, enabling the measurement of the performance of various well known heuristics and fostering the development of the field. Online algorithms are more often than not amenable to analysis under this framework; that is, computing the competitive ratio has proven to be effective—even in cases where the exact shape of the OPT solution is unknown. As well, it has been successfully applied outside the original online paging setting to other applications such as online geometric searching and online approximation to NP-complete problems. Competitive analysis is a relatively simple measure to apply yet powerful enough to quantify, to a large extent, the performance of many online algorithms. The growth of online computation is due in no small part to the effectiveness of this measure. Notwithstanding the wide applicability of competitive analysis, it has been repeatedly observed by various researchers that in certain settings the competitive ratio produces results that are too pessimistic or otherwise found wanting. Indeed, the original paper by Sleator and Tarjan [140] discusses the various drawbacks of the competitive ratio in the case of the paging problem and uses resource augmentation to address some of the observed drawbacks. In her survey of paging algorithms [95], Irani says:

> Some of the reservations which have been raised about the competitive analysis of paging are its inability to discern between LRU and FIFO (algorithms whose performances differ markedly in practice), and the fact that the theoretical competitiveness of LRU is much larger than what is observed in practice. Also unsettling is fact that in the standard competitive model, a fixed amount of lookahead does not give an on-line algorithm any advantage. [...] It seems intuitive that in practice, an on-line algorithm would benefit from some limited lookahead, but this is not reflected in the model.

More details about the shortcomings of competitive analysis are provided in Chapter 2. These shortcomings have led to the introduction of several alternative measures for analysis of online algorithms. For the case of paging, Irani [95] says: *"in fact, a survey of the competitive analysis of paging is in some ways a survey of the refinements of competitive analysis."* In this thesis we review these alternatives and then we propose several new measures. We prove that these new measures overcome some of the shortcomings of competitive analysis. As well, we resolve certain issues which none of the previous alternatives did.

In most online problems we have a sequence of events that happen over time. For instance, for paging we have a sequence of page requests. In other online problems, e.g., online robot navigation, we do not have such an event sequence. Instead, the algorithm does not have complete information about the input, e.g., the robot does not know the map of the terrain. The algorithm then can obtain the missing information by performing some (online) actions. We consider such online problems in Chapter 9.

As a testbed we concentrate on two of the most important problems in the context of online computation: paging and list update, though we briefly discuss other applications in Chapter 9 and Chapter 10. These problems are the standard benchmarks for developing alternative measures and they are ideally suited to this task, given our extensive understanding of them. We know why competitive analysis fails, what are typical sequences in practice and we can better evaluate whether a new technique indeed overcomes known shortcomings. It is important to note that even though well studied, most of the alternative models for these problems are only partially successful in resolving the issues posed by them and as such these testbed problems are still challenging case studies against which to test a new model. We provide some background about these problems in this chapter.

1.1 Paging

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

Due to the importance of this problem, several paging algorithms have been proposed. The most well known paging algorithms are Least-Recently-Used (LRU) and First-In-First-Out (FIFO). When an eviction is required, LRU evicts the page that is least recently used and FIFO evicts the page that was first brought to the cache. Flush-When-Full (FWF) is another algorithm that evicts all the pages that are currently in the cache when a fault occurs and the cache is full. It turns out that there exists a simple and efficient optimal algorithm for the offline version of paging (this is not always the case for online problems, e.g., the list update problem [35]). Longest-Forward-Distance (LFD) evicts the page whose next request is latest, if an eviction is required. It is known [28] that LFD is an optimal offline paging algorithm.

Furthermore, paging algorithms are classified by certain common properties. A *lazy* paging algorithm does not evict a page on a hit, and evicts at most one page on a fault. Thus FWF is not lazy while LRU and FIFO are lazy algorithms. A paging algorithm is called *conservative* if it incurs at most k page faults on any page sequence that contains at most k distinct pages. It can be shown [35] that LRU and FIFO are conservative algorithms and FWF is not. Another class of online paging algorithms are *marking* algorithms. A marking algorithm $\mathcal{A}$ works in phases. Each phase starts right after the last request of the previous phase and consists of the maximal sequence of requests that contains at most k distinct pages. All the pages in the cache are unmarked at the beginning of each phase. We mark any page just after the first request to it. When an eviction is necessary, $\mathcal{A}$ should evict an unmarked page. It is easy to show that LRU and FWF are marking algorithms while FIFO is not. It is known that the competitive ratio of any conservative or marking paging algorithm is k and this is the best possible among deterministic online algorithms [35]. Therefore LRU, FIFO, and FWF all have competitive ratio

k. Unfortunately this is not consistent with our expectations. We will elaborate more on this later in the thesis.

Finally we describe a few other paging algorithms that will be of use. LRU-2 is a paging algorithm proposed by O'Neil et al. for database disk buffering [128]. On a fault, LRU-2 evicts the page whose second to last request is least recent. If there are pages in the cache that have been requested only once so far, LRU-2 evicts the least recently used among them. O'Neil et al. provided experimental results supporting that LRU-2 performs better than LRU in database systems. Boyar et al. proved that LRU-2 has competitive ratio $2k$ [38]. On a fault (with a full cache), Last-In-First-Out (LIFO) evicts the page that is most recently brought to the cache, and Least-Frequently-Used (LFU) evicts the page that has been requested the least since entering the cache. LFU and LIFO do not have constant competitive ratios [35].

1.2 Self-Organizing Data Structures

The *dictionary problem* or the *search problem* is one of the most fundamental problems in theoretical computer science. In it we have a (dynamic) totally ordered set of elements on which we want to efficiently perform a sequence of searches. More specifically we require a data structure that stores a set of n elements and supports the following three basic operations: $Insert(e)$ inserts the element e into the structure, $Delete(e)$ deletes e from the structure, and $Search(e)$ finds the element e. Various efficient data structures have been proposed for the dictionary problem. For example balanced trees such as AVL-trees [1] and red-black trees [24, 86] support all three operations in $\Theta(\log n)$ time in the worst case which is the best possible in the worst case in the comparison-based model. These data structures are static, i.e., they do not change their state on a search operation. Self-Organizing or self-adjusting data structures have a restructuring or self-organizing rule that changes the state of the structure after each operation. Informally, this rule is designed to discover the properties of the given input sequence (in an online manner) and reorganize the structure into a state that presumably favors future operations. Self-organizing data structures have several possible advantages over their static counterparts [141]:

1. Their asymptotic amortized time is never much worse than static structures.
2. Their performance is much better than static structures on skewed input sequences as they adjust according to the input.
3. They do not need to maintain additional information (e.g. balance information for search trees) and therefore they need less space.
4. They usually have simple and easy to implement algorithms for their operations.

Their possible disadvantages are [141]:

- They need more local adjustments (they reorganize their state even on a search).
- Their worst case performance for an individual (non-amortized) operation can be bad.

Two very popular self-organizing data structures for the dictionary problem are unsorted linear lists and binary search trees [10]. The most common self-organizing binary search tree is

the *splay tree* [141]. Several results are known about the performance and structure of splay trees and other self-organizing binary search trees (See e.g. [147, 58, 59, 71, 83, 63, 150]).

Using an unsorted list to support dictionary operations is usually called the *list update* or *list accessing* problem. Consider an unsorted list of ℓ items. Let $\mathcal{A}$ be an arbitrary online list update algorithm. The input to $\mathcal{A}$ is a sequence of n requests that must be served in an online manner. To serve a request to an item x, $\mathcal{A}$ linearly searches the list until it finds x. If x is the i^{th} item in the list, $\mathcal{A}$ incurs a cost i to access x. Immediately after this access, $\mathcal{A}$ can move x to any position closer to the front of the list at no extra cost. This is called a *free exchange*. Also $\mathcal{A}$ can exchange any two consecutive items at unit cost. These are called *paid exchanges*. An efficient algorithm can thus use free and paid exchanges to minimize the overall cost of serving a sequence. This is called the *standard cost model* [10]. Later in this section we will describe an alternative cost model.

List update algorithms were among the first algorithms studied using competitive analysis. Three well known deterministic online algorithms are Move-To-Front (MTF), Transpose (TR), and Frequency-Count (FC). MTF moves the requested item to the front of the list whereas TR exchanges the requested item with the item that immediately precedes it. FC maintains an access count for each item ensuring that the list always contains items in non-increasing order of frequency count. Sleator and Tarjan showed that MTF is 2-competitive, while TR and FC do not have constant competitive ratios [140]. Since then, several other deterministic and randomized online algorithms have been studied using competitive analysis. (See [93, 5, 9] for some representative results.)

Albers introduced the algorithm Timestamp (TS) and showed that it is 2-competitive [5]. After accessing an item a, TS inserts a in front of the first item b that appears before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorder the list. Schulz [137] introduced an infinite (uncountable) family of list update algorithms called Sort-By-Rank (SBR). All algorithms in this family achieve the optimal competitive ratio 2 and they mediate between MTF and TS. Consider a sequence $\sigma = \sigma_1\sigma_2\cdots\sigma_m$ of length m. For an item a and a time $1 \leq t \leq m$, denote by $w_1(a,t)$ and $w_2(a,t)$ the time of the last and the second last access to a in $\sigma_1\sigma_2\cdots\sigma_t$, respectively. If a has not been accessed so far, set $w_1(a,t) = 0$ and if a has been accessed at most once, set $w_2(a,t) = 0$. Then we define $s_1(a,t) = t - w_1(a,t)$ and $s_2(a,t) = t - w_2(a,t)$. Note that after each access, MTF and TS reorganize their lists so that the items are in increasing order by s_1 and s_2, respectively[1]. For a parameter $0 \leq \alpha \leq 1$, SBR(α) reorganizes its list after the t^{th} access so that items are sorted by their α-rank function defined as $r_\alpha(a,t) = (1-\alpha) \times s_1(a,t) + \alpha \times s_2(a,t)$.[2] More formally, upon a request for an item a in time t, SBR(α) inserts a just after the last item b in front of a with $r_\alpha(b,t) < r_\alpha(a,t)$. Furthermore, if there is no such item b or this is the first access to a, SBR(α) inserts a at the front of the list. Therefore SBR(0) is equivalent to MTF and SBR(1) is equivalent to TS except for the handling of the first accesses, i.e., they would be equivalent if TS moved an item that has been accessed only once so far to the front of the list.

While list update algorithms with better competitive ratios tend to have better performance in practice, the validity of the cost model has been debated. More precisely, Martínez and Roura [122] and Munro [126], independently addressed the drawbacks of the standard cost model. Let $(a_1, a_2, \ldots, a_\ell)$ be the list currently maintained by an algorithm $\mathcal{A}$. Martínez and Roura argued that in a realistic setting a complete rearrangement of all items in the list which precede item

[1] For TS, strictly speaking, this applies only to items that have been accessed at list twice.

[2] Schulz [137] denoted this by $r_t(a,\alpha)$, here we follow the convention of [69].

a_i would in practice require time proportional to i, while this has cost proportional to i^2 in the standard cost model. Munro provided the example of accessing the last item of the list and then reversing the entire list. The real cost of this operation in an array or a linear link list should be $O(\ell)$, while it costs about $\ell^2/2$ in the standard cost model. As a consequence, their main objection to the standard model is that it prevents online algorithms from using their true power. They instead proposed a new model in which the cost of accessing the i^{th} item of the list plus the cost of reorganizing the first i items is linear in i. We will refer to this model as the *modified cost model*.

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

1.3 Our Results and Organization of the Thesis

The thesis is structured as follows. In Chapter 2, we formally introduce competitive analysis and study its shortcomings using paging as a case study. In Chapter 3, we present a survey of various alternative measures to the competitive ratio that have been proposed in the literature. In Chapter 4 we introduce Bijective Analysis and Average Analysis as two alternative measures for analysis of online algorithms. The application of these measures to paging and list update appears in Chapters 4 and 5. In Chapter 6 we define relative interval analysis and apply it to paging. In Chapter 7, we study parameterized analysis of paging and list update problems. We analyze the performance of algorithms for these problems in terms of the amount of locality in the input sequence. In Chapter 8, we study two existing models for paging with locality of reference in more detail. In Chapter 9, we apply adaptive analysis to two basic geometric search problems. In Chapter 10, we perform an experimental comparison of various list update algorithms for compression. We conclude in Chapter 11.

Remark. Part of the work presented in this thesis has already been published in the following conferences:

1. S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. *On the separation and equivalence of paging strategies.* In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), pages 229-237.

2. S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. *List update with locality of reference.* In Proceedings of the 8th Latin American Theoretical Informatics Symposium (LATIN '08), pages 399-410.

3. R. Dorrigiv, A. López-Ortiz, and J. I. Munro. *On the relative dominance of paging algorithms.* In Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC '07), pages 488-499.

4. R. Dorrigiv, A. López-Ortiz, and J. I. Munro. *On the relative dominance of paging algorithms.* Theoretical Computer Science, Volume 410, Issues 38-40, pages 3694-3701, 2009.

5. R. Dorrigiv, A. López-Ortiz, and J. I. Munro. *List update algorithms for data compression.* In Proceedings of the IEEE Data Compression Conference (DCC '08), page 512.

6. R. Dorrigiv, M. R. Ehmsen, and A. López-Ortiz. *"Parameterized Analysis of Paging and List Update Algorithms".* In Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA '09), to appear, 2009.

7. R. Dorrigiv, A. López-Ortiz, and J. I. Munro. *An application of self-organizing data structures to compression.* In Proceedings of the 8th International Symposium on Experimental Algorithms (SEA '09), pages 137-148.

and in the following workshops and venues:

8. R. Dorrigiv and A. López-Ortiz. *A survey of performance measures for online algorithms.* ACM SIGACT (Special Interest Group on Automata and Computability Theory) News, 36 (3): 67-81.

9. R. Dorrigiv and A. López-Ortiz. *Adaptive searching in one and two dimensions.* In Proceedings of the the 20th Canadian Conference on Computational Geometry (CCCG '08), pages 215-218.

10. R. Dorrigiv and A. López-Ortiz. *On certain new models for paging with locality of reference.* In Proceedings of the 2nd Workshop on Algorithms and Computation (WALCOM '08), pages 200-209.

11. R. Dorrigiv and A. López-Ortiz. *Closing the gap between theory and practice: new measures for online algorithm analysis.* In Proceedings of the 2nd Workshop on Algorithms and Computation (WALCOM '08), pages 13-24.

12. R. Dorrigiv and A. López-Ortiz. *On Developing New Models, with Paging as a Case Study.* ACM SIGACT (Special Interest Group on Automata and Computability Theory) News, to appear.

Chapter 2

Online Measures

In this chapter we formally define the competitive ratio. We also study in more detail the shortcomings of competitive analysis using paging as case study. Furthermore, we study and compare the basic features of the classical performance measures. Finally we briefly talk about the evaluation of new models.

2.1 Competitive Ratio

Competitive analysis was a major breakthrough in the study of online algorithms. It can be derived from the observation that an online algorithm, in essence, computes a partial solution to a problem using incomplete information. Then, it is only natural to quantify the performance drop due to this absence of information. That is, we compare the quality of the solution obtained by the online algorithm with the one computed in the presence of full information, namely that of the optimal offline algorithm OPT. Without loss of generality we assume that we have a cost minimization online problem. The definitions and discussions can be extended to profit maximization problems in a straightforward way. We denote the cost of an algorithm $\mathcal{A}$ on a sequence σ by $\mathcal{A}(\sigma)$.

Definition 2.1. *An online algorithm $\mathcal{A}$ is said to have competitive ratio c if*

$$\mathcal{A}(\sigma) \leq c \cdot \text{OPT}(\sigma)$$

for all sequences σ.

Some of the early literature considers only algorithms with constant competitive ratio, and all others are termed as algorithms with *unbounded* competitive ratio. Alternatively, we can define a $C(n)$-competitive algorithm as follows.

Definition 2.2. *An online algorithm $\mathcal{A}$ is said to have competitive ratio $C(n)$ if, for all sequences σ we have:*

$$\mathcal{A}(\sigma) \leq C(|\sigma|) \cdot \text{OPT}(\sigma).$$

This definition can be relaxed to describe the asymptotic competitive ratio.

Definition 2.3. *An online algorithm $\mathcal{A}$ is said to have asymptotic competitive ratio $C(n)$ if*

$$\mathcal{A}(\sigma) \leq C(|\sigma|) \cdot \text{OPT}(\sigma) + b$$

for all sequences σ and a constant b.

Equivalently, using the more conventional ratio notation, we have that an algorithm is $C(n)$–competitive if and only if

$$C(n) = \max_{|\sigma|=n} \left\{ \frac{\mathcal{A}(\sigma)}{\text{OPT}(\sigma)} \right\}.$$

We note that this definition has three key components: the max operator, the online cost in the numerator and the optimal offline cost in the denominator. To better understand their role, in the next section we briefly review three well known performance measures used in classical algorithm analysis. Highlighting the differences between these well known measures will help us understand the motivations behind some of the alternative measures we will describe in Chapter 3.

Randomized Algorithms. Unlike deterministic algorithms, competitive analysis of randomized algorithms does not have a single definition. This is because we can have three different adversary models in this case [30]. A randomized online algorithm $\mathcal{A}$ makes some random choices while serving a sequence. In all three models, the adversary knows $\mathcal{A}$ and the probability distribution of its actions on any sequence σ. The distinction comes from the extent to which the adversary knows the outcome of random choices made by $\mathcal{A}$. An *oblivious adversary* does not know the specific actions taken by $\mathcal{A}$ on σ. Thus it does not know the outcome of random choices made by $\mathcal{A}$. We say that $\mathcal{A}$ has asymptotic competitive ratio $C(n)$ against an oblivious adversary if there exists a constant b such that for all sequences σ,

$$E[\mathcal{A}(\sigma)] \leq C(|\sigma|) \cdot \text{OPT}(\sigma) + b,$$

where the expectation is taken over the random choices made by $\mathcal{A}$.

In contrast, an *adaptive adversary* knows the outcome of random choices made by $\mathcal{A}$. Thus it knows the exact actions taken by $\mathcal{A}$ at each step and can use this knowledge to choose a worst-case sequence τ. We have two types of adaptive adversaries, which differ in the cost model. The cost of an *adaptive-offline adversary* on a sequence σ is $\text{OPT}(\sigma)$, i.e., the cost of an optimal offline algorithm on σ. We say that $\mathcal{A}$ has asymptotic competitive ratio $C(n)$ against an adaptive-offline adversary if there exists a constant b such that for all sequences σ,

$$E[\mathcal{A}(\sigma) - C(|\sigma|) \cdot \text{OPT}(\sigma)] \leq b,$$

where the expectation is taken over the random choices made by $\mathcal{A}$. An *adaptive-online adversary* should serve each request in an online manner and its cost is defined accordingly. The competitiveness against an adaptive-online adversary is the same as the one against an adaptive-offline adversary, except that $\text{OPT}(\sigma)$ is replaced by the cost of an adaptive-online adversary on σ.

Thus an adaptive-offline adversary is stronger than an adaptive-online adversary, which in turn is stronger than an oblivious adversary. Oblivious adversaries are analogous to adversaries for randomized offline algorithms and are the most common in analysis of randomized online algorithms. We only consider oblivious adversaries in this thesis. Thus whenever we talk about competitiveness of randomized online algorithms, we implicitly assume an oblivious adversary.

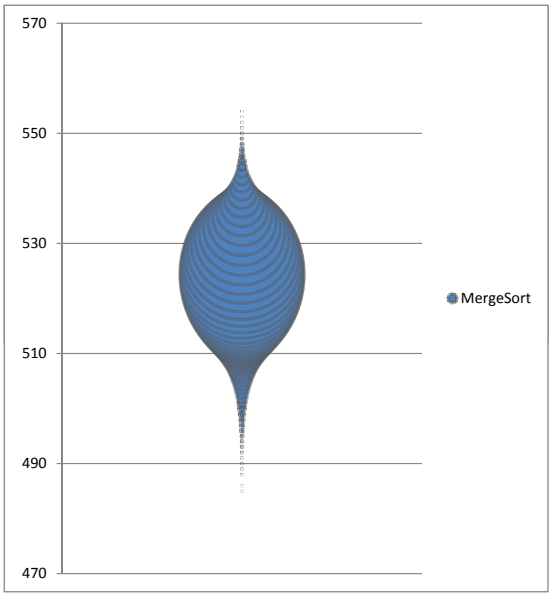

Figure 2.1: Performance of MERGESORT over a billion random permutation of 100 numbers.

2.2 Classical Performance Measures

In this section we discuss the basic features of the classical model with the aim of understanding the differences between the various models. Classical analysis of algorithms uses primarily the worst-case measure to quantify the performance of an algorithm. We briefly review the underlying assumptions and highlight certain key aspects of the measure. The driving motivation for algorithm analysis is to understand, quantify and compare the relative performance of a set of algorithms for a given problem. Traditionally, there has been a preference for measures which return a single numerical value. Such a measure would naturally induce a linear order in the set of algorithms thus allowing for ready comparisons between them. However, as we know, for the case of algorithms in general it is not possible to reduce their performance down to a single number. Hence we must consider measures that are functions of one (or more) parameters, and introduce a method for making pairwise comparisons.

Formally, given an algorithm $\mathcal{A}$ and an input x, $\mathcal{A}\langle x\rangle$ denotes the execution of $\mathcal{A}$ over x. A performance measure μ associates a real positive number to $\mathcal{A}\langle x\rangle$. Typical examples of μ are time, space usage, number of random bits used, and number of I/O operations.

When analyzing an algorithm we aim to determine the value of μ for a given input in as accurate a manner as feasible. Observe that the domain of $\mu(\cdot)$, which consists of all binary strings representing well formed inputs, lacks regularity. This makes it difficult to evaluate or

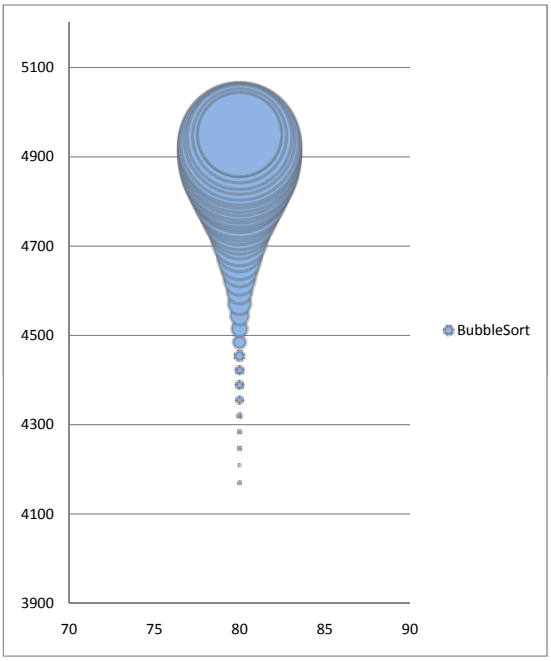

Figure 2.2: Performance of BUBBLESORT over a billion random permutation of 100 numbers.

estimate $\mu(\mathcal{A}\langle x\rangle)$ for a given input x. Hence, we group inputs by size. This draws on the familiar notion developed in elementary school that for a fixed type of problem—say addition or multiplication—its complexity increases with the number of digits in the input. In other words this assumption in the model is at some level naturally supported by experience.

With the grouping by size assumption in place we can now define a new measure—termed a timing function—from the positive integers to the real numbers, i.e. $T_{\mathcal{A}} : \mathbb{Z}^+ \to \mathbb{R}^+$. The function $T_{\mathcal{A}}(n)$, the worst-case time of algorithm $\mathcal{A}$, is defined as

$$T_{\mathcal{A}}(n) = \max_{|x|=n}\{T_{\mathcal{A}}(x)\},$$

where $T_{\mathcal{A}}(x) = \mu(\mathcal{A}\langle x\rangle)$ denotes the time taken by algorithm $\mathcal{A}$ to run on input x.

Note that the worst-case instances for two algorithms can be different. Thus when we compare two algorithms according to worst-case analysis, we might compare the performance of them on different sequences. Although worst-case analysis might seem too pessimistic, it is the standard measure for analysis of algorithms. This is partly because it guarantees the performance of the algorithm in all situations. Furthermore, it is an abstraction that simplifies analysis of algorithms (together with asymptotic analysis).

Worst-case analysis has come to be seen as synonymous to the performance of an algorithm. However, it is important to remember that it is in fact an imprecise approximation and for some

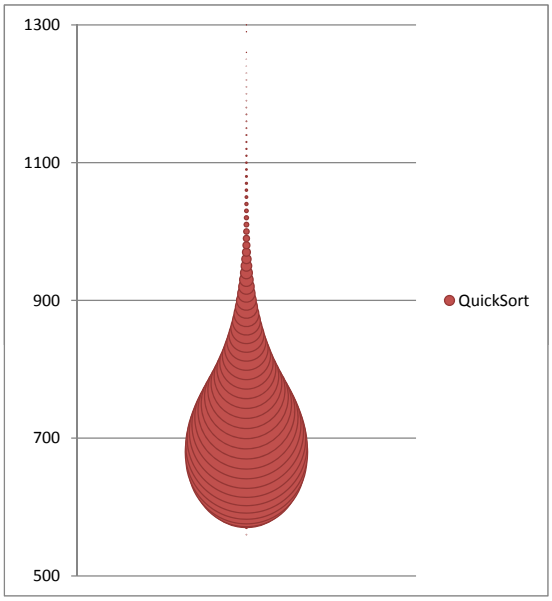

Figure 2.3: Performance of QUICKSORT over a billion random permutation of 100 numbers.

applications some other approaches might be preferable. Indeed, the amount of information lost in this reduction varies significantly with the algorithm. We illustrate this with three algorithms: MERGESORT, BUBBLESORT with early termination, and QUICKSORT. Figures 2.1, 2.2, and 2.3, show the performance of these algorithms over a billion random permutations of 100 numbers. Each dot in the tear drop shape represents the number of comparisons required by an actual input among the billion permutations. We can see that the performance for MERGESORT over most inputs is strongly clustered around 525 ± 10 comparisons—which is reasonably close to the predicted worst-case value of 664 comparisons (see Figure 2.1). The actual observed worst case for MERGESORT over a billion permutations was 554 comparisons. For BUBBLESORT with early termination we have a similar situation where the performance is strongly clustered around 4920 ± 20 comparisons which is very close to the worst case performance of 4950 comparisons (see Figure 2.2). Moreover the worst case bound of 4950 was observed in practice for several of the permutations generated at random. Lastly we have QUICKSORT. For this algorithm we can see that the worst case performance is a terrible representative of its actual performance. QUICKSORT timings are strongly clustered around 700 ± 40 comparisons but the worst case prediction is 4950 comparisons! Even the observed worst case over a billion permutations which was on itself very rare, required 1427 comparisons, falling well short of the predicted worst case.

While this is well known it illustrates that worst case performance is, in all three cases, just an approximation of the actual behaviour we were interested in studying and that the quality of this approximation is variable. Note that the fact that this bound on the performance has been

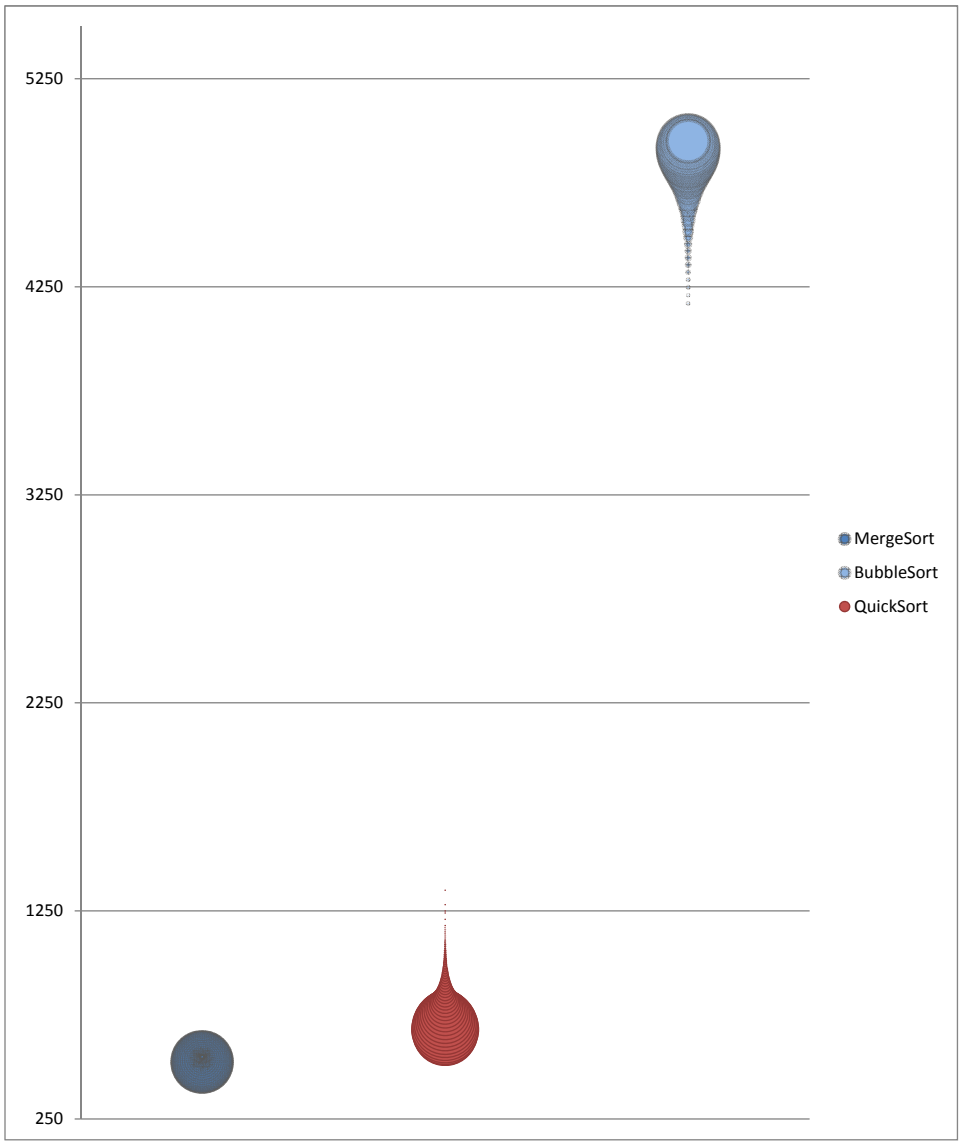

Figure 2.4: Relative performance of sorting algorithms over a billion random permutation of 100 numbers.

mathematically proven is orthogonal to the quality of this representation. The example given also illustrates the ways in which a model or measure can fail and yet still be the preferred tool. In Section 2.5 we will discuss this in more detail.

Observe that in the definition of worst case analysis there are several choices to be made. First we have the choice of measure. For example, by replacing the time measure function μ with a function μ' measuring space or I/O operations we obtain the standard worst case space and I/O-model analysis, respectively. Second we have the choice of the worst case max operator. We could use instead the average operator which gives average-case analysis for any of the measures listed above. In particular we can define average-case running time as follows.

Definition 2.4. *The average-case time of $\mathcal{A}$ under a uniform distribution is defined as*

$$\overline{\mathcal{A}}(n) = \text{avg}_{|x|=n}\{T_{\mathcal{A}}(x)\} = \frac{\sum_{|x|=n} T_{\mathcal{A}}(x)}{\#\{|x| = n\}},$$

where # denotes the cardinality function.

In the case of a general distribution, the average-case time of $\mathcal{A}$ is

$$\overline{\mathcal{A}}(n) = E(T_{\mathcal{A}}(x) \mid |x| = n) = \sum_{|x|=n} T_{\mathcal{A}}(x) Pr(x),$$

where E denotes the expectation function of classical probability. Observe that max, avg, and $E(\cdot)$ are aggregate functions over all inputs of size n. As stated before, the main drawback of average-case analysis is that the input distribution is usually unknown in practice.

Another choice that is made in the definition of worst-case analysis is grouping by input size. There are alternatives to this grouping though none as commonly used as those for the previous two choices. For polynomial algorithms there is *adaptive analysis*. This measure is used on problems in which much simpler instances appear frequently. The idea is to require good performance on all inputs, as compared to only on the worst-case or the average-case. Adaptive analysis takes into account the difficulty of input instances. This means that an algorithm has better performance according to adaptive analysis if it performs well on "easy" instances and not too poorly on "difficult" ones. We define the adaptive performance of an algorithm by normalizing its performance by the difficulty of input. The two main challenges of adaptive analysis are to find a realistic difficulty measure for input instances and to propose algorithms that perform well under such a measure. For example see the survey by Estivill-Castro and Wood [73] for several difficulty measures for the sorting problem.

Definition 2.5. *An algorithm $\mathcal{A}$ is said to be adaptive with respect to a measure $V(x)$ if $T_{\mathcal{A}}(x) = O(V(x))$ for all input x or equivalently*

$$\max_{\forall x} \left\{ \frac{T_{\mathcal{A}}(x)}{V(x)} \right\} \leq c$$

for some constant c.

This expression combines both the max aggregate operator of worst-case analysis and a comparison ratio which is reminiscent of competitive analysis.

Thus, adaptive analysis uses a measure of difficulty to group inputs by a second dimension. For example, in Table 2.1 we list the set of input permutations to a comparison based sort. Each

input size	1	2	3	4
0 transpositions	1	1 2	1 2 3	1 2 3 4
1 transposition		2 1	1 3 2 2 1 3	1 2 4 3 1 3 2 4 2 1 3 4
2 transpositions			2 3 1 3 1 2	1 3 4 2 1 4 2 3 2 1 4 3 2 3 1 4 3 1 2 4
3 transpositions			3 2 1	1 4 3 2 2 3 4 1 2 4 1 3 3 1 4 2 3 2 1 4 4 1 2 3
4 transpositions			3 1 2	2 4 3 1 3 2 4 1 3 4 1 2 4 1 3 2 4 2 1 3
5 transpositions				3 4 2 1 4 2 3 1 4 2 1 3
6 transpositions				4 3 2 1

Table 2.1: Number of transpositions needed to sort an input of size n for $n = 1, 2, 3$, and 4.

column is grouped by the standard measure of input size but further subdivided by the complexity measure of transpositions required to obtain the sorted sequence. Observe that trivially all sequences in the first few k rows, for k constant, can be sorted in time $O(n)$.

For NP-hard problems we have *parameterized analysis*, which uses a set of measures—such as treewidth—to group the inputs by a parameter other than input size. Another example is given by output-sensitive algorithms in which the input space is partitioned by both the size of the input and the size of the output produced by a particular input string.

Observe that these performance measures have, as in the case of the online competitive ratio, three key choices: (i) an aggregate function over inputs; (ii) a numerator with possibly its own aggregation function; and in the case of the average and adaptive measures (iii) a denominator with its own aggregation function and involving an external measure.

2.2.1 Probability Measures

The partition of the input space by size is motivated by the goal of introducing structure to the input space as well as obtaining a mapping from the reals to the reals, since real valued functions are more amenable to analysis. Note that this situation parallels the use of random variables in probability. In this case the probability space often is similarly not amenable to study due to its lack of structure and hence the notion of random variable is introduced. A random variable transforms the space of events (which could as varied as a set of coin tosses or a sample subset of people) into classes which are represented by the pre-image of their numerical value, i.e. the set of events $X^{-1}(r)$ for $r \in R$.

In practice, in nearly all instances the random variable X is some sort of natural counting function, yet the formal definition chooses to ignore this fact and allows for any arbitrarily defined random variable. Contrast this with the analysis of algorithms in which we selected the most natural grouping function (input size). This grouping has only relatively recently been expanded to include a second parameter such as difficulty of the input, treewidth or output size. At the same time it has become increasingly clear that there is a large number of such grouping functions which are natural and worthy of study[1].

Similarly the measurement function Pr which also has a very natural interpretation in mathematics is left unspecified by the axiomatization of Kolmogorov and is only required to be a measurement function assigning the value of one to the entire sample space. In turn in computer science we selected a specific random-variable-of-sorts namely the input size, and then a specific measure or density function, namely max. Thus far the field has only hesitantly considered variations of those, as discussed before. This is in stark contrast to the probabilistic definition in which no judgement is made as to the specific choice of parameters.

In practice, the most commonly used measures aside from time are space and number of I/O operations. In this it can be argued that while this set might not yet be complete it is unlikely to contain many other functions and that by keeping the set of possible measures small leads to stronger theorems, just as in probability theory often theorems assume a specific type of distribution or at least independent identically distributed variables. So in all, the fact that a narrower choice was made in algorithms analysis is not necessarily bad, but it is important to keep in mind that this choice was made and expand this selection if need be.

[1]Indeed, at a recent Dagstuhl meeting on fixed parameter tractability over a dozen different parameters were identified by the attendants as being in common use in the field.

Observe that the competitive ratio can also be understood in this light. In its original form (Definition 2.1) it simply computes the maximum of a normalized measure over the entire input space. As stated before, this definition was enlarged (Definition 2.2) to include a partition of the input space which allows us to describe an algorithm as, say $\log n$-competitive or $\sqrt{n}$-competitive.

Following the same outline, we can generalize the original definition of adaptive algorithms (Definition 2.5) to the notion of $f(n, V(x))$-adaptive algorithms which are those such that $T_{\mathcal{A}}(x) = f(n, V(x))$, for all inputs x of length n. Again this can be expressed as a ratio

$$\max_{|x|=n} \left\{ \frac{T_{\mathcal{A}}(x)}{V(x)} \right\} = f(n, V(x)),$$

which gives a bivariate timing function. Fixed parameter algorithms, approximation algorithms and output-sensitive algorithms also use a second parameter to refine the partition of the input space. For example, consider the performance of Quicksort as shown in Figure 2.3. A well chosen difficulty parameter would partition the performance teardrop into narrower ranges each of which would then presumably be estimated much more accurately by its corresponding maximum value.

In sum, classical measures have made certain parameter choices, which while natural and useful are not necessarily unique. In cases where these choices led to undesirable results one can either perfect the classical model, or introduce a novel measure to handle the anomalous cases. Past experience shows that properly chosen variants have led to interesting and novel algorithms and results.

2.3 Online Performance Measures

As stated before, competitive analysis has some drawbacks. In this section we try to find the roots of these drawbacks and general approaches to overcome them. Consider again the competitive ratio measure $C(n)$:

$$C(n) = \max_{|\sigma|=n} \left\{ \frac{\mathcal{A}(\sigma)}{\mathrm{OPT}(\sigma)} \right\}.$$

This measure suffers in practice from three aspects. One of them is that the denominator is an offline algorithm that has full knowledge of the future request sequence and unbounded computational power. Indeed, in some cases computing the optimal offline algorithm is non-recursive! In certain instances the comparison with such a powerful adversary leads to algorithms of varying degrees of sophistication having the same equally bad competitive ratio.

The other problem with the measure is its pessimistic nature: It concentrates on a single worst-case sequence. For example an algorithm that has good performance in all but one rather complicated instance has the same competitive ratio as an algorithm that always makes a bad decision (even on "trivial" instances) so long as the bad decisions are never worse than that of the rare worst case of the preferred algorithm.

Another problem with competitive analysis is that it cannot be used to directly compare two online algorithms: it uses the concept of an optimal offline algorithm as a baseline for comparing online algorithms. While this may be convenient, it is rather indirect: one could argue that in comparing two online algorithms, all we need to study is the relative cost of the algorithms on the request sequences.

Theoretical Model	Systems Framework
Competitive ratio framework	Fault rate measure
Worst-case analysis	Typical case analysis
Marking algorithms optimal	LRU and variants thereof are best
In practice LRU is best	LRU is impractical
LFD is offline optimal	No analogous concept
Competitive ratio is k	Comp. ratio is at most 4
User is a malicious adversary	User (compiler/programmer) seeks locality of reference
No benefit from lookahead	Lookahead helps

Table 2.2: Contrast of theory versus practice for paging.

Therefore several alternative measures for the quality of online algorithms have been proposed [29, 47, 39, 111, 110, 79, 95, 36, 97, 54, 153]. We study some of these alternatives in Chapter 3. In general, known alternative approaches rely on one or more of the following:

1. Defining a new measure as a substitute for the competitive ratio
2. Limiting the power of the adversary
3. Employing different definitions for the concept of the "cost" of an algorithm
4. Incorporating certain assumptions concerning the request sequences
5. Directly comparing online algorithms, i.e., without a reference to OPT

2.4 Theory versus Practice

As was mentioned earlier, competitive analysis of paging does not lead to satisfactory conclusions which are replicated in practice. With the goal of closing the gap between theory and practice, we examine the difference in assumptions between the theoretical competitive ratio model and the practical systems research approach to paging. We now discuss in detail the differences which appear summarized in Table 2.2.

1. The theoretical model for the study of paging algorithms is the competitive ratio framework, in contrast, the vast majority of systems research on paging uses the fault rate measure, which simply determines the percentage of page requests leading to a page fault. Consider for example a request sequence of 1000000 pages, such that an online algorithm $\mathcal{A}$ has 200 page faults while OPT has twenty faults. This means that $\mathcal{A}$ has a competitive ratio of 10 which is high, while in terms of the fault rate model $\mathcal{A}$ has a page fault rate of 0.02% which is very good. Unfortunately, we cannot use (worst-case) fault rate alone to get meaningful theoretical results about online paging algorithms, as it is still a pessimistic measure. However, we will see in Section 3.14 that the fault rate measure can be combined with some alternative models to give informative results.
2. In the worst case one can devise highly contrived request sequences with a very high competitive ratio for any paging algorithm. Since these sequences do not occur naturally, measuring

the performance of an online algorithm using them does not shed light on the actual relative performance of various algorithms. Practical studies in contrast use an extensive set of real-life request sequences (traces) gathered from diverse set of applications, over which the performance of any online strategy can be measured.

3. It has been empirically well established that LRU (and/or minor variants thereof) are, in practice, preferable paging strategies to all other known paging algorithms [139]. In contrast, the competitive ratio of LRU is the same as FIFO, and it is worse than that of some randomized algorithms. Actually, LRU has the same performance as the naïve algorithm FWF under competitive analysis. The competitive ratio then fails to separate between these algorithms with very different performance in practice. This is one of the most noticeable drawbacks of competitive analysis; most alternative measures can separate the performance of LRU and that of FWF.

4. In terms of practice the theoretical model suggests that LRU might be preferable for "practical" heuristic reasons. In actuality, since paging algorithms are executed concurrently with every page access this limits the complexity of any solution, and hence practical heuristic solutions are simplifications and approximations of LRU.

5. Competitive analysis uses an optimal offline algorithm as a baseline to compare online algorithms. While this may be convenient, it is rather indirect: one could argue that in comparing A to B all we need to study is the relative cost of the algorithms on the request sequences. The indirect comparison to an offline optimal can introduce spurious artifacts due to the comparison of two objects of different types, namely an online and an offline algorithm [2]. As well the offline optimum benefits from aspects other than the difficulty of the instances, namely it can take advantage of knowledge of the future, so regardless of the difficulty of servicing a request it might do better as a consequence of this. In contrast, the fault rate measure uses a direct comparison of the number of faults per access of paging algorithms to determine which one is preferable. Some other models, e.g., the relative worst order ratio [39] and the Max/Max ratio [29] allow for direct comparison of online algorithms.

6. Interestingly, even if algorithms are measured using the competitive ratio, in practice the worst-case request sequence encountered using LRU has (empirical) competitive ratio 4, and most sequences have competitive ratio between two and four [153]. Experimental results suggest that the empirical competitive ratio of LRU is a small constant independent of k [35]. Contrast this with the predicted competitive ratio of k under the theoretical model. Several alternative measures address this issue by proving constant performance ratios for LRU, e.g., loose competitiveness [156] and adequate analysis [129].

7. The offline optimum model implicitly creates an adversarial model in which the paging algorithm must be able to handle all request sequences, including those maliciously designed to foil the paging algorithm. In contrast, in real life, programmers and compilers purposely avoid bad request sequences and try to arrange the data in a way so as to maximize locality of reference in the request sequence (e.g. I/O model [3], or the cache oblivious model [130]). In game theoretical terms, the theoretical competitive model is a zero sum game in which

[2] To illustrate, consider a consumer wishing to purchase a mountain bike. There are two choices which the user evaluates indirectly by comparing them to an "optimal" racing bike. While in general good racing bikes and mountain bikes have common characteristics, such a comparison would award no points for shock absorbers. Similarly, lightness, which is essential in a racing bike is secondary to sturdiness in the case of the mountain bike, and so on and so forth.

the adversary benefits from a badly performing paging algorithm, while in practice paging is a positive sum game in which both the user and the paging algorithm can maximize their respective performances by cooperating and coordinating their strategies. Indeed it has been observed [66] that paging algorithms optimize for locality of reference because this was first observed in real life traces, and now compilers optimize the code to increase locality of reference because those are the sequences over which paging algorithms excel.

8. Lastly, as stated in Chapter 1, finite lookahead does not help in the theoretical model, yet in practice instruction schedulers in many cases know the future request sequence for a small finite lookahead and can use this information to improve the fault rate of paging strategies. This drawback has been addressed in two ways. One way is to propose alternative definitions for lookahead (see Section 3.15.). The other is to use alternative models for analysis of online algorithms: the relative worst order ratio [39], the Max/Max ratio [29], the full access cost model [148], and the comparative ratio [111] reflect the influence of lookahead.

2.5 Evaluating New Models

In the rest of thesis we will see several models for analysis of (online) algorithms. In this section we discuss key aspects of the introduction and evaluation of new models to illustrate the process by which new models become adopted. As observed before, proposing a new model or measure is relatively rare, and widespread adoption of one is even rarer. As such there is a paucity of established techniques on how to gauge the usefulness and validity of a proposed new model.

Often there is a tendency to think in terms of "competing models" as if there were a unique best model out there waiting to be found. In practice, different models are used in different settings depending on the information being sought and the specific application in mind. For example if we are sorting a given number of keys, we might choose to study the problem under the RAM model, the I/O model or the MapReduce model if the number of keys is modest, large or massive, respectively. That is to say, to adopt the MapReduce model we do not need to disprove the validity of the RAM model throughout. All that is required is for the MapReduce model to produce more accurate estimates of the algorithm performance for the particular set of instances that we are interested in, and then we use this model for exactly those instances.

Furthermore, newly introduced models often have to be evaluated on their promise as it might take the combined work of several research efforts before their full potential is grasped[3]. Indeed a well-honed and understood inferior model will initially outperform an ultimately preferable new model that is yet to be fully developed and perfected.

Another often overlooked factor is that models are approximations of reality, not accurate descriptions—hence the use of the word model. This was most notably stated by the mathematician George Box in his maxim "essentially, all models are wrong, but some are useful"[4] [37, page 424]. In fact models can be at times highly inaccurate provided that they fail in well understood and predictable scenarios, in which case we learn not to use them in the first place. For example, we know that the worst case measure is not adequate for the evaluation of randomized algorithms. Naturally this is not considered an indictment of the worst case measure, and rightly so.

[3] Consider for example the introduction of the notion of NP-completeness by S. Cook and the followup paper by Richard Karp the year after, discussing this idea further.

[4] Another variant, also by G. Box, is "remember that all models are wrong; the practical question is how wrong do they have to be to not be useful."

The economist Paul Krugman observed this process of internalization of the limits of well established models and overlooking of their flaws, in the context of commenting on the various models he introduced:

> The models I proposed were incomplete and yet they told meaningful stories. To achieve this aggregate level of description required accepting certain basically silly assumptions of symmetry, yet these silly assumptions tell stories that were persuasive, and that could not be told using the standard model. What I began to realize was that *in economics we are always making silly assumptions; it's just that some of them have been made so often that they come to seem natural.* And so one should not reject a model as silly until one sees where its assumptions lead. Initially your assumptions will surely look peculiar. Consider for example the Arrow-Debreu model of perfect competition with utility maximization and complete markets. This is indeed a wonderful model—not because its assumptions are remotely plausible but because it helps us think more clearly about both the nature of economic efficiency and the prospects for achieving efficiency under a market system. [*emphasis added and excerpted for brevity, see [113] for the complete quote*]

This is not to say that newly proposed models cannot be tested or evaluated. A good model shows promise even if initially flawed and should eventually lead to new insights and predictions that were not explicitly built into the model.

A good model first unifies, then explains and ultimately predicts. Ideally it begins by reducing the facts to basic, self-evident principles which are so obviously true that they verge between the trivial, the overly simplified and the downright circular.[5] Yet they readily and accurately explain the facts in ways that were not otherwise possible.

Observe that the effectiveness of a model is another important consideration. Indeed, we readily trade accuracy for effectiveness in our choice of models. We illustrated this fact with the error observed during the worst case analysis of the sorting algorithms in Section 2.2. In a more recent example we see the same tradeoff being made in the development of non-uniform access-cost memory models:

> A number of models have been proposed [...] The two most widely adopted ones are the input/output model (or I/O model) and the cache-oblivious model. Their success is due to the balance they provide between simplicity, in order to allow the design and analysis of sophisticated algorithms and accuracy in predicting the performance of algorithms on real memory hierarchies. (*excerpted from [2]*)

It is easy to underestimate the amount of precision that we routinely trade in exchange for effectiveness. In this sense almost all probabilistic models are a vast tradeoff between accuracy and effectiveness. For example, given an unlimited budget a doctor might be able to determine if a patient will or will not develop lung cancer. Instead we settle for a probabilistic statement such as "30% of former smokers do". In all three cases discussed above, even though we might be able to produce much better estimates it would be at a disproportionately higher cost and hence not done.

[5] For example, consider Newton's 1st Law: "Objects remain in motion until they are affected by an external force". What is an external force? something that affects the motion of an object.

Chapter 3

Existing Alternative Measures for the Analysis of Online Algorithms

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

3.1 Max/Max Ratio

The Max/Max ratio [29] tries to be more optimistic by comparing the amortized worst case behaviour of the online algorithm with the amortized worst case behaviour of the optimal offline algorithm. Recall that in competitive analysis we compare the two algorithms on the same sequence. However, this approach is sometimes problematic because the existence of only one bad sequence for an online algorithm can drastically change the result. This measure tries to avoid the situation in which a single unusual sequence leads to a very bad ratio. This becomes more clear with the following example used in [29] as a motivation for defining the Max/Max ratio. Consider the problem of buying an insurance policy in an online manner. It is reasonable to pay \$5 a month to insure your car against theft. However, this is not a competitive strategy because the offline adversary can select the scenario in which one will never present a claim to the insurance agent. Conversely if no theft insurance is purchased, then the adversary can select the scenario in which the car is stolen. In the Max/Max ratio, we compare the two algorithms on their respective worst case sequences of the same length.

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

Definition 3.1. *The amortized cost of $\mathcal{A}$ is defined as $M(A) = \limsup_{n\to\infty} M_n(\mathcal{A})$ where $M_n(\mathcal{A}) = \max_{|\sigma|=n} \mathcal{A}(\sigma)/n$. The Max/Max ratio of $\mathcal{A}$ denoted $w_M(\mathcal{A})$ is*

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

where OPT *is an optimal offline algorithm.*

Note that we can directly compare two online algorithms $\mathcal{A}$ and $\mathcal{B}$ using this measure because we have $\frac{M(\mathcal{A})}{M(\mathcal{B})} = \frac{w_M(\mathcal{A})}{w_M(\mathcal{B})}$. Also when considering the Max/Max ratio, lookahead can improve the online performance even in cases where the competitive ratio does not improve [29]. In Chapter 4 we will see that all lazy paging algorithms are equivalent according to the Max/Max ratio.

3.2 Random Order Ratio

The random order ratio [110] is another measure that tries to decrease the dependence on some unusual bad sequences. Let $\mathcal{A}$ be an online algorithm for a minimization problem.

Definition 3.2. *The* random order ratio *of $\mathcal{A}$ is defined as*

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

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

Therefore this measure assumes that all orderings of an input sequence are equally likely. This is a reasonable assumption for some online problems. Kenyon [110] proves a lower and an upper bound on the random order ratio of the Best-Fit algorithm for online bin-packing which are better than the corresponding competitive ratio bounds. However, it seems that this measure is difficult to generalize to other online problems.

3.3 Relative Worst Order Ratio

The relative worst order ratio [39, 48, 40] combines some desirable properties of the Max/Max ratio and the random order ratio. Using this measure we can directly compare two online algorithms. Informally, for a given sequence it considers the worst case ordering of that sequence for each algorithm and compares their behaviours on these orderings. Then it finds among all sequences (not just reorderings) the one that maximizes the worst case performance. Thus this measure can be considered as a modification of the Max/Max ratio in that we consider the worst sequence among those which are permutations of each other instead of considering the worst sequence among all those having the same length as the Max/Max ratio does. It is also related to random order ratio as it considers permutations of a sequence. However instead of taking the expectation of the algorithm's behaviour on all permutations, it considers the permutation with the worst behaviour.

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

Definition 3.3. *[40] Let $S_1(c)$ and $S_2(c)$ be the statements about algorithms $\mathcal{A}$ and $\mathcal{B}$ defined in the following way.*

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

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

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

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

The idea behind this measure is that some online algorithms perform well on some types of sequence orderings and other algorithms perform well on some other types of orderings. Therefore certain algorithms that cannot be compared using competitive analysis may be comparable in this measure. Boyar and Favrholdt show that the relative worst order ratio is transitive [39].

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

In [48], three online algorithms (First-Fit, Best-Fit, and Worst-Fit) for two variants of the seat reservation problem [44] are compared using the relative worst order ratio. All of these three algorithms can be compared in this framework while they have the same performance within the classical competitive analysis framework.

For paging algorithms, LRU is strictly better than FWF with respect to the worst order ratio [40], while these two algorithms have the same competitive ratio. Also a new paging algorithm, Retrospective-LRU (RLRU), is proposed and it is shown that RLRU is better than LRU with respect to the relative worst order ratio. This is in contrast to what the competitive ratio of these algorithms predicts. It is also shown that lookahead is helpful when we consider the relative worst order ratio.

3.4 Loose Competitiveness

Loose competitiveness was first proposed in [153] and later modified in [156]. We describe it for an online minimization problem. It attempts to obtain a more realistic measure by considering the following two aspects in the analysis of online algorithms. First, in many real-life online problems, we can ignore those sequences on which the online algorithm incurs a cost less than a certain threshold. Second, many online problems have a second resource parameter (e.g. size of the cache, number of servers) and input sequences are independent of these parameters. In contrast, in competitive analysis the adversary can select sequences tailored against those parameters. We clarify this situation by considering the paging problem. In this case the problem parameter is k, the size of the cache. Consider the following lower bound on the competitive ratio of any deterministic paging algorithm.

Theorem 3.1. *[140] The competitive ratio of any deterministic online paging algorithm is at least* k.

This result can be easily proven by considering an adversary that selects only $k+1$ pages and at each time requests a page that is not in the cache. For this to work the adversary needs to know the problem parameter k. However, in practice the competitive ratios of many online paging algorithms have been observed to be constant [156], i.e., independent of k. This can be obtained by applying loose competitiveness.

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

Definition 3.4. *[156] An algorithm $\mathcal{A}$ is (ε, δ)-loosely c-competitive if, for any input sequence σ and for any n, at least $(1-\delta)\,n$ of the values $k \in \{1, 2, \ldots, n\}$ satisfy*

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

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

Theorem 3.2. *[156] Every k-competitive paging algorithm is (ε, δ)-loosely c-competitive for any $0 < \varepsilon, \delta < 1$, and $c = (e/\delta)\ln(e/\varepsilon)$, where e is the base of the natural logarithm.*

3.5 Accommodating Ratio and Accommodating Function

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

Now for the accommodating ratio [44] we consider those sequences that can be packed in n bins by a fair optimal offline algorithm. In general we only consider those sequences in which the optimal offline algorithm does not benefit from having more resources than those already available. For the accommodating function [47, 41] we only consider those sequences that can be packed in αn bins by a fair optimal offline algorithm OPT. Boyar et al. [47] argue that if we allow large values of α, then we will have sequences that cannot be handled very well by OPT and this can be unrealistic for some applications. Thus they consider small values of $\alpha \geq 1$.

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

Definition 3.5. *[41] Let Π be an online problem with a fixed amount n of resources. For any $\alpha > 0$, an input sequence σ is said to be an α-sequence, if $\text{OPT}_{\alpha n}(\sigma) = \text{OPT}_{n'}(\sigma)$, for all $n' \geq \alpha n$ (1-sequences are also called accommodating sequences).*

Definition 3.6. *Algorithm $\mathcal{A}$ is c-competitive on α-sequences, if there exists a constant b, such that $\mathcal{A}(\sigma) \leq c \cdot \text{OPT}(\sigma) + b$, for any α-sequence σ. The accommodating function is defined as*

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

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

Several papers [44, 46, 18, 17, 72, 49, 16] use the accommodating ratio as the measure of quality for online algorithms. Boyar et al. [44] give lower bounds and upper bounds for the competitive ratio and accommodating ratio of two versions (unit price and proportional price) of the *seat reservation* problem. They prove these results for Best-Fit, First-Fit, and general deterministic fair algorithms. The two measures (the competitive ratio and the accommodating ratio), agree on the proportional price problem but differ on the unit price problem. The bounds are tight for the proportional price problem, but not in the unit price problem. Bach et al. [18, 17] give tight bounds for the deterministic fair algorithms for the unit price problem. They also consider randomized algorithms and prove some bounds on them.

Several algorithms for the online fair bin packing problem are analyzed in [45]. Upper bounds and lower bounds are given for the accommodating ratio for First-Fit, Worst-Fit, and general algorithms. The lower bound for First-fit is improved in [49]. According to these bounds, First-Fit has strictly better accommodating ratio than Worst-Fit. However if we consider the standard competitive ratio, it can be shown [46] that Worst-Fit behaves strictly better than First-Fit. Therefore the competitive ratio and the accommodating ratio can give contradictory results. Epstein and Favrholdt [72] consider a variation of online fair bin packing in which bins can have different sizes and give upper bounds and lower bounds for the accommodating ratio of several online algorithms.

The unrestricted bin-packing problem is the same as the fair bin packing problem except that we do not require the algorithms to be fair. Azar et al. [16] study this variation of the problem and compare it with fair bin packing using the accommodating ratio. They prove an asymptotically tight bound for the accommodating ratio of First-Fit for the fair bin packing problem. They design an online algorithm called Unfair-First-Fit which has asymptotically better accommodating ratio than First-Fit in the unrestricted bin packing problem. Finally upper bounds on the accommodating ratio of deterministic and randomized algorithms are proven for the unrestricted bin packing problem.

The accommodating function is studied by Boyer et al. for the fair bin packing problem [47]. They prove lower and upper bounds on the accommodating functions of First-Fit, Worst-Fit, and all deterministic fair algorithms for all $\alpha \geq 1$. A variant of the seat reservation problem in which seat changes are allowed is studied in [43]. Lower bounds and upper bounds for the competitive ratio, accommodating ratio, and accommodating function are proven for several algorithms. Finally, Boyer et al. [41] extend the accommodating function to values of $\alpha < 1$. They study the accommodating function of several algorithms for the seat reservation and unrestricted

bin packing problems. For the seat reservation problem, they show that we can separate the performance of three algorithms using the accommodating function at $\alpha = 1/3$, while we cannot do the same using the competitive ratio or the accommodating ratio. They also studied the connection between the accommodating function and the resource augmentation technique [101].

3.6 Access Graph Model

The access graph model was introduced by Borodin et al. to solve two main problems in competitive analysis of online paging algorithms [36]. One of these problems is that the practical performance ratio of LRU is much better than its competitive ratio. We have mentioned the second problem before: Although LRU and FIFO have the same competitive ratio, LRU behaves much better than FIFO in practice. One reason that LRU has good experimental behaviour is that in practice page requests show *locality of reference*. Temporal locality means that when a page is requested it is more likely to be requested in the near future. Spatial locality means that when a page is requested it is more likely that a nearby page will be requested in the near future.

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

Using this model, several interesting results can be obtained [36, 97, 54, 78]. For every graph G and every number k of pages in the fast memory, let $c_k(G)$ denote the best competitive ratio that can be achieved by an online paging algorithm. Borodin et al. prove that the value $c_k(G)$ is computable for every finite access graph G [36]. They also show how to compute the competitive ratio of LRU for every access graph and every k within a factor of two (plus additive constant), and propose a simple algorithm that nearly achieves the best competitive ratio for every access graph. This algorithm, called FAR, evicts, on each fault, the unmarked page in cache whose distance from a marked page is maximum in the access graph. They proved that the competitive ratio of FAR for every undirected access graph and every k is within $O(\log k)$ of the best possible competitive ratio. This was later improved by Irani et al. who showed that the competitive ratio of FAR is $O(c_k(G))$ for any undirected graph G [97]. Experimental results of [36] show that some variations of FAR behave better than LRU in practice. It is also known that the competitive ratio of LRU is at least as good as FIFO on every access graph [54].

Karlin, Phillips, and Raghavan [109] extended the access graph model by assigning probabilities to the edges of the graph. In other words, they assume that the request sequences are generated by a Markov chain process on the set of pages. Given such a Markov chain, the objective is to find an efficient online algorithm that minimizes the expected number of faults per request. They provide an efficient algorithm that achieve bounds within a constant of the best online algorithm.

3.7 Diffuse Adversary Model

The diffuse adversary model [111] tries to refine the competitive ratio by decreasing the power of the adversary. It does this by restricting the set of legal input sequences. Recall that in standard competitive analysis we do not put any restriction on the input sequences and so they can have any distribution. In other words, the online algorithm knows nothing about the distribution of the input sequences. At the other end of the spectrum, in classic average-case analysis of online algorithms, the exact distribution of input sequences is known to the online algorithm. In the diffuse adversary model, the online algorithm does not know the exact distribution, but it knows that it is a member of a class Δ of distributions.

Definition 3.7. *Let $\mathcal{A}$ be an online algorithm for a minimization problem and let Δ be a class of distributions for the input sequences. Then $\mathcal{A}$ is c-competitive against Δ, if there exists a constant b, such that*

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

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

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

This model is applied to paging [111] by considering a class Δ_ε of distributions and proving that LRU has the best competitive ratio against Δ_ε among all deterministic online algorithms. For any sequence ρ of pages and any page p, let $Pr(p|\rho)$ denote the probability that p is the next page requested provided that the request sequence seen so far is ρ. For any $0 \leq \varepsilon \leq 1$, Δ_ε contains distributions in which $Pr(p|\rho) \leq \varepsilon$ for every page p and every page sequence ρ. Young [154, 155] computed the actual competitive ratios of both deterministic and randomized algorithms against Δ_ε. He proves that around the threshold $\varepsilon \approx 1/k$, the best competitive ratios against Δ_ε are $\Theta(\ln k)$ for both deterministic and randomized algorithms. The competitive ratios rapidly become constant for values of ε less than the threshold. For $\varepsilon = \omega(1/k)$, i.e., values greater than the threshold, the competitive ratio rapidly tends to $\Theta(k)$ for deterministic algorithms while it remains $\Theta(\ln k)$ for randomized algorithms. He shows that for $\varepsilon \geq 1/k$, FIFO and FWF have competitive ratio k against Δ_ε. Thus in this case they are outperformed by LRU, which has competitive ratio $\Theta(\ln k)$.

Becchetti [25] argues that the class Δ_ε does not reflect locality of reference. He considers other classes of distributions that model locality of reference. He then proves that LRU achieves good competitive ratios against these classes, while the performance of FWF against them is bad.

3.8 Probabilistic Competitiveness

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

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

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

Definition 3.8. *[95] Let D be a probability distribution on request sequences σ. An algorithm $\mathcal{A}$ is c-competitive against D if there exists a constant, b, such that*

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

Equivalently, we can say that in this definition we compute

$$\frac{E_D[\mathcal{A}(\sigma)]}{E_D[\text{OPT}(\sigma)]}.$$

The alternative is the *instance-wise* definition.

Definition 3.9. *Let D be a probability distribution on request sequences σ. The average-case competitive ratio of an algorithm $\mathcal{A}$ against D is defined as*

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

Fujiwara and Iwama [80] apply average-case competitive analysis to the *ski-rental* problem. In this problem, a skier should decide between renting and purchasing a pair of skis. The cost of purchasing is s times the cost of renting. The problem is online: the skier does not know how many times she would go skiing. It is well known [107] that the optimal deterministic strategy is to buy the skis after $s-1$ rental times. [80] considers the following scenario: At each time, the skier quits skiing with probability $\lambda > 0$. Then the total number of ski trips follows exponential distribution $\lambda e^{-\lambda t}$. Fujiwara and Iwama prove that the algorithm with optimal average-case competitive ratio against this distribution always rents skis if $s \geq 1/\lambda$. Otherwise it buys the skis after $\approx s^2\lambda$ times. This is different from optimal algorithms in competitive analysis and standard average-case analysis frameworks.

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

Definition 3.10. *Let $\mathcal{A}$ be an online algorithm for a minimization problem and let Δ be a class of distributions for the input sequences. Then the average-case competitive ratio of $\mathcal{A}$ against Δ is defined as*

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

Souza [142] calls the competitive ratios defined in Definitions 3.7 and 3.10 *the expected competitive ratio* and *the average performance ratio*, respectively. He argues that for some applications the average performance ratio is preferable to the expected competitive ratio. Bin packing algorithms have been extensively studied using the average performance ratio, e.g., see [56] and references therein. This ratio has been applied to some scheduling problems as well [57, 135, 143]. Note that the average performance ratio is called the expected competitive ratio in these papers.

Actually, the average-case competitive ratio was first introduced in the context of competitive searches on the real line for a target of unknown location. The classic problem in this field is the cow path problem. As traditionally described, a cow reaches a fork on the road and recalls that in one and only one of the two paths there is a pasture field. This problem was first analyzed under the competitive ratio and its solution predates the introduction of the competitive ratio in online algorithms. The optimal solution under the standard competitive ratio metric is 9-competitive. However, on the average, the pasture is discovered at a cost of approximately 4.59 times the optimal path to the pasture. Interestingly enough, the strategy resulting in the optimal average cost is different from the optimal one under the competitive ratio framework [81, 114]. Formally we have

Definition 3.11. *The average competitive ratio, or* average ratio *for short is defined as*

$$E_{\forall\,|\sigma|}\left[\frac{\mathcal{A}(\sigma)}{\mathrm{OPT}(\sigma)}\right].$$

3.9 Smoothed Competitiveness

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

Smoothed analysis of algorithms [145] tries to explain this intriguing behavior without any prior assumptions on the distribution of the input instances. In this model, we randomly perturb (smoothen) the input instances according to a probability distribution f and then analyze the behavior of the algorithm on these perturbed (smoothed) instances. For each input instance $\check{I}$ we compute the neighborhood $N(\check{I})$ of $\check{I}$ which contains the set of all perturbed instances that can be obtained from $\check{I}$. Then we compute the expected running time of the algorithm over all perturbed instances in this neighborhood. The smoothed complexity of the algorithm is the maximum of this expected running time over all the input instances. Intuitively, an algorithm with bad worst case performance can have good smoothed performance if its worst case instances are isolated. Spielman and Teng show [145] that the simplex algorithm has polynomial smoothed complexity. Several other results are known about the smoothed complexity of algorithms [23, 121, 34, 144].

As stated before, competitive analysis is a rather pessimistic measure and an algorithm can have a very bad competitive ratio only because of a few bad sequences. Therefore the competitive ratio is a reasonable choice for applying smoothed analysis. This was first done by Becchetti et al. [26] who introduced *smoothed competitive analysis*. Informally, smoothed competitive analysis is the same as competitive analysis except that we consider the cost of the algorithm on randomly perturbed adversarial sequences. The smoothed competitive ratio of an online algorithm $\mathcal{A}$ for a minimization problem can be formally defined as follows.

Definition 3.12. *[26] The* smoothed competitive ratio *of an algorithm $\mathcal{A}$ is defined as*

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

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

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

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

In [26], the first definition is used but it is remarked that a similar result can be obtained using the second definition. They use the smoothed competitive ratio to analyze the Multi-Level Feedback(MLF) algorithm for processor scheduling in a time sharing multitasking operating system. This algorithm has very good performance in practice but its competitive ratio is very bad. They obtain strictly better ratios using smoothed competitive analysis than with the competitive ratio.

3.10 Search Ratio

The search ratio belongs to the family of measures in which the offline OPT is weakened. It is defined only for the specific case of geometric searches in an unknown terrain for a target of unknown position. Recall that the competitive ratio compares against an all knowing OPT. Indeed, for geometric searches OPT is simply a shortest path algorithm, while the online search algorithm has intricate methods for searching. The search ratio instead considers the case where OPT knows the terrain but not the position of the target. That is, the search ratio compares two search algorithms, albeit one more powerful than the other. By comparing two instances of like objects the search ratio can be argued to be a more meaningful measure of the quality of an online search algorithm. Koutsoupias et al. show that searching in trees exhibits a large competitive ratio regardless of the algorithm, yet under the search ratio framework certain algorithms are far superior to others [112].

3.11 Travel Cost

As stated in Chapter 2, classical complexity time analysis generally uses an unnormalized time measure even though a normalized measure has been defined and proven fruitful in certain settings. This raises the possibility of using an unnormalized cost measure for online algorithms as well. This measure has been used in online geometric searches, in which the main objective is to minimize the length of the longest search sequence, known as the travel cost of the solution. Formally,

Definition 3.13. *The* travel cost *of an online algorithm $\mathcal{A}$ on input σ is given by*

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

For example in the case of an actual search and rescue operation in the high seas minimizing the maximum search time is more relevant than the competitive ratio on any particular point in the search path [75].

3.12 Acceleration Ratio

Acceleration ratio is a worst case measure similar to the competitive ratio. It is defined for some scheduling problems related to anytime algorithms [157]. In an *anytime* algorithm, the output quality is related to the computation time. The behaviour of an anytime algorithm $\mathcal{A}$ on an instance σ is described by a *performance profile* $Q_{\mathcal{A}}(t)$, where $Q_{\mathcal{A}}(t)$ denotes the quality of the output produced by $\mathcal{A}$ in computation time t. It is usually assumed that $Q_{\mathcal{A}}(t)$ is strictly increasing. There are two kinds of anytime algorithms. In an *interruptible* algorithm you can stop the algorithm and ask the result at any time during the execution of the algorithm. On the other hand, in a *contract* algorithm the computation time (contract time) should be given as part of input before the start of its execution. The performance profile $Q_{\mathcal{A}}(t)$ for a contract algorithm $\mathcal{A}$ is the quality of output produced by $\mathcal{A}$ if the computation time t is given to $\mathcal{A}$ prior to its execution.

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

Definition 3.14. *[157] Let $\mathcal{A}$ be a contract algorithm and $\mathcal{B}$ be an interruptible algorithm produced by the sequence of contracts $X = (x_1, x_2, \dots)$, then the acceleration ratio of X, $r \geq 1$, is the smallest constant c for which:*

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

Russell and Zilberstein [134] give a schedule with acceleration ratio of 4 and Zilberstein et al. [157] show that this is the best possible. Bernstein et al. extend the definition to the case that we have m processors [32]. They also give a schedule of acceleration ratio $\frac{(m+1)^{\frac{m+1}{m}}}{m}$ for this case, which is shown to be optimal in [116].

3.13 Full Access Cost Model

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

by $F_{\mathcal{A}}(\sigma)$, is defined as $\mathcal{A}(\sigma)/|\sigma|$, where $|\sigma|$ is the length of σ. As stated before, the fault rate is usually used in systems research for analysis of paging. In Section 3.14 we describe some results using fault rate as the cost model.

Torng [148] proposes the full access cost model for paging. On a hit, the requested page is in fast memory and can be accessed in one time step. On a fault, the corresponding page should be brought to the fast memory in time p and then be accessed in time 1, where p is a parameter of model that is called the *miss penalty*. Thus the cost of a hit is 1 and the cost of a fault is $p+1$. Suppose that an algorithm $\mathcal{A}$ incurs f faults on an input sequence σ. Then the full access cost of $\mathcal{A}$ on σ is defined as $\mathcal{A}(\sigma) = (|\sigma| - f) \times 1 + f \times (p+1) = |\sigma| + fp$. Note that in the standard model the cost of a hit is 0 and the cost of a fault is 1. Also we can consider the standard model as a special case of the full access cost model in which we have $p = \infty$. As stated before this is not an unrealistic assumption as the miss penalty for the paging problem can be very large. For example according to Figure 5.37 on page 441 of [88], it ranges between 7000 and 150000 on typical systems.

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

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

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

3.14 Concave Analysis

In this section we describe a model for paging with locality of reference proposed by Albers et al. [6]. This model is based on the concept of working set by Denning [64]. The *working set* is the small set of pages used by a process at any phase of its execution. Consider a function whose value at n is the size of the working set in a window of size n. Denning [64] shows that if we assume some local statistical regularities in a request sequence, then this function would be increasing and concave.

The idea behind concave analysis [6] is that we can say a request sequence has locality of reference if the number of distinct pages in a window of size n is small. Consider a function that represents the maximum (average) number of distinct pages in a window of size n, in a request

sequence, in terms of n. Extensive experiments with real data show that this function is nearly concave for practical request sequences [6]. Let f be an increasing concave function. There are two possible ways to model locality of reference. In the *Max-Model* we say that a request sequence is consistent with f if the number of distinct pages in any window of size n is at most $f(n)$, for any $n \in \mathcal{N}$. In the *Average-Model* we consider a request sequence consistent with f if the average number of distinct pages in a window of size n is at most $f(n)$, for any $n \in \mathcal{N}$. Now we can model locality by considering only those request sequences that are consistent with f.

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

Definition 3.15. *[6] A function $f : N \to R_+$ is* concave* *if*

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

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

Albers et al. [6] consider the fault rate for measuring the performance of paging algorithms. They define the fault rate of a paging algorithm with respect to a function as follows.

Definition 3.16. *[6] The fault rate of a paging algorithm $\mathcal{A}$ with respect to a* concave* *function f is*

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

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

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

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

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

3.15 Lookahead

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

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

Consider *weak lookahead* as the standard definition of lookahead, i.e., an online algorithm has weak lookahead of size l if it sees the current request together with the next l requests. Young proposes *resource-bounded lookahead* as an alternative for weak lookahead [152]. An online paging algorithm has resource-bounded lookahead of size l if it sees the current request together with the maximal sequence of future requests for which it incurs at most l faults. Young describes a deterministic marking algorithm that achieves the competitive ratio of $\max\{2k/l, 2\}$ in the resource-bounded lookahead model. Note that the number of pages seen by the algorithm at each time depends on its behaviour in the past and also it can be very large.

Albers introduces *strong lookahead* as a more practical model of lookahead [4]. An online paging algorithm with strong lookahead of size l sees the current request together with the minimal sequence of future requests that contains l pairwise distinct pages other than the current request. She shows that a variant of LRU achieves competitive ratio $k - l$ in the strong lookahead model with $l \leq k - 2$ and that this is the best possible for any deterministic online paging algorithm in this model.

Natural lookahead is another model of lookahead for the paging problem that is proposed by Breslauer [50]. An online paging algorithm $\mathcal{A}$ has natural lookahead l if it sees the current request together with the maximal sequence of future requests that contains at most $l + 1$ distinct pages that are not in $\mathcal{A}$'s current cache. He proves that a variation of LRU achieves a competitive ratio $\frac{k+l}{l+1}$ in the natural lookahead model.

Note that these alternative models do not reflect how lookahead works in a CPU pipeline. In the pipeline all one can do is look ahead a constant number of instructions into the future, which does not match any of the definitions above. The effect of lookahead for some other online problems is studied in [55, 94, 87, 103].

3.16 Comparative Ratio

The comparative ratio was proposed by Koutsoupias and Papadimitriou [111] as an extension of the competitive ratio to reflect the influence of lookahead.

Definition 3.17. *For two classes of algorithms A and B, where typically $A \subseteq B$, the comparative*

ratio is defined as follows:

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

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

Koutsoupias and Papadimitriou [111] prove that finite lookahead of size l is beneficial for paging algorithms. Let A be the class of online paging algorithms and B be the class of paging algorithms with lookahead l. They show the influence of lookahead by proving $R(A,B) = \min\{l+1, k\}$.

3.17 Adequate Analysis

Panagiotou and Souza propose adequate analysis as a model for explaining the efficiency of LRU in practice [129]. In their work, they classify request sequences according to some parameters and prove an upper bound on the competitive ratio of LRU as a function of these parameters. Then they argue that, in practice, typical request sequences have parameters that lead to a constant competitive ratio for LRU.

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

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

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

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

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

3.18 Conclusion

In this chapter we reviewed various alternative measures for analysis of online algorithms. We also briefly described their weaknesses and strengths, as well as the results that we can get by using them. Figure 3.1 summarizes the relationships among these measures, as well as other alternative measures that we will introduce later in this thesis.

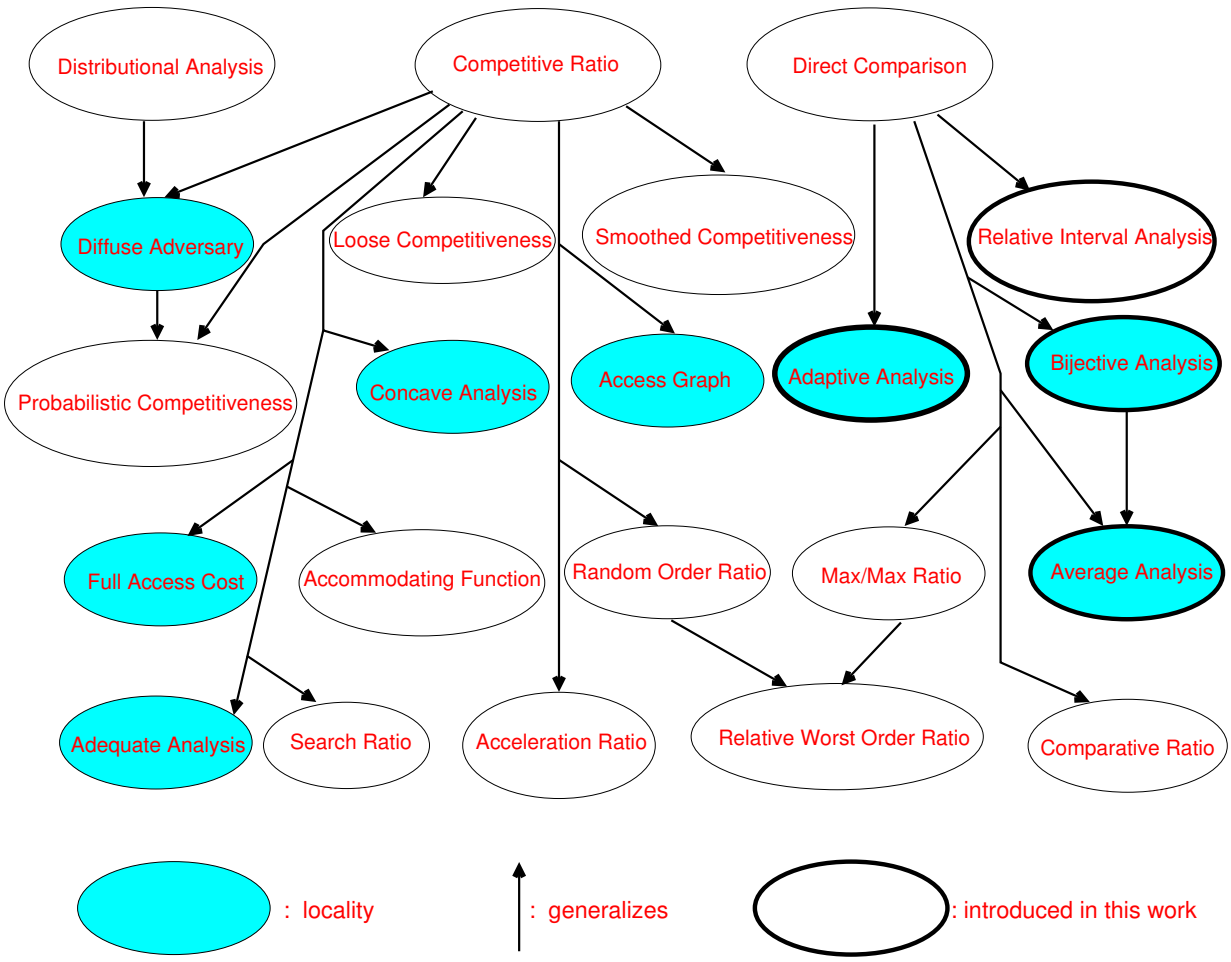

Figure 3.1: Relationships among different alternative measures.

Chapter 4

Bijective Analysis and Average Analysis

In this chapter we introduce two alternative measures for the analysis of online algorithms, namely Bijective Analysis and Average Analysis. These measures are closely related and directly compare two online algorithms on all sequences of the same length. We apply Bijective Analysis and Average Analysis to paging and building upon the ideas of concave analysis by Albers, Favrholdt, and Giel [6], we prove *strict* separation between LRU and all other paging strategies under locality of reference assumptions. That is, we show that LRU is the unique optimum strategy for paging among all deterministic strategies. This is the first deterministic model to provide full theoretical backing to the empirical observation that LRU is preferable in practice.

Our results We begin by showing that all *lazy* paging algorithms are equivalent under Bijective Analysis. In contrast, we show that LRU is strictly better than FWF (note that the latter is not a lazy strategy). Both of these results describe natural, "to-be-expected" properties of the corresponding paging strategies which competitive analysis nevertheless fails to yield. The equivalence of lazy algorithms provides strong evidence of an inherent difficulty in separating algorithms in any general setting. In fact, it implies that in order to obtain a theoretical separation between paging algorithms we must either induce a partition of the space of request sequences (e.g. as in Albers et al. [6]) or assume a distribution on the sequence space (e.g. as in Koutsoupias and Papadimitriou [111], Young [154] and Becchetti [25]). The latter group of approaches uses probabilistic assumptions on the sequence space. However, since we are interested in separating algorithms under a deterministic model, we adopt concave analysis as introduced by Albers et al., which we then apply in the context of Average Analysis. Using this approach, we show formally our main result: namely that LRU is never outperformed in any possible subpartition on the request sequence space induced by concave analysis (cf. Corollary 4.1), while it always outperforms *any other paging algorithm* in at least one subpartition of the request-sequence space (cf. Theorem 4.5). This result proves separation between LRU and all other algorithms and provides theoretical backing to the observation that LRU is preferable in practice.

Structure of chapter In Section 4.1 we give formal definitions of the concepts of Bijective Analysis and Average Analysis. In Subsection 4.2.1 we show strong equivalence between all lazy algorithms according to Bijective Analysis. These results formalize ideas that while perhaps

familiar to many researchers of online algorithms had yet to be proved in a rigorous manner. We also show that LRU is strictly better than FWF under this measure. In Subsection 4.2.2 we present our main result, i.e., separation between LRU and all other paging strategies using Average Analysis coupled with concave analysis.

4.1 Bijective Analysis and Average Analysis

In this section we provide formal definitions of a new technique for comparing the performance of online algorithms. At a high level, this is achieved by pairwise comparisons of the cost of algorithms over all possible request sequences of the same size.

For an online algorithm $\mathcal{A}$ and a request sequence σ, let $\mathcal{A}(\sigma)$ be the cost incurred by $\mathcal{A}$ on σ, and let $\mathcal{I}_n$ be the set of all request sequences of length n. The first comparison model we introduce is *Bijective Analysis*. In this model, we aim to pair-up request sequences for $\mathcal{A}$ and $\mathcal{B}$ using a bijective mapping in such a way that the cost of $\mathcal{A}$ on sequence σ is no more than the cost of $\mathcal{B}$ on the image of σ. More formally, we obtain the following definition.

Definition 4.1. *We say that an online algorithm $\mathcal{A}$ is* no worse *than an online algorithm $\mathcal{B}$ according to Bijective Analysis if there exists an integer $n_0 \geq 1$ such that for each $n \geq n_0$, there is a bijection $b : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ satisfying $\mathcal{A}(\sigma) \leq \mathcal{B}(b(\sigma))$ for each $\sigma \in \mathcal{I}_n$. We denote this by $\mathcal{A} \preceq_b \mathcal{B}$. Otherwise we denote the situation by $\mathcal{A} \not\preceq_b \mathcal{B}$. Similarly, we say that $\mathcal{A}$ and $\mathcal{B}$ are* the same *according to Bijective Analysis if $\mathcal{A} \preceq_b \mathcal{B}$ and $B \preceq_b A$. This is denoted by $\mathcal{A} \equiv_b \mathcal{B}$. Finally we say $\mathcal{A}$ is* better *than $\mathcal{B}$ according to Bijective Analysis if $\mathcal{A} \preceq_b \mathcal{B}$ and $\mathcal{B} \not\preceq_b \mathcal{A}$. We denote this by $\mathcal{A} \prec_b \mathcal{B}$.*

Observe that, as in the Max/Max ratio, this measure considers sequences of the same length and allows direct comparison of two online algorithms. However, it induces a comparison of their performance on *all* sequences in $\mathcal{I}_n$, rather than only on the worst sequence. A related, and less stringent comparison model can be obtained by considering the average number of faults that a paging algorithm incurs on request sequences of a certain length.

Definition 4.2. *We say that an online algorithm $\mathcal{A}$ is* no worse *than an online algorithm $\mathcal{B}$ according to Average Analysis if there exists an integer $n_0 \geq 1$ such that for each $n \geq n_0$, $\sum_{I \in \mathcal{I}_n} \mathcal{A}(I) \leq \sum_{I \in \mathcal{I}_n} \mathcal{B}(I)$. We denote this by $\mathcal{A} \preceq_a \mathcal{B}$. Otherwise we denote the situation by $\mathcal{A} \not\preceq_a \mathcal{B}$. $\mathcal{A} \equiv_a \mathcal{B}$, and $\mathcal{A} \prec_a \mathcal{B}$ are defined as for Bijective Analysis.*

Observation 4.1. *If $\mathcal{A} \not\preceq_a \mathcal{B}$, then $\mathcal{A} \not\preceq_b \mathcal{B}$. As well, if $\mathcal{A} \preceq_b \mathcal{B}$, then $\mathcal{A} \preceq_a \mathcal{B}$ with similar statements holding for $\equiv_b$ and $\prec_b$.*

Example 4.1. *We use a simple example to illustrate the above definitions. Let $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ be three online algorithms and $\mathcal{I}_n = \{\sigma_1, \sigma_2, \cdots, \sigma_{10}\}$ be the set of all possible request sequences of length n. Suppose that the cost of $\mathcal{A}$, $\mathcal{B}$, C, and the optimal offline algorithm OPT on request sequences are as follows:*

σ	$\mathcal{A}(\sigma)$	$\mathcal{B}(\sigma)$	$\mathcal{C}(\sigma)$	$OPT(\sigma)$
σ_1	5	6	6	3
σ_2	7	8	8	2
σ_3	3	4	3	3
σ_4	9	4	3	1
σ_5	7	10	8	3
σ_6	5	7	6	4
σ_7	3	6	4	2
σ_8	7	6	7	5
σ_9	5	8	5	2
σ_{10}	7	10	9	3

We have $\sum_\sigma \mathcal{A}(\sigma) = 58$, $\sum_\sigma \mathcal{B}(\sigma) = 69$, and $\sum_\sigma \mathcal{C}(\sigma) = 59$. Therefore $\mathcal{A} \prec_a \mathcal{B}$, $\mathcal{A} \prec_a \mathcal{C}$, and $\mathcal{C} \prec_a \mathcal{B}$. We have $\mathcal{B} \not\preceq_b \mathcal{A}$, because $\mathcal{B} \not\preceq_a \mathcal{A}$. We also have $\mathcal{A} \preceq_b \mathcal{B}$ by considering the bijection that maps $\sigma_1, \sigma_2, \ldots, \sigma_{10}$ to σ_1, σ_2, σ_3, σ_5, σ_6, σ_7, σ_4, σ_9, σ_8, and σ_{10}, respectively. Therefore $\mathcal{A}$ is better than $\mathcal{B}$ according to Bijective Analysis, i.e., $\mathcal{A} \prec_b \mathcal{B}$. Note that although $\mathcal{A}$ is better than $\mathcal{C}$ according to Average Analysis, the two algorithms are not comparable according to Bijective Analysis. Since $\mathcal{A} \prec_a \mathcal{C}$, we conclude that $\mathcal{C} \not\preceq_b \mathcal{A}$. We also have $\mathcal{A} \not\preceq_b \mathcal{C}$ because $\mathcal{C}$ incurs a cost less than 5 on 3 sequences while $\mathcal{A}$ incurs a cost less than 5 only on 2 sequences. As a last example, consider the competitive ratio of these algorithms. The competitive ratio of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ is 9, 4, and 4 respectively. Although $\mathcal{A}$ seems to have better overall performance than $\mathcal{B}$ and $\mathcal{C}$, its bad performance on a single sequence, namely σ_4, results in a bad competitive ratio.

Suitability of the Measure Note that rather than considering a worst case sequence, these measures take into account all sequences of the same length. To be precise, Bijective Analysis compares the performance of two algorithms over pairs of different inputs of the same size. A natural question is if this is a reasonable comparison. To answer this, it is necessary to briefly review standard worst case analysis. Worst case analysis of an algorithm A considers the running time of A over all possible inputs of a given size n and selects as representative for this set the maximum or worst case time observed in that class. Let $I_{A,n}$ denote this worst case input of size n for A. Now when the worst case performance of A is compared to that of algorithm B, worst case analysis compares the execution time of A on $I_{A,n}$ with that of B on $I_{B,n}$. Observe that in general $I_{A,n} \neq I_{B,n}$ and hence Bijective Analysis is no different than worst case analysis in terms of pairing different inputs of the same size. The main difference is that Bijective Analysis studies the performance of both algorithms across the entire spectrum on inputs of size n as opposed to the worst case. This is similar to average case analysis which also measures performance across all inputs of a given size.

4.2 LRU Separation

4.2.1 Separation Between Certain Paging Algorithms

As a warm-up, we will first show that LRU is better than FWF according to Bijective Analysis (recall that these algorithms have the same competitive ratio). We also prove that all lazy

algorithms are equivalent according to Bijective Analysis. For the remainder of this section let N denote the number of pages in slow memory.

Lemma 4.1. LRU $\preceq_b$ FWF.

Proof. We prove that for every $n \geq 1$ there is a bijection $b_n : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ so that $\text{LRU}(\sigma) \leq \text{FWF}(b_n(\sigma))$ for each $\sigma \in I_n$. We show this by induction on n, the length of request sequences. For $n \leq k$, this is trivial as each sequence has at most k distinct pages and LRU and FWF behave the same. Assume that it is true for all $n \leq h$, where $h \geq k$. We prove that it is also true for $n = h + 1$. We define a new bijection $b_{h+1} : \mathcal{I}_{h+1} \leftrightarrow \mathcal{I}_{h+1}$, which maps the continuations of a sequence σ to the continuations of the sequence $b_h(\sigma)$ in the image and more specifically, maps LRU's last-fault continuation sequences of σ into FWF's last-fault continuation sequences of $b_h(\sigma)$.

First assume that σ has at least k distinct pages. Then LRU's cache contains k pages after serving σ. Therefore, there are k last-hit sequences and $N - k$ last-fault sequences in the continuation of σ for LRU. In turn, FWF's cache contains at most k pages after serving $b_h(\sigma)$. Thus there are at least $N - k$ last-fault sequences for FWF in the continuation of σ.

Alternatively, if σ has $k' < k$ distinct pages then from our construction $b(\sigma)$ also contains exactly k' distinct pages and the number of last-fault and last-hit continuations for each algorithm match. Hence in either case the number of last-fault sequences in LRU is no larger than the number of last-fault sequences for FWF and we can define an injective mapping b_{h+1} from the former into the latter. We then arbitrarily map the remaining (last-hit) LRU continuation sequences of σ to the remaining unused sequences in the continuation of $b_h(\sigma)$. Clearly from the construction the bijection maps a request sequence in LRU to a request sequence of FWF with the same or more number of page faults, as claimed. □

This shows that LRU's performance is as good as FWF's. We now show that the converse does not hold.

Lemma 4.2. FWF $\not\preceq_b$ LRU.

Proof. We prove this by contradiction. Assume that we have FWF $\preceq_b$ LRU and so there is an $n_0 \geq 1$ so that for each $n \geq n_0$ we have the bijection $b_n : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$. Recall that we can partition a sequence into a number of consecutive phases so that each phase contains exactly k distinct pages. LRU incurs at most k faults in each phase. On the other hand, FWF empties its cache at the beginning of each phase and incurs exactly k faults in each phase. Therefore we have $\text{LRU}(\sigma) \leq \text{FWF}(\sigma)$ for each sequence σ. Thus the desired bijection exists only if we have $\text{FWF}(\sigma) = \text{LRU}(\sigma)$ for every sequence of length $n \geq n_0$. Consider a sequence $\sigma = p_1 p_2 \dots p_h$ ($h \geq n_0$) so that σ contains at least k distinct pages and p_h is the first page of a phase. Therefore p_h causes FWF to flush its cache, which now contains only one page after serving σ. Now consider the set of continuations of σ. The number of last-fault sequences among these for LRU and FWF is $N - k$ and $N - 1$, respectively. Therefore there are at least $k - 1$ sequences for which the cost of LRU is strictly less than the cost of FWF and hence a bijection as required does not exist. □

Combining Lemma 4.1 and Lemma 4.2 we obtain strict separation between the performance of LRU and FWF.

Theorem 4.1. LRU $\prec_b$ FWF.

Note that we can use exactly the same argument as in Lemma 4.1 to prove the following theorems.

Theorem 4.2. FIFO $\preceq_b$ FWF, FIFO $\preceq_b$ LRU, *and* LRU $\preceq_b$ FIFO. *Thus we have* LRU $\equiv_b$ FIFO.

Theorem 4.3. *Let $\mathcal{A}$ and $\mathcal{B}$ be two arbitrary lazy algorithms. Then we have $\mathcal{A} \equiv_b \mathcal{B}$.*

Remark 1. *Theorem 4.3 implies that all lazy paging algorithms are equivalent according to the Max/Max ratio as well. In general, when two algorithms are equivalent according to Bijective Analysis then they have the same Max/Max ratio, as their worst performance on sequences of the same length would be equal.*

These results explain why most measures are unable to separate lazy algorithms such as LRU and FIFO, in that the multisets of the costs incurred by the algorithms on all requests of the same length are identical. In fact, Theorem 4.3 supports the observation that unless we constrain the space of request sequences to the ones most often appearing in practice, it is unlikely to separate different paging algorithms. Since locality of reference is a definitive characteristic of typical sequences in paging, this naturally leads to the question of how it affects the comparison of algorithms, according to our measures.

4.2.2 Paging with Locality of Reference

Above, we proved that all lazy algorithms are strongly equivalent according to Bijective Analysis. However, this analysis ignored that in practice request sequences exhibit *locality of reference*. We follow the Max-Model of concave analysis for modelling locality of reference (cf. Section 3.14). Let $\mathcal{I}^f$ denote the set of request sequences that are consistent with a given concave* function f as defined in page 34. We can apply Bijective and Average Analysis over this restricted set of sequences, by adapting Definition 4.1 and Definition 4.2 appropriately, i.e., by replacing the set $\mathcal{I}$ with the set $\mathcal{I}^f$. We denote the corresponding relations by $\mathcal{A} \preceq_b^f \mathcal{B}$, $\mathcal{A} \preceq_a^f \mathcal{B}$, etc. Note that we can make any sequence consistent with f by repeating every request a sufficient number of times. Therefore even if we restrict sequences to those with high locality of reference, there is a still a worst case sequence for LRU that is consistent with f. Therefore the competitive ratio of LRU is the same as in the standard model[1]. Figure 4.1 illustrates the partition of the request-sequence space induced by the choice of function f. Observe that the performance of a paging algorithm is now evaluated within the subset of request sequences of a given length whose locality of reference is consistent with f.

Note that the inductive argument used to prove that all lazy algorithms are equivalent according to Bijective Analysis does not necessarily carry through under concave analysis. The problem is that the bijective mapping could map a sequence consistent with f to a sequence which is not consistent with f.

Consider a fixed concave* function f. Let $\mathcal{I}_n^f$ denote sequences of length n in $\mathcal{I}^f$ and $\mathcal{A}$ be an arbitrary paging algorithm. We call a sequence *bad* for $\mathcal{A}$ if $\mathcal{A}$ incurs a fault on its last request; otherwise we call it a *good* sequence for $\mathcal{A}$. Let $B_h(\mathcal{A})$ be the number of sequences in $\mathcal{I}_h^f$ that are bad for $\mathcal{A}$. For a sequence $\sigma \in \mathcal{I}_h^f$, let $B_{h+1}(\mathcal{A} \,|\, \sigma)$ denote the number of sequences in $\mathcal{I}_{h+1}^f$

[1]This is one of the reasons that Albers et al. [6] use the *fault rate*, instead of overall cost, as a performance measure.

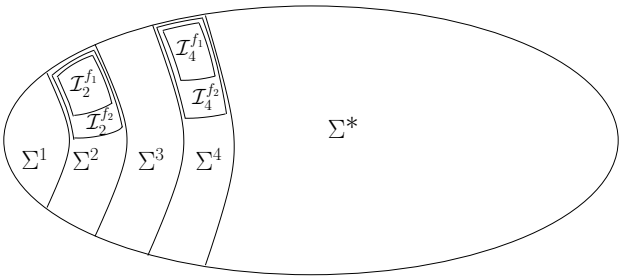

Figure 4.1: Partition of the input space induced by different choices of f.

that have σ as their prefix and are bad for $\mathcal{A}$. Define $G_h(\mathcal{A})$ and $G_{h+1}(\mathcal{A} \mid \sigma)$ in an analogous way for good sequences. Intuitively, a good algorithm maintains its good sequences in the set of sequences with high locality of reference and hence can safely perform the continuation. Observe that LRU naturally fits this criterion: the most recently accessed pages are exactly those that are in its cache, and therefore good (i.e. last-hit) sequences for LRU are more likely to be sequences with high locality of reference. We formalize this intuition in the rest of this section.

Lemma 4.3. *For any integer $h > 0$ and any paging algorithm $\mathcal{A}$, $B_h(\mathrm{LRU}) \leq B_h(\mathcal{A})$.*

Proof. We prove this by induction on h. If $h = 1$, then every sequence of $\mathcal{I}_h$ is consistent with f and each algorithm incurs a fault on its last request (recall that algorithms start with an empty cache). Therefore we have $B_1(\mathrm{LRU}) = |\mathcal{I}_1^f| = N = B_1(\mathcal{A})$. If $h > 1$, consider an arbitrary sequence $\sigma \in \mathcal{I}_{h-1}^f$. If σ has at most k distinct pages, then LRU and $\mathcal{A}$ have the same pages in their cache after serving σ and therefore $B_h(\mathrm{LRU} \mid \sigma) = B_h(\mathcal{A} \mid \sigma)$. Otherwise LRU has filled its cache with k pages after serving σ, while A's cache contains at most k pages. The next page requested can be an arbitrary page, provided that adding that page does not violate consistency with f.

From the definition of f, repeating the last request of a sequence $\sigma \in \mathcal{I}_{h-1}^f$ is always consistent with f. Repeating the second to last sequence may or may not be consistent with f, however if the second to last sequence is not consistent neither is any other request. This implies that for every good request for A that is consistent with f, there is a good request for LRU that is consistent with f. Hence $G_h(\mathrm{LRU} \mid \sigma) \geq G_h(\mathcal{A} \mid \sigma)$. Now since the good and bad continuations form a partition of the set of continuations of σ consistent with f, the inequality above implies $B_h(\mathrm{LRU} \mid \sigma) \leq B_h(\mathcal{A} \mid \sigma)$. To conclude observe that $B_h(X) = \sum_{\sigma \in \mathcal{I}_{h-1}} B_h(X \mid \sigma)$ for any algorithm X. Hence

$$B_h(\mathrm{LRU}) = \sum_{\sigma \in \mathcal{I}_{h-1}} B_h(\mathrm{LRU} \mid \sigma) \leq \sum_{\sigma \in \mathcal{I}_{h-1}} B_h(\mathcal{A} \mid \sigma) = B_h(\mathcal{A})$$

as claimed. □

Lastly, we show that LRU strictly outperforms all other paging algorithms.

Definition 4.3. *Let m be an integer, $\mathcal{A}$ and $\mathcal{B}$ be online algorithms, and f be a* concave* *function. $\mathcal{A}$ is said to* (m, f)-dominate *$\mathcal{B}$ if we have*

$$\sum_{\sigma \in \mathcal{I}_m^f} \mathcal{A}(\sigma) \leq \sum_{\sigma \in \mathcal{I}_m^f} \mathcal{B}(\sigma).$$

$\mathcal{A}$ *is said to* dominate $\mathcal{B}$ *if there exists an integer* $m_0 \geq 1$ *so that for each* $m \geq m_0$ *and every* concave* *function* f, $\mathcal{A}$ (m, f)*-dominates* $\mathcal{B}$.

Observation 4.2. $\mathcal{A} \preceq_a^f \mathcal{B}$ *if and only if there exists an integer* $m_0 \geq 1$ *so that* $\mathcal{A}$ (m, f)*-dominates* $\mathcal{B}$ *for each* $m \geq m_0$.

Lemma 4.4. *For every paging algorithm* $\mathcal{A}$, LRU *dominates* $\mathcal{A}$.

Proof. Let f be an arbitrary concave* function and m be a positive integer. For any $1 \leq i \leq m$, let $\mathcal{F}_{i,m}(\mathcal{A})$ be the number of sequences in $\mathcal{I}_m^f$ for which $\mathcal{A}$ incurs a fault on the i^{th} request. We will show that $\mathcal{F}_{i,m}(\text{LRU}) \leq \mathcal{F}_{i,m}(\mathcal{A})$ for any $1 \leq i \leq m$ which will imply optimality of LRU. For $i = 1$, we have $\mathcal{F}_{1,m}(\text{LRU}) = \mathcal{F}_{1,m}(\mathcal{A}) = |\mathcal{I}_m^f|$. Now assume that $i > 1$. Let σ be an arbitrary sequence of length $i-1$, and let T_σ denote the set of sequences in $\mathcal{I}_m^f$ that have σ as their prefix. Denote by $\mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma)$ the number of sequences in T_σ for which $\mathcal{A}$ incurs a fault on the i^{th} request.

If σ contains at most k distinct pages, then LRU and $\mathcal{A}$ behave the same on σ and we have $\mathcal{F}_{i,m}(\text{LRU} \,|\, \sigma) = \mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma)$. Assume then that σ has more than k distinct pages. We can partition T_σ into four subsets: (1) T_σ^1: sequences in which neither LRU nor $\mathcal{A}$ incur a fault on the i^{th} page request, (2) T_σ^2: sequences in which both LRU and $\mathcal{A}$ incur a fault on the i^{th} page request, (3) T_σ^3: sequences in which $\mathcal{A}$ incurs a fault on the i^{th} page request, but LRU does not, and (4) T_σ^4: sequences in which LRU incurs a fault on the i^{th} page request, but $\mathcal{A}$ does not.

We have $\mathcal{F}_{i,m}(\text{LRU} \,|\, \sigma) = |T_\sigma^2| + |T_\sigma^4|$, and $\mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma) = |T_\sigma^2| + |T_\sigma^3|$. We show that $|T_\sigma^4| \leq |T_\sigma^3|$ by proving that there exists a one-to-one mapping d from T_σ^4 to T_σ^3. For $1 \leq q \leq 4$, let P_σ^q be the set of pages that are requested as the i^{th} page of a sequence in T_σ^q. Let $\rho \in \mathcal{I}_i^f$ be a sequence that contributes to $B_i(\text{LRU} \,|\, \sigma)$ and p be its i^{th} request. Denote by τ the sequence of length m obtained by appending requests to page p to ρ. Since ρ is consistent with f and p is the last request of ρ, τ is consistent with f and thus $\tau \in T_\sigma$. Note that LRU incurs a fault on the i^{th} request of τ. Therefore τ belongs to either T_σ^2 or T_σ^4 and p is in one of the sets P_σ^2 and P_σ^4. Also any page in $P_\sigma^2 \cup P_\sigma^4$ contributes to $B_i(\text{LRU} \,|\, \sigma)$. To see this, let p be a page in $P_\sigma^2 \cup P_\sigma^4$. Then p is the i^{th} request of a sequence $\rho \in T_\sigma^2 \cup T_\sigma^4$. Let $\tau \in \mathcal{I}_i^f$ be the prefix of ρ that contains i requests. LRU incurs a fault on the last request of τ. Thus τ is a bad sequence for LRU and p contributes to $B_i(\text{LRU} \,|\, \sigma)$. Hence we have $B_i(\text{LRU} \,|\, \sigma) = |P_\sigma^2| + |P_\sigma^4|$. Using analogous arguments we get $B_i(\mathcal{A} \,|\, \sigma) = |P_\sigma^2| + |P_\sigma^3|$. We know that $B_i(\text{LRU} \,|\, \sigma) \leq B_i(\mathcal{A} \,|\, \sigma)$ from the proof of Lemma 4.3; therefore $|P_\sigma^4| \leq |P_\sigma^3|$ and there is a one-to-one mapping r from P_σ^4 to P_σ^3.

We use the mapping r to define the desired mapping d. Consider an arbitrary sequence $S = p_1 p_2 \ldots p_m \in T_\sigma^4$. Let $p_i = x$ and $y = r(x)$. According to definitions we know that on the i^{th} request of a sequence in T_σ, x is a fault for LRU and a hit for $\mathcal{A}$, while y is a hit for LRU and a fault for $\mathcal{A}$. Let $\sigma_x \in \mathcal{I}_i^f$ be the sequence obtained by appending the page x to σ, and $\sigma_y \in \mathcal{I}_i^f$ be the sequence obtained by appending the page y to σ. On serving σ_x, the last page (x) is a fault for LRU; therefore x is not among the last k distinct pages in σ. LRU does not incur a fault on the last page of σ_y; thus y is among the last k distinct pages of σ. Hence if starting from the i^{th} request, we convert each x in a sequence in T_{σ_x} to y, we will obtain a sequence that is consistent with f, i.e., a sequence in T_{σ_y}. This gives us a one-to-one mapping from T_{σ_x} to T_{σ_y}. By a similar process for the pages in P_σ^4, we obtain a one-to-one mapping from T_σ^4 to T_σ^3. Therefore

$$|T_\sigma^4| \leq |T_\sigma^3| \Rightarrow \mathcal{F}_{i,m}(\text{LRU} \,|\, \sigma) \leq \mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma). \tag{4.1}$$

Since

$$\mathcal{F}_{i,m}(\text{LRU}) = \sum_{\sigma \in \mathcal{I}_{i-1}} \mathcal{F}_{i,m}(\text{LRU} \mid \sigma) \tag{4.2}$$

and

$$\mathcal{F}_{i,m}(\mathcal{A}) = \sum_{\sigma \in \mathcal{I}_{i-1}} \mathcal{F}_{i,m}(\mathcal{A} \mid \sigma), \tag{4.3}$$

we get $F_{i,m}(\text{LRU}) \leq F_{i,m}(\mathcal{A})$. We also have

$$\sum_{\sigma \in \mathcal{I}_m^f} \text{LRU}(\sigma) = \sum_{1 \leq i \leq m} \mathcal{F}_{i,m}(\text{LRU}) \tag{4.4}$$

and

$$\sum_{\sigma \in \mathcal{I}_m^f} \mathcal{A}(\sigma) = \sum_{1 \leq i \leq m} \mathcal{F}_{i,m}(\mathcal{A}). \tag{4.5}$$

Therefore

$$\sum_{\sigma \in \mathcal{I}_m^f} \text{LRU}(\sigma) \leq \sum_{\sigma \in \mathcal{I}_m^f} \mathcal{A}(\sigma). \tag{4.6}$$

Thus LRU (m, f)-dominates $\mathcal{A}$ for every concave* function f, and every integer $m \geq 1$. Hence LRU dominates $\mathcal{A}$. □

Corollary 4.1. *For any* concave* *function* f *and any paging algorithm* $\mathcal{A}$, LRU $\preceq_a^f \mathcal{A}$.

Therefore LRU is *an* optimal algorithm when we restrict request sequences to those with high locality of reference. A natural question is whether or not LRU is a unique optimal algorithm. The following theorem answers this question in the affirmative.

Theorem 4.4. *No paging algorithm (other than* LRU*) dominates* LRU.

Proof. Consider a paging algorithm $\mathcal{A}$ which is different from LRU. We will show that there exists a concave* function f such that $\mathcal{A} \not\preceq_a^f$ LRU. In fact, we will prove a slightly stronger result, we will show that this holds for every concave* function other than the identity. First, if the content of the cache of LRU and $\mathcal{A}$ is always the same there is nothing to show. Hence we focus on situations where their cache content differs. As in the proof of Lemma 4.4 we need only focus on T_σ^3 and T_σ^4. We use the same notation as that proof.

Let σ' denote a sequence consistent with f up to the point at which LRU's cache contents first differ from those of $\mathcal{A}$. That is, a request for a page p caused an eviction of a page p_a in LRU while it caused the eviction of a page p_b in $\mathcal{A}$. We have $P_{\sigma'}^3 \subseteq \{p_b\}$ and $P_{\sigma'}^4 \subseteq \{p_a\}$. We add some requests to p to get a sequence σ for which $P_\sigma^3 = \{p_b\}$. Let $|\sigma| = i - 1$. We can use the same one-to-one mapping d that was defined in the proof of Lemma 4.4 to show that $|T_\sigma^4| \leq |T_\sigma^3|$. Thus each sequence $\rho \in T_\sigma^4$ is mapped to a distinct sequence $d(\rho) \in T_\sigma^3$. We prove that $|T_\sigma^4| < |T_\sigma^3|$ by showing that there is a sequence ρ' in T_σ^3 such that $d(\rho) \neq \rho'$ for any $\rho \in T_\sigma^4$. Let $Q = \{q_0, q_1, \ldots, q_r\}$ be pages that do not appear in σ and σ^0 be the suffix of σ starting just after the last request for p_a. Observe that the number of distinct pages in $\sigma^0 p_a$ and $\sigma^0 p_b$ is $k+1$

and k, respectively. Let ρ_1 and ρ_2 be continuations of σp_a and σp_b, respectively, defined as follows. We add new pages from Q to ρ_1 and ρ_2 until adding a new page q_t causes an inconsistency with f in the suffix starting with $\sigma^0 p_a$ in ρ_1. This will always eventually occur as the percentage of distinct pages in the sequence as extended approaches $(a+m)/(b+m)$ where m is the length of the continuation past σp_a, a is the number of distinct pages in σ before p_a and $|\sigma| = b$. Observe that as m goes to infinity this ratio approaches one. This contradicts the easy to verify fact that every concave* function except the identity is such that $f(n) \leq (1-\varepsilon)n$ where ε is a positive constant that depends on f. Note that ρ_2 (but not ρ_1) would remain consistent with f as the number of distinct pages in $\sigma^0 p_b$ is one less than the number of distinct pages in $\sigma^0 p_a$. There is a small caveat in this construction. We might first violate consistency with f in a subsequence τ of ρ_1 and ρ_2 that does not start with σ^0. There are two cases:

1. τ starts after σ^0. Let q be the last page of τ, i.e., the first inconsistency occurred by adding q and let q' be the page requested just before q. In this case we add a sufficient number of requests to q' just before the first request to q to resolve the inconsistency.

2. τ starts before σ^0, say at the m^{th} request. We change our construction slightly to rule out this case. Let σ^1 be the prefix of σ that ends just before the start of σ^0 and let C be the set of pages that are either in the cache of LRU after serving σp_a or in the cache of $\mathcal{A}$ after serving σp_b. Denote by $R = \{r_1, \ldots, r_t\}$ the pages not in C that are requested in σ^1, ordered by their last request in σ^1. Now to construct ρ_1 and ρ_2, we use pages from $r_1, r_2, \ldots, r_t, q_1, \ldots, q_r$ in this order (note that we might not need all these pages to get our desired sequences). We claim that we cannot have an inconsistency with f in a subsequence of ρ_1 that starts at position m before the desired inconsistency at the suffix that starts with σ^0. Assume for the sake of contradiction that the first inconsistency occurs after adding a page q on a subsequence that starts at position m while the subsequence of ρ_1 that starts by σ^0 remains consistent with f. If $q = r_u \in R$, then since this is the first inconsistency in ρ_1, r_u is a new page in τ and has not been requested before. Therefore τ does not contain r_v for $v > u$. However the suffix of σ that starts with σ^0 also has all requests to r_v for $v \leq u$ and thus τ does not contain any additional pages, as compared to σ^0. Hence σ^0 contains the same number of distinct pages as τ while τ is a longer sequence, so σ^0 cannot be consistent with f, which is a contradiction. Else $q = q_u \in Q$. Since we have already requested all pages of R, a page requested by τ has also been requested in the subsequence that starts by σ^0.

Now consider the sequence S_2 that we have constructed in this way. We have $S_2 \in T^3_\sigma$ but it is easy to see that no sequence in T^4_σ is mapped by d to ρ_2 (in fact $d(\rho_1) = \rho_2$ but ρ_1 is not consistent with f). Therefore $|T^4_\sigma| < |T^3_\sigma|$ which leads to

$$\sum_{\sigma \in \mathcal{I}^f_m} \text{LRU}(\sigma) \leq \sum_{\sigma \in \mathcal{I}^f_m} \mathcal{A}(\sigma),$$

for any $m > |S_1|$ by the same arithmetic manipulation as in equations 4.1-4.6 in the proof of Lemma 4.4.

□

From Bijective Analysis analysis we know that on unconstrained sequences many algorithms match the performance of LRU. The previous lemmas show that outside of that realm, LRU is the sole optimum.

Theorem 4.5. *Let $\mathcal{A}$ be a paging algorithm other than* LRU. *Then there is a* concave* *function f so that $\mathcal{A} \not\preceq_a^f$* LRU *which implies $\mathcal{A} \not\preceq_b^f$* LRU.

4.3 Conclusions

In this chapter we introduced Bijective Analysis and Average Analysis as two new techniques for comparing the performance of online algorithms. These measures compare online algorithms over all sequences of the same length, rather than the worst case sequences alone. In this line of research we set as a goal to propose a model that would more accurately reflect the performance of known online paging algorithms. After examining a variety of options, we chose the model of Albers et al. as the best starting point for our endeavours. We then proceeded to further refine the measure using Bijective Analysis. It then became apparent that this combined new model, proposed purely from first principles, would naturally provide separation between well known paging algorithms.

We demonstrated how the proposed measure can be applied to paging. We showed that the new comparison techniques overcome some of the shortcomings of competitive analysis, namely they are able to separate the performance of LRU and FWF. In the next chapter, we will show that they also reflect the influence of lookahead for paging. We next proved that all lazy algorithms are equivalent according to Bijective Analysis. This result provides an intuitive explanation why it is difficult, or even impossible, for several known measures to distinguish between the performance of known algorithms. Although a negative result at first sight, it suggests that unless one considers typical request sequences for the problem, it is not likely that a meaningful measure can separate any two algorithms.

In view of this result, we turned our attention to the definitive property of typical request sequences for the paging problem, namely locality of reference. As a model of sequences with locality of reference, we relied on a natural model due to Albers et al. [6], namely concave analysis. We then showed that combining average and concave analysis, LRU emerges as the sole optimal online paging algorithm. More specifically, we proved that when we restrict the input to sequences with high locality of reference, LRU is never outperformed by another paging algorithm according to Average Analysis, while it outperforms any other paging algorithm according to Average Analysis (and thus according to Bijective Analysis as well). Since, in practice, input sequences are known to exhibit locality of reference, this justifies theoretically why LRU is believed to have the best practical performance among online paging algorithms.

Chapter 5

Other Applications of Bijective and Average Analysis

In Chapter 4 we introduced Bijective Analysis and Average Analysis as two new models for the analysis of online algorithms and applied them to paging. In this chapter, we apply these models to other problem domains. First, we provide concrete evidence that in the context of paging, lookahead is beneficial. Specifically, we show that under Bijective Analysis, LRU with lookahead as small as one (that is, the sequence is revealed to the algorithm as overlapping consecutive pairs of requests) is strictly better than LRU without any lookahead. Second, we turn our attention to another fundamental problem in online computation, namely the *list update* problem. We first show that under the modified cost model, all online list update algorithms are equivalent according to Bijective Analysis. We then address the issue of locality of reference in the context of list update. In particular, we show how the model of Albers et al. [6] can be extended, so as to properly capture the effect of locality of reference in list update applications related to compression algorithms. We provide experimental results obtained on the Calgary Corpus, which is frequently used as a standard benchmark for evaluating the performance of compression algorithms (and by extension for list update algorithms, e.g. [19]). We thus resolve the open problem posed by Hester and Hirschberg [89], in that we provide a theoretical model which captures the effect of locality in list update applications. Our main result proves that under both the standard cost model as well as under the modified cost formulation, MTF is the unique optimal algorithm according to Average Analysis.

Our results on list update also address a question posed by Martínez and Roura [122], namely the question of defining an alternative measure to the competitive ratio that demonstrates the superiority of MTF in the modified cost model. This is motivated by the observation that in the modified cost model, all list update algorithms have asymptotically the same non-constant competitive ratio [122]. Our results provide evidence that Bijective and Average Analysis are not tied to the paging problem, but rather can be applied, with success, to other online optimization problems.

5.1 Influence of Lookahead

In this section we demonstrate that Bijective Analysis can properly capture the effects of lookahead in the paging problem. We consider the setting in which the paging algorithm has the

capacity to know, at any given point while serving a request sequence, the next ℓ pages that will be requested.

Let LRU(ℓ) be the modification of LRU defined for a lookahead of size ℓ as follows [4]: on a fault, LRU(ℓ) evicts the page in the cache that is least recently used among the pages that are not in the current lookahead. We will show that our model reflects the influence of lookahead in that LRU(ℓ) $\prec_b$ LRU, i.e. LRU(ℓ) is better than LRU according to Bijective Analysis. Intuitively, we obtain that LRU(ℓ) $\preceq_b$ LRU because LRU(ℓ) behaves almost the same as LRU except when it knows it can perform better.

Lemma 5.1. *The cost of* LRU(ℓ) *is no more than the cost of* LRU *on any given sequence of requests, that is* LRU(ℓ) $\preceq_b$ LRU.

Proof. Assume for the sake of contradiction that there is a sequence $\sigma = p_1 \dots p_m$ on which LRU(ℓ) incurs strictly more faults than LRU. Let a be the smallest index so that p_a is a hit for LRU and a fault for LRU(ℓ). Suppose that the most recent eviction of p_a by LRU(ℓ) is at time r on the request p_r. Therefore we have $p_i \neq p_a$ for $r \leq i \leq a$ and furthermore LRU does not evict p_a at any time t, where $r \leq t \leq a$. Let $p_{r_1}, p_{r_2}, \dots, p_{r_k}$ be the pages in LRU's cache at time r so that p_{r_i} is less recently used than p_{r_j} at time r if and only if $i < j$. Note that since a is the smallest index so that p_a is a hit for LRU and a fault for LRU(ℓ), LRU incurs a fault on p_r and evicts p_{r_1}. Note that $p_{r_x} = p_a$ for some $1 < x \leq k$. Let $\mathcal{L}_r$ the set of pages in the lookahead of size ℓ at time r. We consider two cases:

Case 1: All the pages $p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}$ are in LRU(ℓ)'s cache at time r. Since LRU(ℓ) evicts $p_{r_x} = p_a$ at time r, we should have $p_{r_i} \in \mathcal{L}_r$ for $1 \leq i \leq x-1$. Let y be the largest index so that $r \leq y \leq r + \ell$, $p_y \in \{p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}\}$, and LRU incurs a fault at time y. Note that since $p_{r_1} \in \mathcal{L}_r$ and LRU evicts p_{r_1} at time r, y exists. We claim that LRU should evict $p_{r_x} = p_a$ at time y. Since p_a has not been requested between times r and y, the only pages that can be less recently used than p_a are $p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}$. Assume, by way of contradiction, that at time y, LRU evicts the page p_{r_z} for some $1 \leq z \leq x-1$. Note that p_{r_z} cannot be a request between times r and y; otherwise p_a would be less recently used than it. We know that $p_{r_z} \in \mathcal{L}_r$ and therefore p_{r_z} will be requested at least once between the times $y+1$ and $r+\ell$. The first such request is a fault on a page (p_{r_z}) that is in $\{p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}\}$; this contradicts the choice of y. Therefore p_a is the least recently used page for LRU at time y and LRU evicts it. This contradicts the fact that LRU does not evict p_a on a fault at any time $r \leq t \leq a$.

Case 2: There is a page $p \in \{p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}\}$ that is not in LRU(ℓ)'s cache at time r. Let $r' < r$ be the last time that LRU(ℓ) has evicted p. Since p is in LRU's cache and not in LRU(ℓ)'s cache at time r, we have $p_i \neq p$ for $r' \leq i \leq r-1$ and furthermore LRU does not evict p at any time t, where $r' \leq t \leq r-1$. This reduces to the situation discussed at the beginning of this proof, with $a = r$ and $r = r'$. Since a is a finite number and we strictly decrease our time of interest, after a finite number of applications of case 2 this reduces to case 1.

Thus we cannot have any request on which LRU(ℓ) incurs a fault and LRU does not, and hence LRU(ℓ) does not incur more faults than LRU on any sequence. □

Lemma 5.2. *There exists a sequence in which* LRU(ℓ) *outperforms* LRU, *therefore* LRU $\not\preceq_b$ LRU(ℓ).

Proof. From Lemma 5.1, we know that LRU has the same or higher number of page faults as LRU(ℓ) on each sequence of length at least n_0. So it suffices to show that on at least one sequence LRU(ℓ) strictly outperforms LRU. Consider a sequence σ of size $n_1 \geq n_0$, as in Definition 4.1, which contains several consecutive copies of the subsequence $p_1 p_2 \ldots p_k p_{k+1}$. LRU incurs a fault on all pages of σ and therefore the cost of LRU on σ is n_1. The cost of LRU(ℓ) on the other hand is $n_1/(\ell+1)$. □

Lemmas 5.1 and 5.2 imply the following theorem.

Theorem 5.1. LRU(ℓ) $\prec_b$ LRU.

5.2 The List Update Problem

Recall the problem posed by Martínez and Roura: "an important open question is whether there exist alternative ways to define competitiveness such that MTF and other good online algorithms for the list update problem would be competitive, even for the [modified] cost model". In this section we address this open problem by showing that under locality of reference assumptions, MTF is the unique optimal algorithm for list update. First, we show that under the modified cost model all list update algorithms are equivalent. This result parallels the equivalence of all lazy paging algorithms under Bijective Analysis as shown in Subsection 4.2.1.

Theorem 5.2. *Let $\mathcal{A}$ and $\mathcal{B}$ be two arbitrary online list update algorithms. Under the modified cost model, we have $\mathcal{A} \equiv_b \mathcal{B}$.*

Proof. We prove that for every $n \geq 1$ there is a bijection $b_n : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ so that $\mathcal{A}(\sigma) \leq \mathcal{B}(b_n(\sigma))$ for each $\sigma \in I_n$. We show this by induction on n, the length of the input sequence. Since $\mathcal{A}$ and $\mathcal{B}$ start with the same initial list, they incur the same cost on each sequence of length one. Therefore the statement trivially holds for $n = 1$. Assume that it is true for $n = k$. Thus there is a bijection $b_k : \mathcal{I}_k \leftrightarrow \mathcal{I}_k$ so that $\mathcal{A}(\sigma) \leq \mathcal{B}(b_k(\sigma))$ for each $\sigma \in I_k$. Let σ be an arbitrary sequence of length k and $\sigma' = b_k(\sigma)$. Denote by $\mathcal{I}_{k+1}(\sigma)$ the set of sequences in $\mathcal{I}_{k+1}$ which have σ as their prefix. We map $\mathcal{I}_{k+1}(\sigma)$ to $\mathcal{I}_{k+1}(\sigma')$ as follows. Let $\mathcal{L}(\mathcal{A}, \sigma) = (a_1, a_2, \ldots, a_\ell)$ be the list maintained by $\mathcal{A}$ after serving σ and $\mathcal{L}(\mathcal{B}, \sigma') = (b_1, b_2, \ldots, b_\ell)$ be the list maintained by $\mathcal{B}$ after serving σ'. Consider an arbitrary sequence $\sigma_1 \in \mathcal{I}_{k+1}(\sigma)$ and let its last element be a request to item a_i. We map σ_1 to the sequence $\sigma_2 \in \mathcal{I}_{k+1}(\sigma')$ that has b_i as its last request. Since $\mathcal{A}(\sigma) \leq \mathcal{B}(\sigma')$ and $\mathcal{A}$'s cost on the last request of σ_1 is the same as $\mathcal{B}$'s cost on the last request of σ_2, we have $\mathcal{A}(\sigma_1) \leq \mathcal{B}(\sigma_2)$. Therefore we get the desired mapping from $\mathcal{I}_{k+1}(\sigma)$ to $\mathcal{I}_{k+1}(\sigma')$. We obtain a bijection $b_{k+1} : \mathcal{I}_{k+1} \leftrightarrow \mathcal{I}_{k+1}$ by considering the above mapping for each sequence $\sigma \in \mathcal{I}_k$. Thus our induction statement is true and we have $\mathcal{A} \preceq_b \mathcal{B}$. Using a similar argument, we can show $\mathcal{B} \preceq_b \mathcal{A}$. Therefore we have $\mathcal{A} \equiv_b \mathcal{B}$. □

We term a list update algorithm *economical* if it does not use paid exchanges, assuming the standard cost model. Since an economical list update algorithm does not incur any cost for reorganizing the list we can prove the following statement using an argument analogous to the proof of Theorem 5.2.

Corollary 5.1. *All economical online list update algorithms are equivalent according to Bijective Analysis under the standard cost model.*

The results above suggest that so long as we consider the space of all possible request sequences, all on-line list update algorithms are equivalent in a strong sense. In the following section we address the issue of locality of reference in typical list update sequences, and its effect in the relative performance of online algorithms.

5.2.1 List Update with Locality of Reference

Unlike the paging problem, the prevalence of locality of reference in list update is less self-evident. In practice, input sequences of list update algorithms indeed exhibit locality of reference [89, 137, 35] and on-line list update algorithms try to take advantage of this property [89, 132]. In particular, locality is prevalent in applications of list update algorithms for data compression (see e.g. [19]).

The lack of formal models led Hester and Hirschberg [89] to pose the question of providing a satisfactory formal definition of locality of reference for list update as an open problem. In [12] we addressed the problem of list update under locality of reference. To the best of our knowledge, this was the first formal study of locality of reference for list update. Recently, Albers and Lauer [7] proposed another model for list update with locality of reference that is based on the concept of runs; here a run is a subsequence of requests to the same list item. It has been commonly assumed, based on intuition and experimental evidence, that MTF is the best algorithm on sequences with high locality of reference. For instance, Hester and Hirschberg [89] claim: "move-to-front performs best when the list has a high degree of locality" (see also [8], page 327). The results of Albers and Lauer [7] confirm, in a theoretical manner, that MTF has excellent performance at high locality.

Following the model of concave analysis for the paging problem of Albers et al. [36], we say that a request sequence σ for list update is *consistent* with f if the maximum number of distinct items in a window of w consecutive items in σ is at most $f(w)$. In section 5.2.1 we provide experimental results that demonstrate that for applications of list update related to data compression, the function f has an overall concave shape. Hence, the experimental evidence suggest that concave analysis can be applied not only in the paging problem but also in list update.

Perhaps not surprisingly, this measure seems to parallel MTF's behaviour as the latter has been tailored to benefit from locality of reference. This should not be construed as a drawback of the measure, but rather as evidence of the fact that the design of the MTF algorithm incorporates the presence of locality of reference into its choices. Our theoretical proof of the optimality of MTF in this context is then perhaps not surprising, yet this fact had eluded proof until now.

As with the paging problem, we can apply bijective and/or Average Analysis by restricting the set of request sequences to those consistent with f. We will make use of the notation introduced in Subsection 4.2.2.

Lemma 5.3. *For every online list update algorithm $\mathcal{A}$,* MTF *dominates $\mathcal{A}$.*

Proof. Let f be an arbitrary concave* function and m be a positive integer. For any $1 \leq i \leq m$, let $\mathcal{F}_{i,m}(\mathcal{A})$ be the total cost $\mathcal{A}$ incurs on the i^{th} request of all sequences in $\mathcal{I}_m^f$. We will first show that $\mathcal{F}_{i,m}(\text{MTF}) \leq \mathcal{F}_{i,m}(\mathcal{A})$ for any $1 \leq i \leq m$. For $i = 1$, we have $\mathcal{F}_{1,m}(\text{MTF}) = \mathcal{F}_{1,m}(\mathcal{A})$, as all algorithms start with the same list. Now suppose that $i > 1$. Let σ be an arbitrary sequence of length $i-1$, T_σ denote the set of all sequences in $\mathcal{I}_m^f$ that have σ as their prefix, and $\mathcal{F}_{i,m}(\mathcal{A} \mid \sigma)$ be the total cost $\mathcal{A}$ incurs on the i^{th} request of all sequences in T_σ. Denote by

$\mathcal{L}(\text{MTF}, \sigma) = (a_1, a_2, \ldots, a_\ell)$ and $\mathcal{L}(\mathcal{A}, \sigma) = (b_1, b_2, \ldots, b_\ell)$ the lists maintained by MTF and $\mathcal{A}$ after serving σ, respectively. Suppose that c_j (resp., d_j) sequences in T_σ have a_j (resp., b_j) as their i^{th} request, for $1 \leq j \leq \ell$. Note that $\sum_{1 \leq j \leq l} c_j = \sum_{1 \leq j \leq l} d_j = |T_\sigma|$ and $(d_1, d_2, \ldots, d_\ell)$ is a permutation of $(c_1, c_2, \ldots, c_\ell)$.

We first show that $c_{j+1} \leq c_j$ for $1 \leq j < \ell$. Let C_j and C_{j+1} denote the set of sequences in T_σ that have a_j and a_{j+1} as their i^{th} request. We provide a one-to-one mapping from C_{j+1} to C_j which proves that $|C_{j+1}| \leq |C_j|$. We map every sequence τ in C_{j+1} to a sequence τ' in C_j by replacing every a_j with a_{j+1} and every a_{j+1} by a_j, starting from position i. Since a_j occurs before a_{j+1} in MTF's list after serving σ, we know that the last request to a_j occurs after the last request to a_{j+1} in σ. Therefore if τ is consistent with f, so is τ'. Thus every sequence in C_{j+1} is mapped to a unique sequence in C_j and we have $c_{j+1} = |C_{j+1}| \leq |C_j| = c_j$.

Therefore $(c_1, c_2, \ldots, c_\ell)$ is a permutation of $(d_1, d_2, \ldots, d_\ell)$ in non-increasing order, and thus $\mathcal{F}_{i,m}(\text{MTF} \,|\, \sigma) = \sum_{1 \leq j \leq \ell} j \times c_j \leq \sum_{1 \leq j \leq \ell} j \times d_j = \mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma)$. Now since

$$\mathcal{F}_{i,m}(\text{MTF}) = \sum_{\sigma \in \mathcal{I}_{i-1}} \mathcal{F}_{i,m}(\text{MTF} \,|\, \sigma) \text{ and } \mathcal{F}_{i,m}(\mathcal{A}) = \sum_{\sigma \in \mathcal{I}_{i-1}} \mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma),$$

we get $\mathcal{F}_{i,m}(\text{MTF}) \leq F_{i,m}(\mathcal{A})$. We have

$$\sum_{\sigma \in \mathcal{I}_m^f} \text{MTF}(\sigma) = \sum_{1 \leq i \leq m} \mathcal{F}_{i,m}(\text{MTF}) \leq \sum_{1 \leq i \leq m} \mathcal{F}_{i,m}(\mathcal{A}) = \sum_{\sigma \in \mathcal{I}_m^f} \mathcal{A}(\sigma).$$

Thus MTF (m, f)-dominates $\mathcal{A}$ for every concave* function f, and every integer $m \geq 1$. Hence MTF dominates $\mathcal{A}$. □

Corollary 5.2. *For any* concave* *function* f *and any online list update algorithm* $\mathcal{A}$*,*

$$\text{MTF} \preceq_a^f \mathcal{A}.$$

Corollary 5.2 demonstrates that MTF is an optimal algorithm according to Average Analysis on sequences with locality of reference. As in the case of the paging problem (and optimality of LRU), a natural question is whether MTF is a unique optimum or not. The following theorem shows that no other list update algorithm can be optimal in this model (assuming that the algorithm is oblivious of the function f.)

Theorem 5.3. *No online list update algorithm (other than* MTF *itself) dominates* MTF*.*

Proof. Assume by way of contradiction that an online list update algorithm $\mathcal{A}$ dominates MTF and that $\mathcal{A}$ is different from MTF. According to the definition, there exists an integer $m_0 \geq 1$ so that for each $m \geq m_0$ and every concave* function f, $\mathcal{A}$ (m, f)-dominates MTF, i.e.,

$$\sum_{\sigma \in \mathcal{I}_m^f} \mathcal{A}(\sigma) \leq \sum_{\sigma \in \mathcal{I}_m^f} \text{MTF}(\sigma).$$

Following the proof of Lemma 5.3, this holds only if $\mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma) = \mathcal{F}_{i,m}(\text{MTF} \,|\, \sigma)$ for every $m \geq m_0$, $2 \leq i \leq m$, and every sequence σ of length $i-1$. Let $\sigma \in \mathcal{I}_{i-1}^f$ be a sequence so that $\mathcal{L}(\mathcal{A}, \sigma)$ is different from $\mathcal{L}(\text{MTF}, \sigma)$, k be the largest index so that $x = a_k \neq b_k = y$ (for a_k and b_k defined as in Lemma 5.3), and p be the smallest index so that $\sigma[p..i-1]$ contains at most $k-1$ distinct items. Select the concave* function f so that $\lfloor f(i-p) \rfloor = \lfloor f(i-p+1) \rfloor = k-1$. Observe

that $\sigma[p..i-1] = \{a_1, a_2, \ldots, a_{k-1}\}$, $y \in \{a_1, a_2, \ldots, a_{k-1}\}$, and $x = a_k$. Therefore $y \in \sigma[p..i-1]$, and $x \notin \sigma[p..i-1]$. Thus we have $c_k = 0 < d_k$ (the sequence of length $m > i$ obtained by repeating y in all positions starting from i^{th} position is consistent with f). Therefore

$$\mathcal{F}_{i,m}(\text{MTF} \,|\, \sigma) = \sum_{1 \le j \le \ell} j \times c_j < \sum_{1 \le j \le \ell} j \times d_j = \mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma),$$

which is a contradiction. □

We have thus obtained the following.

Theorem 5.4. *Let $\mathcal{A}$ be an online list update algorithm other than* MTF. *Then* MTF $\preceq_a^f \mathcal{A}$ *and there exists at least one* concave* *function f so that*

$$\mathcal{A} \not\preceq_a^f \text{MTF}, \qquad \textit{which implies} \qquad \mathcal{A} \not\preceq_b^f \text{MTF}.$$

We can prove separation with respect to Bijective Analysis between MTF and specific algorithms, e.g., TR, for a much larger family of concave* functions.

Theorem 5.5. *For all* concave* *functions f such that $f(\ell) < \ell$ (ℓ is the size of list),*

$$\text{TR} \not\preceq_b^f \text{MTF}.$$

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list. Assume by way of contradiction that TR $\preceq_b^f$ MTF. Therefore there is an integer $n_0 \ge 1$ so that for each $n \ge n_0$, there is a bijection $b : \mathcal{I}_n^f \leftrightarrow \mathcal{I}_n^f$ satisfying $\text{TR}(\sigma) \le \text{MTF}(b(\sigma))$ for each $\sigma \in \mathcal{I}_n^f$. Now consider a sequence σ of length $m \ge n_0$ obtained by considering the prefix of size m of the infinite sequence $a_\ell a_{\ell-1} a_\ell a_{\ell-1} \ldots$. TR incurs a cost of ℓ on each request and we have $\text{TR}(\sigma) = m \times \ell$. Note that σ is consistent with f, because it has two distinct items.[1] Thus $\sigma \in \mathcal{I}_m^f$ and from the assumption there should exist some sequence $\sigma' \in \mathcal{I}_m^f$ so that $m \times \ell = \text{TR}(\sigma) \le \text{MTF}(\sigma')$. Therefore MTF should incur a cost of ℓ on each request of σ'. Hence σ' should be a prefix of the sequence $a_\ell a_{\ell-1} a_{\ell-2} \ldots a_1 a_\ell a_{\ell-1} a_{\ell-2} \ldots a_1 \ldots$. Note however that any window of size ℓ in σ' has ℓ distinct items. Since we started with $f(\ell) < \ell$, σ' is not consistent with f and this contradicts the assumption that $\sigma' \in \mathcal{I}_m^f$. □

Experimental Results and Analysis

In this section we test the validity of the locality of reference assumption as described in Section 5.2.1 against experimental data. For our experiments, we considered the fourteen files of the Calgary Compression Corpus [151] which are frequently used as a standard benchmark for file compression. Recall that list update algorithms can be used in a very direct way in file compression. For each file, we computed the maximum number of characters in windows of all possible sizes, up to the size of the whole file. Figures 5.1 and 5.2 show the resulting graphs. Note that since we observed that the maximum number of distinct items does not change much as we increase the size of window to values more than 3500, we only show the results for windows of size up to 3500.

As can be seen from these graphs, the curves have an overall concave shape. We should note that for some of the input files, the function we obtained is not concave for some intervals.

[1] We can assume that $f(2) = 2$ because otherwise we are restricted to sequences that contain only one item.

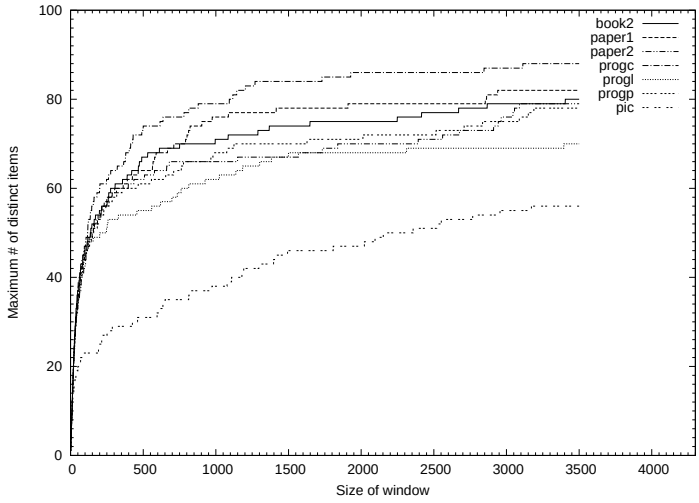

Figure 5.1: Maximum number of distinct characters in windows of size up to 3500 for the files in Calgary Compression Corpus.

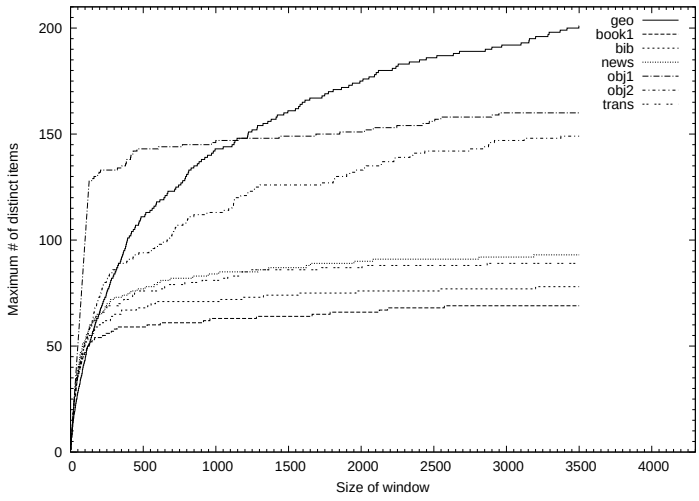

Figure 5.2: Maximum number of distinct characters in windows of size up to 3500 for the files in Calgary Compression Corpus.

However, this is not a major concern, since we can bound said function by any concave function f which is such that $f(i)$ is an upper bound on the maximum number of distinct items in windows of size i. For instance, we can take the upper convex hull of the data points. In fact, Albers et al. [6] observed that similar non-concavity (mostly localized within small intervals) was present in their experimental results in typical request sequences for the paging problem. Albers et al. put forth this argument to justify the fact that local small deviations from concavity do not impose a serious problem.

Albers and Mitzenmacher [8] compared the efficiency of MTF and TS algorithms for compressing the files of the Calgary Compression Corpus. After accessing an item a, TS inserts a in front of the first item b that appears before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorganize the list. They compared MTF and TS in two settings: with or without Burrows-Wheeler transform (BWT). Informally, BWT transforms a string to one of its permutations that has more locality of reference, which is hence more readily compressible [51, 106]. We will describe the BWT in more detail in Chapter 10. Their results show that although TS outperforms MTF on compression without BWT, MTF usually has better performance when we use BWT. This is consistent with our results as BWT is a transform designed with the goal of increasing the locality of reference in the representation of the string.

5.3 Conclusions

In this chapter we applied Bijective Analysis and Average Analysis to paging with lookahead and list update. We first showed that unlike competitive analysis, Bijective Analysis reflects the influence of lookahead for paging. We then turned our attention to the list update problem, in particular under the cost formulation of Martínez and Roura, and Munro. We showed that under this cost model, all algorithms for list update are equivalent according to Bijective Analysis, a result that parallels the equivalence of all lazy paging algorithms. Next, we investigated the issue of locality of reference in list update, which, unlike the paging problem, has received considerably less attention in the literature. We provided experimental evidence that the model of Albers et al. can be adapted to applications of list update that emerge from text compression. We then applied Average Analysis over request sequences exhibiting locality of reference, and proved that MTF is the unique optimal algorithm.

In future work we intend to apply bijective and Average Analysis to other online problems. Natural candidates here are the k-server problem and file caching. We would be very interested to extend these techniques so as to be able to compare randomized online algorithms. Since the introduction of these measures, further applications of these techniques emerged. More specifically, Angelopoulos and Schweitzer [14] extended the optimality of LRU to Bijective Analysis. In addition, Boyar, Irani and Larsen [42] showed that the greedy algorithm is optimal according to Bijective Analysis for the 2-server problem on three co-linear points. Last, but not least, we would be interested in the study of relaxed versions of Bijective Analysis, e.g., by requiring that $A(\sigma) \leq cB(\pi(\sigma))$, for some constant c.

Chapter 6

Relative Interval Analysis

In this chapter we give a finer separation of several known paging algorithms using a new technique called *relative interval analysis*. This technique compares the fault rate of two paging algorithms across the entire range of inputs of a given size rather than in the worst case alone. Using this technique we characterize the relative performance of several paging algorithms. We also show that lookahead is beneficial for a paging algorithm under this model as well.

In Chapter 4 we introduced Bijective Analysis and Average Analysis which combined with the locality model of Albers et al. [6], shows that LRU is the sole optimal paging algorithm on sequences with locality of reference. This resolved an important disparity between theory and practice of online paging algorithms, namely the superiority in practice of LRU.

A remaining question however, is how to characterize the full spectrum of performance of the various known paging algorithms. The competitive ratio focuses on the worst case which in this setting is known to be the same for all algorithms. In this chapter we compare instead the performance of two algorithms across the entire range of inputs; in that comparison we use the fault rate measure instead of the competitive ratio. Aside from artifacts introduced by the comparison to an offline algorithm, practitioners find the fault rate a better indicator of performance. The idea behind the fault rate is that sequences on which $\mathcal{A}$ incurs very few faults compared to the number of requests are not that important, even if the number of faults happens to be much higher than what can be achieved by an offline (or even online) optimum. We show this using an example. Let $\mathcal{A}$ and $\mathcal{B}$ two online paging algorithms. Suppose that the fault rate of $\mathcal{A}$ is generally much lower than that of $\mathcal{B}$, so clearly $\mathcal{A}$ is preferable to $\mathcal{B}$. However, if there happens to be an "easy" sequence σ of length 1000000 on which $\mathcal{A}$ incurs 100 faults, $\mathcal{B}$ incurs 10 faults and optimal offline algorithm can serve σ by only 2 faults, then the competitive ratio of $\mathcal{A}$ is 50 while that of $\mathcal{B}$ is 5 suggesting the opposite of the logical conclusion. Note that the fault rate of $\mathcal{A}$ and $\mathcal{B}$ on σ is 0.01 and 0.001, respectively, which is miniscule and thus of no relevance to the actual performance of a system using either algorithm.

Our results In this chapter we aim to provide a tool for finer study and separation—as compared to the competitive ratio—of the relative performance characteristics of online paging algorithms. We propose the relative interval which directly compares two online paging algorithms $\mathcal{A}$ and $\mathcal{B}$, without any reference to the optimal offline algorithm. They are compared across their entire performance spectrum (rather than on the worst case alone) using a normalized measure

Table 6.1: Summary of the results for relative intervals of several paging algorithms.

	LRU	FWF	FIFO	LIFO	LRU-2
LRU					
FWF	$[0, \frac{k-1}{k}]$				
FIFO	$\supseteq [-\frac{k-1}{k}, \frac{k-1}{2k-1}]$	$[-\frac{k-1}{k}, 0]$			
LIFO	$[-\frac{k-1}{k}, 1]$		$[-\frac{k-1}{k}, 1]$		
LRU-2	$\supseteq [-\frac{k-1}{2k}, \frac{k-1}{k+1}]$				

of performance, similar to the fault rate. Informally the relative interval of two algorithms reflects the range of the difference between the fault rate of those algorithms. For every two online paging algorithms $\mathcal{A}$ and $\mathcal{B}$ we define a relative interval $\mathcal{I}(\mathcal{A}, \mathcal{B}) = [\alpha, \beta]$, where $-1 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$. Intuitively, $\beta > -\alpha$ shows that $\mathcal{B}$ is better than $\mathcal{A}$ the according to the relative interval. The greater the difference, the better $\mathcal{B}$ is compared to $\mathcal{A}$. Table 6.1 shows the summary of our results about the relative intervals of well known paging algorithms. These results show that LRU and FIFO are better than FWF, a result expected from practice and experience, yet not fully reflected by the competitive ratio model. We also show that the relative interval has another good feature, namely we prove that it reflects the influence of lookahead.

6.1 Relative Interval

In relative interval analysis we directly compare two online algorithms, i.e., we do not use the optimal offline algorithm as the baseline of our comparisons. Let $\mathcal{A}$ and $\mathcal{B}$ be two online algorithms for the same minimization problem, e.g., paging. Denote the cost of $\mathcal{A}$ on a sequence σ by $\mathcal{A}(\sigma)$. We define the following two functions:

$$Min_{\mathcal{A},\mathcal{B}}(n) = \min_{|\sigma|=n} \{\mathcal{A}(\sigma) - \mathcal{B}(\sigma)\},$$

and

$$Max_{\mathcal{A},\mathcal{B}}(n) = \max_{|\sigma|=n} \{\mathcal{A}(\sigma) - \mathcal{B}(\sigma)\}.$$

Using these functions we define

$$Min(\mathcal{A}, \mathcal{B}) = \liminf_n \frac{Min_{\mathcal{A},\mathcal{B}}(n)}{n}, \quad \text{and} \quad Max(\mathcal{A}, \mathcal{B}) = \limsup_n \frac{Max_{\mathcal{A},\mathcal{B}}(n)}{n}.$$

Note that $Min(\mathcal{A}, \mathcal{B}) = -Max(\mathcal{B}, \mathcal{A})$ and $Max(\mathcal{A}, \mathcal{B}) = -Min(\mathcal{B}, \mathcal{A})$. Now we are ready to define the *relative interval* of $\mathcal{A}$ and $\mathcal{B}$ as

$$\mathcal{I}(\mathcal{A}, \mathcal{B}) = [Min(\mathcal{A}, \mathcal{B}), Max(\mathcal{A}, \mathcal{B})].$$

This interval gives useful information about the relative performance of $\mathcal{A}$ and $\mathcal{B}$. If $Max(\mathcal{A}, \mathcal{B}) > |Min(\mathcal{A}, \mathcal{B})|$ then $\mathcal{B}$ has better performance than $\mathcal{A}$ in this model. In particular, if $\mathcal{I}(\mathcal{A}, \mathcal{B}) = [0, \beta]$ for $\beta > 0$ we say that $\mathcal{B}$ *dominates* $\mathcal{A}$. Note that in this case $\mathcal{A}$ does not have better performance than $\mathcal{B}$ on any sequence (asymptotically), while $\mathcal{B}$ outperforms $\mathcal{A}$ on some sequences. Also if $Max(\mathcal{A}, \mathcal{B})$ is close to 0 then we can conclude that $\mathcal{A}$ is not much worse than $\mathcal{B}$ on any sequences. We can interpret other situations in an analogous way.

Example 6.1. *We use a simple example to illustrate the above definitions. Let $\mathcal{A}$ and $\mathcal{B}$ be two online algorithms and $\mathcal{I}_n = \{\sigma_1, \sigma_2, \cdots, \sigma_{10}\}$ be the set of all possible request sequences of length $n = 5000$. Suppose that the cost of $\mathcal{A}$, $\mathcal{B}$, and the optimal offline algorithm OPT on request sequences are as follows:*

σ	$\mathcal{A}(\sigma)$	$\mathcal{B}(\sigma)$	$OPT(\sigma)$	$\mathcal{A}(\sigma) - \mathcal{B}(\sigma)$
σ_1	*200*	*800*	*100*	*-600*
σ_2	*600*	*550*	*500*	*50*
σ_3	*70*	*90*	*40*	*-20*
σ_4	*20*	*10*	*2*	*10*
σ_5	*1000*	*2000*	*700*	*-1000*
σ_6	*500*	*600*	*200*	*-100*
σ_7	*120*	*100*	*80*	*20*
σ_8	*1386*	*2007*	*1360*	*-621*
σ_9	*140*	*270*	*50*	*-130*
σ_{10}	*2007*	*2000*	*1981*	*7*

We have $Min_{\mathcal{A},\mathcal{B}}(n) = -1000$ and $Max_{\mathcal{A},\mathcal{B}}(n) = 50$. The difference between the fault rates of $\mathcal{A}$ and $\mathcal{B}$ on these sequences ranges between -20% and 1%. Restricting our attention to sequences of length 5000 we can say $\mathcal{I}(\mathcal{A},\mathcal{B}) = [-20\%, 1\%]$. Therefore $\mathcal{A}$ is better than $\mathcal{B}$ according to relative interval analysis. On the other hand, the competitive ratio of $\mathcal{A}$ and $\mathcal{B}$ on these sequences is 10 (occurring on σ_4) and 8 (occurring on σ_1), respectively. Although $\mathcal{A}$ has better performance than $\mathcal{B}$ on most sequences, its bad performance on a single sequence, namely σ_4, results in a worse competitive ratio.

We compute the value of $Min(\mathcal{A},\mathcal{B})$ and $Max(\mathcal{A},\mathcal{B})$ for various choices of $\mathcal{A}$ and $\mathcal{B}$. In some cases we obtained bounds or approximation of the values thereof. We say that $[\alpha, \beta]$ approximates the relative interval of $\mathcal{A}$ and $\mathcal{B}$ if $Min(\mathcal{A},\mathcal{B}) \leq \alpha$ and $\beta \leq Max(\mathcal{A},\mathcal{B})$. We denote this by $\mathcal{I}(\mathcal{A},\mathcal{B}) \supseteq [\alpha, \beta]$.

6.2 Comparison to Other Measures

We can directly compare two online algorithms using relative interval analysis. Recall from Chapter 3 that the other measures which directly compare two online algorithms are the Max/Max ratio, the relative worst order ratio, Bijective Analysis, and Average Analysis. The Max/Max ratio reflects the influence of lookahead, but it does not provide better separation than the competitive ratio, e.g., LRU and FWF are equivalent under this measure. The relative worst order ratio reflects the advantage of lookahead and separates the performance of LRU and FWF. Thus this measure gives results comparable to the relative interval. However, it is not intuitive why we should consider all permutations of a sequence for comparing two online paging algorithms (this might be more straightforward for other problems, e.g., bin packing.). Furthermore, the relative interval provides a more comprehensive measure by considering the whole range of possible differences between the performance of two algorithms. Note that in the Max/Max ratio and the relative worst order ratio we only consider the worst case sequence (among all sequences of the same length and all permutations of a sequence, respectively).

The influence of lookahead is reflected by bijective analysis and average analysis. These measures also separate the performance of LRU and FWF. However, all lazy paging algorithms are equivalent according to Bijective Analysis and Average Analysis. Therefore most paging algorithms have the same performance under these measures. However, under the locality of reference model of [6] LRU is the unique optimal deterministic online algorithm under average analysis. This is consistent with the well known fact that LRU is the superior paging algorithm in practice. The relative interval does not reflect the unique optimality of LRU, but this also applies to all other measures that do not incorporate the locality of reference assumption. An interesting extension to this work would be to combine the relative interval with models for locality of reference. We believe that this will lead to better separation results. For example, although LRU beats LIFO under the relative interval, they have close performance. This is not consistent with practice, where LRU behaves much better than LIFO. However, note that LIFO and LRU are equivalent under plain bijective analysis and average analysis. Their performances are only separated when we assume locality of reference. We believe that this would be the case for the relative interval as well.

Finally, we explain a potential drawback of the relative interval. This drawback is due to the fact that the relative interval does not consider the whole distribution of the difference between the performance of two algorithms. Instead, it only considers the extreme values (minimum and maximum). This dependence on extreme cases simplifies the model and makes it more practical, but it leads to the same potential shortcomings as measures based on the worst-case analysis. We elaborate this using a simple example. Consider the difference between the performance of two algorithms $\mathcal{A}$ and $\mathcal{B}$ on all sequences of length n. Assume that $\mathcal{A}(\sigma) - \mathcal{B}(\sigma)$ is -2000 for a single sequence and it gets values between -1 and 1000 for all other sequences. This shows that a single sequence can considerably affect the relative interval.

6.3 Relative Interval Applied to Paging Algorithms

In this section we compare several well known paging algorithms using the new model.

Theorem 6.1. *For any two online paging algorithms $\mathcal{A}$ and $\mathcal{B}$,*

$$0 \leq Max(\mathcal{A}, \mathcal{B}) \leq 1.$$

Proof. For any n, there is a sequence σ of length n so that $\mathcal{A}(n) = n$, i.e., $\mathcal{A}$ incurs a fault on every request of σ. This sequence can be obtained by requesting, at each step, the page that is evicted by $\mathcal{A}$ in the previous step. $\mathcal{B}$ incurs at most n faults on every sequence of length n. Therefore $\mathcal{B}(\sigma) \leq n$ and $\mathcal{A}(\sigma) - \mathcal{B}(\sigma) \geq 0$. Thus $\max_{|\sigma|=n}\{\mathcal{A}(\sigma) - \mathcal{B}(\sigma)\} \geq 0$. Since this holds for every n, we have $Max(\mathcal{A}, \mathcal{B}) \geq 0$. For the upper bound, note that for every sequence σ of length n, we have

$$\mathcal{A}(n) \leq n \;\Rightarrow\; \mathcal{A}(n) - \mathcal{B}(n) \leq n \;\Rightarrow\; \frac{A(n) - B(n)}{n} \leq 1.$$

Therefore $Max(\mathcal{A}, \mathcal{B}) \leq 1$. □

Corollary 6.1. *For any two online paging algorithms $\mathcal{A}$ and $\mathcal{B}$,*

$$-1 \leq Min(\mathcal{A}, \mathcal{B}) \leq 0.$$

Theorem 6.2. $\mathcal{I}(\text{FWF}, \text{LRU}) = [0, \frac{k-1}{k}]$.

Proof. Let σ be an arbitrary sequence of length n and consider the partition of σ to phases of marking algorithms. At each phase, FWF incurs exactly k faults while LRU incurs at most k faults. Therefore the cost of LRU on σ is at most the cost of FWF on σ and we have $Min(\text{FWF}, \text{LRU}) \geq 0$. According to Corollary 6.1 we have $Min(\text{FWF}, \text{LRU}) = 0$. At each phase, LRU incurs at least one fault and each phase has length at least k. Therefore $Max(\text{FWF}, \text{LRU}) \leq \frac{k-1}{k}$. Consider the following sequence for some arbitrary integer m: $\sigma = \{p_1 p_2 \cdots p_k p_{k+1} p_2 p_3 \cdots p_k\}^m$. We have $\text{FWF}(\sigma) = 2k \times m$ and $\text{LRU}(\sigma) = k + 1 + 2 \times (m-1)$ and thus $Max(\text{FWF}, \text{LRU}) \geq \frac{k-1}{k}$. □

We can extend this argument to all lazy conservative and lazy marking algorithms.

Theorem 6.3. *Let $\mathcal{A}$ be an arbitrary lazy conservative or lazy marking algorithm. Then we have* $\mathcal{I}(\text{FWF}, \mathcal{A}) = [0, \frac{k-1}{k}]$.

Proof. Let σ be an arbitrary sequence and φ be an arbitrary marking phase of σ. Any marking algorithm incurs at most k faults on φ. Also since φ contains k distinct pages, any conservative algorithm incurs at most k faults in this phase as well. Thus $\mathcal{A}$ incurs at most k faults on φ. Since FWF incurs exactly k faults on φ we have $Min(\text{FWF}, \mathcal{A}) \geq 0$. Then we get $Min(\text{FWF}, \mathcal{A}) = 0$ by applying Corollary 6.1. Now we prove $Max(\text{FWF}, \mathcal{A}) \geq \frac{k-1}{k}$ by constructing the following sequence σ which contains $k+1$ distinct pages and starts by $p_1 p_2 \cdots p_k p_{k+1}$. After this the sequence contains several blocks of size k. At the beginning of each block B the cache of FWF contains only one page, say p. Also since $\mathcal{A}$ is a lazy paging algorithm its cache contains all of these pages save one (say q). B consists of requests to each of these distinct pages with the exception of p, thus it has length exactly k. In addition, we make sure that the request to q is the last request in B. Thus the last request of B is a fault for both algorithms on which $\mathcal{A}$ evicts one page while FWF flushes its cache and brings only that last request to its cache. Now we can construct a new block in a similar way. Therefore on each block of length k, FWF and $\mathcal{A}$ incur cost k and 1, respectively and we have $Max(\text{FWF}, \mathcal{A}) \geq \frac{k-1}{k}$. At each marking phase φ, FWF incurs k faults and $\mathcal{A}$ incurs at least one fault. Also φ has length at least k. Therefore $Max(\text{FWF}, \mathcal{A}) \leq \frac{k-1}{k}$. □

Theorem 6.4. *For any conservative algorithm $\mathcal{A}$ and any online algorithm $\mathcal{B}$, we have* $Max(\mathcal{A}, \mathcal{B}) \leq \frac{k-1}{k}$.

Proof. Let σ be an arbitrary sequence of length n and partition σ into blocks so that $\mathcal{B}$ incurs a fault only on the first request of each block. Therefore each block has at most k distinct pages and $\mathcal{A}$ incurs at most k faults in each block. Let $b_1, b_2, \cdots, b_d$ be the sizes of blocks of σ. Then we have $\mathcal{B}(\sigma) = d$ and $\mathcal{A}(\sigma) \leq \sum_{b_i \leq k} b_i + \sum_{b_i > k} k$. Therefore

$$\begin{aligned}\frac{\mathcal{A}(\sigma) - \mathcal{B}(\sigma)}{n} &\leq \frac{\sum_{b_i \leq k} b_i + \sum_{b_i > k} k - d}{\sum_{b_i \leq k} b_i + \sum_{b_i > k} b_i} \\ &\leq \frac{\sum_{b_i \leq k} (b_i - 1) + \sum_{b_i > k} (k-1)}{\sum_{b_i \leq k} b_i + \sum_{b_i > k} k}.\end{aligned}$$

If $b_i \leq k$, we have $\frac{b_i-1}{b_i} \leq \frac{k-1}{k}$ and thus

$$\frac{\sum_{b_i \leq k}(b_i - 1)}{\sum_{b_i \leq k} b_i} \leq \frac{k-1}{k}.$$

Also we have

$$\frac{\sum_{b_i > k}(k - 1)}{\sum_{b_i > k} k} \leq \frac{k-1}{k}.$$

Therefore $\frac{\mathcal{A}(\sigma)-\mathcal{B}(\sigma)}{n} \leq \frac{k-1}{k}$. Since this is true for any σ, we have $Max(\mathcal{A}, \mathcal{B}) \leq \frac{k-1}{k}$. □

Theorem 6.5. $\mathcal{I}(\text{LIFO}, \text{LRU}) = [-\frac{k-1}{k}, 1]$.

Proof. Since LRU is conservative, according to Theorem 6.4 we have

$$Max(\text{LRU}, \text{LIFO}) \leq \frac{k-1}{k} \Rightarrow Min(\text{LIFO}, \text{LRU}) \geq -\frac{k-1}{k}.$$

Now consider the sequence $\sigma = \{p_1 p_2 \cdots p_k p_{k+1}\}^m$. LRU incurs a fault on every request of σ while LIFO incurs a fault on every kth request. Thus

$$Min(\text{LIFO}, \text{LRU}) \leq -\frac{k-1}{k} \Rightarrow Min(\text{LIFO}, \text{LRU}) = -\frac{k-1}{k}.$$

For the other direction consider the sequence $\sigma = p_1 p_2 \cdots p_k p_{k+1} \{p_k p_{k+1}\}^m$. LIFO incurs a fault on every request while LRU only incurs a fault on the first $k+1$ pages. Since m can be arbitrarily large, we have $Max(\text{LIFO}, \text{LRU}) \geq 1$. This, together with Theorem 6.1 imply $Max(\text{LIFO}, \text{LRU}) = 1$. □

A similar argument on the same sequences implies the following theorem.

Theorem 6.6. $\mathcal{I}(\text{LIFO}, \text{FIFO}) = [-\frac{k-1}{k}, 1]$.

Theorem 6.7. $\mathcal{I}(\text{FIFO}, \text{LRU}) \supseteq [-\frac{k-1}{k}, \frac{k-1}{2k-1}]$.

Proof. $Max(\text{FIFO}, \text{LRU})$: Consider the following sequence σ that consists of $k+1$ distinct pages: σ starts with $\sigma_0 = p_1 p_2 \ldots p_k$. After this initial subsequence, σ consists of several blocks. Each block starts right after the previous block and contains $2k-1$ requests to k distinct pages. The first k blocks of σ are shown in Fig. 6.1. The blocks repeat after this, i.e., the $(k+1)$th block is the same as the first block, the $(k+2)$th block is the same as the second block and so on. It is easy to verify that FIFO incurs a fault on the last k requests of each block while LRU only incurs a fault on the middle request of every block. Let σ have m blocks. Then we have $\text{FIFO}(\sigma) = k + m \times k$ and $\text{LRU}(\sigma) = k + m$. Therefore

$$\frac{\text{FIFO}(\sigma) - \text{LRU}(\sigma)}{|\sigma|} = \frac{m(k-1)}{k + m(2k-1)},$$

and for sufficiently large values of m, this value becomes arbitrarily close to $\frac{k-1}{2k-1}$.

$Min(\text{FIFO}, \text{LRU})$: Consider the following sequence σ' that consists of $k+1$ distinct pages: σ' starts with $\sigma'_0 = p_1 p_2 \ldots p_k p_{k-1} p_{k-2} \ldots p_1$. After this initial subsequence, σ' consists of m blocks. The first k blocks of σ' are shown in Fig. 6.2. The blocks repeat after this, e.g., the $(k+1)$th

$$\begin{pmatrix}
p_{k-1} & p_{k-2} & \cdots & p_2 & p_1 & \mathbf{p_{k+1}} & p_1 & p_2 & \cdots & p_{k-1} \\
p_{k-2} & p_{k-3} & \cdots & p_1 & p_{k+1} & \mathbf{p_k} & p_{k+1} & p_1 & \cdots & p_{k-2} \\
p_{k-3} & p_{k-4} & \cdots & p_{k+1} & p_k & \mathbf{p_{k-1}} & p_k & p_{k+1} & \cdots & p_{k-3} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
p_k & p_{k-1} & \cdots & p_3 & p_2 & \mathbf{p_1} & p_2 & p_3 & \cdots & p_k
\end{pmatrix}$$

Figure 6.1: Blocks of the sequence σ in the proof of Theorem 6.7; each row of the matrix represents a block.

$$\begin{pmatrix}
\mathbf{p_{k+1}} & p_k & p_{k-1} & \cdots & p_3 & p_2 \\
\mathbf{p_1} & p_{k+1} & p_k & \cdots & p_4 & p_3 \\
\mathbf{p_2} & p_1 & p_{k+1} & \cdots & p_5 & p_4 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\mathbf{p_k} & p_{k-1} & p_{k-2} & \cdots & p_2 & p_1
\end{pmatrix}$$

Figure 6.2: Blocks of sequence σ' in the proof of Theorem 6.7.

block is the same as the first block. It is easy to verify that LRU incurs a fault on all k requests of each block while FIFO only incurs a fault on the first request of every block. Then we have $\text{LRU}(\sigma) = k + m \times k$ and $\text{FIFO}(\sigma) = k + m$. Therefore

$$\frac{\text{FIFO}(\sigma) - \text{LRU}(\sigma)}{|\sigma|} = \frac{-m(k-1)}{k+mk},$$

and for sufficiently large values of m, this value becomes arbitrarily close to $-\frac{k-1}{k}$.

□

Theorem 6.8. $Max(\text{LRU-2}, \text{LRU}) \geq \frac{k-1}{k+1}$.

Proof. Consider the sequence σ obtained by repeating m times the block $p_1 p_2 \ldots p_{k-1} p_k p_k p_{k-1} \ldots p_1 p_{k+1} p_{k+1}$. In the first block, LRU incurs $k+1$ faults. In every other block, it only incurs two faults, one on the first request to p_k, and the other on the first request to p_{k+1}. Therefore we have $\text{LRU}(\sigma) = k+1+2(m-1) = 2m+k-1$. LRU-2 incurs $k+1$ faults in the first block and $2k$ faults in every other block; it has a hit only on the second requests to p_k and p_{k+1} in each block (other than the first block). Therefore we have $\textit{LRU-2}(\sigma) = k+1+2k(m-1) = 2km-k+1$. Thus

$$\frac{\textit{LRU-2}(\sigma) - \text{LRU}(\sigma)}{|\sigma|} = \frac{2km-k+1-2m-k+1}{m(2k+2)} = \frac{m(2k-2)-2k+2}{m(2k+2)},$$

and for sufficiently large values of m, this value becomes arbitrarily close to $\frac{2k-2}{2k+2} = \frac{k-1}{k+1}$. □

Theorem 6.9. $Max(\textit{LRU-2}, \text{LRU}) \leq \frac{k-1}{k}$.

Proof. Let σ be an arbitrary sequence of length n and partition σ into blocks so that LRU incurs a fault only on the first request of each block. Let $B_1, B_2, \ldots, B_d$ be the blocks of σ, and b_i be the size of block B_i. Then we have $\mathrm{LRU}(\sigma) = d$ and $\mathit{LRU\text{-}2}(\sigma) \leq \sum_{1 \leq i \leq d} b_i$. We show that LRU-2 incurs at most k faults in each block. We first show B_i contains at most k distinct pages. B_1 contains requests to one page and LRU-2 incurs one fault on it. Consider an arbitrary block B_i for $i > 1$, let p be the first request of B_i, and let p_1, p_2, ..., p_{k-1} be the $k-1$ most recently used pages before the block B_i in this order. We have $p \notin P = \{p_1, p_2, \ldots, p_{k-1}\}$, because LRU incurs a fault on p. We claim that each request of B_i is either to p or to a page of P. Assume for the sake of contradiction that B_i contains a request to a page $q \notin \{p\} \cup P$ and consider the first request to q in B_i. All pages $p, p_1, p_2, \ldots, p_{k-1}$ are requested since the previous request to q. Therefore at least k distinct pages are requested since the last request to q and LRU incurs a fault on q. This contradicts the definition of a block. Therefore B_i contains at most k distinct pages.

We claim that LRU-2 incurs at most one fault on every page q in block B_i. Assume that this is not true and LRU-2 incurs two faults on a page q in B_i. Therefore q is evicted at some point after it is first requested in B_i. Assume that this eviction happens on a fault in a request to a page r and consider the pages that are in LRU-2's cache just before that request. Since $r \in \{p\} \cup P$ is not in the cache and $|\{p\} \cup P| = k$, there is a page $s \notin \{p\} \cup P$ in the cache. Let t be the time of the last request to p_{k-1} before the block B_i. The last request to s is before t, while the second to last request to q is at time t or afterwards. Therefore LRU-2 does not evict q on this fault, which is a contradiction. Hence LRU-2 incurs at most k faults in each block of σ. Therefore

$$\begin{aligned}\frac{\mathit{LRU\text{-}2}(\sigma) - \mathrm{LRU}(\sigma)}{n} &\leq \frac{\sum_{b_i \leq k} b_i + \sum_{b_i > k} k - d}{\sum_{b_i \leq k} b_i + \sum_{b_i > k} b_i} \\ &\leq \frac{\sum_{b_i \leq k} (b_i - 1) + \sum_{b_i > k} (k-1)}{\sum_{b_i \leq k} b_i + \sum_{b_i > k} k}.\end{aligned}$$

If $b_i \leq k$, we have $\frac{b_i - 1}{b_i} \leq \frac{k-1}{k}$ and thus

$$\frac{\sum_{b_i \leq k} (b_i - 1)}{\sum_{b_i \leq k} b_i} \leq \frac{k-1}{k}.$$

Also we have

$$\frac{\sum_{b_i > k} (k-1)}{\sum_{b_i > k} k} \leq \frac{k-1}{k}.$$

Therefore

$$\frac{\mathit{LRU\text{-}2}(\sigma) - \mathrm{LRU}(\sigma)}{n} \leq \frac{k-1}{k}.$$

Since this is true for any σ, we have $\mathit{Max}(\mathit{LRU\text{-}2}, \mathrm{LRU}) \leq \frac{k-1}{k}$. □

Theorem 6.10. $\mathit{Min}(\mathit{LRU\text{-}2}, \mathrm{LRU}) \leq -\frac{k-1}{2k}$.

Proof. Consider the following sequence σ that consists of $k+1$ distinct pages. σ starts with $\sigma_0 = p_1 p_2 \ldots p_k$. After this initial subsequence, σ consists of m blocks. Each block starts right after the previous block. The ith block consists of one of the subsequences shown in Figure 6.3, depending on the parity of i. It is easy to verify that LRU incurs a fault on the last k requests

Odd:	p_k	p_{k-1}	$\dots$	p_2	p_1	p_{k+1}	p_k	p_{k-1}	$\dots$	p_3	p_2
Even:	p_2	p_3	$\dots$	p_k	p_{k+1}	p_1	p_2	p_3	$\dots$	p_{k-1}	p_k

Figure 6.3: A block of sequence σ in the proof of Theorem 6.10.

of each block while LRU-2 only incurs a fault on the middle request of every block, i.e., p_{k+1} in *Odd* blocks and p_1 in *Even* blocks. Then we have $\text{LRU}(\sigma) = k + m \times k$ and $\textit{LRU-2}(\sigma) = k + m$. Therefore

$$\frac{\textit{LRU-2}(\sigma) - \text{LRU}(\sigma)}{|\sigma|} = \frac{-m(k-1)}{k + m(2k)},$$

and for sufficiently large values of m, this value becomes arbitrarily close to $-\frac{k-1}{2k}$.

□

Therefore while *Max(*LRU-2,LRU*)* is almost 1, we have proven so far an upper bound of almost -1/2 for *Min(*LRU-2,LRU*)*. A natural question is whether we can improve this bound, i.e., prove that *Min(*LRU-2,LRU*)* is less than -1/2. We believe that this is not true and prove it for the case that we only have $k+1$ distinct pages (note that all our examples are using $k+1$ distinct pages). While this is a counterintuitive result, in the sense that LRU-2 is preferable in practice it adds to our understanding of the relative advantages of LRU and LRU-2. This results indicates that in the fault rate model LRU is also preferable to LRU-2 and hence additional assumptions need to be made (such as the independency of requests [123]) so that it accurately reflects the superiority of LRU-2 observed in practice.

Theorem 6.11. *If we have at most $k+1$ distinct pages then Min(*LRU-2,LRU*)*$\geq -1/2$.

Proof. We call a request a "disparity" if it is a fault for LRU and a hit for LRU-2. Note that only a disparity may reduce the value of *Min(*LRU-2,LRU*)*. Consider an arbitrary sequence $\sigma = \sigma_1\sigma_2 \dots \sigma_n$ and an arbitrary page p. Let S be the set of all k distinct pages other than p. We prove that between any two requests for p in σ causing a disparity there should be a request to p that is not a disparity. Assume for the sake of contradiction that this is not the case: σ_a and σ_b are disparity requests to p, and there is no request to p between them. Let σ_x be the last request to p before σ_a. Since p_a is a fault for LRU, it has evicted p between p_x and p_a. Therefore all members of S are requested between p_x and p_a. Similarly all pages of S are requested between p_a and p_b. Since p is at LRU-2's cache right before p_a, there should be at least one page in S that is not in its cache at that time. As all pages of S are requested between p_a and p_b, LRU-2 incurs at least one fault in this interval. Let p_y be the last request between p_a and p_b on which LRU-2 incurs a fault. We claim that LRU-2 should evict p on the request p_y. Assume that LRU-2 evicts a page $q \in S$ on the fault p_y. The next request to q would be a fault for LRU-2 and since p_y is its last fault between p_a and p_b and q is requested in this range, we conclude that q has been requested between p_a and p_y. However this means that the second last request to q is after p_x, while the second last request to p is at p_x. Thus LRU-2 should evict p at p_y, and p_b is a fault for LRU-2 which is a contradiction. Hence corresponding to each request that may reduce the value of *Min(*LRU-2,LRU*)* there is at least one request that does not. This proves the bound of $-1/2$ for *Min(*LRU-2,LRU*)*. □

Influence of lookahead

We demonstrate that the relative interval reflects the effects of lookahead. Consider the algorithm LRU(ℓ) as defined in Subsection 5.1. We showed there that LRU(ℓ) incurs no more faults than LRU on any sequence. Therefore $Min(\text{LRU}, \text{LRU}(\ell)) = 0$. Now consider the sequence $\sigma = \{p_1 p_2 \dots p_k p_{k+1}\}^m$. LRU incurs a fault on every request of σ while LRU (ℓ) incurs a fault on every lth request. Hence

$$Max(\text{LRU}, \text{LRU}(\ell)) \geq 1 - 1/l,$$

and thus LRU(ℓ) dominates LRU.

6.4 Conclusions and Open Questions

We introduced a fault rate based metric to compare paging algorithms and using this metric, we showed the superiority of LRU and FIFO over FWF. The metric also reflects the beneficial influence of lookahead.

Several natural open questions remain: filling in the remaining entries in Table 6.1 as well as refining the bounds that are not tight. Additionally we believe that the relative interval can be of interest in other online settings and even perhaps for the comparison of offline algorithms.

Chapter 7

Parameterized Analysis of Online Algorithms

As noted in Chapter 3, it is well-established that input sequences for paging and list update have locality of reference. In this chapter we analyze the performance of algorithms for these problems in terms of the amount of locality in the input sequence. We define a measure for locality that is based on Denning's working set model and express the performance of well known algorithms in term of this parameter.

This introduces parameterized-style analysis to online algorithms. The idea is that rather than normalizing the performance of an online algorithm by an (optimal) offline algorithm, we explicitly express the behavior of the algorithm in terms of two more natural parameters: the size of the cache and Denning's working set measure. This technique creates a performance hierarchy of paging algorithms which better reflects their intuitive relative strengths. Also it reflects the intuition that a larger cache leads to a better performance. We obtain similar separation between list update algorithms. Lastly, we show that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the parameterized case, which matches experimental results.

Recall that locality of reference for paging means that when a page is requested it is more likely to be requested in the near future. In the early days of computing, Denning recognized the locality of reference principle and modeled it using the well known *working set* model [64, 65]. He defined the working set of a process as the set of most recently used pages and addressed thrashing using this model. According to [66], the word "locality" was first used by Denning in discussions with Dennis and Belady. They noticed that programs showed locality behavior even when it was not explicitly planned. After the introduction of the working set model, the locality principle has been adopted in operating systems, databases, hardware architectures, compilers, and many other areas. Therefore it holds even more so today. Indeed, [66] states "locality of reference is one of the cornerstones of computer science."

Our results We apply parameterized analysis to paging and list update. We express the performance of well known paging and list update algorithms in terms of some measures of locality of reference. For paging, this leads to better separation than the competitive ratio. Furthermore, in contrast to competitive analysis it reflects the intuition that a larger cache leads to a better performance. We also provide experimental results that justify the applicability of our

measure in practice. We obtain bounds on the parameterized performance of several list update algorithms and prove the superiority of MTF. We also apply our measures to randomized list update algorithms and show that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the parameterized case.

7.1 Parameterized Analysis of Paging Algorithms

As stated before input sequences for paging show locality of reference in practice. We want to express the performance of paging algorithms on a sequence in terms of the amount of the locality in that sequence. Therefore we need a measure that assigns a number proportional to the amount of locality in each sequence. None of the previously described models provide a unique numerical value as a measure of locality of reference[1]. We define a quantitative measure for non-locality of paging instances.

Definition 7.1. *For a sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ we define $d_\sigma[i]$ as either $k+1$ if this is the first request to page σ_i, or otherwise, the number of distinct pages that are requested since the last request to σ_i (including σ_i).*[2] *Now we define $\overline{\lambda}(\sigma)$, the "non-locality" of σ, as $\overline{\lambda}(\sigma) = \frac{1}{|\sigma|}\sum_{1\le i\le|\sigma|} d_\sigma[i]$. We denote the non-locality by $\overline{\lambda}$ if the choice of σ is clear from the context.*

If σ has high locality of reference, the number $d_\sigma[i]$ of distinct pages between two consecutive requests to a page is small for most values of i and thus σ has low non-locality. Note that while this measure is related to the working set model [64] and the locality model of [6], it differs from both in several aspects. Albers et al. [6] consider the maximum/average number of distinct pages in all windows of the same size, while we consider the number of distinct pages requested since the last access to each page. Also our analysis does not depend on a concave function f whose identification for a particular application might not be straightforward. Our measure is also closely related to the working set theorem in area of self-organizing data structures [141]. For binary search trees (like splay trees), the working set bound is defined as $\sum_{1\le i\le|\sigma|} \log(d_\sigma[i]+1)$. The logarithm can be explained by the logarithmic bounds on most operations in binary search trees. Thus our measure of locality of reference can be considered as variant of this measure in which we remove the logarithm.

Example 7.1. *Let $k = 3$ denote the size of the cache and consider the request sequence $\sigma = (p_1, p_2, p_3, p_4, p_5, p_4, p_5, p_3, p_5, p_5, p_1, p_2, p_6, p_4, p_5)$. We have $d_\sigma[i] = k+1 = 4$ for both $1 \le i \le 5$ and $i = 13$ as these are the first requests to their corresponding pages. Following Definition 7.1 we have $d_\sigma[6] = 2$, $d_\sigma[7] = 2$, $d_\sigma[8] = 3$, $d_\sigma[9] = 2$, $d_\sigma[10] = 1$, $d_\sigma[11] = 5$, $d_\sigma[12] = 5$, $d_\sigma[14] = 6$, $d_\sigma[15] = 5$. Thus*

$$\overline{\lambda}(\sigma) = \frac{1}{|\sigma|}\sum_{1\le i\le|\sigma|} d_\sigma[i] = \frac{1}{15}(5\times 4 + 2 + 2 + 3 + 2 + 1 + 5 + 5 + 4 + 6 + 5) = \frac{44}{15}.$$

The rest of this section is organized as follows. In Subsection 7.1.1 we study the non-locality measure of sequences from a set of real-life traces. We then provide bounds on the performance

[1] Formally a measure is a function that assigns a numerical non-negative value to an object, assigns the value of zero to the empty set and is additive over disjoint objects.

[2] Asymptotically, and assuming the number of requests is much larger than the number of distinct pages, any constant can replace $k+1$ for the $d_\sigma[i]$ of the first accesses.

	espresso	li	eqntott	compress	tomcatv	ear	sc	swm	gcc
Distinct	3913	3524	9	189	5260	1614	561	3635	2663
$\overline{\lambda}$	193.1	195.2	1.7	2.3	348.3	34.1	5.4	166.7	90.6
Ratio	4.9%	5.5%	19.3%	1.2%	6.6%	2.1%	1.0%	4.6%	3.4%

Table 7.1: Locality of address traces collected from SPARC processors running the SPEC92 benchmarks.

of well known paging algorithms in terms of the non-locality measure in Subsection 7.1.2. In Subsection 7.1.3 we apply our model to the case where the input is restricted to sequences with high locality of reference in model of Albers et al. [6]. We see two other possibilities for locality measure of paging in Subsection 7.1.4. Finally, in Subsection 7.1.5 we discuss the connection between our approach and adaptive analysis.

7.1.1 Experimental Evaluation of the Measure

In order to check validity of our measure we ran some experiments on traces of memory reference streams from the NMSU TraceBase [149]. Here we present the results of our experiments on address traces collected from SPARC processors running the SPEC92 benchmarks. We considered a page size of 2048 bytes and truncated them after 40000 references. The important thing to notice is that these are not special cases or artificially generated memory references, but are access patterns which a real-life implementation of any paging algorithm might face. The results for the corresponding nine program traces are shown in Table 7.1. The first row shows the number of distinct pages, the second row shows $\overline{\lambda}$, and finally the third row shows the ratio of the actual locality to the worst possible locality. The worst possible locality of a trace asymptotically equals the number of distinct pages in that trace. It is clear from the low ratios that in general these traces exhibit high locality of reference as defined by our measure.

7.1.2 Theoretical Results

Next we analyze several well known paging algorithms in terms of the non-locality parameter. We consider the fault rate, the measure usually used by practitioners. Recall that the fault rate of a paging algorithm $\mathcal{A}$ on a sequence σ is defined as $\mathcal{A}(\sigma)/|\sigma|$, i.e., the number of faults $\mathcal{A}$ incurs on σ normalized by the length of σ. The fault rate of $\mathcal{A}$, F_A, is defined as the asymptotic worst case fault rate of $\mathcal{A}$ on any sequence. The bounds are in the worst case sense, i.e., when we say $F_{\mathcal{A}} \geq f(\overline{\lambda})$ we mean that there is a sequence σ such that $\frac{\mathcal{A}(\sigma)}{|\sigma|} \geq f(\overline{\lambda}(\sigma))$ and when we say $F_{\mathcal{A}} \leq g(\overline{\lambda})$ we mean that for every sequence σ we have $\frac{\mathcal{A}(\sigma)}{|\sigma|} \leq g(\overline{\lambda}(\sigma))$. Also for simplicity, we ignore the details related to the special case of the first few requests (the first block or phase). Asymptotically and as the size of the sequences grow, this can only change the computation by additive lower order terms. Most of the proofs below use constructions similar to the ones found in [35]. This is an advantage as we reuse the same techniques to prove sharper results.

Lemma 7.1. *For any deterministic paging algorithm $\mathcal{A}$, $\frac{\overline{\lambda}}{k+1} \leq F_{\mathcal{A}} \leq \frac{\overline{\lambda}}{2}$.*

Proof. For the lower bound consider a slow memory containing $k+1$ pages. Let σ be a sequence of length n obtained by first requesting $p_1, p_2, \ldots, p_k, p_{k+1}$, and afterwards repeatedly requesting

the page not currently in $\mathcal{A}$'s cache. Since $\frac{\mathcal{A}(\sigma)}{|\sigma|} = n/n = 1$, and $\overline{\lambda}$ is at most $k+1$ (there are $k+1$ distinct pages in σ), the lower bound follows.

For the upper bound, consider any request sequence σ of length n. If the i^{th} request is a fault charged to $\mathcal{A}$, then $d_\sigma[i] \geq 2$ (otherwise σ_i cannot have been evicted). Hence, $2\mathcal{A}(\sigma) \leq \sum_{i=1}^{n} d_\sigma[i]$ and the upper bound follows. □

We now show that LRU attains the best possible performance in terms of $\overline{\lambda}$.

Theorem 7.1. $F_{\text{LRU}} = \frac{\overline{\lambda}}{k+1}$.

Proof. It follows from Lemma 7.1 and the observation that LRU faults on the request σ_i if and only if $d_\sigma[i] \geq k+1$, which implies $\text{LRU}(\sigma) \leq \frac{\overline{\lambda}}{k+1}|\sigma|$. □

Next, we show a general upper bound for conservative and marking algorithms.

Lemma 7.2. *Let $\mathcal{A}$ be a conservative or marking algorithm, then $F_{\mathcal{A}} \leq \frac{2\overline{\lambda}}{k+3}$.*

Proof. Let σ be an arbitrary sequence and let φ be an arbitrary phase in the decomposition of σ. $\mathcal{A}$ incurs at most k faults on φ. For any phase except the first, the first request in φ, say σ_i, is to a page that was not requested in the previous phase, which contained k distinct pages. Hence, $d_\sigma[i] \geq k+1$. There are at least $k-1$ other requests in φ to $k-1$ distinct pages, which all could have been present in the previous phase. But these pages contribute at least $\sum_{j=1}^{k-1}(j+1) = k-1+\frac{k^2-k}{2}$ to $\overline{\lambda}$. It follows that the contribution of this phase to $|\sigma|\overline{\lambda}$ is at least $k+1+k-1+\frac{k^2-k}{2} = \frac{k^2+3k}{2}$. Hence,

$$\frac{\mathcal{A}(\sigma)}{|\sigma|\overline{\lambda}} \leq \frac{k}{\frac{k^2+3k}{2}} = \frac{2}{k+3} \Rightarrow F_{\mathcal{A}} \leq \frac{2\overline{\lambda}}{k+3}.$$

□

There is a matching lower bound for FWF.

Lemma 7.3. $F_{\text{FWF}} \geq \frac{2\overline{\lambda}}{k+3}$.

Proof. Consider $\sigma = \{p_1 p_2 \dots p_k p_{k+1} p_k p_{k-1} \dots p_2\}^n$. $\text{FWF}(\sigma) = 2kn$, since FWF faults on all the requests. Now consider any block except the first. First, consider a page p_i, $2 \leq i \leq k$. The first and second request to p_i contribute i and $k+2-i$ to $|\sigma|\overline{\lambda}$, respectively, giving a total contribution of $k+2$. The requests to p_1 and p_{k+1} contribute $k+1$ each. Hence, the total contribution (for all phases except the first) is $(k-1)(k+2)+2(k+1) = k^2+3k$. Therefore

$$\frac{\text{FWF}(\sigma)}{|\sigma|\overline{\lambda}} = \frac{2k}{k^2+3k} = \frac{2}{k+3}.$$

□

Thus FWF has approximately twice as many faults as LRU on sequences with the same locality of reference, in the worst case. FIFO also has optimal performance in terms of $\overline{\lambda}$.

Lemma 7.4. $F_{\text{FIFO}} \leq \frac{\overline{\lambda}}{k+1}$.

Proof. Let σ be an arbitrary sequence. Consider a fault σ_i on a page p and let $\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_m}$ be the requests to p since p last entered the cache. By definition all the requests $\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_m}$ are hits and p is evicted between σ_{j_m} and σ_i. Observe that for p to get evicted at least k distinct pages other than p have to be requested since p entered the cache, hence $d_\sigma[i] + \sum_{h=1}^{m} d_\sigma[j_h] \geq k+1$. It follows that for each fault charged to FIFO we have at least a contribution of $k+1$ to $|\sigma|\overline{\lambda}$. □

Lemma 7.5. $F_{\text{LFU}} \geq \frac{2\overline{\lambda}}{k+3}$.

Proof. Consider the (usual) sequence $\sigma = p_1^n p_2^n \ldots p_{k-1}^n \{p_k p_{k+1}\}^n$, where $\text{LFU}(\sigma) = k - 1 + 2n$. For $|\sigma|\overline{\lambda}$, each of the pages $p_1, p_2, \ldots, p_{k-1}$ contributes $k + 1 + n - 1 = k + n$, and the pages p_k and p_{k+1} contribute $(k+1) + 2(n-1)$ each. Hence, $|\sigma|\overline{\lambda} = (k-1)(k+n) + 2(k+2n-1) = (k+3)n + k^2 + k - 2$, and therefore

$$\frac{\text{LFU}(\sigma)}{|\sigma|\overline{\lambda}} = \frac{2n + k - 1}{(k+3)n + k^2 + k - 2},$$

which becomes arbitrarily close to $\frac{2}{k+3}$ as n grows. □

In contrast LIFO has much poorer performance than most other paging algorithms (the worst possible) in terms of $\overline{\lambda}$.

Lemma 7.6. $F_{\text{LIFO}} \geq \frac{\overline{\lambda}}{2}$.

Proof. Consider the sequence $\sigma = p_1 p_2 \ldots p_k p_{k+1} \{p_k p_{k+1}\}^n$. We have $\text{LIFO}(\sigma) = k + 1 + 2n$ and $|\sigma|\overline{\lambda} = (k+1)(k+1) + 2 \cdot 2n$, and the bound follows. □

Recall that LRU-2 (defined in Section 1.1) has been shown to have good performance in certain instances. The following result confirms this.

Lemma 7.7. $\frac{2k\overline{\lambda}}{(k+1)(k+2)} \leq F_{\text{LRU-2}} \leq \frac{2\overline{\lambda}}{k+1}$.

Proof. Let σ be the sequence $\{p_1 p_2 \ldots p_{k-1} p_k p_k p_{k-1} \ldots p_1 p_{k+1} p_{k+1}\}^n$. Now, consider any repetition except the first. LRU-2 faults on all requests except the second request to p_k and the second request to p_{k+1}, giving a total of $2k$ faults. The first request to p_i, $1 \leq i \leq k-1$, contributes i to $|\sigma|\overline{\lambda}$ and the second request to p_i contributes $k + 2 - i$. Hence, each of these $k-1$ pages contributes $k+2$. For the pages p_k and p_{k+1}, their first request contributes $k+1$ and the second only 1 to $|\sigma|\overline{\lambda}$. This gives a total contribution of $(k+1)(k+2)$, and asymptotically the result follows.

For the upper bound, consider three consecutive faults on some page p. At least k other distinct pages should be requested since the first fault on p (at least $k-1$ other pages with at least 2 request and at least one other page). □

While no deterministic on-line paging algorithm can have competitive ratio better than k, there are randomized algorithms with better competitive ratio. The randomized marking algorithm MARK, introduced by Fiat et al. [77], is $2H_k$-competitive, where H_k is the k^{th} harmonic number. On a fault, MARK evicts a page chosen uniformly at random from among the unmarked

pages. Let σ be a sequence and $\varphi_1, \varphi_2, \ldots, \varphi_m$ be its phase decomposition. A page requested in phase φ_i is called *clean* if it was not requested in phase φ_{i-1} and *stale* otherwise. Let c_i be the number of clean pages requested in phase φ_i. Fiat et al. proved that the expected number of faults MARK incurs on phase φ_i is $c_i(H_k - H_{c_i} + 1)$.

Lemma 7.8. $F_{\text{MARK}} = \frac{2\overline{\lambda}}{3k+1}$.

Proof. Let σ be $\{p_1 p_2 \ldots p_k p_{k+1} p_{k+2} \ldots p_{2k} p_k p_{k-1} \ldots p_1 p_{2k} \ldots p_{k+1}\}^n$. This sequence has $4n$ phases. All pages of each phase are clean. Therefore we have $c_i = k$ for $1 \leq i \leq 4n$ and the expected number of faults MARK incurs on each phase is $k \times (H_k - H_k + 1) = k$. Thus $E(\text{MARK}(\sigma)) = 4nk$. We have $|\sigma|\overline{\lambda} = 4n(k + 1 + k + 2 + \cdots + 2k) = 4n(k^2 + k(k+1)/2) = 2n(3k^2 + k)$. Hence

$$\frac{E(\text{MARK}(\sigma))}{|\sigma|\overline{\lambda}} = \frac{4nk}{2n(3k^2+k)} = \frac{2}{3k+1},$$

which proves the lower bound.

For the upper bound, consider an arbitrary sequence σ and let $\varphi_1, \varphi_2, \ldots, \varphi_m$ be its phase decomposition. Suppose that the i^{th} phase φ_i has c_i clean pages. Therefore the expected cost of MARK on phase i is at most $c_i(H_k - H_{c_i} + 1)$. The first request to the j^{th} clean page in a phase contributes at least $k+j$ to $|\sigma|\overline{\lambda}$ (k pages from previous phase and $j-1$ clean pages that have been seen so far). The first request to the j^{th} stale page in a phase contributes at least $j+1$. Therefore the contribution of phase i to $|\sigma|\overline{\lambda}$ is at least $\sum_{j=1}^{c_i}(k+j) + \sum_{j=1}^{k-c_i}(j+1) = (2c_i^2 - 2c_i + k^2 + 3k)/2$, and

$$\frac{E(\text{MARK}(\sigma))}{|\sigma|\overline{\lambda}} \leq \frac{2c_i(H_k - H_{c_i} + 1)}{2c_i^2 - 2c_i + k^2 + 3k},$$

where $1 \leq c_i \leq k$. This is an increasing function in terms of c_i and attains its maximum at $c_i = k$. Thus we have

$$\frac{E(\text{MARK}(\sigma))}{|\sigma|\overline{\lambda}} \leq \frac{2k(H_k - H_k + 1)}{2k^2 - 2k + k^2 + 3k} = \frac{2}{3k+1}.$$

□

Finally we prove bounds on the performance of LFD. Recall that on a fault, LFD evicts the page whose next request is farthest in the future.

Lemma 7.9. $\frac{\overline{\lambda}}{3k+1} \leq F_{\text{LFD}} \leq \frac{2\overline{\lambda}}{3k+1}$.

Proof. Let σ be $\{p_1 p_2 \ldots p_k p_{k+1} p_{k+2} \ldots p_{2k} p_k p_{k-1} \ldots p_1 p_{2k} \ldots p_{k+1}\}^n$. This sequence has $4n$ phases. Each two consecutive phases of σ contain $2k$ distinct pages. LFD contains at most k pages in its cache before serving these phases and thus it would incur at least k faults on serving any two consecutive phases. Thus we have $\text{LFD}(\sigma) \geq 2kn$. We have $|\sigma|\overline{\lambda} = 4n(k+1+k+2+\cdots+2k) = 4n(k^2 + k(k+1)/2) = 2n(3k^2+k)$. Hence

$$\frac{\text{LFD}(\sigma)}{|\sigma|\overline{\lambda}} \geq \frac{2nk}{2n(3k^2+k)} = \frac{1}{3k+1}.$$

Recall that any randomized algorithm can be viewed as a probability distribution on a set of deterministic algorithms. Since the performance of LFD on any sequence is at least as good as the performance of any deterministic algorithm on that sequence, the performance of LFD is no worse than the expected performance of a randomized algorithm on any sequence. Thus the upper bound follows from Lemma 7.8. □

Algorithm	Lower Bound	Upper Bound
Deterministic	$\frac{\overline{\lambda}}{k+1}$	$\frac{\overline{\lambda}}{2}$
LRU	$\boldsymbol{\frac{\overline{\lambda}}{k+1}}$	$\boldsymbol{\frac{\overline{\lambda}}{k+1}}$
Marking	$\frac{\overline{\lambda}}{k+1}$	$\frac{2\overline{\lambda}}{k+3}$
FWF	$\boldsymbol{\frac{2\overline{\lambda}}{k+3}}$	$\boldsymbol{\frac{2\overline{\lambda}}{k+3}}$
FIFO	$\boldsymbol{\frac{\overline{\lambda}}{k+1}}$	$\boldsymbol{\frac{\overline{\lambda}}{k+1}}$
LFU	$\frac{2\overline{\lambda}}{k+3}$	$\frac{\overline{\lambda}}{2}$
LIFO	$\boldsymbol{\frac{\overline{\lambda}}{2}}$	$\boldsymbol{\frac{\overline{\lambda}}{2}}$
LRU-2	$\frac{2k\overline{\lambda}}{(k+1)(k+2)}$	$\frac{2\overline{\lambda}}{k+1}$
MARK	$\boldsymbol{\frac{2\overline{\lambda}}{3k+1}}$	$\boldsymbol{\frac{2\overline{\lambda}}{3k+1}}$
LFD	$\frac{\overline{\lambda}}{3k+1}$	$\frac{2\overline{\lambda}}{3k+1}$

Table 7.2: Bounds for paging.

The results are summarized in Table 7.2. According to these results, LRU and FIFO have optimal performance among deterministic algorithms. Marking algorithms can be twice as bad and FWF is among the worst marking algorithms. LIFO has the worst performance possible and LRU-2 is almost twice as bad as LRU. The performance of the randomized algorithm MARK is better than any deterministic algorithms: it behaves almost 2/3 better than LRU. Observe that LFD, an optimal offline algorithm, is only a factor of 3 better than LRU under this measure. Contrast this with the competitive ratio for which LFD is k times better than LRU.

7.1.3 Inputs with Locality of Reference

We can further incorporate a locality of reference assumption by restricting the inputs to those with high locality of reference in the Max-model proposed by Albers et al. [6] (Cf. Section 3.14). We model locality of reference by restricting the input to $\mathcal{I}_f$, the set of sequences that are consistent with f. Table 7.3 shows the fault rate of paging algorithms in terms of $\overline{\lambda}$ when we restrict the input to $\mathcal{I}_f$. Recall that $f^{-1}(m)$ denotes the smallest size of a window in a sequence consistent with f that contains m distinct pages, i.e., $f^{-1}(m) = \min\{n \in \mathbb{N} \mid f(n) \geq m\}$. Most of the proofs below use constructions similar to the ones given by Albers et al. [6].

Lemma 7.10. *For any deterministic online paging algorithm $\mathcal{A}$,*

$$\frac{(k-1)\overline{\lambda}}{k(k-1)+f^{-1}(k+1)-2} \leq F_{\mathcal{A}}(f) \leq \frac{\overline{\lambda}}{2}.$$

Proof. The upper bound follows from the general upper bound proved in Lemma 7.1. For the lower bound, given the function f and algorithm $\mathcal{A}$, we construct a sequence σ as follows. We use

Algorithm	Lower Bound	Upper Bound
General	$\frac{\overline{\lambda}}{k+b}, \quad b = \frac{f^{-1}(k+1)-2}{k-1}$	$\frac{\overline{\lambda}}{2}$
Marking	$\frac{\overline{\lambda}}{k+b}, \quad b = \frac{f^{-1}(k+1)-2}{k-1}$	$\frac{2\overline{\lambda}}{k+1+2b}, \quad b = \frac{f^{-1}(k+1)-1}{k}$
LRU	$\frac{\overline{\lambda}}{k+b}, \quad b = \frac{f^{-1}(k+1)-2}{k-1}$	$\frac{\overline{\lambda}}{k+b}, \quad b = \frac{f^{-1}(k+1)-2}{k-1}$
FWF	$\frac{2\overline{\lambda}}{k+1+2b}, \quad b = \frac{f^{-1}(k+1)-1}{k}$	$\frac{2\overline{\lambda}}{k+1+2b}, \quad b = \frac{f^{-1}(k+1)-1}{k}$
FIFO	$\frac{\overline{\lambda}}{k+b}, \quad b = \frac{f^{-1}(k+1)-1}{k-1}$	$\frac{\overline{\lambda}}{k-1+b}, \quad b = \frac{f^{-1}(k+1)}{k}$

Table 7.3: The fault rate of paging algorithms in terms of $\overline{\lambda}$ with respect to a concave* function f.

$k+1$ distinct pages. The sequence σ is constructed in phases, each with length $f^{-1}(k+1)-2$, where each phase consists of $k-1$ blocks. Each block contains several requests to the page that was not in $\mathcal{A}$'s cache just before the first request of the block. Hence, $\mathcal{A}$ faults on the first request of each block and incurs $k-1$ faults on each phase. In each phase, block j, $1 \leq j \leq k-1$, starts with request $f^{-1}(j+1)-1$. The construction is well-defined and is consistent with f ([6, Theorem 1]). For an upper bound on the non-locality of a phase, observe that since there are only $k+1$ distinct pages, the first request of each block of the phase contributes at most $k+1$ to $|\sigma|\overline{\lambda}$. Each of the following requests in the block, contributes 1. Since there are $k-1$ blocks, the total contribution of the first request to each and all of the blocks to $|\sigma|\overline{\lambda}$ is at most $(k+1)(k-1)$. Since there are $f^{-1}(k+1)-2-(k-1)$ other requests in a phase, each contributing 1 to $|\sigma|\overline{\lambda}$, we get a total contribution of at most $(k+1)(k-1)+f^{-1}(k+1)-2-(k-1) = k(k-1)+f^{-1}(k+1)-2$, and the result follows. □

Lemma 7.11. $F_{\mathrm{LRU}}(f) \leq \frac{(k-1)\overline{\lambda}}{k(k-1)+f^{-1}(k+1)-2}$.

Proof. Let σ be an arbitrary sequence consistent with f. Partition σ into phases such that each phase contains $k-1$ faults made by LRU, except possibly the last, and is maximal subject to that constraint. Hence, the request just before and just after a phase is a fault for all phases except the first and last phases. Let φ be any such phase. In [6, Theorem 2] it is shown that φ has length at least $f^{-1}(k+1)-2$. Since LRU faults on the i^{th} request if and only if $d_\sigma[i] \geq k+1$, each of the $k-1$ faults made by LRU in φ contributes at least $k+1$ to $|\sigma|\overline{\lambda}$. All other requests contribute at least 1. Hence, the total contribution to $|\sigma|\overline{\lambda}$ is at least $(k+1)(k-1)+f^{-1}(k+1)-2-(k-1) = k(k-1)+f^{-1}(k+1)-2$, and the upper bound follows.

□

Lemma 7.12. *For any marking algorithm $\mathcal{A}$, $F_{\mathcal{A}}(f) \leq \frac{2k\cdot\overline{\lambda}}{k(k+1)+2(f^{-1}(k+1)-1)}$.*

Proof. Let σ be an arbitrary sequence and consider the decomposition of σ. $\mathcal{A}$ incurs at most k faults on each phase. For any phase φ except the last, the next phase begins with a request to

a page not in φ. Hence, the subsequence consisting of φ and the first request of the next phase contains $k+1$ distinct pages and has length at least $f^{-1}(k+1)$. It follows that the length of φ is at least $f^{-1}(k+1)-1$. For any phase φ except the first, the first request, say i, in φ is to a page that was not requested in the previous phase, which contained k distinct pages. Hence, $d_\sigma[i] \geq k+1$. Now, consider any phase φ except for the first and last phase. The first request contributes at least $k+1$ to $|\sigma|\overline{\lambda}$. There are at least $k-1$ other requests in φ to $k-1$ distinct pages, which all could have been present in the previous phase. These pages contribute at least $\sum_{j=1}^{k-1}(j+1) = k-1+\frac{k^2-k}{2}$ to $|\sigma|\overline{\lambda}$. The remaining $f^{-1}(k+1)-1-k$ requests in P (requests to pages already requested in φ) all contribute at least 1. It follows that the contribution to $|\sigma|\overline{\lambda}$ of φ is at least

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

Hence,

$$\frac{\mathcal{A}(\sigma)}{|\sigma|\overline{\lambda}} \leq \frac{k}{\frac{k(k+1)+2(f^{-1}(k+1)-1)}{2}} = \frac{2k}{k(k+1)+2(f^{-1}(k+1)-1)}.$$

□

Lemma 7.13. $F_{\mathrm{FWF}}(f) \geq \frac{2k\cdot\overline{\lambda}}{k(k+1)+2(f^{-1}(k+1)-1)}$.

Proof. We construct a sequence σ as follows. We use $k+1$ pages, $p_1, p_2, \ldots, p_{k+1}$. σ consists of phases (corresponding to the phases of FWF) and each phase is composed of k blocks, where each block is a subsequence of requests to the same page. In each phase, block j, $1 \leq j \leq k$, has length $f^{-1}(j+1)-f^{-1}(j)$. By Proposition [6, Proposition 1], the block lengths are well-defined, i.e., they are non-zero, non-decreasing in a phase, and the total length of a phase is $f^{-1}(k+1)-1$.

In the first phase, the jth block consists of requests to p_j ($1 \leq j \leq k$). Now we inductively define the $(i+1)$st phase. The first block consists of $f^{-1}(2)-f^{-1}(1) = 2-1 = 1$ requests to the unique page that was unmarked at the end of the ith phase. That causes FWF to fault and flush the cache (and all pages become unmarked). The second block consists of requests to the page that was requested in the last block of the ith phase. The third block consists of requests to the page that was requested in the second to last block of the ith phase. The following $k-2$ blocks are defined similarly. By [6, Theorem 4], the construction is consistent with f.

FWF faults k times in each phase. For the non-locality of the sequence consider any phase except the first or the least. The first request contributes $k+1$ to $|\sigma|\overline{\lambda}$. There are $k-1$ other distinct pages requested in the phase, and the first request to each of these contributes $\sum_{j=1}^{k-1}(j+1) = k-1+\frac{k^2-k}{2}$. Each of the remaining $f^{-1}(k+1)-1-k$ requests (requests to pages already requested in the phase), contributes one unit to $|\sigma|\overline{\lambda}$, and the result follows as in the previous lemma. □

Lemma 7.14. $\frac{(k-\frac{1}{k})\overline{\lambda}}{(k-1)(k+1)+f^{-1}(k+1)-1} \leq F_{\mathrm{FIFO}}(f) \leq \frac{k\cdot\overline{\lambda}}{(k-1)(k+1)+f^{-1}(k+1)}$.

Proof. The lower bound follows from the construction in [6, Theorem 5]. For the non-locality, observe that the first request in each block contributes $k+1$ units to $|\sigma|\overline{\lambda}$. The following request to p_k contributes 2 and any following request to p_k contributes 1. Hence, each block contributes

$k+1+|\text{block}|$, and since there are $k-1$ blocks in a phase and k phases in a super phase, the total contribution to $|\sigma|\overline{\lambda}$ of a super phase is

$$k(k-1)(k+1)+|\text{super phase}| = k(k-1)(k+1)+k(f^{-1}(k+1)-1),$$

and the lower bound follows.

For the upper bound, first consider a fault σ_i for a page p and let $\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_m}$ be the requests to p since p last entered the cache. By definition all the requests $\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_m}$ are hits and p is evicted between request σ_{j_m} and σ_i. Observe that for p to get evicted at least k distinct pages different from p have to be requested since it entered the cache, hence $d_\sigma[i]+\sum_{h=1}^{m} d_\sigma[j_h] \geq k+1$. It follows that for each fault charged to FIFO we have at least a contribution of $k+1$ to $|\sigma|\overline{\lambda}$. Now, partition the sequence into phases that contains exactly k faults by FIFO and starts with a fault. Since a subsequence consisting of a phase and the following request, contains $k+1$ faults it must have a length of at least $f^{-1}(k+1)$. Hence, a phase has length at least $f^{-1}(k+1)-1$. By the above observation the k faults in a phase contribute $k(k+1)$ to $|\sigma|\overline{\lambda}$. The remaining $f^{-1}(k+1)-1-k$ requests contribute at least 1. It follows that the total contribution of a phase is $k(k+1)+f^{-1}(k+1)-1-k = (k-1)(k+1)+f^{-1}(k+1)$. The upper bound follows. □

7.1.4 Alternative Measures of Non-locality

A natural question is whether there are better measures for non-locality of paging instances. In this subsection, we consider two other natural non-locality measures for paging. We will see that these measures are not as effective as $\overline{\lambda}$. Thus, $\overline{\lambda}$ is a good, if not the right, measure for non-locality of paging sequences. The first alternative is a simple measure that is based on the phase decomposition of sequences.

Definition 7.2. *For a sequence σ, let $|D(\sigma)|$ be the number of phases of σ. We define $\tilde{\lambda}(\sigma)$, the "non-locality" of σ, as $|D(\sigma)|/|\sigma|$. We denote the non-locality by $\tilde{\lambda}$ if the choice of σ is clear from the context.*

We can easily get results comparable to competitive analysis using this simple definition of non-locality. First we obtain a lower bound on the performance of any deterministic online algorithm.

Lemma 7.15. *For any deterministic online algorithm $\mathcal{A}$ we have $F_{\mathcal{A}} \geq k \cdot \tilde{\lambda}$.*

Proof. We construct a sequence σ that contains $k+1$ distinct pages by requesting the page that is not in $\mathcal{A}$'s cache at each time. $\mathcal{A}$ incurs a fault on each request of σ and we have $\mathcal{A}(\sigma)/|\sigma| = 1$. Since the length of each phase is at least k, we have $|D(\sigma)| \leq |\sigma|/k \Rightarrow \tilde{\lambda}(\sigma) \leq 1/k$. Therefore $\mathcal{A}(\sigma)/|\sigma| \geq k \cdot \tilde{\lambda}(\sigma)$. □

As in competitive analysis, all marking and conservative paging algorithms have optimal performance. This follows from the fact that any marking or conservative algorithm incurs at most k faults in each phase. The performance of LIFO and LFU cannot be bounded in terms of $\tilde{\lambda}$. For LIFO, consider the sequence $\sigma = p_1 p_2 \ldots p_k p_{k+1}\{p_k p_{k+1}\}^n$ for an arbitrary integer n. LIFO incurs a fault on all requests of σ, while we have $|D(\sigma)| = 2$. Therefore $\text{LIFO}(\sigma)/|\sigma| = 1 = \frac{|\sigma|}{2}\tilde{\lambda}(\sigma)$. For LFU, consider the sequence $\sigma = p_1^n p_2^n \ldots p_{k-1}^n\{p_k p_{k+1}\}^n$ for an

arbitrary integer n. We have $|D(\sigma)| = 2$ and LFU incurs a fault on all last $2n$ requests. Hence $\mathrm{LFU}(\sigma) = 2n/|\sigma| = n \cdot \tilde{\lambda}(\sigma)$. Since we can select an arbitrarily large n, LFU does not have a bounded fault rate in terms of $\tilde{\lambda}$.

Thus the phase-based definition of non-locality does not give better separation results than competitive analysis. A more elaborate definition can be obtained as follows.

Definition 7.3. *Consider a sequence σ. We call a request "non-local" if it is the first request to a page or at least k distinct pages have been requested since the previous request to this page in σ. The non-locality of σ, $\hat{\lambda}$, is defined as the number of non-local requests in σ, divided by $|\sigma|$.*

If a sequence has high locality of reference, there are not many distinct pages between two consecutive requests to a page. Therefore there are not many non-local requests and the sequence has small non-locality. First we show that LRU achieves the optimal fault rate in terms of $\hat{\lambda}$.

Theorem 7.2. $F_{\mathrm{LRU}} = \hat{\lambda}$.

Proof. LRU always maintains in its cache the last k distinct pages that are requested. Therefore a request is a fault for LRU if and only if it is a non-local request. Thus we have $\mathrm{LRU}(\sigma)/|\sigma| = \hat{\lambda}(\sigma)$. □

Lemma 7.16. *For any deterministic online paging algorithm $\mathcal{A}$, $F_{\mathcal{A}} \geq \hat{\lambda}$.*

Proof. Consider a sequence σ obtained by requesting an item that is not in $\mathcal{A}$'s cache at each time. We have $\mathcal{A}(\sigma) = |\sigma|$. On the other hand, σ has at most $|\sigma|$ non-local requests and we have $\hat{\lambda}(\sigma) \leq 1$. Therefore $\mathcal{A}(\sigma)/|\sigma| = 1 \geq \hat{\lambda}(\sigma)$ □

The following lemma shows that marking algorithms are a reasonable choice in general, even if not always optimal.

Lemma 7.17. *For any conservative or marking algorithm $\mathcal{A}$, we have $F_{\mathcal{A}} \leq k \cdot \hat{\lambda}$.*

Proof. Let σ be an arbitrary sequence and let φ be an arbitrary phase of the decomposition of σ. We know that $\mathcal{A}$ incurs at most k faults on φ. We claim that the first request of φ is always non-local. If this is the first phase, then this is the first request to a page and is non-local by definition. Otherwise, it should be different from k distinct pages that are requested in the previous phase. Therefore it is not requested in the previous phase and at least k distinct pages are requested since the last request to this page. Thus we have at most k faults and at least one non-local request in each phase and this proves the desired upper bound. □

Other well known algorithms are not optimal in terms of $\hat{\lambda}$.

Lemma 7.18. $F_{\mathrm{FIFO}} = k \cdot \hat{\lambda}$.

Proof. Upper bound follows from Lemma 7.17. For the lower bound, consider a sequence σ that starts with $\sigma_0 = p_1 p_2 \dots p_k p_1 p_2 \dots p_{k-1} p_{k+1} p_1 p_2 \dots p_{k-1}$ and contains $k+1$ distinct pages.. After the initial subsequence σ_0, σ consists of several blocks. Each block starts right after the previous block and contains $2k-1$ requests to k distinct pages. Let p be the page that is not in the cache at the beginning of a block B, q be the page that is requested just before B, and P be the set of $k-1$ pages that are requested in the previous block and are different from q. B starts by an

$$
\begin{array}{cccccccccc}
 & & & & p_1 & p_2 & \cdots & p_k & & \\
p_1 & p_2 & p_3 & \cdots & p_{k-1} & p_{k+1} & p_1 & p_2 & p_3 & \cdots & p_{k-1} \\
p_{k+1} & p_1 & p_2 & \cdots & p_{k-2} & p_k & p_{k+1} & p_1 & p_2 & \cdots & p_{k-2} \\
p_k & p_{k+1} & p_1 & \cdots & p_{k-3} & p_{k-1} & p_k & p_{k+1} & p_1 & \cdots & p_{k-3} \\
 & & & & \cdots & \cdots & & & & &
\end{array}
$$

Figure 7.1: The sequence σ in the proof of Lemma 7.18.

arbitrary permutation π of P, then has a request to page p, and finally ends by another copy of π. Initial blocks of a possible representation of σ are shown in Figure 7.1.

It is easy to verify that FIFO incurs a fault on the last k requests of each block while only the middle request of every block is non-local. Let n be the number of blocks of σ. We have $\text{FIFO}(\sigma) = n \cdot k$ and $\hat{\lambda}(\sigma) = n$ and the lower bound follows. □

We can obtain a similar lower bound for FWF by considering the sequence obtained by sufficient repetitions of the pattern $p_1 p_2 \dots p_k p_{k+1} p_k p_{k-1} \dots p_2$.

Lemma 7.19. $F_{\text{FWF}} = k \cdot \hat{\lambda}$.

Lemma 7.20. *The fault rate of* LFU *and* LIFO *cannot be bounded in terms of* $\hat{\lambda}$.

Proof. Consider the sequence $\sigma = p_1^n p_2^n \dots p_{k-1}^n \{p_k p_{k+1}\}^n$. LFU incurs a fault on the last $2n$ requests of σ. Only the first request to a page is non-local in σ and we have $\hat{\lambda}(\sigma) = (k+1)/|\sigma|$. Therefore $\text{LFU}(\sigma)/|\sigma| = 2n/|\sigma| = \frac{2n}{k+1}\hat{\lambda}(\sigma)$. Since n can be selected arbitrarily larger than k, the fault rate of LFU is not bounded in terms of $\hat{\lambda}$. For LIFO, consider the sequence $p_1 p_2 \dots p_k p_{k+1} \{p_k p_{k+1}\}^n$. LIFO incurs a fault on all requests of σ, while we have $\hat{\lambda}(\sigma) = (k+1)/|\sigma|$. Therefore $\text{LIFO}(\sigma)/|\sigma| = 1 = \frac{|\sigma|}{k+1}\hat{\lambda}(\sigma)$. □

Theorem 7.3. $F_{\text{LRU-2}} = k \cdot \hat{\lambda}$.

Proof. [Lower bound] Let σ be the sequence introduced in the proof of Lemma 7.7, i.e., $\sigma = \{p_1 p_2 \dots p_{k-1} p_k p_k p_{k-1} \dots p_1 p_{k+1} p_{k+1}\}^n$. As we proved there, LRU-2 incurs $2k$ faults in each block. In each block except the first, only two requests are non-local, namely the first request to p_k and the first request to p_{k+1}. Consider a page p_i for $1 \le i \le k-1$ and a block b_j for $2 \le j \le n$. There are at most $k-1$ distinct pages between the first request to p_i in b_j and the previous request to p_i (which is in the previous block), since p_k is not requested in this period. Thus the first request to p_i in b_j is local. Also p_{k+1} is not requested between the two requests to p_i in b_j. Therefore the second request to p_i in b_j is local too. The first request to p_k and the first request to p_{k+1} in b_j are non-local. Thus at each block we have two non-local requests, while LRU-2 incurs $2k$ faults and the lower bound follows.

[Upper bound] Let σ be an arbitrary sequence of page requests. Partition σ into a set of consecutive blocks so that each block consists of a maximal sequence that contains exactly one non-local request. Note that each block starts with a non-local request and all other requests of the block are local. We prove that LRU-2 incurs at most k faults in each block. Let $B_1, B_2, \ldots, B_m$ be the blocks of σ. B_1 contains requests to one page and LRU-2 incurs one fault on it. Consider an arbitrary block B_i for $i > 1$, let p be the first request of B_i, and let p_1, p_2, $\ldots$, p_{k-1} be the $k-1$ most recently used pages before the block B_i in this order. We have $p \notin P = \{p_1, p_2, \ldots, p_{k-1}\}$, because p is a non-local request. We claim that each request of B_i is either to p or to a page of P. Assume for the sake of contradiction that B_i contains a request to a page $q \notin \{p\} \cup P$ and consider the first request to q in B_i. All pages $p, p_1, p_2, \ldots, p_{k-1}$ are requested since the previous request to q. Therefore at least k distinct pages are requested since the last request to q and q is non-local. This contradicts the definition of a block. Hence B_i contains at most k distinct pages.

We claim that LRU-2 incurs at most one fault on every page q in phase B_i. Assume that this is not true and LRU-2 incurs two faults on a page q in B_i. Therefore q is evicted again at some point after its first request in B_i. Assume that this eviction happened on a fault on a page r and consider the pages that are in LRU-2's cache just before that request. Since $r \in \{p\} \cup P$ is not in the cache and $|\{p\} \cup P| = k$, there is a page $s \notin \{p\} \cup P$ in the cache. The last request to s is before the last request to p_{k-1} before the block B_i, while the second last request to q is after this request. Therefore LRU-2 does not evict q on this fault, which is a contradiction. Thus, LRU-2 contains at most k distinct pages in each block and incurs at most one fault on each page. Hence, $\text{LRU-2}(\sigma)/|\sigma| \leq km/|\sigma| = k \cdot \hat{\lambda}(\sigma)$. □

The performance of MARK in terms of $\hat{\lambda}$ is worse than LRU but better than other well known deterministic algorithms.

Theorem 7.4. $F_{\text{MARK}} = H_k \cdot \hat{\lambda}$.

Proof. [Lower bound] Consider the sequence $\sigma = \{p_1 p_2 \ldots p_k p_{k+1} p_k p_{k-1} \ldots p_2\}^n$ for some integer n. σ has $2n$ phases, each odd numbered phase has the form $p_1 p_2 \ldots p_k$ and each even numbered phase has the form $p_{k+1} p_k \ldots p_2$. Also each phase has only one clean page, namely its first request. Therefore we have $c_i = 1$ for $1 \leq i \leq 2n$ and the expected number of faults MARK incurs on each phase is $1 \times (H_k - H_1 + 1) = H_k$. Thus $E(\text{MARK}(\sigma)) = 2nH_k$. Only the first request of each phase is non-local and we have $\hat{\lambda}(\sigma) = 2n/|\sigma|$. Hence

$$\frac{E(\text{MARK}(\sigma))}{|\sigma|} = 2nH_k \cdot \frac{\hat{\lambda}(\sigma)}{2n} = H_k \cdot \hat{\lambda}(\sigma).$$

[Upper bound] Consider an arbitrary sequence σ and let $\varphi_1, \varphi_2, \ldots, \varphi_m$ be its phase decomposition. Suppose that the i^{th} phase φ_i has c_i clean pages. Therefore the expected cost of MARK on σ is $\sum_{i=1}^{n} c_i(H_k - H_{c_i} + 1) \leq \sum_{i=1}^{n} c_i H_k$. The first request to a clean page in a phase is non-local because it is not among the k distinct pages that are requested in the previous phase. Therefore we have $|\sigma|\hat{\lambda}(\sigma) \geq \sum_{i=1}^{n} c_i$. We have

$$\frac{E(\text{MARK}(\sigma))}{|\sigma|} \leq \frac{\sum_{i=1}^{n} c_i H_k}{\sum_{i=1}^{n} c_i} \cdot \hat{\lambda}(\sigma) = H_k \cdot \hat{\lambda}(\sigma).$$

□

Algorithm	Lower Bound	Upper Bound
Deterministic	$\hat{\lambda}$	unbounded
LRU	$\hat{\lambda}$	$\hat{\lambda}$
Marking	$\hat{\lambda}$	$k \cdot \hat{\lambda}$
FWF	$k \cdot \hat{\lambda}$	$k \cdot \hat{\lambda}$
FIFO	$k \cdot \hat{\lambda}$	$k \cdot \hat{\lambda}$
LFU	unbounded	unbounded
LIFO	unbounded	unbounded
LRU-2	$k \cdot \hat{\lambda}$	$k \cdot \hat{\lambda}$
MARK	$H_k \cdot \hat{\lambda}$	$H_k \cdot \hat{\lambda}$
LFD	$(M-k)\hat{\lambda}/M$	$\hat{\lambda}$

Table 7.4: Bounds for paging algorithms in terms of $\hat{\lambda}$.

If there is no restriction on the number of distinct pages in the sequence, we can prove a lower bound of $\hat{\lambda}$ on the fault rate of LFD, by always requesting a page that has not been requested so far. Each request is non-local and LFD incurs a fault on it. Thus LFD does not have a better performance than LRU on this general settings. If we restrict the number of distinct pages to M then the performance of LFD could be better than LRU. In this case we can get a lower bound of $(M-k)/M$ on the fault rate of LFD by considering the sequence $\sigma = \{p_1, p_2, \ldots, p_M\}^n$. In each block there are M distinct pages. Since the size of cache is k, LFD incurs at least $M-k$ faults in each block. All requests are non-local and we have $\hat{\lambda}(\sigma) = 1$. Thus $\text{LFD}(\sigma)/|\sigma| = (M-k)/M = \frac{M-k}{M}\hat{\lambda}(\sigma)$. The results are summarized in Table 7.4.

Comparison of three measures We have seen three possible measures for non-locality of sequences for paging. We get different results in our parameterized framework by using each measure. The measure $\tilde{\lambda}$ is simplest among them, but it does not give separation results better than standard competitive analysis. Therefore it shares most of the shortcomings of competitive analysis described in Chapter 2. The remaining two measures provide better separation between paging algorithms. They both separate the performance of LRU from FWF. However, they do not always give equivalent results. According to $\overline{\lambda}$, LRU and FIFO are both optimal deterministic online algorithms. Also the performance of MARK is better than LRU. In contrast, when we consider $\hat{\lambda}$, LRU has strictly better performance than both FIFO and MARK. Furthermore, FIFO, LRU-2, and FWF have the same performance. The performance of MARK is between LRU and FIFO. It seems that the definition of $\hat{\lambda}$ is tailored to the behavior of LRU, as it considers a clear distinction between local and non-local requests. We remark that a similar distinction was used in adequate analysis [129]. A good feature of $\overline{\lambda}$ is that we get better performance by increasing the size of the cache, while the reverse is true for the other measures. Overall, $\hat{\lambda}$ and $\overline{\lambda}$ both have their own merits and shortcomings, but $\overline{\lambda}$ seems to be the preferable choice.

7.1.5 Adaptive Analysis

In this subsection we study the connection between the $\overline{\lambda}$ measure and adaptive analysis. Recall from Subsection 2.2 that the adaptive performance of an algorithm is obtained by describing its traditional worst-case performance in terms of the size and difficulty of the instance. Observe that the competitive ratio can be seen as a special case of adaptive analysis, namely the case where the measure of difficulty is the performance of the off-line OPT. Our model can be expressed in terms of adaptive analysis by considering the non-locality of each sequence as its difficulty measure.

Thus we can generalize this framework to other online problems too. For each problem, we can choose the measure that best reflects the difficulty of the input. As in the case of parameterized complexity and previous adaptive analysis results, choosing the right measure of difficulty is a non-trivial task which can require several iterations. For example see the survey by Estivill-Castro and Wood [73] for several difficulty measures for the sorting problem. In the case of on-line problems, it is unlikely that the off-line OPT is a good measure for all or even most cases. We have seen several alternative measures for paging in this section.

In certain online problems, competitive analysis might force the algorithm to make a move that is suboptimal in most cases except for a pathological worst case scenario. If the application is such that these pathological cases are agreed to be of lesser importance, then the online strategy can perform somewhat more poorly in these and make the choice that is best for the common case. This means that the input is no longer assumed to be adversarially constructed. This better reflects the case of paging, in which programmers, compilers, instruction schedulers and optimized virtual machines (such as HotSpot) go to great lengths to maintain and increase locality of reference in the code. Hence it is more realistic to assume that paging sequences are not adversarial and that furthermore, the user/programmer fully expects code with low locality of reference to result in a degradation in performance. The same observation has been made in scenarios such as online robot exploration and network packet switching, in which a robot vacuuming a room or a router serving a packet sequence need only concentrate in well behaved common cases. A vacuuming robot need not efficiently vacuum a maze, neither does the router have to keep up with denial-of-service floods. We will see adaptive analysis of robot navigation in Chapter 9.

7.2 Parameterized Analysis of List Update Algorithms

In this section we study the parameterized complexity of list update algorithms in terms of locality of reference. We define the non-locality of sequences for list update in an analogous way to the corresponding definition for paging (Definition 7.1). The only differences are:

1. We do not normalize the non-locality by the length of the sequence, i.e., $\overline{\lambda}(\sigma) = \sum_{1 \leq i \leq |\sigma|} d_\sigma[i]$.
2. If σ_i is the first access to an item we assign the value ℓ to $d_\sigma[i]$[3].

Theorem 7.5. *For any deterministic online list update algorithm $\mathcal{A}$ we have* $\overline{\lambda} \leq \mathcal{A}(\sigma) \leq \ell \cdot \overline{\lambda}$.

[3] As for paging, asymptotically, and assuming the number of requests is much larger than ℓ, any constant can replace ℓ for the $d_\sigma[i]$ of the first accesses.

Proof. [Upper bound] Consider an arbitrary sequence σ of length n. Since the maximum cost that $\mathcal{A}$ incurs on a request is ℓ, we have $\mathcal{A}(\sigma) \leq n\ell$. We have $d_\sigma[i] \geq 1$ for all values of i. Thus $\overline{\lambda} \geq n$. Therefore $\frac{\mathcal{A}(\sigma)}{\overline{\lambda}} \leq \frac{n\ell}{n} = \ell$.

[Lower bound] Consider a sequence σ of length n obtained by requesting the item that is in the last position of list maintained by $\mathcal{A}$ at each time. We have $\mathcal{A}(\sigma) = n\ell$. Also we have $d_\sigma[i] \leq \ell$ because σ has at most ℓ distinct items. Therefore $\overline{\lambda} \leq n\ell$, and $\frac{\mathcal{A}(\sigma)}{\overline{\lambda}} \geq \frac{n\ell}{n\ell} = 1$. □

Theorem 7.6. MTF *is optimal in terms of* $\overline{\lambda}$*:* $\mathrm{MTF}(\sigma) \leq \overline{\lambda}$.

Proof. Consider the i^{th} request of σ. If this is the first request to item σ_i, then $d_\sigma[i] = \ell$, while the cost of MTF on σ_i is at most ℓ. Otherwise, the cost of MTF is $d_\sigma[i]$. Thus the cost of MTF on σ_i is at most $d_\sigma[i]$. Hence, $\mathrm{MTF}(\sigma) \leq \sum_{1 \leq i \leq n} d_\sigma[i] = \overline{\lambda}$, and the upper bound follows. Theorem 7.5 shows that this bound is tight. □

The following lemmas show that other well known list update algorithms are not optimal in terms of $\overline{\lambda}$.

Lemma 7.21. $\mathrm{TR}(\sigma) \geq \frac{\ell \cdot \overline{\lambda}}{2}$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list. Consider a sequence σ of length n obtained by several repetitions of pattern $a_\ell a_{\ell-1}$. We have $\mathrm{TR}(\sigma) = n \cdot \ell$. Also we have $d_\sigma[i] = \ell$ for $1 \leq i \leq 2$ and $d_\sigma[i] = 2$ for $2 < i \leq n$. Therefore $\overline{\lambda} = 2\ell + 2n - 4$, and

$$\frac{\mathrm{TR}(\sigma)}{\overline{\lambda}} = \frac{n \cdot \ell}{2\ell + 2n - 4},$$

which becomes arbitrarily close to $\ell/2$ as n grows. □

Lemma 7.22. $\mathrm{FC}(\sigma) \geq \frac{(\ell+1)\overline{\lambda}}{2} \approx \frac{\ell \cdot \overline{\lambda}}{2}$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list and n be an arbitrary integer. Consider the following sequence: $\sigma = a_1^n a_2^n a_3^n \ldots a_\ell^n$. On serving σ, FC does not change the order of items in its list and incurs cost $\sum_{i=1}^{\ell} ni = n \sum_{i=1}^{\ell} i = n\frac{\ell(\ell+1)}{2}$. We have $\overline{\lambda} = \sum_{i=1}^{\ell}(\ell + (n-1)) = \ell \cdot n + \ell^2 - \ell$. Therefore

$$\frac{\mathrm{FC}(\sigma)}{\overline{\lambda}} = \frac{n\frac{\ell(\ell+1)}{2}}{\ell \cdot n + \ell^2 - \ell} = \frac{n(\ell+1)}{2(n + \ell - 1)},$$

which approaches $\frac{\ell+1}{2}$ as n grows. □

Lemma 7.23. $\mathrm{TS}(\sigma) \geq \frac{2\ell \cdot \overline{\lambda}}{\ell+1} \approx 2\overline{\lambda}$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list and n be an arbitrary integer. Consider the sequence σ obtained by the repetition of the block $a_\ell^2 a_{\ell-1}^2 \ldots a_1^2$ n times. Let B be an arbitrary block of σ. Each item a_i is accessed twice in B. TS does not move a_i after its first access in B, because each item has been accessed twice since the last access to a_i. After the second access, TS moves the item to the front of the list. Therefore each access is to the last item of the list and TS incurs a cost of ℓ on each access. Thus, we have $\mathrm{TS}(\sigma) = 2\ell^2 n$. Next we compute $\overline{\lambda}$.

The first and second access to a in block B contribute ℓ and 1 to $\overline{\lambda}$, respectively. Thus we have $\overline{\lambda} = \ell(\ell+1)n$. Therefore

$$\frac{\mathrm{TS}(\sigma)}{\overline{\lambda}} = \frac{2\ell^2 n}{\ell(\ell+1)n} = \frac{2\ell}{\ell+1}.$$

□

Observe that parameterized analysis by virtue of its finer partition of the input space resulted in the separation of several of the strategies which were not separable under the classical model. This introduces a hierarchy of algorithms better reflecting the relative strengths of the strategies considered above. We can also apply the parameterized analysis to randomized list update algorithms by considering their expected cost.

In the next theorem we show that, surprisingly, certain randomized algorithms which are superior to MTF in the standard model are not so in the parameterized case. Observe that in the competitive ratio model a deterministic algorithm must serve a pathological, rare worst case even if at the expense of a more common but not critical case, while a randomized algorithm can hedge between the two cases, hence in the classical model the randomized algorithm is superior to the deterministic one. In contrast, in the parameterized model the rare wost case, if pathological, has a larger non-locality measure, leading to a larger $\overline{\lambda}$ factor. Hence such a cases can safely be ignored, with a resulting overall increase in the measured quality of the algorithm.

The algorithm *Bit*, considers a bit $b(a)$ for each item a and initializes these bits uniformly and independently at random. Upon an access to a, it first complement $b(a)$, then if $b(a) = 0$ it moves a to the front, otherwise it does nothing. Bit has competitive ratio 1.75, thus outperforming any deterministic algorithm [131]. In the parameterized model this situation is reversed.

Theorem 7.7. $E(Bit(\sigma)) \geq \frac{(3\ell+1)\overline{\lambda}}{2\ell+2} \approx 3\overline{\lambda}/2$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list and n be an arbitrary integer. Consider the sequence $\sigma = \{a_\ell^2 a_{\ell-1}^2 \ldots a_1^2\}^n$. Let σ_i and σ_{i+1} be two consecutive accesses to a_j. After two consecutive accesses to each item, it will be moved to the front of the list with probability 1. Therefore a_j is in the last position of the list maintained by Bit at the time of request σ_i and Bit incurs cost ℓ on this request. After this request, Bit moves a_j to the front of the list if and only if $b(a_j)$ is initialized to 1. Since $b(a_j)$ is initialized uniformly and independently at random, this will happen with probability 1/2. Therefore the expected cost of Bit on σ_{i+1} is $\frac{1}{2}(\ell+1)$ and the expected cost of Bit on σ is $n\ell(\ell + \frac{\ell+1}{2})$. We have $\overline{\lambda} = \ell(\ell+1)n$. Therefore

$$\frac{E(Bit(\sigma))}{\overline{\lambda}} = \frac{n \cdot \ell(\ell + \frac{\ell+1}{2})}{\ell(\ell+1)n} = \frac{3\ell+1}{2\ell+2}.$$

□

The results are summarized in Table 7.5. According to these results, MTF has the best performance among well known list update algorithms. TS has performance at least twice as bad as MTF. The performance of TR and FC is at least $\ell/2$ times worse than MTF. The performance of Bit is worse than MTF, while its competitive ratio is better. Experimental results of [19] show that MTF has better performance than Bit in practice. Thus our measure leads to more realistic result than competitive analysis in this case.

Algorithm	Lower Bound	Upper Bound
General	$\overline{\lambda}$	$\ell \cdot \overline{\lambda}$
MTF	$\overline{\lambda}$	$\overline{\lambda}$
TR	$\frac{\ell \cdot \overline{\lambda}}{2}$	$\ell \cdot \overline{\lambda}$
FC	$\approx \frac{\ell \cdot \overline{\lambda}}{2}$	$\ell \cdot \overline{\lambda}$
TS	$\approx 2\overline{\lambda}$	$\ell \cdot \overline{\lambda}$
Bit	$\approx \frac{3}{2}\overline{\lambda}$	

Table 7.5: Bounds for list update.

7.3 Conclusions

We applied parameterized analysis in terms of locality of reference to paging and list update algorithms and showed that this model gives promising results. The plurality of results shows that this model is effective in that we can readily analyze well known strategies. Using a finer, more natural measure we separated paging and list update algorithms which were otherwise indistinguishable under the classical model. We showed that a randomized algorithm which is superior to MTF in the classical model is not so in the cooperative case, which matches experimental evidence. This confirms that the ability of the online adaptive algorithm to ignore pathological worst cases can lead to the selection of algorithms that are more efficient in practice.

Chapter 8

Paging with Locality of Reference

As stated before, an apparent reason for the shortcomings of competitive analysis for paging is that the model does not take into account the locality of reference evidenced by actual input sequences. In this chapter we consider two previously proposed models of locality of reference, described in Section 3.13 and Section 3.14. We observe that even if we restrict the input to sequences with high locality of reference in these models the performance of every online algorithm in terms of the competitive ratio does not improve. Then we prove that locality of reference is useful under some other cost models, which suggests that a new model combining aspects of both proposed models can be preferable. We also slightly modify one of the models to show that the performance of randomized marking algorithm MARK improves as locality of reference increases. Finally we generalize the existing models to several variants of the caching problem.

In Chapter 3 we described several models for paging with locality of reference [36, 97, 148, 6, 129]. In this chapter we consider the models proposed by Torng [148] and Albers et al. [6]. Recall from Section 3.13 that $L(\sigma, k)$ is the average phase length in the decomposition of σ. Torng models locality by restricting the input to *a-local sequences*: sequences σ for which we have $L(\sigma, k) \geq a \cdot k$. Recall that in the working set model, a request sequence has high locality of reference if the number of distinct pages in a window of size n is small. For a concave function f, we say that a request sequence is consistent with f if the number of distinct pages in any window of size n is at most $f(n)$, for any $n \in \mathcal{N}$. In this way, Albers et al. model locality by restricting the input to sequences that are consistent with a concave function f. Refer to Sections 3.13 and 3.14 for complete definition of the models, as well as the relevant results. Torng shows that marking algorithms perform better on sequences with high locality under the *full access cost model*. Recall that Torng [148] uses the full access cost model, while Albers et al. [6] use the fault rate.

Our Results. First we formally show that under the standard cost model the competitive ratio of every online paging algorithm remains the same under the Albers et al. and the Torng locality of reference models. Hence, new results about paging algorithms necessitate a change to the cost model. We apply the fault rate cost model to the k-phase model and the full access cost model to the working set model. These two had been previously studied under the alternate combination. The full access cost model compares online algorithms to the optimal offline algorithm, while the fault rate cost model does not. Therefore there is not a direct relationship between the two cost models and our results cannot be directly concluded from the results of Torng [148] and Albers et al. [6].

Furthermore, we propose a new model for locality of reference and show that the randomized marking algorithm of [77] benefits from the locality of reference assumption. Finally we apply this assumption to the caching problem which is a generalization of the paging problem for which pages have different sizes and retrieval costs. We extend the existing models of locality of reference and show that certain caching algorithms perform better on sequences with good locality of reference under this model.

8.1 Limitations of the Competitive Ratio Model

We prove that restricting input sequences to those with high locality of reference is not reflected in an improvement on the competitive ratio.

Observation 8.1. *If we restrict the input to sequences that are consistent with a concave function f, the competitive ratio of deterministic online paging algorithms does not improve.*

Proof. The proof idea is the same as the one used to show that finite lookahead does not improve the competitive ratio of online paging algorithms [35]. Let $\mathcal{A}$ be a deterministic paging algorithm and σ be an arbitrary sequence. We can obtain a sequence I' such that σ' is consistent with f, $\mathcal{A}(\sigma) = \mathcal{A}(\sigma')$, and $\text{OPT}(\sigma) = \text{OPT}(\sigma')$. σ' is obtained by repeating each request of σ a sufficient number of times. Since we only consider demand paging algorithms, every paging algorithm has the same number of faults on σ and σ'. □

A similar result for the k-phase model can be proven, as we can make any sequence a-local by repeating each request a times.

Observation 8.2. *If we restrict the input to a-local sequences, the competitive ratio of deterministic online paging algorithms does not change.*

8.2 The Fault Rate of the k-phase Model

We obtain new results based on the k-phase model by considering the fault rate as the cost model. The fault rate of an algorithm $\mathcal{A}$ on a-local sequences is defined as $F_{\mathcal{A}}(a) = \inf\{r \mid \exists\, n \in \mathbb{N} : \forall \sigma,\, L(\sigma, k) \geq ak,\, |\sigma| \geq n : F_{\mathcal{A}}(\sigma) \leq r\}$. Also recall that $F_{\mathcal{A}}(\sigma)$ denotes the fault rate of $\mathcal{A}$ on an arbitrary sequence σ. We obtain the following bound for the fault rate of marking algorithms.

Theorem 8.1. *Let $\mathcal{A}$ be an arbitrary marking algorithm and $a > 0$ be a constant. Then $F_{\mathcal{A}}(a) \leq 1/a$.*

Proof. Consider an arbitrary a-local sequence σ of size at least n. We show that $F_{\mathcal{A}}(\sigma) \leq 1/a$. Consider the decomposition $D(\sigma, k)$ of σ. $\mathcal{A}$ does not fault more than once on a page p in a phase φ because after the first fault, it marks p and does not evict it in the reminder of φ. Since each phase contains at most k distinct pages, $\mathcal{A}$ does not fault more than k times in a phase. Thus $\mathcal{A}$ incurs at most $k \cdot |D(\sigma, k)|$ faults on σ and we have $F_{\mathcal{A}}(\sigma) \leq \frac{k \cdot |D(\sigma,k)|}{|\sigma|}$. Using

$$L(\sigma, k) = |\sigma|/|D(\sigma, k)|,$$

we get

$$F_{\mathcal{A}}(\sigma) \leq \frac{k}{L(\sigma,k)} \leq \frac{k}{ak} = 1/a.$$

□

This theorem shows that the fault rate of any marking algorithm decreases as the locality of reference of the input increases. Note that this holds for every algorithm $\mathcal{A}$ that incurs at most k faults in each phase. Since any phase contains at most k distinct pages, we obtain the following result.

Corollary 8.1. *Let $\mathcal{A}$ be an arbitrary conservative algorithm and $a > 0$ be a constant. Then $F_{\mathcal{A}}(a) \leq 1/a$.*

8.3 The Working Set Model under Full Access Cost Model

In this section we apply the full access cost model to the working set model. Earlier we proved that the standard competitive ratio does not improve for sequences with high locality of reference in this model. Now we show that the competitive ratio of the classical algorithms in the full access cost model improves for such sequences.

First we use some results of Albers et al. [6] about the fault rate of paging algorithms. These results are expressed in term of f^{-1}, the inverse function of f, defined in Section 3.14. Recall that $f^{-1}(m)$ denotes the minimum size of a window that contains at least m distinct pages.

Theorem 8.2. *The competitive ratio of* LRU *with respect to a concave function f in the full access cost model is at most*

$$\frac{p \cdot k \cdot (k-1) + k(f^{-1}(k+1) - 2)}{p \cdot (k-1) + k(f^{-1}(k+1) - 2)}.$$

Proof. Albers et al. proved that the fault rate of LRU is at most $\frac{k-1}{f^{-1}(k+1)-2}$ [6]. Consider an arbitrary sequence σ that is consistent with f. Suppose that LRU and OPT incur m and m' faults on σ, respectively. We have

$$\frac{m}{|\sigma|} \leq \frac{k-1}{f^{-1}(k+1)-2} \quad \Longrightarrow \quad |\sigma| \geq \frac{m \cdot (f^{-1}(k+1)-2)}{k-1}.$$

Since $m' \geq m/k$, we have $\text{OPT}_{FA}(\sigma) = p \cdot m' + |\sigma| \geq p \cdot m/k + |\sigma|$. We also have $\text{LRU}_{FA}(\sigma) = p \cdot m + |\sigma|$, and therefore

$$\frac{\text{LRU}_{FA}(\sigma)}{\text{OPT}_{FA}(\sigma)} \leq \frac{p \cdot m + \frac{m \cdot (f^{-1}(k+1)-2)}{k-1}}{p \cdot m/k + \frac{m \cdot (f^{-1}(k+1)-2)}{k-1}} = \frac{p \cdot k \cdot (k-1) + k(f^{-1}(k+1) - 2)}{p \cdot (k-1) + k(f^{-1}(k+1) - 2)}.$$

Since σ was an arbitrary sequence, this proves the theorem. □

As p increases, the upper bound of Theorem 8.2 approaches k. When p is not too large, the term $(f^{-1}(k+1) - 2)$ becomes important. For a fixed p, the larger the value of $f^{-1}(k+1)$, the better the upper bound of the theorem. This supports our intuition that LRU has better performance on sequences with more locality of reference.

It is also known that $F_{\text{FIFO}}(f) \leq \frac{k}{f^{-1}(k+1)-1}$ and $F_{\mathcal{A}}(f) \leq \frac{k}{f^{-1}(k+1)-1}$ for any marking algorithm $\mathcal{A}$ [6]. We can use these results to prove the following theorem in an analogous way to Theorem 8.2.

Theorem 8.3. *Let $\mathcal{A}$ be a marking algorithm or* FIFO*. The competitive ratio of $\mathcal{A}$ with respect to a concave function f in the full access cost model is at most*

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

Finally we prove a result for all marking and conservative algorithms.

Theorem 8.4. *Let $\mathcal{A}$ be a marking or conservative algorithm. The competitive ratio of $\mathcal{A}$ with respect to a concave function f in the full access cost model is at most*

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

Proof. Let σ be a sequence consistent with f and consider the decomposition $D(\sigma, k)$ of σ. We know that $\mathcal{A}$ incurs at most k faults in each phase. Let m denote the number of faults $\mathcal{A}$ incurs on σ. We have $m \leq k \cdot |D(\sigma, k)| \Rightarrow |D(\sigma, k)| \geq m/k$. Each phase has length at least $f^{-1}(k)$ because σ is consistent with f. Therefore $|\sigma| \geq |D(\sigma, k)| \cdot f^{-1}(k) \geq m \cdot f^{-1}(k)/k$, and

$$\frac{\mathcal{A}_{FA}(\sigma)}{\text{OPT}_{FA}(\sigma)} \leq \frac{p \cdot m + |\sigma|}{p \cdot m/k + |\sigma|} \leq \frac{p \cdot m + m \cdot f^{-1}(k)/k}{p \cdot m/k + m \cdot f^{-1}(k)/k} = \frac{p \cdot k + f^{-1}(k)}{p + f^{-1}(k)}.$$

□

8.4 A New Model for Locality of Reference

In this section we introduce a new model for locality of reference that can be used to show that the randomized marking algorithm MARK benefits from locality of reference. We described MARK in Section 7.1. Intuitively, a sequence with locality of reference does not have many clean pages in a phase. Recall that a page is called clean if it is not requested in the previous phase. In order to formalize this intuition we generalize the k-phase model as follows.

Definition 8.1. *Let σ be a sequence and consider its phase decomposition. For constants $a > 1$ and $b < 1$, σ is called (a, b)-local if $L(\sigma, k) \geq ak$ and each phase of $D(\sigma, k)$ has at most bk clean pages.*

Now we can define the fault rate of an algorithm $\mathcal{A}$ on (a, b)-local sequences, $F_{\mathcal{A}}(a, b)$, by restricting the input sequences to (a, b)-local sequences.

The following theorem shows that MARK works better on sequences that are "more" local.

Theorem 8.5. *For any constants $a > 1$ and $b < 1$,*

$$F_{\text{MARK}}(a, b) \leq \frac{b \cdot (H_k - H_{bk} + 1)}{a}.$$

Proof. Consider an arbitrary (a, b)-local sequence σ such that $|\sigma| \geq n$. We need to show that $F_{\text{MARK}}(\sigma) \leq \frac{b \cdot (H_k - H_{bk} + 1)}{a}$. Consider the phase decomposition of σ. Let l_i denote the number of clean pages of phase φ_i. Fiat et al. proved that the expected number of faults of MARK in phase φ_i, f_i, is at most $B_i = l_i \cdot (H_k - H_{l_i} + 1)$ [77]. Note that $1 \leq l_i \leq bk$; the first page of each phase is clean and σ is an (a, b)-local sequence. Since B_i is strictly increasing for $l_i \leq k$, we get $f_i \leq b \cdot k \cdot (H_k - H_{bk} + 1)$. Therefore the expected number of faults that MARK incurs on σ is at most $bk \cdot (H_k - H_{bk} + 1) \cdot |D(\sigma, k)|$. On the other hand we have $|\sigma| \geq ak \cdot |D(\sigma, k)|$. Therefore

$$F_{\text{MARK}}(\sigma) \leq \frac{bk \cdot (H_k - H_{bk} + 1) \cdot |D(\sigma, k)|}{ak \cdot |D(\sigma, k)|} = \frac{b \cdot (H_k - H_{bk} + 1)}{a}.$$

□

Since $H_n \approx \ln n$, we obtain an upper bound of $b \cdot (1 - \ln b)/a$ for $F_{\text{MARK}}(a, b)$. Thus the fault rate of MARK decreases as a increases and b decreases. Note that several other results can be obtained by imposing more restrictions on input sequences. For example if σ is an a-local sequence that contains only $k + 1$ distinct pages we have $l_i = 1$ for each phase φ_i and therefore

$$F_{\text{MARK}}(\sigma) \leq \frac{1 \cdot (H_k - H_1 + 1)|D(\sigma, k)|}{ak|D(\sigma, k)|} = \frac{H_k}{ak}.$$

8.5 Caching with Locality of Reference

In the paging problem, all pages have the same size and the same retrieval cost on a fault. However, in some applications such as caching files on the Web, pages have different sizes and the cost of bringing a page to the cache varies for different pages. We can generalize the paging problem in different ways. These generalized variants of the problem are usually called *caching* problems. We can have different models for the caching problem [156, 96]:

- **General Model**: pages have arbitrary sizes and arbitrary retrieval costs.
- **Weighted Caching**: pages have uniform sizes, but they can have arbitrary retrieval costs (weights).
- **Fault Model**: pages have arbitrary sizes, however, they have uniform retrieval costs.
- **Bit Model**: pages have arbitrary sizes and the retrieval cost is proportional to their size.

Each of these models is appropriate for certain applications. Irani describes various applications in Web caching that are best modeled using the Fault/Bit model [96]. In this section we study the behaviour of marking caching algorithms on sequences with high locality of reference.

8.5.1 Weighted Caching

For weighted caching, we need some new notation. Consider an online paging algorithm $\mathcal{A}$. Each page π has a weight $w(\pi)$. In the full access cost model, the cost of a hit is 1 and the cost of a fault on a page π is $p \cdot w(\pi) + 1$ for some parameter p. Let $W_{\mathcal{A}}(\sigma)$ be the total weight of pages on which $\mathcal{A}$ incurs a fault when it serves a sequence σ and $W_{\text{OPT}}(\sigma)$ be the

same value for the optimal offline algorithm. Define the average weight of faults in a phase as $AW_{\mathcal{A}}(\sigma) = W_{\mathcal{A}}(\sigma)/|D(\sigma,k)|$ and $AW_{\text{OPT}}(\sigma) = W_{\text{OPT}}(\sigma)/|D(\sigma,k)|$. Note that the full access cost of $\mathcal{A}$ and OPT on σ is $|\sigma|+p\cdot W_{\mathcal{A}}(\sigma)$ and $|\sigma|+p\cdot W_{\text{OPT}}(\sigma)$, respectively. Let $C_{FA}(\mathcal{A},\sigma) = \frac{|\sigma|+p\cdot W_{\mathcal{A}}(\sigma)}{|\sigma|+p\cdot W_{\text{OPT}}(\sigma)}$; then we have $C_{FA}(\mathcal{A}) = \sup_\sigma C_{FA}(\mathcal{A},\sigma)$.

Now assume that σ is an a-local sequence, i.e. $L(\sigma,k) \geq ak$ for some constant $a > 1$. We have

$$C_{FA}(\mathcal{A},\sigma) = \frac{L(\sigma,k) + p\cdot AW_{\mathcal{A}}(\sigma)}{L(\sigma,k) + p\cdot AW_{\text{OPT}}(\sigma)} \leq \frac{ak + p\cdot AW_{\mathcal{A}}(\sigma)}{ak + p\cdot AW_{\text{OPT}}(\sigma)}.$$

Note that the standard competitive ratio of $\mathcal{A}$ is

$$C(\mathcal{A}) = \sup_\sigma \frac{AW_{\mathcal{A}}(\sigma)}{AW_{\text{OPT}}(\sigma)}.$$

Therefore when p is large, $C_{FA}(\mathcal{A})$ approaches the standard competitive ratio. For smaller values of p, $C_{FA}(\mathcal{A})$ improves as the locality of reference increases.

8.5.2 Bit Model

There is a close connection between the Bit model and the full access cost model. Let $s(\pi)$ denote the size of a page π and k be the size of cache. In the Bit model, the retrieval cost of π is $r\cdot s(\pi)$ for some fixed constant r. In the full access cost model, the cost of a hit is 1 and the cost of a fault is $p+1$ for some parameter p. Therefore we can have a generalization of the full access cost model for the Bit model as follows. The cost of a hit is 1 and the cost of a fault on a page π is $q\cdot s(\pi)+1$ for some parameter q.

We obtain results in this model using the idea of a k-decomposition. However since pages can have arbitrary sizes, we modify the definition of the decomposition. We upper bound the total size of distinct pages in a phase, rather than the number of distinct pages. For an input sequence σ and an integer $m > 1$, the m-decomposition $D(\sigma,m)$ is defined as partitioning σ into consecutive phases so that each phase is a maximal subsequence that contains a set Π of distinct pages such that the total of pages in Π adds up to at most m units of information.[1] Note that each phase may contain a set of distinct pages whose total size is strictly less than m. $|D(\sigma,m)|$ and $L(\sigma,m)$ are as before.

A marking algorithm in this model works in phases. At the beginning of each phase all pages in the cache are unmarked. A page is marked when it is requested. On a fault, the algorithm brings the requested page to the cache and evicts as many (unmarked) pages as necessary from the cache to make room for this page. If all pages in the cache are marked, the phase ends and all pages are unmarked. As before, we call a sequence σ a-local if $L(\sigma,k) \geq ak$ for some constant $a > 1$. Also assume that we have normalized the sizes of pages so that the smallest pages have unit size.

Theorem 8.6. *Let $\mathcal{A}$ be an arbitrary marking algorithm on an a-local input sequence. Then under the Bit model we have $C_{FA}(\mathcal{A}) \leq 1 + q/a$.*

Proof. Consider an arbitrary a-local sequence σ. $\mathcal{A}$ incurs at most one fault on any page π in any phase φ because π is marked after the first fault and will not be evicted in the remaining

[1] Depending on the application, the unit of information can be: bit, byte, word, etc.

steps of φ. Since the total size of distinct pages in a phase is at most k, the full access cost of $\mathcal{A}$ on σ is at most $|\sigma| + |D(\sigma, k)| \cdot (q \cdot k)$. On the other hand, according to the definition of the decomposition, the optimal offline algorithm should incur at lease one fault in each phase and therefore its full access cost is at least $|\sigma| + |D(\sigma, k)| \cdot (q \cdot 1)$. Therefore we get

$$C_{FA}(\mathcal{A}, \sigma) \leq \frac{|\sigma| + |D(\sigma, k)| \cdot (q \cdot k)}{|\sigma| + |D(\sigma, k)| \cdot (q \cdot 1)} = \frac{L(\sigma, k) + q \cdot k}{L(\sigma, k) + q \cdot 1}.$$

Now since σ is an a-local sequence, $L(\sigma, k) \geq ak$ and

$$C_{FA}(\mathcal{A}, \sigma) \leq \frac{ak + q \cdot k}{ak + q \cdot 1} = 1 + \frac{(k-1)}{ak/q + 1} < 1 + q/a.$$

This completes the proof as σ is an arbitrary a-local sequence. □

8.6 Conclusions

In this chapter we studied selected models for paging with locality of reference. In particular we proved that in general the competitive ratio does not improve on input sequences with high locality of reference under the models of Torng [148] and Albers et al. [6]. We also proposed a new model for locality of reference and proved that the randomized marking algorithm has better fault rate on sequences with high locality of reference. Finally we generalized the existing models to several variants of the caching problem.

Chapter 9

Adaptive Searching in One and Two Dimensions

Searching in a geometric space is an active area of research, predating computer technology. The applications are varied, ranging from robotics to search-and-rescue operations in the high seas [133, 120], to avalanche rescue [33], to office discovery automation [90, 60, 91], to scheduling of heuristic algorithms for solvers searching an abstract solution space [104, 105, 119, 13, 116]. Within academia, the field has seen two marked boosts in activity. The first was motivated by the loss of weaponry off the coast of Spain in 1966 in what is known as the Palomares incident and of the USS Thresher and Scorpion submarines in 1963 and 1966 respectively [133, 146]. A second renewed thrust took place in the late 1980s when the applications for autonomous robots became apparent.

Geometric searching has proven to be a fertile ground within computational geometry for the design and analysis of search and recognition strategies under various initial conditions [92, 90, 52, 60, 61, 115, 117]. The basic search scenarios consist of exploring a one dimensional object, such as a path or office corridor, usually modeled as the real line, and of exploring a two dimensional scene, such as a room or a factory floor, usually modelled as a polygonal scene. However, in spite of numerous advances in the theoretical understanding of both of these scenarios, so far such solutions have generally had a limited impact in practice.

Over the years various efforts have been made to address this situation, both in terms of isolated research papers attempting to narrow the gap, as well as in organized efforts such as the Algorithmic Foundations of Robotics conference and the Dagstuhl seminars on online robotics which bring together theoreticians and practitioners. From these it is apparent that the cost model and hence the solutions obtained from theoretical analysis do not fully reflect real life constraints. Several efforts have been made to resolve this, such as including the turn cost, the scanning cost, and error in navigation and reckoning [62, 74, 102, 117, 115].

In this chapter we address one more shortcoming of the standard model. Consider for example a vacuuming robot—such as Roomba(TM). Such a robot explores the environment using sophisticated motion planning algorithms with the goal of attaining complete coverage of the floor surface within a reasonable amount of time. It is not hard to devise worst case floor plans (such as complex mazes) which would not be covered very efficiently. In practice this is not a concern since (i) most rooms are relatively simple and (ii) if the robot ever encounters such a complex scene a drop in performance is only to be expected and users would not mind a severe

degradation in the time required to complete the task. As discussed previously, this naturally leads to the concept of adaptive algorithms, in which on simpler inputs the robot must perform more efficiently than on more complex ones.

In this chapter we consider adaptive analysis of two basic geometric primitives: searching on the real line and looking around the corner. Searching on the real line consists of finding a target t on the real line located at an unknown distance d (in either direction) from a search robot. The robot detects t upon contact. The optimal strategy visits the rays under a doubling strategy with competitive ratio of 9 [27, 82, 20, 118]. We refer the reader to the survey of Alpern and Gal [11] for a thorough discussion. However upon being presented by the optimal doubling strategy practitioners routinely report that they find the answer non-intuitive and generally "not optimal". This holds for the optimal strategy for either the average or the worst case. There are several non-mutually exclusive explanations for this disparity. In particular we incorporate the observation that in some settings, exploration is a valuable task in which case the goal is to simultaneously minimize the time to the target, and maximize the amount of information gained during the search. For this case we obtain an optimal strategy that is, subjectively, more pleasing to practitioners.

For the second case study we consider searching around a corner. Icking et al. [92] provided an algorithm with competitive ratio $c \approx 1.21218$ and proved that this is the best competitive ratio possible. We extend this result by applying adaptive analysis to this problem.

9.1 Searching on the Real Line

Without loss of generality, we assume the robot searches starting from the origin x_1 units to left, then it returns to the origin and moves past it x_2 units to the right. In general in the ith phase, it goes x_i units from origin to the left or right (depending on the parity of i) and returns to the origin. The search ends when the robot finds the target. In the *doubling strategy* we have $x_i = 2^{i-1}$. In the standard cost model, we minimize the ratio of the distance travelled by the robot to the straight distance from the target to the origin, which is termed the competitive ratio. As stated before, the doubling method has competitive ratio 9, which is optimal.

In order to reflect the underutilization of robot resources when traversing a region that has already been explored, we propose a dual cost model. It costs one unit whenever the robot traverses one unit of distance of unknown territory, while it costs c units ($c \geq 1$) when the robot traverses a region that has already been explored.

In order to find the worst case for doubling method under the new cost model, assume that the target is located at distance $2^k + \varepsilon$ from the origin, for some integer k. Therefore robot will find the target at phase $k+3$. For $3 \leq i \leq k+2$, let $C(i)$ be the cost robot incurs at phase i. At phase i, the robot goes 2^{i-1} units away from the origin and then returns to the origin. Of the first 2^{i-1} units, 2^{i-3} units are already explored and $2^{i-1} - 2^{i-3} = 3 \times 2^{i-3}$ units are newly explored. All 2^{i-1} units on the robot's return to the origin are already explored. Therefore we have $C(i) = 2^{i-3}(5c+3)$. Thus the total cost of the first $k+2$ phases is $(1+c) + (2+2c) + \sum_{i=3}^{k+2} 2^{i-3}(5c+3) = (5 \times 2^k - 2)c + 3 \times 2^k$. In the last phase, the robot finds the target at distance $2^k + \varepsilon$, incurring cost $2^k c + \varepsilon$. Thus the competitive ratio of doubling is

$$\frac{(6 \times 2^k - 2)c + 3 \times 2^k + \varepsilon}{2^k + \varepsilon},$$

which becomes arbitrarily close to $6c+3$ as k grows. Note that for $c=1$ we get the standard competitive ratio of 9.

Observe that the doubling might no longer be the optimal strategy under the new model. As usual we consider the family of geometric search strategies A_r: we have $x_i = r^{i-1}$ for an arbitrary real number $r>1$ (the doubling strategy corresponds to $\mathcal{A}_2$). Using arguments similar to the analysis of the doubling method, the cost of robot at phase $3 \le i \le k+2$ is $C(i) = r^{i-3}((r^2+1)c+(r^2-1))$ and the total cost of $\mathcal{A}_r$ is

$$\left(r+1+\left(r^2+1\right)\left(\frac{r^k-1}{r-1}\right)+r^k\right)c+\left(r^2-1\right)\left(\frac{r^k-1}{r-1}\right)+\varepsilon.$$

Thus the competitive ratio of A_r for this case is

$$CR(\mathcal{A}_r) = \frac{\left(r+1+\left(r^2+1\right)\left(\frac{r^k-1}{r-1}\right)+r^k\right)c+(r+1)\left(r^k-1\right)+\varepsilon}{r^k+\varepsilon},$$

which becomes arbitrarily close to $(r+1)\left(1+\frac{rc}{r-1}\right)$ as k grows. Through symbolic manipulation, we find out that the competitive ratio is minimized for $r = 1+\sqrt{\frac{2c}{c+1}}$. As c goes to ∞, this optimal value of r goes to $1+\sqrt{2} = 2.414213\ldots$ with a search cost of $\left(3+2/\sqrt{2}\right)c+2+\sqrt{2} \approx 5.83c+3.41$. This improves over the $6c+3$ cost of doubling for large c.

Furthermore, this is optimal, as it can be shown using the Gal-Schuierer functional theorem [82, 136] as follows. For any given strategy, let $X = x_0, x_1, x_2, \ldots$ denote the (infinite) sorted sequence of turn points incurred by the strategy. Then using ideas similar to [119] we can lower bound the competitive ratio by $CR \ge cost(ALG)/cost(OPT)$, where $cost(ALG) = (x_0+cx_0)+(x_1+cx_1)+(x_2-x_0+cx_0+cx_2)+\ldots+(x_{k+1}-x_{k-1}+cx_{k-1}+cx_{k+1})+cx_k$, and $cost(OPT) = x_k$. Therefore, we have that

$$CR(X,k) \ge \frac{(c+1)\sum_{i=0}^{k+1} x_i + (c-1)\sum_{i=0}^{k-1} x_i + cx_k}{x_k} \tag{9.1}$$

Let $X^{+i} = (x_i, x_{i+1}, \ldots)$ denote the suffix of a sequence $X = (x_0, x_1, \ldots)$ starting at x_i.

Theorem 9.1 ([136]). *Let $X = (x_0, x_1, \ldots)$ be a sequence of positive numbers, r an integer, and $a = \limsup_{n\to\infty}(x_n)^{1/n}$, for $a \in \mathbb{R}\cup\{+\infty\}$. If F_k, $k \ge 0$, is a sequence of functionals which satisfy*
(1) $F_k(X)$ only depends on $x_0, x_1, \ldots, x_{k+r}$,
(2) $F_k(X)$ is continuous, $\forall x_i > 0$, with $0 \le i \le k+r$,
(3) $F_k(\alpha X) = F_k(X)$, $\forall \alpha > 0$,
(4) $F_k(X+Y) \le \max(F_k(X), F_k(Y))$, and
(5) $F_{k+i}(X) \ge F_k(X^{+i})$, $\forall i \ge 1$,
then $\sup_{0\le k<\infty} F_k(X) \ge \sup_{0\le k<\infty} F_k(A_r)$.

It is not hard to verify that the hypotheses of the theorem hold for the modified cost model, and hence it suffices to consider x_i of the form r^{i-1} in the expression for $CR(X,k)$ above. Note that the left-hand side of inequality 9.1 above is precisely the expression we derived when upper-bounding the competitive ratio. Therefore, substituting r with $1+\sqrt{2}$ yields a lower bound on

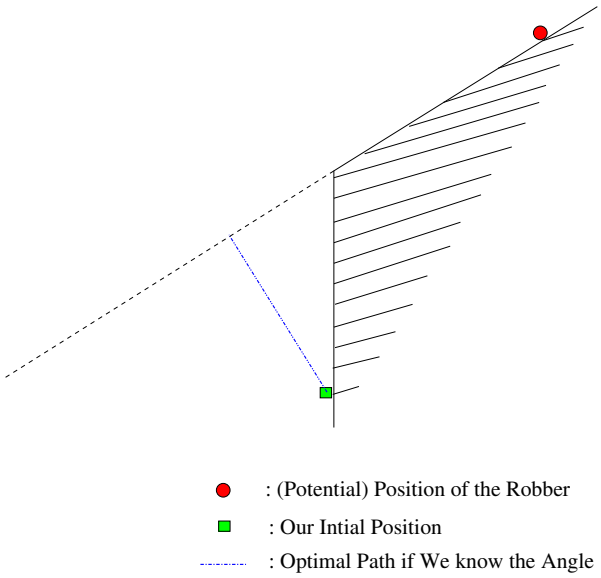

Figure 9.1: Discovering the presence of the robber using minimum movement.

$CR(X, k)$ which is identical to the upper bound, which in turn implies that the geometric strategy with $r = 1 + \sqrt{2}$ is in fact asymptotically optimal.

We can extend the dual cost model to cases in which $c < 1$, i.e., revisiting is less expensive than discovering. The case $c = 0$ can also be used to model two sequential communicating searchers. If $c < 0$, the robot can reduce its cost by revisiting the discovered territories ad infinitum and no optimal strategy exists. For $0 < c < 1$, we can use an analysis analogous to the case $c \geq 1$ to show that A_r with $r = 1 + \sqrt{\frac{2c}{c+1}}$ is optimal. For $c = 0$, the optimal strategy is A_r with $r = 1 + \varepsilon$ for a very small constant ε and this leads to the competitive ratio $2 + \varepsilon$.

9.2 Looking Around a Corner

Consider the following scenario: a robber might be hiding behind a corner. Naturally one would like to determine as soon as possible if this is the case. The situation is complicated by the fact that two walls forming the corner are not necessarily perpendicular. The question is what path to follow so that we discover at the earliest possible time the presence/absence of the robber. Observe that if we knew the angle formed by the wall, we would simply move on a direction perpendicular to it, to the closest point formed by the extension of the wall (See Figure 9.1). We will formally define the problem in the next paragraph. An optimal competitive strategy for this problem is presented in [92]. In this section we consider the problem in a man-made environment in which there is a preferential occurrence of orthogonal and near orthogonal angles. We wish to explore the change in the nature of the solution when this assumption is made.

We follow the same approach as [92] and formulate the problem using a differential equation. Therefore we use similar terminology and notation and just highlight the differences between the methods; refer to [92] for omitted details. First we formally define the problem. Figure 9.2 shows a typical instance of the problem. The corner is placed at the origin O and one of its halflines coincides with the negative y axis. The other halfline of the corner makes an angle $0 \leq \varphi \leq \pi$

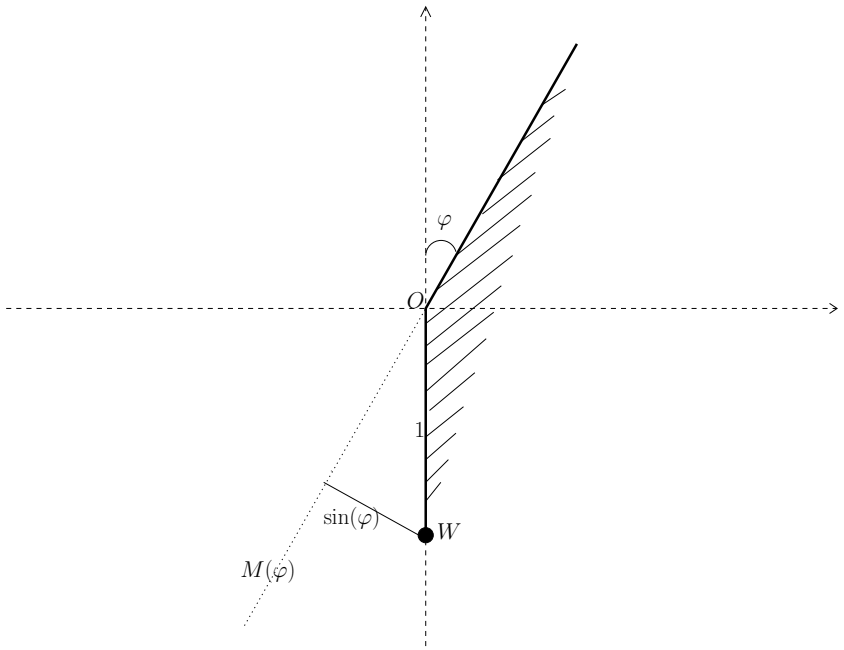

Figure 9.2: A typical instance of the corner problem.

with the positive y axis. A mobile robot is located at point $W = (0, -1)$ and is equipped with an on-board vision system facing O. When $\varphi > 0$, the robot cannot see the other halfline (wall) of the corner and his goal is to discover that (invisible) halfline by minimum movement. The robot sees the invisible line the first time it visits any point on the prolongation $M(\varphi)$ of the invisible line. Let $a(\varphi)$ be the distance between W and $M(\varphi)$. We have

$$a(\varphi) = \begin{cases} \sin\varphi & \text{if } 0 \le \varphi \le \pi/2 \\ 1 & \text{if } \pi/2 < \varphi \le \pi \end{cases} \tag{9.2}$$

If the robot knows φ then it can discover the invisible wall by the optimal movement $a(\varphi)$. However this is not the case and the robot should come up with a strategy S that works for all $0 \le \varphi \le \pi$. Let $A_S(\varphi)$ be the length of the path generated by S from W to the first point on $M(\varphi)$. Then the *competitive function* of S is defined as $f_S(\varphi) = \frac{A_S(\varphi)}{a(\varphi)}$ and the *competitive factor* of S is defined as $c_S = \sup_{\varphi\in(0,\pi]} f_S(\varphi)$.

In practical robot navigation most corners have angles close to $\pi/2$ and usually we do not have angles close to 0 or π. As a first attempt for applying adaptive analysis ideas we consider $d(\varphi) = 1/\sqrt{\sin\varphi}$ as difficulty measure. Figure 9.3 shows the behaviour of $d(\varphi)$ for $0 < \varphi < \pi$. We normalize the competitive function further by $d(\varphi)$ and the new competitive function is defined as

$$g_S(\varphi) = \frac{f_S(\varphi)}{d(\varphi)} = \begin{cases} \frac{A_S(\varphi)}{\sqrt{\sin\varphi}} & \text{if } 0 \le \varphi \le \pi/2 \\ A_S(\varphi)\sqrt{\sin\varphi} & \text{if } \pi/2 < \varphi \le \pi \end{cases}$$

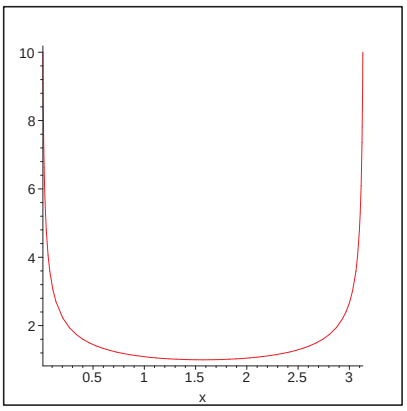

Figure 9.3: Plot of $d(\varphi) = \frac{1}{\sqrt{\sin\varphi}}$ for $0 < \varphi < \pi$.

Icking et al. [92] describe the strategies by curves of form $S = (\varphi, s(\varphi))$ in polar coordinates about O that satisfy certain properties, e.g., $s(0) = 1$. They show that the optimal competitive strategy is given by the solution to

$$f_R(\varphi) = \frac{A_R(\varphi)}{\sin\varphi} = c,$$

for all $\varphi \in [0, \pi/2]$ and for some constant c (the smallest c if there are several solutions). For our cost model, the corresponding equation becomes $g_R(\varphi) = \frac{A_R(\varphi)}{\sqrt{\sin\varphi}} = c$. We have

$$A_R(\varphi) = c\sqrt{\sin\varphi} \Rightarrow \frac{c\cos\varphi}{2\sqrt{\sin\varphi}} = A_R'(\varphi) = \sqrt{r'^2(\varphi) + r^2(\varphi)}$$

$$\Rightarrow r'(\varphi) = -\sqrt{\frac{c^2\cos^2\varphi}{4\sin\varphi} - r^2(\varphi)}.$$

We take the negative square root because in an optimal strategy the robot should always come closer to the corner. By replacing $r(\varphi)$ by $cu(\varphi)$ we get the differential equation

$$u'(\varphi) + \sqrt{\frac{\cos^2\varphi}{4\sin\varphi} - u^2(\varphi)} = 0, \tag{9.3}$$

with initial condition $u(0) = 1/c$. Therefore, our problem reduces to:

Problem Find the minimum $c > 1$, such that the ordinary differential equation (9.3) has a solution on some interval $[0, \sigma] \subseteq [0, \pi/2]$, subject to the following constraints:

1. $u(0) = 1/c$
2. $u(\varphi) > 0$ for $\varphi \in [0, \sigma]$
3. $u(\sigma) = 0$

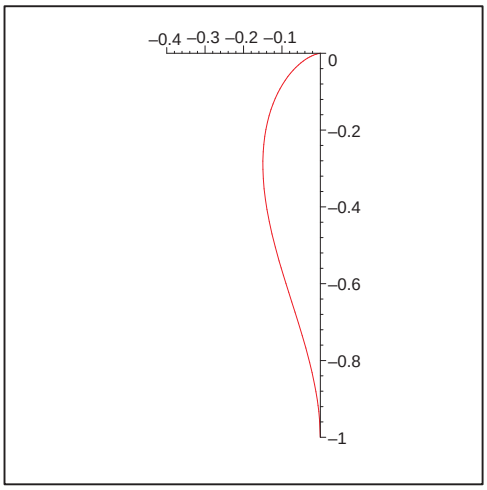

Figure 9.4: Robot's optimal path in the new model.

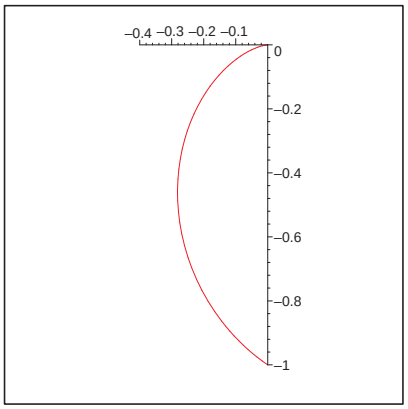

Figure 9.5: Robot's optimal path in the previous model.

Since this type of differential equations generally do not have a closed form we use numerical methods to compute the solution $c \approx 1.08$. The strategy with this competitive factor is shown in Figure 9.4. We can prove the optimality of this strategy using arguments analogous to [92].

The optimal strategy in the standard model is shown in Figure 9.5. It has competitive factor ≈ 1.21 [92]. Observe that since less weight is given to small angles the solution takes a shorter path to reach sightlines for angles around $\pi/4$.

Chapter 10

List Update Algorithms for Data Compression

List update algorithms have been widely used as subroutines in compression schemes, most notably as part of Burrows-Wheeler compression. The Burrows-Wheeler transform (BWT), which is the basis of many state-of-the-art general purpose compressors applies a compression algorithm to a permuted version of the original text. List update algorithms are a common choice for this second stage of BWT-based compression. As well, list update algorithms have been shown to be a reasonable alternative to simple compression schemes such as Huffman coding [31]. In this chapter we perform an experimental comparison of various list update algorithms both as stand alone compression mechanisms and as a second stage of the BWT-based compression. Our experiments show MTF outperforms other list update algorithms in practice after BWT. This is consistent with the intuition that BWT increases locality of reference and the predicted result from the locality of reference models described in Chapters 4 and 7. Lastly, we observe that due to an often neglected difference in the cost models, good list update algorithms may be far from optimal for BWT compression and construct an explicit example of this phenomena. This is a fact that had yet to be supported theoretically in the literature.

10.1 Introduction

It has long been observed that list update algorithms can be used in compression. In 1986, Bentley et al. [31] proposed a compression scheme that uses MTF as a subroutine. They proved that their compression scheme, based on MTF is guaranteed to be within twice the compression ratio of the best static Huffman code. Experimentally their algorithm performs even better achieving compression ratios equal or better than Huffman's. In principle MTF can be replaced with any other online list update algorithm, which may or may not improve the compression rate. Albers and Mitzenmacher [8] studied the use of timestamp and showed theoretical and experimental evidence for its efficiency in data compression. Several online list update algorithms were compared according to their efficiency in compression by Bachrach et al. [19]. Surprisingly, their results show that some algorithms with bad competitive ratios outperform those that are optimal according to competitive analysis in terms of compression ratio.

A second application of list update is to Burrows and Wheeler compression. The Burrows-Wheeler transform (BWT) rearranges a string of symbols to one of its permutations and in doing

so brings the issue of higher order entropy into play. Then MTF is used to encode this transform in a way similar to the scheme proposed by Bentley et al. [31]. The resulting scheme is shown to be very effective in theory and practice and many improvements and several variants have been proposed [51, 106, 53, 127, 67, 76, 21, 22]. The well known compression program bzip2 [138] is based on the BWT.

Our study was motivated by recent theoretical results on the impact of locality of reference assumptions for online algorithms [12, 68], as described in Chapters 4 and 7. Compression via list update hinges on an implicit assumption that the text (raw or after the BWT transform) exhibits locality of reference which can then be used advantageously by list update algorithms. In this paper we systematically study different sensible choices for the list update algorithm as well as for the basic compressor.

Our Results. We perform an experimental comparison of the latest list update algorithms for compression, both in stand alone form and as part of BWT based compression. We show that in most cases MTF is the best choice. Additionally, we observe that list update algorithms optimize for a similar but different objective than a compressor and give an example of an algorithm which is a good choice for list update but not for compression, a fact that had yet to be reported in the literature.

10.2 Preliminaries

10.2.1 Compression Schemes

Bentley et al. [31] proposed using list update algorithms as subroutines in compression. The idea is simple enough: both the encoder and the decoder maintain a list L of all symbols in the file and agree on some online list update algorithm $\mathcal{A}$ as well as an initial arrangement for L. The encoder encodes every symbol by its current position in L and then rearranges L according to $\mathcal{A}$. It uses some variable length prefix-free binary code to transmit these integers (positions). Since the decoder knows the initial arrangement of L and the list update algorithm, it can maintain the same list as the encoder and recover all the symbols. Several variable length prefix-free binary codes can be used in this scheme, e.g., Elias encoding, δ-encoding, and ω-encoding. We refer the reader to [19] for a full description.

10.2.2 Burrows-Wheeler Transform.

Burrows and Wheeler [51] introduced the idea of a preprocessing phase based on the BWT which is combined with a compression scheme on the transformed text. Informally, the BWT rearranges a string of symbols to one of its permutations in a reversible way so that the resulting string is "more compressible" or has more "locality of reference". The permutation is such that high order entropy is in line with locality of reference. Recall that a string has high locality of reference if when a symbol occurs in some position of the string, it is more likely to occur in a nearby position. For a detailed explanation of the BWT we refer the reader to [51, 106].

10.3 Competitiveness of List Update Algorithms for Compression

A list update algorithm $\mathcal{A}$ incurs cost i to access the ith item of the list. However, when we use $\mathcal{A}$ as a subroutine for compression we need $\Theta(\log i)$ bits to represent that the symbol is at the ith position of the list. Other papers that have studied the use of list update algorithms in compression are silent on this issue and apparently simply assumed that competitive list update algorithms are also competitive for compression. We show via an example that this is not necessarily the case, i.e. there exist algorithms which are competitive under one model but not the other.

Consider the MOVE-FRACTION (MF) family of deterministic list update algorithms as introduced by Sleator and Tarjan [140]. Upon a request to an item in the ith position, MF(k) moves that item $\lceil i/k \rceil$-1 positions towards the front. MF(k) is known to be $2k$-competitive [140], therefore algorithm MF(2) is 4-competitive for list update. We show that under the $\Theta(\log i)$ cost model, MF(2) does not have constant competitive ratio. Let the cost of compressing for an item in the ith position be $c\lfloor \log i \rfloor + b$ for some constants c and b. For simplicity assume that we have $l = 2^p$ symbols for some integer p. Suppose that symbols are initially ordered as $a_1 a_2 \cdots a_l$ in the list. Now consider the sequence $\sigma_1 = a_l^p$. On the ith request to a_l, MF(2) incurs cost at least $c\lfloor \log \frac{2^p}{2^{i-1}} \rfloor + b = c(p - i + 1) + b$ and moves a_l to a position of index at least $\frac{2^p}{2^i}$. Therefore the cost of MF(2) on σ_1 is at least

$$\sum_{i=1}^{p} (c(p - i + 1) + b) = \frac{cp(p+1)}{2} + bp = \Theta(\log^2 l).$$

On the other hand, MTF moves a_l to the front of the list and incurs cost $c\lfloor \log l \rfloor + b + (p-1)b = (b + c) \log l$ on σ_1. Thus the cost of OPT on this sequence is at most $(b + c) \log l = \Theta(\log l)$. We can request the item that is now in the lth position of MF(2)'s list p times. Therefore the competitive ratio of MF(2) is at least

$$\frac{c \times \log l(\log l + 1)/2 + b \log l}{(b + c) \log l} = \frac{c(\log l + 1)}{2(b + c)} + \frac{b}{b + c} = \Theta(\log l),$$

which is not a constant. The same holds for MF(k) for $k \geq 3$. This fact had been observed empirically by Bachrach et al. [19], who reported on the poor performance of this family for data compression purposes. It remains an open question to determine the competitive ratios of the various list update algorithms under the $c\lfloor \log i \rfloor + b$ cost of access model.

10.4 Experimental Results

We consider two experimental setups. The first one consists of a straightforward compression scheme similar to that of Bentley et al. [31] or Albers et al. [8]. While in practice these compression techniques are unlikely to be of use, the study of their behaviour allows us to understand their differences and advantages. The second setup consists of the realistic setting of BWT based compression. To be more precise, given a text we compute its BWT and then compare the role of various list update algorithms for compressing the transformed string.

10.4.1 Experimental Settings

We compare the compression ratios achieved by different list update algorithms on files in the Calgary Corpus [151] and the Canterbury Corpus [15]. These are standard benchmarks for data compression. We consider the list update algorithms described in Section 1.2 (namely Move-To-Front (MTF), Transpose (TR), Frequency-Count (FC), Timestamp (TS), and Sort-By-Rank (SBR)) as well as MTF′; this algorithm, on the ith access to an item a, moves a to the front of the list if i is even and does not change a's position if i is odd. We considered two implementations for frequency-count depending on the order of items with the same frequency count. In FC, an item that is less recently used precedes an item that is more recently used and has equal frequency count. FC′ adopts the reverse of this ordering. We consider different parameters for SBR since a compressor can, at time of compression, select the parameter α which achieves the most compression and then prepend the compressed file with the choice of α. If not explicitly mentioned otherwise, we use the standard prefix integer encoding of Elias [70] that encodes an integer i using $1 + 2\lfloor \log i \rfloor$ bits. Observe that nonetheless we propose and evaluate other alternative ways for encoding integers.

10.4.2 Sort-By-Rank Parameters

Recall that SBR (0) is equivalent to MTF and SBR (1) is equivalent to TS modulo handling of the first few accesses. Additionally, intuitively SBR (α) mediates between the behaviour of MTF and TS. We test this intuition in the case of compression. Figure 10.1 shows the percentage of the size of the file obtained using SBR(α) as compared to the original file size for different values of parameter α and four files of the Calgary Corpus. Figure 10.2 presents the results for the same files after applying BWT. Observe that as α goes from 0 to 1, the behaviour of SBR (α) goes from MTF to TS. This change in behaviour is faster for small values of α. Although SBR (α) usually achieves best compression for the extremal values of α ($\alpha = 0$ or $\alpha = 1$), there are a few cases in which the optimal value of α is different. For example for file book1 after BWT, the best compression is 32.34% and it is achieved by SBR (0.32).

10.4.3 Comparing List Update Algorithms

We compare the effect of different list update algorithms on text files of the Calgary Corpus and the Canterbury Corpus before and after BWT. Table 10.1 shows their performance as stand alone compression algorithms while Table 10.2 shows their performance as a second stage of BWT compression. From Table 10.1 we can see that TR and FC usually outperform MTF and TS. This is in contrast with competitive analysis in which MTF and TS are superior to TS and FC. MTF has the worst performance on all the files and TR is the best algorithm in most cases. MTF′ and FC′ always have performance close to their variants, i.e., MTF and FC, respectively. Note that the results for MTF and TS were also reported by Albers and Mitzenmacher [8], who observed that TS outperforms MTF. SBR(0.5) always mediated between the performance of MTF and TS. Thus our experimental results are not consistent with theory. This has been observed by other researches as well [19].

However, for the BWT transform of the files, the situation is different. Table 10.2 shows that in this case MTF has the best performance for most of the files. In general, MTF and TS (and thus MTF′ and SBR(0.5)) have comparable performance and always outperform FC

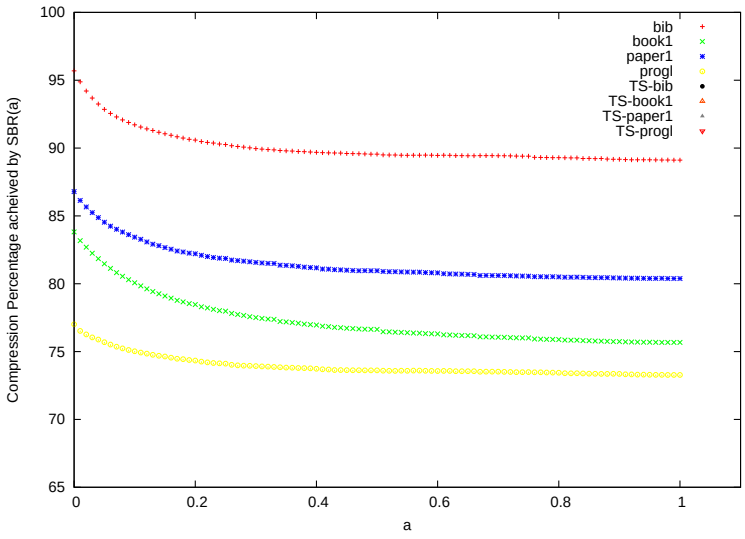

Figure 10.1: Compression using SBR(α) for files in the Calgary Corpus in terms of α.

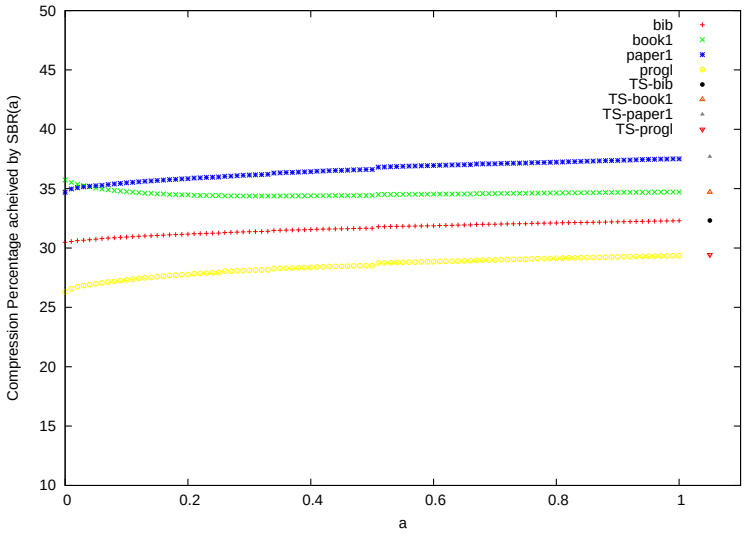

Figure 10.2: Compression using SBR(α) for files in the Calgary Corpus after BWT in terms of α.

File	Size (bytes)	MTF	SBR (0.5)	TS	FC	TR	MTF′	FC′
bib	111261	95.69	89.55	89.08	**81.42**	81.64	94.16	**81.42**
book1	768771	83.82	76.64	75.67	81.34	**69.62**	81.27	81.34
book2	610856	84.35	78.36	77.55	75.74	**72.44**	82.35	75.74
news	377109	88.50	82.68	82.20	88.10	**77.87**	87.08	87.99
paper1	53161	86.79	80.96	80.35	79.48	**74.87**	85.19	79.45
paper2	82199	84.47	78.34	77.43	79.27	**71.02**	82.26	80.45
progc	39611	88.74	84.02	83.62	81.59	**77.67**	88.16	81.54
progl	71646	77.01	73.62	73.25	82.61	**69.02**	76.50	82.40
progp	49379	81.09	76.15	75.45	82.41	**71.64**	80.00	81.68
trans	93695	87.58	84.96	84.59	91.21	**83.02**	87.36	91.18
alice29.txt	152089	83.69	76.49	75.62	74.74	**69.24**	81.48	74.85
asyoulik.txt	125179	88.54	81.71	80.67	79.36	**73.92**	86.44	79.36
cp.html	24603	92.45	91.47	91.90	87.12	**85.53**	93.01	88.31
fields.c	11150	84.05	80.05	79.81	74.41	**73.52**	83.73	74.25
grammar.lsp	3721	79.82	75.30	73.45	77.88	**68.56**	78.63	77.69
lcet10.txt	426754	82.50	76.07	75.23	79.14	**69.91**	80.42	85.39
plrabn12.txt	481861	86.16	77.67	76.29	88.48	**69.98**	82.85	88.48

Table 10.1: Compression of the Calgary and Canterbury Corpus without BWT.

and TR. The compression ratio they achieve after the BWT is much better than without the BWT, as one would expect given that the BWT increases the amount of locality in the string. The superiority of MTF to other algorithms is consistent with the recent result of Angelopoulos et al. proving that MTF outperforms all other online list update algorithm on sequences with high locality of reference [12]. Hence, this provides evidence that the locality of reference model proposed accurately reflects reality. We emphasize that our focus here is comparing the effect of different list update algorithms and therefore we have not applied any post-optimizations to the compression scheme, in the presumption that these optimizations are orthogonal and hence would generally benefit all schemes equally.

File	Size (bytes)	MTF	SBR (0.5)	TS	FC	TR	MTF′	FC′
bib	111261	**30.49**	31.66	32.32	93.42	39.81	31.99	93.33
book1	768771	35.74	**34.42**	34.71	76.63	36.31	36.04	76.50
book2	610856	31.14	**31.03**	31.48	80.44	35.31	31.96	80.11
news	377109	**36.21**	37.75	38.67	85.27	44.90	38.26	85.53
paper1	53161	**34.70**	36.62	37.70	83.42	47.73	36.87	83.34
paper2	82199	**34.86**	35.35	36.04	79.00	41.28	36.17	76.46
progc	39611	**35.04**	37.32	38.54	79.03	51.09	37.54	78.91
progl	71646	**26.31**	28.52	29.43	81.23	36.18	28.33	79.77
progp	49379	**26.00**	29.08	30.22	89.11	41.13	28.57	86.08
trans	93695	**24.12**	27.64	28.71	96.08	41.52	26.76	90.22
alice29.txt	152089	33.15	**32.97**	33.45	81.34	37.43	33.99	81.37
asyoulik.txt	125179	36.96	**36.53**	37.08	83.08	41.50	37.79	80.96
cp.html	24603	**36.10**	38.22	39.37	91.54	49.62	38.58	89.03
fields.c	11150	**29.96**	33.24	35.03	80.73	51.75	32.84	78.74
grammar.lsp	3721	**34.64**	38.48	40.85	74.36	52.06	39.26	70.63
lcet10.txt	426754	30.76	**30.55**	31.01	76.49	34.21	31.51	76.64
plrabn12.txt	481861	36.30	**35.23**	35.57	78.88	37.13	36.65	78.88

Table 10.2: Compression of the Calgary and Canterbury Corpus after BWT.

We also observe that FC and FC′ perform badly compared to other algorithms. One explanation for this is the fact that FC considers the global rather than local environment. For example

if an item is frequently accessed near the beginning and then it is not accessed at all, FC will maintain it close to the front of the list.

10.4.4 Alternative Techniques for Encoding of Integers

We consider other possibilities for the last step of list update based compression schemes, i.e., the prefix-free binary code for integers. All these algorithms apply the following intuition: there is considerable locality of reference in the BWTs of text files intuitively a competitive list update algorithm leads to a sequence with many small integers. Hence, the algorithms we considered assign smaller codes to small integers. In particular, there are many "1"s in the sequence. Therefore almost all these algorithms use a form of run length encoding on "1"s.

RL(1)+Elias. This algorithm combines Elias encoding with run length encoding for the value 1, i.e. when the encoded integer is 1, the following Elias-encoded integer shows the number of consecutive 1's starting from that 1. Otherwise, is the next integer encoded in Elias encoding.

File	Size (bytes)	MTF	SBR (0.5)	TS	FC	TR	MTF′	FC′
bib	111261	**27.87**	28.92	29.55	93.42	37.06	29.28	93.42
book1	768771	35.78	**34.50**	34.77	76.78	36.46	36.02	78.68
book2	610856	29.72	**29.56**	30.00	80.52	33.98	30.48	80.53
news	377109	**35.51**	36.82	37.71	85.33	43.96	37.37	85.50
paper1	53161	**34.60**	36.32	37.38	83.36	47.56	36.64	84.96
paper2	82199	**34.59**	35.01	35.66	79.00	41.02	35.80	78.96
progc	39611	**34.83**	36.89	38.07	79.15	50.83	37.15	82.32
progl	71646	**24.15**	26.17	27.07	81.25	33.96	26.07	84.32
progp	49379	**23.87**	26.68	27.80	89.14	38.92	26.29	91.77
trans	93695	**20.92**	24.26	25.31	95.58	38.32	23.46	102.71

Table 10.3: Compression of the Calgary Corpus using RL(1)+Elias after BWT.

RL(1)+1-2. This algorithm encodes 1 with a single bit 0, and encodes all other numbers with their binary representations prepended by 1. We need $\lceil \log_2 l \rceil$ bits for this binary representation. For most of the cases, this gives a code of length 8 for each integer greater than 1, as $64 \leq l < 128$. Also it uses run length on "1"s.

File	Size (bytes)	MTF	SBR (0.5)	TS	FC	TR	MTF′	FC′
bib	111261	**36.36**	37.77	38.18	87.44	43.09	37.44	87.44
book1	768771	59.33	**57.89**	57.92	98.47	59.05	59.25	96.34
book2	610856	47.94	**47.89**	48.10	97.96	50.78	48.47	97.96
news	377109	**51.60**	53.52	54.16	97.87	58.28	53.09	97.87
paper1	53161	**52.15**	54.29	54.93	88.66	62.02	53.56	88.66
paper2	82199	**54.25**	54.92	55.35	99.97	59.67	55.24	99.97
progc	39611	**50.31**	53.00	53.93	85.96	61.76	52.06	99.40
progl	71646	**36.93**	40.08	41.04	99.76	47.21	38.94	99.76
progp	49379	**36.20**	39.70	40.80	99.72	48.68	37.97	99.72
trans	93695	**30.01**	34.98	35.70	90.49	45.81	31.78	99.99

Table 10.4: Compression of the Calgary Corpus using RL(1)+1-2 after BWT.

RL(1)+2-2-3: This algorithm encodes 1 and 2 with "00" and "01", respectively, and encodes all other numbers with their binary representations prepended by 1. It also uses run length on "1"s.

RL(1)+1-5-6-17: This algorithm encodes 1 by "0", 2 to 9 by "10000", "10001", ..., "10111", 10 to 17 by "110000", "110001", ..., "110111", and integers greater than 17 by their binary representation prepended by "111". Note that there are $l-17$ such numbers, and so we can use a fixed code of length $\lceil \log_2 (l-17) \rceil$ for their binary representations. It also uses run length on "1"s, i.e., when it encodes a "1" the following integer, encoded using the same scheme, denotes the number of consecutive ones started from that "1".

Tables 10.3-10.6 show the performance of these algorithms on text files of the Calgary Corpus after BWT. According to these results, RL(1)+Elias leads to the best compression among these algorithms, then RL(1)+5-6-17, then RL(1)+2-2-3, and finally RL(1)+1-2. Comparing Table 10.3 to Table 10.2 shows that using RL(1) improves the compression factor for most list update algorithms. For algorithms other than FC and FC′, the improvement is considerable. This can be explained by the fact that BWTs of text files have many repetitions. Each such repetition leads to a 1 in the sequence of integers. Therefore we will have many 1's and RL(1) should be effective. Also according to Tables 10.3-10.6, replacing Elias with other proposed integer encodings does not give better compression ratios.

Modified Huffman. Inspired by the fact that there are many blocks of "1"s in the integer sequence we treat them as symbols of our alphabet. Thus our alphabet is $\{1, 2, \cdots, l, 11, 111, \cdots, 1^n\}$, where 1^n means n consecutive "1"s. Then Huffman encodes the elements of this alphabet. The

File	Size (bytes)	MTF	SBR (0.5)	TS	FC	TR	MTF′	FC′
bib	111261	**31.74**	32.83	33.32	86.54	38.76	32.78	86.54
book1	768771	48.54	**47.29**	47.51	93.96	48.94	48.67	94.98
book2	610856	39.16	**39.05**	39.47	97.93	42.77	39.75	97.93
news	377109	**44.63**	45.94	46.66	97.89	51.72	45.98	97.89
paper1	53161	**44.31**	46.15	47.03	88.68	55.53	45.97	88.68
paper2	82199	**45.67**	46.42	47.02	88.92	52.06	46.78	88.92
progc	39611	**42.64**	44.85	45.73	83.80	55.28	44.27	86.35
progl	71646	**31.09**	33.16	33.94	86.96	41.10	32.55	86.96
progp	49379	**29.87**	32.87	33.80	97.04	43.30	31.53	99.70
trans	93695	**26.40**	29.71	30.64	88.14	41.64	27.90	92.06

Table 10.5: Compression of the Calgary Corpus using Algorithm RL(1)+2-2-3 after BWT.

results are shown in Table 10.7. Note that we should also encode the Huffman tree. This cost becomes negligible for large files, especially if one considers implicit representations of the portions of the Huffman code corresponding to 1^k. Indeed the Huffman tree has an impact in the order of 0.3% of compressed size uniformly across the different variants for these rather modest file sizes.

According to these results, this scheme outperforms all other algorithms in our study. Figure 10.3 reports the mean, median and variance of the comparison of other compression algorithms to the modified Huffman algorithm.

10.4.5 Splay Trees

Recall from Section 1.2 that list update algorithms belong to the area of self-organizing data structures and that another well known self-organizing data structure is the splay tree [141]. The splay tree is a binary search tree which applies a splay operation after each access to an item. This operation reorganizes the tree such that the most recently accessed item is moved to the root of the tree. Splay trees are believed to have good performance on sequences with high locality of reference. The working set theorem of [141] shows that splay trees have the working set property. The working property is based on the idea that an operation on a recently accessed item should take less time. Informally, a structure has the working set property if it performs well on sequences with high locality of reference. As stated before there is usually high locality of reference in texts (especially after applying BWT) and thus splay trees are in principle good candidates for text compression. Jones [100] and Grinberg et al. [85] have already studies the application of splay trees to data compression, but they did not consider the BWT.

We studied the effect of using splay trees instead of list update algorithms in our compression

File	Size (bytes)	MTF	SBR (0.5)	TS	FC	TR	MTF′	FC′
bib	111261	**29.54**	30.61	31.10	82.72	37.22	30.53	82.25
book1	768771	40.43	**39.41**	39.50	74.74	40.77	40.41	73.34
book2	610856	33.50	**33.49**	33.76	77.62	36.98	33.98	77.64
news	377109	**38.36**	39.68	40.44	82.37	45.62	39.69	82.68
paper1	53161	**37.80**	39.51	40.33	76.96	48.98	39.20	78.38
paper2	82199	**38.10**	38.54	39.04	77.75	43.43	38.92	77.72
progc	39611	**37.90**	39.92	40.94	75.69	51.50	39.53	84.28
progl	71646	**26.93**	29.02	29.89	80.58	35.73	28.37	83.62
progp	49379	**26.73**	29.40	30.52	85.70	40.28	28.29	86.80
trans	93695	**22.53**	26.10	27.01	90.77	38.38	24.03	96.63

Table 10.6: Compression of the Calgary Corpus using 1-5-6-17+RL(1) after BWT.

schemes. We constructed a splay tree on the characters of the text file. Each character corresponds to a node of the tree and has a binary code that corresponds to the path from the root to its node, i.e., starting from the root, append 0 for each left traversal and 1 for each right traversal. Note that as we proceed with the compression process, the tree changes dynamically and thus the codes for characters are changing as well. Since characters can be in internal nodes, the corresponding codes are not prefix-free. To obtain a prefix-free code, we first add a single 1 to the beginning of each code. Then we consider the number that corresponds to this binary representation and encode these integers using Elias encoding. Note that the code for the root character would be 1.

We can also apply alternative techniques for encoding integers proposed in Subsection 10.4.4. We tested the RL(1)+Elias and the modified Huffman techniques. The compression percentages obtained by applying these schemes to the text files of the Calgary Corpus after BWT are shown in Table 10.8. According to these results, the modified Huffman algorithm is again the best technique for encoding integers and splay trees lead to less compression compared to the best list update algorithms.

10.5 Conclusions

We have considered a variety of list update algorithms in the context of data compression with and without the Burrows-Wheeler transform. We observed that list update algorithms optimize for a similar but different objective than a compressor and gave an example of an algorithm which is a good choice for list update but not for compression. Our experiments showed that competitive list update algorithms are not very effective as compressors without BWT, while they perform well after BWT. We also considered several schemes for encoding the sequence of

File	Size (bytes)	MTF	SBR (0.5)	TS	FC	TR	MTF′	FC′
bib	111261	**26.25**	27.01	27.53	65.70	33.29	27.34	65.70
book1	768771	32.54	**31.66**	31.89	56.91	33.49	32.71	56.86
book2	610856	27.70	**27.61**	27.99	59.93	31.58	28.30	59.94
news	377109	**33.44**	34.25	34.91	64.55	39.64	34.75	64.63
paper1	53161	**32.96**	34.17	35.06	59.46	42.78	34.51	59.48
paper2	82199	**32.39**	32.75	33.30	58.72	37.65	33.33	58.67
progc	39611	**33.21**	34.76	35.64	62.38	44.96	34.99	64.78
progl	71646	**23.43**	24.82	25.53	60.39	31.22	24.91	61.83
progp	49379	**23.22**	25.40	26.26	62.51	34.74	25.24	62.56
trans	93695	**20.42**	22.99	23.84	65.59	33.45	22.51	71.19

Table 10.7: Compression of the Calgary Corpus using Modified Huffman after BWT.

integers that is obtained after applying a list update algorithm. Furthermore, we experimentally tested the efficacy of splay trees in data compression and observed that they are not as effective as list update algorithms.

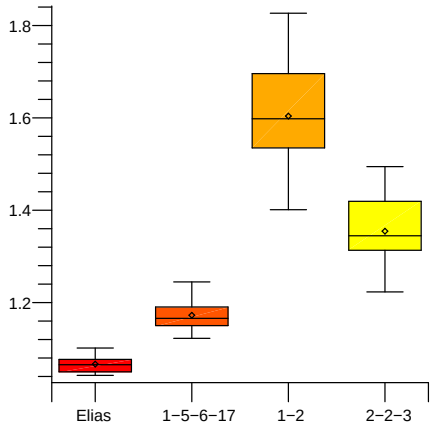

Figure 10.3: Relative compression ratio versus modified Huffman. For each file, Modified Huffman equals 1.

File	Size (bytes)	Elias	RL(1)+Elias	Modified Huffman
bib	111261	37.76	35.14	31.43
book1	768771	44.91	44.94	40.40
book2	610856	38.53	37.12	33.75
news	377109	46.05	45.35	40.49
paper1	53161	43.63	43.53	38.94
paper2	82199	43.71	43.44	38.97
progc	39611	44.14	43.95	39.06
progl	71646	32.14	29.98	27.43
progp	49379	31.34	29.21	26.63
trans	93695	28.92	25.71	23.32

Table 10.8: Compression of the Calgary Corpus using splay Tree Based Schemes after BWT.

Chapter 11

Conclusions

In this thesis we introduced several new models for analysis of online algorithms. We applied these models to paging and list update and showed that they overcome many of the shortcomings of competitive analysis for these problems.

In Chapter 4 we introduced Average Analysis and Bijective Analysis as two new models for comparing the performance of online algorithms. They directly compare two algorithms on all sequences of the same length. We proved that LRU is better than FWF according to these models. However, all lazy paging algorithms have the same performance with respect to Bijective Analysis and Average Analysis. By incorporating the locality of reference assumption of [6], we proved that LRU is the unique best online paging algorithm according to Average Analysis. This resolved a long standing open problem. In Chapter 5 we showed that Bijective Analysis reflects influence of lookahead. Furthermore, we proved superiority of MTF to other online list update algorithms in the presence of locality of reference.

In Chapter 6 we introduced relative interval analysis. In this model we consider the difference between the performance of two algorithms on sequences of the same length. We compared paging algorithms using this model. We also proved that relative interval analysis shows the effect of lookahead for paging. An obvious open problem is filling the missing cells in Table 6.1. Another open problem is combining relative interval analysis with a model for paging with locality of reference.

So far, these three models have been applied to a few other online problems. Recently, Boyar et al. [42] applied Bijective Analysis to a variant of the server problem. Paging is the only problem studied under relative interval analysis. A remaining task for the future is to apply these models to other online algorithms. We remark that Average Analysis, Bijective Analysis, and relative interval analysis can be used to compare the performance of offline algorithms too. For example, we can have two approximation algorithms with vastly different performance in practice that have the same approximation ratio. As for competitive ratio, this can be due to the pessimistic nature of the approximation ratio. In this case, we might get more useful information by comparing the two algorithms using Average Analysis, Bijective Analysis, or relative interval analysis.

In Chapter 7 we applied adaptive analysis to paging and list update. We provided a few natural definitions for the amount of locality in input sequences for paging and list update. We then expressed the performance of well known paging and list update algorithms in terms of these measures of locality. The performance of paging algorithms improves as we increase the size of

the cache, while the reverse holds for competitive analysis. Also the performance of LRU is only a constant factor worse than OPT.

In Chapter 8 we further studied two existing models for paging with locality of reference. We proved that these models are not effective if we use standard cost models, while they work well with some alternative cost models. We also extended these locality models to a few variants of caching. In Chapter 9 we applied adaptive analysis to two basic geometric search problems, namely searching on the real line and looking around a corner. Our results showed that optimal solutions in the new model are different from the standard models and are closer to what we expect in practice. A remaining future work is to extend these ideas to more complex search problems, e.g., exploring a polygon.

In Chapter 10 we tested the performance of various list update algorithms in the context of data compression. We provided and analyzed results of comprehensive experiments. Our results showed that list update algorithms are effective when used in combination with BWT and MTF usually is the best algorithm in this context. Also splay trees are not as effective as MTF. Furthermore, we observed that we should use a different cost model for analyzing the performance of list update algorithms as part of a compression schema. We showed that performance of list update algorithms can be different in this cost model. A remaining open problem is to compute the competitive ratio of well known list update algorithms in the new cost model.

References

[1] G. M. Adel'son-Vel'skii and E. M. Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1262, 1962. 4

[2] P. Afshani, C. Hamilton, and N. Zeh. Cache-oblivious range reporting with optimal queries requires superlinear space. In *Proceedings of the 25th annual symposium on Computational geometry (SoCG '09)*, pages 277–286, 2009. 21

[3] A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988. 19

[4] S. Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, 1997. 35, 49

[5] S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998. 5

[6] S. Albers, L. M. Favrholdt, and O. Giel. On paging with locality of reference. *Journal of Computer and System Sciences*, 70(2):145–175, 2005. 33, 34, 38, 42, 47, 48, 55, 56, 59, 67, 68, 72, 73, 74, 84, 86, 87, 90, 111

[7] S. Albers and S. Lauer. On list update with locality of reference. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP '08)*, pages 96–107, 2008. 51

[8] S. Albers and M. Mitzenmacher. Average case analyses of list update algorithms, with applications to data compression. *Algorithmica*, 21(3):312–329, 1998. 51, 55, 98, 100, 101

[9] S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, 56(3):135–139, 1995. 5

[10] S. Albers and J. Westbrook. Self-organizing data structures. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms — The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 13–51. Springer-Verlag, 1998. 4, 5

[11] S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous*. Kluwer Academic Publishers, 2003. 92

[12] S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. List update with locality of reference. In *Proceedings of the 8th Latin American Symposium on Theoretical Informatics (LATIN '08)*, pages 399–410, 2008. 51, 99, 104

[13] S. Angelopoulos, A. López-Ortiz, and A. M. Hamel. Optimal scheduling of contract algorithms with soft deadlines. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI '08)*, pages 868–873, 2008. 91

[14] S. Angelopoulos and P. Schweitzer. Paging and list update under bijective analysis. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09)*, pages 1136–1145, 2009. 55

[15] R. Arnold and T. C. Bell. A corpus for the evaluation of lossless compression algorithms. In *Data Compression Conference*, pages 201–210, 1997. 101

[16] Y. Azar, J. Boyar, L. Epstein, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen. Fair versus unrestricted bin packing. *Algorithmica*, 34(2):181–196, 2002. 26

[17] E. Bach, J. Boyar, L. Epstein, L. M. Favrholdt, T. Jiang, K. S. Larsen, G. Lin, and R. Van Stee. Tight bounds on the competitive ratio on accommodating sequences for the seat reservation problem. *Journal of Scheduling*, 6(2):131–147, 2003. 26

[18] E. Bach, J. Boyar, T. Jiang, K. S. Larsen, and G. Lin. Better bounds on the accommodating ratio for the seat reservation problem. In *Proceedings of the 6th Annual International Computing and Combinatorics Conference (COCOON '00)*, pages 221–231, 2000. 26

[19] R. Bachrach, R. El-Yaniv, and M. Reinstaedtler. On the competitive theory and practice of list accessing algorithms. *Algorithmica*, 32(2):201–246, 2002. 48, 51, 82, 98, 99, 100, 101

[20] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993. 92

[21] B. Balkenhol and S. Kurtz. Universal data compression based on the Burrows-Wheeler transformation: theory and practice. *IEEE Transactions on Computers*, 49(10):1043–1053, 2000. 99

[22] B. Balkenhol, S. Kurtz, and Y. M. Shtarkov. Modifications of the Burrows and Wheeler data compression algorithm. In *Data Compression Conference*, pages 188–197, 1999. 99

[23] C. Banderier, K. Mehlhorn, and R. Beier. Smoothed analysis of three combinatorial problems. In *Proceedings of the 28th Symposium on Mathematical Foundations of Computer Science (MFCS '03)*, pages 198–207, 2003. 30

[24] R. Bayer. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1:290–306, 1972. 4

[25] L. Becchetti. Modeling locality: A probabilistic analysis of LRU and FWF. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA '04)*, pages 98–109, 2004. 28, 38

[26] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, G. Schafer, and T. Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03)*, pages 462–471, 2003. 30, 31

[27] A. Beck. On the linear search problem. *Israel Journal of Mathematics*, 2:221–228, 1964. 92

[28] L. A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966. 3

[29] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994. 18, 19, 20, 22, 23

[30] 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. 9

[31] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986. 98, 99, 100

[32] D. S. Bernstein, T. J. Perkins, S. Zilberstein, and L. Finkelstein. Scheduling contract algorithms on multiple processors. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 702–706, 2002. 32

[33] T. C. Biedl, M. Hasan, J. D. Horton, A. López-Ortiz, and T. Vinar. Searching for the center of a circle. In *Proceedings of the 14th Canadian Conference on Computational Geometry (CCCG '02)*, pages 137–141, 2002. 91

[34] A. Blum and J. Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA '02)*, pages 905–914, 2002. 30

[35] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998. 3, 4, 19, 51, 68, 85

[36] 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. 18, 27, 51, 84

[37] G. E. P. Box and N. R. Draper. *Empirical model-building and response surface*. John Wiley & Sons, Inc., 1986. 20

[38] J. Boyar, M. R. Ehmsen, and K. S. Larsen. Theoretical evidence for the superiority of LRU-2 over LRU for the paging problem. In *Proceedings of the 4th Workshop on Approximation and Online Algorithms (WAOA '06)*, pages 95–107, 2006. 4

[39] J. Boyar and L. M. Favrholdt. The relative worst order ratio for online algorithms. *ACM Transactions on Algorithms*, 3(2), 2007. 18, 19, 20, 23, 24

[40] J. Boyar, L. M. Favrholdt, and K. S. Larsen. The relative worst-order ratio applied to paging. *Journal of Computer and System Sciences*, 73(5):818–843, 2007. 23, 24

[41] J. Boyar, L. M. Favrholdt, K. S. Larsen, and M. N. Nielsen. Extending the accommodating function. *Acta Informatica*, 40(1):3–35, 2003. 25, 26

[42] J. Boyar, S. Irani, and K. S. Larsen. A comparison of performance measures for online algorithms. In *Proceedings of the 11th International Algorithms and Data Structures Symposium (WADS '09)*, pages 119–130, 2009. 55, 111

[43] J. Boyar, S. Krarup, and M. N. Nielsen. Seat reservation allowing seat changes. *Journal of Algorithms*, 52(2):169–192, 2004. 26

[44] J. Boyar and K. S. Larsen. The seat reservation problem. *Algorithmica*, 25(4):403–417, 1999. 24, 25, 26

[45] J. Boyar, K. S. Larsen, and M. N. Nielsen. The accommodating function – a generalization of the competitive ratio. Technical Report PP-1998-24, Department of Mathematics and Computer Science, Odense University, 1998. 26

[46] J. Boyar, K. S. Larsen, and M. N. Nielsen. Separating the accommodating ratio from the competitive ratio. Technical Report PP-1999-03, Department of Mathematics and Computer Science, Odense University, 1999. 26

[47] J. Boyar, K. S. Larsen, and M. N. Nielsen. The Accommodating Function: A generalization of the competitive ratio. *SIAM Journal on Computing*, 31(1):233–258, 2001. 18, 25, 26

[48] J. Boyar and P. Medvedev. The relative worst order ratio applied to seat reservation. *ACM Transactions on Algorithms*, 4(4), 2008. 23, 24

[49] J. Boyar and M. N. Nielsen. An improved lower bound on first-fit's accommodating ratio for the unit price bin packing problem. Technical Report PP-1999-11, Department of Mathematics and Computer Science, Odense University, 1999. 26

[50] D. Breslauer. On competitive on-line paging with lookahead. *Theoretical Computer Science*, 209(1-2):365–375, 1998. 35

[51] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, DEC SRC, 1994. 55, 99

[52] K. F. Chan and T. W. Lam. An on-line algorithm for navigating in an unknown environment. *International Journal of Computational Geometry and Applications*, 3:227–244, 1993. 91

[53] B. Chapin. Switching between two on-line list update algorithms for higher compression of Burrows-Wheeler transformed data. In *Data Compression Conference*, pages 183–192, 2000. 99

[54] M. Chrobak and J. Noga. LRU is better than FIFO. *Algorithmica*, 23(2):180–185, 1999. 18, 27

[55] F. R. K. Chung, R. L. Graham, and M. E. Saks. A dynamic location problem for graphs. *Combinatorica*, 9:111–131, 1989. 35

[56] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Company, 1997. 30

[57] E. G. Coffman Jr. and E. N. Gilbert. On the expected relative performance of list scheduling. *Operations Research*, 33(3):548–561, 1985. 30

[58] R. Cole. On the dynamic finger conjecture for splay trees. part II: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000. 5

[59] R. Cole, B. Mishra, J. P. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. part I: Splay sorting log n-block sequences. *SIAM Journal on Computing*, 30(1):1–43, 2000. 5

[60] A. Datta, C. A. Hipke, and S. Schuierer. Competitive searching in polygons—beyond generalized streets. In *Proceedings of the 6th International Symposium on Algorithms and Computation (ISAAC '95)*, pages 32–41, 1995. 91

[61] A. Datta and C. Icking. Competitive searching in a generalized street. In *Proceedings of the 10th Annual Symposium on Computational Geometry (SoCG '94)*, pages 175–182, 1994. 91

[62] E. D. Demaine, S. P. Fekete, and S. Gal. Online searching with turn cost. *Theoretical Computer Science*, 361(2-3):342–355, 2006. 91

[63] E. D. Demaine, D. Harmon, J. Iacono, and M. Patrascu. Dynamic optimality - almost. *SIAM Journal on Computing*, 37(1):240–251, 2007. 5

[64] P. J. Denning. The working set model for program behaviour. *Communications of the ACM*, 11(5):323–333, 1968. 33, 66, 67

[65] P. J. Denning. Working sets past and present. *IEEE Transactions on Software Engineering*, SE-6(1):64–84, 1980. 66

[66] P. J. Denning. The locality principle. *Communications of the ACM*, 48(7):19–24, 2005. 20, 66

[67] S. Deorowicz. Improvements to Burrows-Wheeler compression algorithm. *Software, Practice, and Experience*, 30(13):1465–1483, 2000. 99

[68] R. Dorrigiv, M. R. Ehmsen, and A. López-Ortiz. Parameterized analysis of paging and list update algorithms. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA '09)*, 2009. to appear. 99

[69] R. Dorrigiv, A. López-Ortiz, and J. I. Munro. An application of self-organizing data structures to compression. In *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA '09)*, pages 137–148, 2009. 5

[70] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975. 101

[71] A. Elmasry. On the sequential access theorem and deque conjecture for splay trees. *Theoretical Computer Science*, 314(3):459–466, 2004. 5

[72] L. Epstein and L. M. Favrholdt. On-line maximizing the number of items packed in variable-sized bins. *Acta Cybernetica*, 16(1):57–66, 2003. 26

[73] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992. 14, 80

[74] S. P. Fekete, R. Klein, and A. Nüchter. Online searching with an autonomous robot. *Computational Geometry: Theory and Applications*, 34(2):102–115, 2006. 91

[75] A. Felner, R. Stern, A. Ben-Yair, S. Kraus, and N. Netanyahu. PHA*: Finding the shortest path with A* in an unknown physical environment. *Journal of Artificial Intelligence Research*, 21:631–670, 2004. 31

[76] P. M. Fenwick. The Burrows-Wheeler Transform for block sorting text compression: principles and improvements. *The Computer Journal*, 39(9):731–740, 1996. 99

[77] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991. 70, 85, 88

[78] A. Fiat and Z. Rosen. Experimental studies of access graph based heuristics: Beating the LRU standard? In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, pages 63–72, 1997. 27

[79] A. Fiat and G. J. Woeginger. Competitive odds and ends. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms — The State of the Art*, volume 1442 of *LNCS*, pages 385–394. Springer-Verlag, 1998. 18

[80] H. Fujiwara and K. Iwama. Average-case competitive analyses for ski-rental problems. *Algorithmica*, 42(1):95–107, 2005. 29

[81] S. Gal. Search games with mobile and immobile hider. *SIAM Journal of Control and Optimization*, 17(1):99–122, 1979. 30

[82] S. Gal. *Search Games*. Academic Press, 1980. 92, 93

[83] G. F. Georgakopoulos. Splay trees: a reweighing lemma and a proof of competitiveness vs. dynamic balanced trees. *Journal of Algorithms*, 51(1):64–76, 2004. 5

[84] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966. 2

[85] D. Grinberg, S. Rajagopalan, R. Venkatesan, and V. K. Wei. Splay trees for data compression. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms (SODA '95)*, pages 522–530, 1995. 107

[86] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS '78)*, pages 8–21, 1978. 4

[87] M. Halldórsson and M. Szegedy. Lower bounds on on-line graph coloring. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms (SODA '92)*, pages 211–216, 1992. 35

[88] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman, second edition, 1996. 33

[89] J. H. Hester and D. S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17(3):295, 1985. 48, 51

[90] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM Journal on Computing*, 31(2):577–600, 2001. 91

[91] C. Icking. *Motion and Visibility in Simple Polygons*. PhD thesis, Fernuniversität Hagen, 1994. 91

[92] C. Icking, R. Klein, and L. Ma. How to look around a corner. In *Proceedings of the 5th Canadian Conference on Computational Geometry (CCCG '93)*, pages 443–448, 1993. 91, 92, 94, 96, 97

[93] S. Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, 1991. 5

[94] S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11(1):53–72, 1994. 35

[95] S. Irani. Competitive analysis of paging. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms — The State of the Art*, volume 1442 of *LNCS*, pages 52–73. Springer-Verlag, 1998. 2, 18, 29

[96] S. Irani. Page replacement with multi-size pages and applications to web caching. *Algorithmica*, 33(3):384–409, 2002. 88

[97] 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. 18, 27, 84

[98] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973. 2

[99] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314, 1974. 2

[100] D. W. Jones. Application of splay trees to data compression. *Communications of the ACM*, 31(8):996–1007, 1988. 107

[101] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000. 27

[102] T. Kamphans. *Models and Algorithms for Online Exploration and Search*. PhD thesis, University of Bonn, 2005. 91

[103] M. Kao and S. R. Tate. Online matching with blocked input. *Information Processing Letters*, 38(3):113–116, 1991. 35

[104] M.-Y. Kao, Y. Ma, M. Sipser, and Y. Yin. Optimal constructions of hybrid algorithms. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*, pages 372–381, 1994. 91

[105] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '93)*, pages 441–447, 1993. 91

[106] H. Kaplan, S. Landau, and E. Verbin. A simpler analysis of Burrows-Wheeler-based compression. *Theoretical Computer Science*, 387(3):220–235, 2007. 55, 99

[107] A. R. Karlin. On the performance of competitive algorithms in practice. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms — The State of the Art*, volume 1442 of *LNCS*, pages 373–384. Springer-Verlag, 1998. 29

[108] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1–4):77–119, 1988. 2

[109] A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov paging. *SIAM Journal on Computing*, 30(3):906–922, 2000. 27

[110] C. Kenyon. Best-fit bin-packing with random order. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '96)*, pages 359–364, 1996. 18, 23

[111] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000. 18, 20, 28, 35, 36, 38

[112] E. Koutsoupias, C. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP '96)*, pages 280–289, 1996. 31

[113] P. Krugman. How I work. Available at: http://web.mit.edu/krugman/www/howiwork.html. 21

[114] A. López-Ortiz. *On-line target searching in bounded and unbounded domains*. PhD thesis, University of Waterloo, 1996. 30

[115] A. López-Ortiz. *On-line Target Searching in Bounded and Unbounded Domains*. PhD thesis, Dept. of Comp. Sci., University of Waterloo, 1996. 91

[116] A. López-Ortiz, S. Angelopoulos, and A. M. Hamel. Optimal scheduling of contract algorithms for anytime problems. In *Proceedings of the Twenty First National Conference on Artificial Intelligence*, 2006. 32, 91

[117] A. López-Ortiz and S. Schuierer. Simple, efficient and robust strategies to traverse streets. In *Proceedings of the 7th Canadian Conference on Computational Geometry (CCCG '95)*, pages 217–222, 1995. 91

[118] A. López-Ortiz and S. Schuierer. The ultimate strategy to search on m rays? *Theoretical Computer Science*, 261(2):267–295, 2001. 92

[119] A. López-Ortiz and S. Schuierer. Online parallel heuristics and robot searching under the competitive framework. In *SWAT*, pages 260–269, 2002. 91, 93

[120] A. López-Ortiz and G. Sweet. Parallel searching on a lattice. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG '01)*, pages 125–128, 2001. 91

[121] B. Manthey and R. Reischuk. Smoothed analysis of binary search trees. *Theoretical Computer Science*, 378(3):292–315, 2007. 30

[122] C. Martínez and S. Roura. On the competitiveness of the move-to-front rule. *Theoretical Computer Science*, 242(1–2):313–325, 2000. 5, 48

[123] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, pages 115–130, 2003. 64

[124] K. Mehlhorn. Sorting presorted files. In *Proceedings 4th GI Conference on Theoretical Computer Science*, pages 199–212, 1979. iii

[125] K. Mehlhorn. *Data Structures and Algorithms: 1. Searching and Sorting*. Springer, Berlin, 1984. iii

[126] J. I. Munro. On the competitiveness of linear search. In *Proceedings of the 8th Annual European Symposium on Algorithms (ESA '00)*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345, 2000. 5

[127] D. A. Nagy and T. Linder. Experimental study of a binary block sorting compression scheme. In *Data Compression Conference*, pages 439–448, 2003. 99

[128] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 297–306, 1993. 4

[129] K. Panagiotou and A. Souza. On adequate performance measures for paging. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*, pages 487–496, 2006. 19, 36, 79, 84

[130] H. Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1999. 19

[131] N. Reingold and J. Westbrook. Randomized algorithms for the list update problem. Technical Report YALEU/DCS/TR-804, Department of Computer Science, Yale University, June 1990. 82

[132] N. Reingold, J. Westbrook, and D. D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994. 51

[133] H. R. Richardson and L. D. Stone. Operations analysis during the underwater search for Scorpion. *Naval Research Logistics Quarterly*, 18(2):141–157, 1971. 91

[134] S. J. Russell and S. Zilberstein. Composing real-time systems. In *Proceedings of the 12th International Conference on Artificial Intelligence*, pages 212–217, 1991. 32

[135] M. Scharbrodt, T. Schickinger, and A. Steger. A new average case analysis for completion time scheduling. *Journal of the ACM*, 53(1):121–146, 2006. 30

[136] S. Schuierer. Lower bounds in on-line geometric searching. *Computational Geometry: Theory and Applications*, 18(1):37–53, 2001. 93

[137] F. Schulz. Two new families of list update algorithms. In *Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC '98)*, pages 99–108, 1998. 5, 51

[138] J. Seward. bzip2, a program and library for data compression. http://www.bzip.org/. (accessed on December 5, 2009). 99

[139] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. John Wiley & Sons, 2002. 19

[140] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. 2, 5, 24, 100

[141] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985. 2, 4, 5, 67, 107

[142] A. Souza. *Average Performance Analysis*. PhD thesis, Swiss Federal Institute of Technology Zürich, 2006. 30

[143] A. Souza and A. Steger. The expected competitive ratio for weighted completion time scheduling. *Theory of Computing Systems*, 39(1):121–136, 2006. 30

[144] D. A. Spielman and S. Teng. Smoothed analysis of termination of linear programming algorithms. *Mathematical Programming*, 97(1–2):375–404, 2003. 30

[145] D. A. Spielman and S. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004. 30

[146] L. D. Stone. What's happened in search theory since the 1975 Lanchester prize? *Operations Research*, 37(3):501–506, 1989. 91

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

[148] E. Torng. A unified analysis of paging and caching. *Algorithmica*, 20(2):175–200, 1998. 20, 33, 84, 90

[149] New Mexico State University. Homepage of new mexico state university tracebase (online). Available at: http://tracebase.nmsu.edu/tracebase.html (accessed on December 5, 2009). 68

[150] C. C. Wang, J. Derryberry, and D. D. Sleator. O (log log n)-competitive dynamic binary search trees. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '06)*, pages 374–383, 2006. 5

[151] I. H. Witten and T. Bell. The Calgary/Canterbury text compression corpus. Anonymous ftp from ftp.cpsc.ucalgary.ca /pub/text.compression/corpus/text.compression.corpus.tar.Z. 53, 101

[152] N. E. Young. *Competitive paging and dual-guided on-line weighted caching and matching algorithms*. PhD thesis, Department of Computer Science, Princeton University, 1991. 35

[153] N. E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994. 18, 19, 24

[154] N. E. Young. Bounding the diffuse adversary. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA '98)*, pages 420–425, 1998. 28, 38

[155] N. E. Young. On-line paging against adversarially biased random inputs. *Journal of Algorithms*, 37(1):218–235, 2000. 28

[156] N. E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002. 19, 24, 25, 88

[157] S. Zilberstein, F. Charpillet, and P. Chassaing. Real-time problem-solving with contract algorithms. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1008–1015, 1999. 32