Competitive Analysis of Paging: A Survey

Sandy Irani

Information and Computer Science Department, University of California, Irvine, CA

Abstract. This paper is a survey of competitive analysis of paging. We present proofs showing tight bounds for the competitive ratio achievable by any deterministic or randomized online algorithm. We then go on to discuss variations and refinements of the competitive ratio and the insights they give into the paging problem. Finally, we variations to the online paging problem to which competitive analysis has been applied.

1 Introduction

The paging problem has inspired several decades of theoretical and applied research and has now become a classical problem in computer science. This is due to the fact that managing a two level store of memory has long been, and continues to be, a fundamentally important problem in computing systems. The paging problem has also been one of the cornerstones in the development of the area of online algorithms. Starting with the seminal work of Sleator and Tarjan which initiated the recent interest in the competitive analysis of online algorithms, the paging problem has motivated the development of many important innovations in this area.

1.1 Definitions

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

If we knew the future, the decision would be clear. It has long been known that the optimal offline algorithm, called MIN, is the algorithm which always evicts the page whose next request is furthest in the future [Bel66]. Unfortunately, in practice paging decisions are made without knowledge of future requests. Typically, a paging algorithm must be *on-line*, meaning that it must make the decision of which page to evict without knowing which pages will be requested in the future.

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

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

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

Motivated by these observations, Sleator and Tarjan proposed the idea of *competitive analysis*. [1] In competitive analysis, the performance of the online algorithm is compared to the performance of the optimal offline algorithm. Let $\texttt{cost}_A(\sigma)$ be the cost incurred by an online algorithm A on the input sequence σ. In the case of paging, σ is a sequence of page requests, and $\texttt{cost}_{k,A}(\sigma)$ is the number of page faults incurred by algorithm A on the sequence σ when the fast memory can hold k virtual memory pages. Let OPT be the optimal offline algorithm, and let $\texttt{cost}_{k,OPT}(\sigma)$ be the cost incurred by the optimal offline algorithm on input σ when the fast memory can hold k virtual memory pages.

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

$$\texttt{cost}_{k,A}(\sigma) \leq c \cdot \texttt{cost}_{k,OPT}(\sigma) + b.$$

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

Notice that we are not placing any computational restrictions on the algorithm – we are simply measuring, from an information theoretic point of view, what kind of solution quality can be obtained given the fact that the decisions have to be made with partial information.

2 Deterministic Algorithms

Sleator and Tarjan give tight bounds on the best competitive ratio which can be achieved by any deterministic online paging algorithm [ST85]. They show that two commonly used paging algorithms achieve a competitive ratio of k. These algorithms are First-In-First-Out (FIFO) which on a fault evicts the page that was placed in the fast memory least recently and Least-Recently-Used (LRU) which on a fault evicts the page that was used least recently. They then show a lower bound of k for the competitive ratio achievable by any online algorithm, thus establishing the optimality of LRU and FIFO by the competitive measure.

Below is the proof for the lower bound. Lower bounds for the competitive ratio are often proven using an adversary-style argument, as is common in a more traditional worst-case algorithmic analysis. The idea is that the algorithm plays against an adversary who concocts the worst-case scenario for the algorithm. In competitive analysis, however, the adversary has two tasks. First, it must devise a costly input sequence for the algorithm. Then it must service that sequence, showing an upper bound on the optimal cost for that sequence. The argument below is an example of this type of lower bound argument.

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

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

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

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

Sleator and Tarjan actually proved a generalization of this result by allowing the online and offline algorithms to have different memory capacities. Let k_{on} be the number of pages which can fit into the online algorithm's fast memory. Let k_{opt} be the number of pages which can fit into the optimal algorithm's fast memory. They show tight bounds of $k_{on}/(k_{on} - k_{opt} + 1)$ for the competitive ratio of the optimal online algorithm as long as $k_{on} \geq k_{opt}$. If $k_{on} < k_{opt}$, the competitive ratio is not bounded.

Now we turn to proving upper bounds for deterministic online algorithms. Instead of showing the upper bound for LRU directly, we show an upper bound for a general class of algorithms called *marking* algorithms. This class was formally defined by Karlin, Manasse, Sleator, and Rudolph [KMRS88] and includes LRU.

A marking algorithm proceeds in phases. At the beginning of a phase all the nodes are unmarked. Whenever a page is requested, it is marked. On a fault, the marking algorithm evicts an unmarked page (chosen by a rule specified by the algorithm), and brings in the requested page. A phase ends just before the first fault after every page in the fast memory is marked (equivalently, a phase ends just before the request to the $k+1^{st}$ distinct page requested in the phase). At this point all the nodes become unmarked and a new phase begins. Marking algorithms and the notion of phases are key concepts which continually reappear in the study of the competitive analysis of paging.

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

Theorem 2. *[KMRS88] Any marking algorithm is k-competitive.*

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

Thus, the optimal algorithm must fault.

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

□

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

3 Randomized Algorithms

Some care must be taken in discussing competitive analysis of randomized online algorithms since there is no unique notion of an adversary. The issue which distinguishes different adversaries is the extent to which they know the outcomes of random choices made by the algorithm and how they themselves service the sequence they generate. A discussion of the different types of adversaries is beyond the scope of this article, so we will suffice it to say that our discussion will always assume an *oblivious* adversary.[2] Such an adversary must choose the entire request sequence σ, without any knowledge either of the outcome of the coin tosses or of the specific actions taken as a result of the coin tosses. However, the oblivious adversary does know the algorithm itself including the probability distribution of actions taken for a given input. Formally, we say that a randomized online algorithm A is c-competitive if there exists a constant b such that on every request sequence σ,

$$E[\mathtt{cost}_{k,A}(\sigma)] \leq c \cdot \mathtt{cost}_{k,OPT}(\sigma) + b.$$

(See [BDBK+90] for a more detailed discussion of the different adversaries.)

[2] The lower bound of k for the competitive ratio of any deterministic paging algorithm shown in Section 2 can easily be extended to apply to randomized algorithms against an adaptive online adversary. Thus, no advantage can be gained in using randomization against an adaptive adversary.

4 Upper Bounds

4.1 Memoryless randomized paging

We begin our discussion of randomized paging algorithms with the simplest randomized algorithm which we call RANDOM: on a fault, evict a random page from the cache. This is a memoryless algorithm: the only information used in making replacement decisions is which pages are currently in the cache.

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

Theorem 3. *[RS89] Any memoryless paging algorithm A has competitive ratio at least k against an oblivious adversary.*

4.2 Memory versus Randomness.

Raghavan and Snir also make the interesting observation that RANDOM uses $\lceil \log k \rceil$ random bits at every fault while FIFO is a deterministic algorithm which can be implemented using $\lceil \log k \rceil$ bits of memory. Note that $\lceil \log k \rceil$ bits are always necessary at every fault in order to specify which page to evict. To implement FIFO using $\lceil \log k \rceil$ bits of memory, a counter is kept which always has a value in the range $\{0, 1, \ldots, k-1\}$. The pages are stored as a circular array in memory and the counter points to the next page to be evicted. When the next fault occurs, the requested page replaces the page in the location specified by the counter. Then the counter is incremented mod k.

This suggests a whole family of algorithms which are hybrids of RANDOM and FIFO. The algorithm A^i uses i bits of memory and $j = \lceil \log k \rceil - i$ random bits on each fault. Let $I = 2^i$ and $J = k/I$. The pages in memory are organized in a $I \times J$ matrix. The i bits specify a row of the matrix. On a fault, A^i evicts a page chosen randomly from the row pointed to by the i-bit counter. The counter is then incremented mod i. Raghavan and Snir prove the following theorem, demonstrating the trade-off between random bits and memory:

Theorem 4. *[RS89] The competitive ratio of A^i is k.*

4.3 The Randomized Marking Algorithm

Although no memoryless randomized algorithm has competitiveness below k against oblivious adversaries, there do exist non-memoryless randomized algorithms which beat the deterministic bound. Fiat, Karp, Luby, McGeogh, Sleator and Young [FKL+91] were able to show that a randomized marking algorithm has competitive ratio $2H_k$ against an oblivious adversary, where H_k is the k^{th} harmonic number. The algorithm, called RMA (for Randomized Marking Algorithm), is the marking algorithm which on a fault, evicts an unmarked page chosen uniformly at random from the set of unmarked pages.

Theorem 5. *[FKL+91] The randomized marking algorithm RMA has competitive ratio $2H_k$ against any oblivious adversary, where H_k is the k^{th} harmonic number.*

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

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

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

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

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

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

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

Now we must prove a lower bound for the optimal cost. We claim that

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

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

A much more complicated algorithm was shown to be H_k competitive by McGeoch and Sleator[MS91]. Recently, Achlioptas, Chrobak and Noga have shown a simpler algorithm which is also H_k-competitive [ACN96]. Their algorithm uses $O(k^2 \log k)$ bits of memory and $O(k^2)$ time per request. The algorithm is based on a very elegant characterization of the *work function* for the paging problem which was developed by Koutsoupias and Papadimitriou in [KP94]. Achlioptas, Chrobak and Noga also prove that the competitive ratio of RMA is exactly $2H_k - 1$.

4.4 Lower Bounds

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

Definition 6. Let $\mathcal{P}$ be a probability distribution on request sequences σ. An algorithm A is *c-competitive against $\mathcal{P}$* if there exists a constant, b, such that

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

Definition 7. Let $c_A^{\mathcal{P}}$ denote the infimum over all c such that A is c-competitive against $\mathcal{P}$, and let c_R denote the competitive ratio of a randomized algorithm R against an oblivious adversary.

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

Theorem 8.

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

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

We now illustrate the use of Theorem 8 for proving lower bounds on c_R by showing a lower bound for randomized paging algorithms against an oblivious adversary. The bound comes close to matching the upper bound shown in Section 4.3. The following theorem is due to Fiat, Karp, Luby, McGeoch, Sleator and Young [FKL+91].

Theorem 9. *Let R be any randomized paging algorithm. If the number of pages is greater than or equal to $k+1$, where k is the size of the cache, the competitive ratio of R against any oblivious adversary is greater than or equal to H_k.*

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

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

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

Once again, we divide the sequence of page requests into non-overlapping phases such that each phase contains maximal runs of requests to at most k distinct pages. As we have seen, if there are r distinct phases then the optimal algorithm can service the sequence with at most $r+1$ faults. At the beginning of a phase, OPT replaces the one page currently in the cache that will not be requested in

the phase. Therefore, if $N(m)$ is the random variable which is the number of phases in a sequence σ of length m (generated from $\mathcal{P}$), then the expected offline cost satisfies

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

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

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

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

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

Therefore, we have that

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

Applying Theorem 8 yields a lower bound of H_k on c_R. □

5 Variations on Competitive Analysis

It would appear at this point that the case has been closed on the competitive analysis of paging since tight lower bounds for both deterministic and randomized algorithms have been determined. Interestingly, however, these results have been instrumental in illustrating some of the deficiencies with competitive analysis, thus opening up new lines of research in addressing these shortcomings. In fact, a survey of the competitive analysis of paging is in some ways a survey of the refinements of competitive analysis, in general.

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 online algorithm any advantage. If an algorithm is allowed to see l requests into the future, the adversary can choose to repeat each request l times (called the "l-stuttered version" in [KP94]), thus taking away any advantage in having the lookahead. Meanwhile, the cost for the optimal algorithm does not change. It seems intuitive that in practice, an online algorithm would benefit from some limited lookahead, but this is not reflected in the model.

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

Another feature missing from the model is that it is based on the assumption that the online algorithm has absolutely no knowledge whatsoever about the upcoming sequence. However, this may be an unnecessarily pessimistic assumption. It is possible that by preprocessing the program generating the request sequence or by observing past behavior of the program, one can glean some useful information about future requests. One would like to use knowledge of a program's access pattern to improve paging performance.

Most of the refinements of competitive analysis which we discuss attempt to address some or all of these issues.

5.1 Access Graphs

The competitive analysis results conflict with practical experience on paging in at least two ways. First, FIFO and LRU have the same competitiveness, even though in practice LRU usually outperforms FIFO. Secondly, LRU usually incurs much less than k times the optimal number of faults, even though its competitiveness is k.

The reason for the practical success of LRU has long been known: most programs exhibit *locality of reference*. This means that if a page is referenced, it is more likely to be referenced in the near future (temporal locality) and pages near it in memory are more likely to be referenced in the near future (spatial locality). Indeed, a two-level store is only useful if request sequences are *not* arbitrary.

Motivated by these observations, Borodin, Irani, Raghavan and Schieber [BIRS91] introduce a graph theoretic model which restricts page requests so that they conform to a notion of locality of reference. An *access graph* $G = (V, E)$ for a program is a graph that has a vertex for each page that the program can reference. Locality of reference is imposed by the edge relation – the pages that can be referenced after a page p are the neighbors of p in G or p itself. Thus, a request sequence σ must be a walk on G. The specific walk that is generated is determined only at execution time since it depends on the input given to the program. The definition of competitiveness remains the same as before, except for this restriction on the request sequences. Let $c_{A,k}(G)$ denote the competitiveness of an on-line algorithm A on access graph G with k pages of fast memory. We denote by $c_k(G)$ the infimum of $c_{A,k}(G)$ over on-line algorithms A with k pages of fast memory. Thus $c_k(G)$ is the best that any on-line algorithm can do on the access graph G.

An access graph may be either directed or undirected. An undirected access graph might be a suitable model when the page reference patterns are governed by the data structures used by the program. For example, if a program performs operations on a tree data structure, and the mapping of the tree nodes to pages of virtual memory represents a contraction of a tree, then the appropriate access graph might be a tree. For a program doing picture processing or matrix computations, the access graph is likely to resemble a mesh. Alternatively, if we were to completely ignore data, and focus only on the flow of control inherent in the structure of the program, a directed access graph might be a more suitable model.

Typical questions of interest in this model are: (1) How do LRU and FIFO compare on different access graphs? (2) Given the access graph model of a program, what is the performance of LRU on that program (i.e., what is $c_{LRU,k}(G)$)? (3) Given an access graph G, what is $c_k(G)$? can a good paging algorithm be tailor-made given an access graph G? (4) Is there a "universal" algorithm whose

competitiveness is close to $c_k(G)$ on every access graph? (5) What is the power of randomization in the access graph model?

Borodin *et al.* give an in-depth study of LRU. The main results determine $c_{LRU,k}(G)$ for every G and k to within a factor of two (plus additive constant). The technique uses combinatorial properties of small subgraphs of G involving the number of articulation nodes in each subgraph. Another useful way of viewing their tight bounds for LRU is as a characterization of "bad" access graphs for LRU. A refinement of the analysis shows that LRU is optimal among on-line algorithms for the important special case when G is a tree. In recent paper by Chrobak and Noga, [CN97], it was proven that the competitive ratio of LRU is at most that of FIFO on every access graph [CN97].

Borodin *et al.* devise an extremely simple and natural universal algorithm (FAR) and prove the following theorem

Theorem 10. *[BIRS91] For any undirected G and k, $c_{FAR,k}(G) \leq 2 + 4c_k(G)\lceil \log 2k \rceil$.*

This was later improved by Irani, Karlin and Phillips who show

Theorem 11. *[IKP92] For any undirected G and k, $c_{FAR,k}(G) = O(c_k(G))$.*

FAR is a marking algorithm which evicts the unmarked page whose distance in the access graph to to a marked page is maximum. The intuition behind FAR is as follows: it is known that the optimal (off-line) paging algorithm on any sequence is to evict the page which corresponds to the node whose next request occurs furthest in the future. FAR attempts to approximate this behavior by vacating a node that is far from the set of marked pages, and thus likely to be requested far in the future.

The class of *structured program graphs* defined in [BIRS91] are directed access graphs that represent a subset of the stream of instruction references made by a structured program. Borodin *et al.* analyze a simple generalization of FAR, called 2FAR, that is optimal for structured program graphs in which all strongly connected components have at most $k+1$ nodes [BIRS91]. Irani *et al.* introduce a variant of 2FAR, called EVEN, and prove the following theorem:

Theorem 12. *[IKP92] The algorithm EVEN is strongly competitive on the class of structured program graphs. In other words, for any structured program graph G, $c_{k,EVEN}(G) = O(c_k(G))$.*

In a third paper on the access graph model, Fiat and Karlin show a simple randomized universal algorithm with competitive ratio $O(c^R(G))$ for every undirected access graph G. $c^R(G)$ is the best possible competitive ratio achievable by any randomized algorithm on the access graph G.

Fiat and Karlin also consider the multiple pointer case motivated by the observation that in practice there may be multiple flows of control through the virtual memory pages. The different paths through an access graph may represent a multiprogramming environment or operations on multiple data structures within a single application program. If there are m pointers in the access graph, then the next request is to one of the current locations of the pointers or one of their neighbors in the access graph. Denote by $c_m(G)$ the best competitive ratio of any online paging algorithm operating on input sequences generated by m pointers walking on access graph G. Fiat and Karlin show an algorithm whose competitive ratio for any m on every access graph G is $O(c_m(G))$. For the $m = 1$ case, their proof reduces to a simpler proof of the fact that the competitive ratio of FAR is $O(c(G))$.

Fiat and Rosen in [FR97] consider several variations of FAR and evaluate then empirically in relation to FAR and LRU. Their algorithms are truly online in that they build the access graph on the fly instead

of as a preprocessing step. The edges in the graph have weights which are decreased whenever the edge is traversed. They also add a notion of "forgetfulness" in that the edge weights are periodically increased so that edges which have not been recently traversed in the sequence will have a higher relative weight. This allows for the graph to adjust dynamically to the particular phase of the program execution. Their algorithms empirically outperform LRU on most of their traces which are generated from a variety of applications. They also often outperform the version of FAR which was originally proposed with the access graph model.

Open Problem 13. One of the major drawbacks of the universal algorithms for access graphs is that they require a considerable amount of time to determine which page to evict on a fault. Is there an efficient way to perform some of this computation in a preprocessing step instead of as the program is executing? One idea is to embed paging instructions in the code of the program telling the operating system to evict a certain page or switch to a particular page replacement policy. Is there an efficient algorithm which can preprocess a program and determine a set of effective instructions to insert into the code of the program?

5.2 Probabilistic Analysis

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

General Distributions. Lund, Phillips and Reingold consider probabilistic analysis of paging in the context of designing interfaces between IP networks and connection-oriented networks [LPR94]. Although their motivation is different from page replacement policies, the abstract formulation of the problem turns out to be the same. Lund *et al.* consider more general distributions over input sequences than those generated by markov chains. Fortunately, their algorithm requires only limited information about the request sequence. At each point in the request sequence, the algorithm needs to know for every pair of pages in memory (p, q), the probability that the next request to p occurs before the next request to q. We will denote this probability by $w(p, q)$. They prove that for any set of k pages in memory at any point in time, there is a *dominating distribution* over the pages in memory. A dominating distribution has the property that if p is chosen according the distribution, then for each page q in fast memory,

$$E[w(p, q)] \leq \frac{1}{2}.$$

They use this fact to show that if the evicted page is always chosen according to a dominating distribution, then the expected fault rate is at most a factor of 4 times the cost of the optimal online algorithm.

Open Problem 14. Lund, Phillips and Reingold note that the dominating distribution over k pages can be computed by solving a linear program over k variables. Suppose that the distribution over input sequences is generated by a markov chain. Thus, for a given set of k pages, the dominating distribution is static. On every fault, the contents of the fast memory is altered by one page. Is there a way to compute the new dominating distribution from the previous one more efficiently than recomputing the entire linear programming problem?

5.3 Diffuse Adversaries

In the same spirit of restricting the input sequence to the online algorithm, Koutsoupias and Papadimitriou define *the diffuse adversary model* where the input is generated by a distribution [KP94]. Although the online algorithm does not know the exact distribution, it does know that it is chosen from a class Δ of distributions. The adversary can pick the worst $\mathcal{D} \in \Delta$ for an online algorithm A. Once this choice is made, the cost of A on input σ is compared to the optimal offline cost on σ, where σ is picked according to $\mathcal{D}$. The competitive ratio of A is

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

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

Koutsoupias and Papadimitriou illustrate their new measure by defining the class of distributions Δ_ϵ which is the set of all distributions such that for any possible prefeix ρ and any page a, the probability that the next request is a given that the sequence seen so far is ρ, is at most ϵ. The smallest ϵ can be is $1/n$, where n is the total number of pages, in which case the only distribution in the class is the one which generates every sequence uniformly at random. The largest ϵ can be is 1 in which case every distribution is contained in Δ_ϵ. They prove that for any ϵ, LRU is the the algorithm which achieves the optimal competitive ratio against a diffuse adversary which chooses from the class Δ_ϵ. Furthermore, they show that for $k = 2$, the optimal competitive ratio is in between $1+\sqrt{\epsilon}/2$ and $1+2\sqrt{\epsilon}$. For larger k, they obtain a description of (although not a closed form for) the optimal competitive ratio as a function of ϵ.

5.4 Comparative Ratio

In another variation of the competitive ratio, called the *comparative ratio*, Koutsoupias and Papadimitriou pit the online algorithm against less powerful adversaries than the optimal offline algorithm [KP94]. Consider two classes of algorithms $\mathcal{A}$ and $\mathcal{B}$. Typically, $\mathcal{A} \subseteq \mathcal{B}$. $\mathcal{B}$ may have a more powerful information regime or more computational resources. The comparative ratio of $\mathcal{A}$ and $\mathcal{B}$ is

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

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

One appealing feature of comparative analysis is that it allows one to address the anomaly observed in competitive analysis that a fixed amount of lookahead does not give any advantage. The idea is that if an online algorithm competes against a class of algorithms with limited lookahead instead of an omniscient algorithm, it does gain some advantage with additional lookahead. The authors show that $c(\mathcal{L}_0, \mathcal{L}_l) = l + 1$ for paging, where $\mathcal{L}_i$ is the class of all online algorithms with lookahead i.

Open Problem 15. It would be nice to be able to say that an online algorithm which plays against an adversary with limited lookahead does better with more lookahead (i.e., for $i < j < l$, $c(\mathcal{L}_i, \mathcal{L}_l) > c(\mathcal{L}_j, \mathcal{L}_l)$). What are tight bounds for $c(\mathcal{L}_i, \mathcal{L}_l)$ as a function of i and l?

5.5 Loose Competitiveness

Young's definition of *loose competitiveness* [You91b] is based on the observation that lower bounds for online algorithms often use an adversary which carefully tailors the request sequence to the particular "hardware configuration" of the algorithm. In the case of paging, the hardware configuration is simply k, the number of pages that can fit in fast memory. Thus, we are evaluating the algorithm on its most damaging request sequence, designed to take advantage of its specific circumstances. In evaluating the loose competitiveness of an algorithm, we require that the adversary concoct a sequence which is bad for an algorithm under most circumstances (i.e., most values of k). The idea was first introduced in [You91b], however, the following definition is taken from [You97] which has generalized and improved bounds.

Definition 16. Algorithm A is (ϵ, δ)-loosely c-competitive if for any request sequence σ, and any n, at least $(1-\delta)n$ of the values of $k \in \{1, 2, \ldots, n\}$ satisfy

$$\mathtt{cost}_{A,k}(\sigma) \leq \max\{c \cdot \mathtt{cost}_{OPT}(\sigma), \epsilon|\sigma|\}.$$

There are actually two differences between this definition and traditional competitiveness. The first is that the algorithm only has to be competitive on most values of k. The second is that the algorithm does not have to be competitive if its fault rate is small (at most ϵ times the length of the sequence).

Using loose-competitiveness, the competitive ratio decreases dramatically:

Theorem 17. *[You91b] Every* $\left(\frac{k_{on}}{k_{on}-k_{off}+1}\right)$*-competitive algorithm is* (ϵ, δ)*-loosely competitive for any* $0 < \epsilon, \delta < 1$ *and*

$$c = e\frac{1}{\delta}\lceil \ln \frac{1}{\epsilon} \rceil.$$

k_{on} *is the number of pages that can fit in the online algorithm's fasy memory and* k_{off} *is the number of pages that can fit in the optimal offline algorithm's fast memory.*

Theorem 18. *[You91b] Every* $O\left(\ln \frac{k_{on}}{k_{on}-k_{off}}\right)$*-competitive algorithm is* (ϵ, δ)*-loosely competitive for any* $0 < \epsilon, \delta < 1$ *and*

$$c = O(1 + \ln \frac{1}{\delta} + \ln\ln \frac{1}{\epsilon}).$$

These theorems were actually proven for a generalization of the paging problem discussed in Section 6.

Open Problem 19. The idea of loose competitiveness has the potential to shed light on many other problems in the area of online algorithms. In particular, for many scheduling problems, lower bounds are proven by concocting job arrival sequences which are bad for the particular number of processors (or more generally, hardware configuration) available to the algorithm. Are these lower bounds still attainable under the model of loose competitiveness?

5.6 Measuring total memory access time

Torng [Tor95] adds a new twist to the paging problem by making the argument that the correct cost for paging algorithms should be total memory access time and not simply the number of page faults. This adds an extra parameter to the paging problem which is the additional time to access a page in slow memory versus the time to access a page in fast memory. That is, if the access time for fast memory is 1, then the access time for slow memory is $1+s$. If an algorithm A incurs f faults in a sequence of n requests, then the cost of A on that sequence is

$$\texttt{cost}_A = (n-f)\cdot 1 + f\cdot(s+1).$$

Using this cost model, Torng re-examines the competitive ratio of paging algorithms. Thus, we are still interested in bounding $\texttt{cost}_{k,A}(\sigma)/\texttt{cost}_{k,OPT}(\sigma)$, but $\texttt{cost}_{k,A}$ and $\texttt{cost}_{k,OPT}$ are defined to be the total memory access time instead of the number of faults. Note that the previous analysis assumes that $s=\infty$ so that hits do not contribute to memory access time. Using this model, Torng generalizes the previous bounds for the standard algorithms.

Theorem 20. *[Tor95] Any marking algorithm achieves a competitive ratio of*

$$\frac{k(s+1)}{k+s} \approx \min\{k, s+1\}.$$

Theorem 21. *[Tor95] The randomized marking algorithm achieves a competitive ratio of* $\min\{2H_s, 2H_k\}$ *against an oblivious adversary.*

Torng also proves a lower bound showing that Theorem 20 is tight. Probably more interesting is that there is a very natural notion of locality of reference with this model. Define $L(\sigma,k)$ to be the average phase length in the request sequence σ with k slots of fast memory. Phases here are defined in an identical manner as they were in the definition of marking algorithms. Naturally, the larger $L(\sigma,k)$ is, the more locality of reference exhibited by the sequence. For sequences with large values for $L(\sigma,k)$, some improvement can be gained.

Theorem 22. *[Tor95] If the adversary is restricted to sequences where* $L(\sigma,k) > as$*, then any marking algorithm achieves a competitive ratio of* $(1+\frac{k-1}{a+1})$.

Theorem 23. *[Tor95] If the adversary is restricted to sequences where* $L(\sigma,k) > kas$*, where* $a < 1/2$*, then the competitive ratio of the Randomized Marking Algorithm is at most* $2+2(H_k - H_{2ak}) \approx 2(1+\ln\frac{1}{2a})$*. If* $a \geq 1/2$*, then the competitive ratio is* $(1+\frac{1}{2a})$.

Another appealing feature of this model is that allowing the online algorithm some lookahead does give it some advantage. For example, the l-stuttered version of a sequence gives both the online algorithm and offline algorithm additional hits which contributes an equal additive cost to the online and offline algorithms and serves to reduce the competitive ratio.

Consider the following natural generalization of LRU with lookahead: on a fault, evict the page not in the lookahead buffer which was accessed the least recently. If all pages in the fast memory are in the lookahead buffer, then evict the page whose next reference is farthest in the future. We denote this version of LRU with lookahead l by $LRU(l)$. $LRU(l)$ proves difficult to analyze in Torng's model, so he considers a simple variant called $LRU(l,k)$ which behaves exactly like $LRU(l)$, except that it restricts its lookahead to the requests in the current phase.

Theorem 24. *[Tor95] The competitive ratio of $LRU(l,k)$ is at most*

$$\min\left\{2+\sqrt{\frac{2ks}{l}}, s+1\right\},$$

when $l < sk$. If $l \geq sk$, the competitive ratio is at most 2.

5.7 Lookahead

Various refinements of the competitive ratio which we have seen so far have resulted in a model in which some limited amount of lookahead does give an online algorithm some advantage. The issue of lookahead has also been addressed in a more direct manner by simply changing the definition of lookahead l.

Strong lookahead. Albers defines the notion of *strong lookahead l* which means that the algorithm can see the minimal prefix of the remaining sequence which contains requests to l distinct pages [Alb93]. She also considers the algorithm $LRU(l)$ defined in Section 5.6 and proves the following theorem:

Theorem 25. *[Alb93] The competitive ratio of $LRU(l)$ under the strong lookahead model is exactly $k-l$ when $l \leq k-2$.*

The following theorem establishes that this upper bound is tight:

Theorem 26. *[Alb93] No deterministic algorithm can obtain a competitive ratio of less than $k-l$ with strong lookahead l when $l \leq k-2$.*

Albers also examines a variant of her model in which requests arrive in blocks of l distinct requests. The motivation behind this model is that in practice a paging algorithm will be allowed some amount of lookahead because the arrival of requests is bursty. This is more likely to result in batch arrivals than a steady stream in which there is always a backlog of l unserved requests. The algorithm $LRU(l)-blocked$ behaves exactly like $LRU(l)$ except that it limits its lookahead to the requests in the current block. Albers proves the following theorem.

Theorem 27. *[Alb93] LRU(l)-blocked is $(k-l+1)$-competitive when $l \leq k-2$.*

Albers also shows that a variant of the Randomized Marking Algorithm is $(2H_{k-l})$-competitive with strong lookahead l. She proves that this bound is tight to within a factor of 2.

Resource bounded lookahead. Young defines the notion of *resource-bounded lookahead l* in which the algorithm can see the maximal prefix of future requests for which the algorithm must incur l faults [You91a]. Young proves a generalization of the traditional competitive ratio by allowing the online

algorithm to potentially have more fast memory than the offline algorithm. Let k_{on} be the number of pages that can fit into the online algorithm's fast memory. Let k_{off} be the number of pages that can fit into the offline algorithm's fast memory. Let $a = k_{on} - k_{off}$.

Theorem 28. *[You91a] There is a marking-like algorithm which achieves a competitive ratio of at most*

$$\max\left\{2\frac{k_{on}+a+l}{a+l+1}, 2\right\},$$

with resource-bounded lookahead l.

Interestingly, the above result shows a direct trade-off between extra memory and lookahead. Young shows a lower bound of $\frac{k_{on}+a+l}{a+l+1}$ for the competitive ratio of any online deterministic algorithm with resource bounded lookahead. He then shows a variant of Randomized Marking which achieves a competitive ratio of $2(\ln\frac{k}{l}+1)$ and gives a lower bound of $\ln\frac{k+l}{l} - \ln\ln\frac{k+l}{l} - \frac{2}{l}$ for any randomized paging algorithm with resource bounded lookahead l.

Breslauer improves Young's upper bound by showing that $LRU(l)$ achieves a competitive ratio of exactly $\frac{k_{on}+a+l}{a+l+1}$ with resource bounded lookahead l [Bre96]. He also defines an alternate definition of lookahead which he calls *natural lookahead*. In this model, the algorithm is allowed to see the maximal prefix of future requests which contain l distinct pages not in the algorithm's fast memory. Breslauer proves that $LRU(l)$ achieves a competitive ratio of $\frac{k_{on}+a+l}{a+l+1}$ and that this is the best competitive ratio that can be achieved under his model of lookahead.

6 Variations on the Paging Problem

This survey has mainly addressed algorithms for the basic online page replacement problem and different measures which can be used to evaluate them. There have also been variations on the basic page replacement problem itself which have been examined using competitive analysis. In this section, we give a very brief survey of some of those problems.

6.1 Weighted Caching

In some cases, the cost of bringing a page into memory may vary, even though the number of different pages which the cache can hold remains fixed. That is, each page has a fixed weight which denotes the cost of bringing that page into memory. Since this problem is a generalization of paging, the lower bound of k on the competitive ratio of any deterministic paging algorithm holds for weighted caching. The first matching upper bound was proven by Chrobak, Karloff, Payne and Vishwanathan in [CKPV91] who proved that an extension of FIFO called BALANCE is k-competitive. Neal Young, then showed that a whole class of algorithms called Greedy-Dual are $k_{on}/(k_{on} - k_{off} + 1)$-competitive [You91b], where k_{on} and k_{off} are the sizes of the online and offline caches respectively. This class of algorithms include BALANCE as well as a generalization of LRU.

6.2 Multi-Size Pages

The paging problem in which pages have varying sizes has been examined by Irani in [Ira97]. This problem is motivated by cache management for World Wide Web documents where documents held in

the cache can vary dramatically in size, depending largely on the type of information they contain (e.g. text, video, audio). Again, since this is a generalization of paging, the lower bound of k still applies for the competitive ratio of any deterministic online algorithm. It is relatively straight-forward to prove that the natural extension of LRU is k-competitive in this case. k is the ratio of the size of the cache to the size of the smallest page.

It is also proven that a version of Neal Young's Greedy Dual algorithm which is a generalization of LRU is k-competitive for the multi-size weighted cache problem where pages have varying sizes as well as a weight associated with each page [CI97]. Weighted caching is especially pertinent in the multi-size paging problem since the time to cache a web document can vary greatly depending on the location of the document and network traffic. Subsequently, Young proved that the entire range of algorithms covered by Greedy-Dual are k-competitive [You97]. In fact, he proved it is $k_{on}/(k_{on}-k_{off}+1)$-competitive [You91b], where k_{on} and k_{off} are the sizes of the online and offline caches respectively. The proof is a generalization and simplification of the proof in [You91b]. He also has bounds for the loose competitiveness which can be achieved for this problem which improve, generalize and simplify the results from [You91b]. (See Section 5.5 for a definition of loose competitiveness).

Interestingly, when the pages have varying sizes, the optimal algorithm is not nearly as straightforward as in the uniform-size case. Irani considers two cost models. In the first, (the FAULT model), the cost to bring a page into the cache is constant, regardless of its size. In the second, (the BIT model), the cost to bring a page into memory is proportional to its size. Offline algorithms for both cost models are shown which obtain approximation factors of $O(\log k)$. Randomized online algorithms for both cost models are shown which are $O(\log^2 k)$-competitive. In addition, if the input sequence is generated by a known distribution, algorithms for both cost models are given whose expected cost is within a factor of $O(\log^2 k)$ of any other online algorithm.

6.3 Paging for Shared Caches

Consider the following problem. The page replacement algorithm is given a sequence of requests which is an interleaving of p different request sequences are known in advance. The online aspect of the problem is that the paging algorithm does not know how the request sequences will be interleaved until it sees each request of the sequence in an online manner. The problem models the situation where there are p applications which are simultaneously sharing a cache. Each application uses knowledge about its own request sequence to enable the paging algorithm to make optimal paging decisions about its own request sequence. However, it is unknown how fast each application will progress in its request sequence relative to the other applications. The problem was originally studied by Cao, Felten and Li who showed that a competitive ratio of $2p + 2$ can be achieved, where p is the number of processes [PC94]. Barve, Grove and Vitter use randomization to improve this bound to $2H_{p-1} + 2$, where H_p is the p^{th} harmonic number [RB95]. They also show a lower bound of H_{p-1} for the competitive ratio of any randomized online algorithm.

References

[ACN96] D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. In *Proc. 4th European Symposium on Algorithms*, LNCS 1136, pages 419–430. Springer, 1996.

[Alb93] S. Albers. The influence of lookahead in competitive paging algorithms. In *First Annual European Symposium on Algorithms*, pages 1–12, 1993.

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

[Bel66] L.A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.

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

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

[Bre96] D. Breslauer. On competitive on-line paging with lookahead. In *13th Annual Symposium on Theoretical Aspects of Computer Science.*, pages 593–603, 1996.

[CI97] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. Technical report, 1343, Dept. of Computer Sciences, University of Wisconsin -Madison, May 1997. A shorter version appears in the 2nd Web Caching Workshop, Bould er, Colorado, June 1997.

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

[CN97] M. Chrobak and J. Noga. LRU is better than FIFO. Submitted for publication, 1997.

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

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

[FR97] Amos Fiat and Ziv Rosen. Experimental studies of access graph based heuristics: Beating the lru standard? In *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 63–72, 1997.

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

[Ira97] Sandy Irani. Page replacement with multi-size pages and applications to web caching. In *Proc. 29rd ACM Symposium on Theory of Computing*, pages 701–710, 1997.

[KMRS88] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[KP94] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. In *Proc. 25th Symposium on Foundations of Computer Science*, pages 394–400, 1994.

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

[LPR94] C. Lund, S. Phillips, and N. Reingold. Ip over connection-oriented networks and distributional paging. In *35th IEEE Symposium on Foundations of Computer Science*, pages 424–435, 1994.

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

[PC94] K. Li P. Cao, E.W. Felten. Application-controlled file caching policies. In *Proc. for the Summer USENIX Conference*, 1994.

[RB95] J.S. Vitter R.D. Barve, E.F. Grove. Application-controlled paging for a shared cache. In *Proc. for the 36th Annual Symposium on Foundations of Computer Science*, pages 204–213, 1995.

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

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

[Tor95] E. Torng. A unified analysis of paging and caching. In *36th IEEE Symposium on Foundations of Computer Science*, pages 194–203, 1995.

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

[You91a] N. Young. Dual-guided on-line weighted caching and matching algorithms. Technical Report Tech. Report CS-TR-348-91, Comp. Sci. Dept., Princeton University, 1991.

[You91b] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 1991. To appear. Rewritten version of "On-line caching as cache size varies", in The 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, 241-250, 1991.

[You97] N. Young. Online file caching. Submitted for publication., 1997.