Markov Paging *

Anna R. Karlin † Steven J. Phillips‡ Prabhakar Raghavan§

April 22, 1997

Abstract

This paper considers the problem of paging under the assumption that the sequence of pages accessed is generated by a Markov chain. We use this model to study the fault-rate of paging algorithms. We first draw on the theory of Markov decision processes to characterize the paging algorithm that achieves optimal fault-rate on any Markov chain. Next, we address the problem of devising a paging strategy with low fault-rate for a given Markov chain. We show that a number of intuitive approaches fail. Our main result is a polynomial-time procedure that, on any Markov chain, will give a paging algorithm with fault-rate at most a constant times optimal. Our techniques also show that some algorithms that do poorly in practice fail in the Markov setting, despite known (good) performance guarantees when the requests are generated independently from a probability distribution.

1 Introduction

This paper considers the problem of paging in a two-level store under the assumption that the sequence of pages accessed (henceforth *reference string*) is generated by a Markov chain. Each page of virtual memory is represented by a state (or node) of the Markov chain M, whose transition probabilities p_{ij} specify the probability that an access to i is immediately followed by an access to j.

Sleator and Tarjan [18] initiated the worst-case study of paging, introducing a style of worst-case analysis that has come to be known as *competitive analysis* [11]. We wish

*A prelimiary version of this paper appeared in the Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science, 1992.

†Department of Computer Science and Engineering, University of Washington, Seattle, WA. karlin@cs.washington.edu

‡AT&T Bell Laboratories, Murray Hill, NJ. Supported by NSF Grant CCR-9010517, and grants from Mitsubishi and OTL. This work was done while at Stanford University and DEC SRC. phillips@research.att.com

§IBM Almaden Research Center, Almaden, CA. pragh@almaden.ibm.com

to address some shortcomings of the traditional adversarial analysis of paging [18]. Borodin *et al.* [2] provided an important step towards bridging the gap between such worst-case analysis and reality by providing a model that captured the essential aspects of locality of reference in the reference string. Here we follow their lead, and assume that the reference string is generated by a Markov chain (thereby removing the adversary's role).

We refer to on-line paging with the reference string generated by such a Markov chain as *Markov paging*. We focus on questions such as: given a Markov chain, what is the best paging algorithm for reference strings generated by it, and how can this algorithm be computed? is there a simple algorithm that performs near-optimally on every Markov chain? can Markov paging explain why some paging algorithms perform poorly in practice?

Some salient features of our work are:

- There is recent interest in the systems community in designing paging algorithms that adapt to the locality characteristics of a program [5, 14]. Thus insights derived from theoretical studies may have an impact on implementations.

- Markov paging, like the access graph model in [2, 10], offers a clean theoretical abstraction for locality of reference in a program. Unlike the models in [2, 10, 11, 18], there is no adversary generating the reference string, much as in a real program. Further, certain simple properties of real programs — such as the fact that a data-dependent loop typically gets executed many times before exiting — can be modeled well.

- Practitioners study the *page-fault rate* for a program and paging algorithm, rather than competitiveness. Page-fault rate has little meaning in an adversarial model, but is eminently suited to a probabilistic model. In Markov paging, there is (as we shall see) a precise meaning to the page-fault rate of an algorithm, as well as the best page-fault rate achievable on the Markov chain. Note that if each request is drawn independently from a probability distribution, the problem of devising the paging algorithm with lowest fault-rate has an easy solution [9].

- Markov paging enables us to provide a mathematical basis for the poor performance of certain paging algorithms (such as *random replacement* and *frequency count*). For example, the *Frequency Count* algorithm can be bad in the Markov paging setting — whereas one might think that on a probabilistic input this algorithm would perform well.

- Somewhat surprisingly, we find that several plausible approaches to devising paging algorithms will fail for Markov paging. For example, all *marking algorithms*, provably best in an adversarial model, suffer from page-fault rates that are far from optimal in Markov paging. This includes probabilistic versions of the FAR algorithm of [2, 10]. We also find that natural approximations of the optimal off-line paging algorithm MIN are far from optimal in Markov paging.

- We present an on-line paging algorithm that is computable in polynomial time and achieves a fault-rate within a constant times the best possible, on every Markov chain.

While our approach enables us to move away from competitive analysis towards a performance measure of greater interest to practitioners, two practical limitations of our work should be noted. Firstly, in practical settings, page reference sequences may not be accurately modeled by a Markov chain in which pages are equated with chain states, since each page contains many memory locations. For example, a single page may have a number of basic blocks of code, and the next page to be referenced will depend on which basic block the program is currently in. Secondly, while the algorithm we present is polynomial-time computable, it requires the estimation and storage of commute times between pairs of pages of memory. The time and memory involved are considerable, so a simpler algorithm may be more applicable in practice.

1.1 Model

We have n pages that may be either in memory or on disk. Only k pages may be in memory at any time. An *on-line paging algorithm* is presented with a sequence of page requests. If the page currently requested is in memory, (a *hit*), no cost is incurred. However, if the page requested is not in memory (*a fault*), it must be fetched from the disk into memory for a unit cost. Furthermore, if there are already k pages in memory, one of the pages in memory must be evicted to make room for the requested page. The decision of which page to evict must be made by the paging algorithm without detailed knowledge of future requests.

We now augment this basic model with a Markov source of references. Let M be an irreducible Markov chain whose state space is the set of n pages, and let A be an on-line paging algorithm. We define $f_A(M, k)$ to be the long-term frequency of faults incurred by A running on page request sequences generated by M and using a memory that can hold k pages. This is formalized in Section 2. It is also shown in Section 2 that there is an *optimal fault-rate* for every Markov chain M, which we denote by $f^*(M, k)$.

Note that we are not concerned here with the problem of inferring the Markov chain by observation of the page request sequence. We assume that the transition probabilities are already known. For example, they could be approximated to arbitrarily accuracy by sampling a large enough initial prefix of the reference string.[1]

1.2 Related previous work

The underpinnings of competitive paging were laid in [11]. The literature of computer performance modeling and analysis contains related work, both theoretical and

[1] For this reason, the results we obtain assuming the Markov chain is known also hold in a limiting sense in the case where the Markov chain is not known.

empirical. Denning [6] (and references therein) developed the *working set* model of program behavior for capturing locality of reference. Spirn[19] gives a comprehensive survey of models for program behavior. Franaszek and Wagner [9] studied a model in which every request is drawn independently from the same probability distribution. Shedler and Tung [17] and Lewis and Shedler [13] study paging in a Markov chain whose states represent *LRU stack distances*, a model convenient for studying the *least recently used* (LRU) paging algorithm. Denning and Spirn showed empirically that in order for a first-order Markov model to reasonably approximate real program behavior, it is necessary to separate data and instruction reference streams. Borodin *et al.* [2] and Irani *et al.* [10] have studied locality of reference in paging, but with an adversarial model.

Irani, Karp, Kearns and Luby (private communication, 1991) have studied other on-line problems when requests are independently drawn from a probability distribution. Vitter and Krishnan [20] consider the problem of prefetching into a cache when the reference string is generated by a Markov source (or mth order Markov source). Under the assumption that there is always sufficient time to prefetch as many pages as wanted, Vitter and Krishnan show that data compression techniques can be used to obtain algorithms with optimal limiting page fault rates.

1.3 Guided tour of the paper

Section 2 draws on the theory of Markov decision processes to characterize paging algorithms that achieve optimal fault-rate on any Markov chain. Theorem 1 shows that there is a memoryless, deterministic optimal on-line algorithm. Theorem 2 shows that there is a linear program that determines the optimal on-line algorithm, but this algorithm may have running time exponential in k. Section 3 shows that a number of plausible approaches for designing an efficient and provably good Markov paging algorithm will fail. Our main result, in Section 4, gives an efficiently computable paging algorithm whose fault-rate is within a constant factor of the best possible on every Markov chain.

2 Markov decision theory and optimal paging

We begin by studying the relationship between page replacement policies and Markov decision theory [7].

A *Markov decision process* can be described as follows. Consider a discrete time Markov chain, whose state at time t is Y_t. After each observation of the system, one of a set of possible actions is taken. Say that K_i is the set of actions possible when the system is in state i. Let A_t be the action taken at time t. A (possibly randomized) policy R is a set of functions $D_a(H_{t-1}, Y_t)$, where a is an action in K_{Y_t} meaning that if H_{t-1} is the history of states and actions up to time $t-1$, and Y_t is the state at time t, then the probability action a is taken at time t is $D_a(H_{t-1}, Y_t)$. The actions can

be such that they change the state of the system. We define this precisely by saying that $q_{ij}(a)$ is the probability of the system being in state j at the next instant, given that the system is in state i and action a is taken, i.e.

$$Pr(Y_{t+1} = j \mid H_{t-1}, Y_t = i, A_t = a) = q_{ij}(a).$$

An additional set of parameters associated with a Markov decision process are costs: when the chain is in state i and action a is taken, a known cost w_{ia} is incurred.

Let $S_{R,T}$ be the expected cost of operating a system up to time T using policy R. ($S_{R,T} = \sum_{0 \le t \le T} \sum_j \sum_a P_R(Y_t = j, A_t = a) w_{ja}$.)

A standard cost criterion in Markov decision theory is to minimize the expected average cost per unit time, i.e. to find a policy R to minimize

$$\limsup_{T \to \infty} \frac{S_{R,T}}{T}. \tag{1}$$

We formalize the notion of expected faulted rate $f_A(M, k)$ of a paging algorithm A running on request sequences generated by the Markov chain M as follows. Let S be the state space of M (remember S is just the set of pages). Define the following augmented Markov chain whose state space is S'. A state in S' has two components: r and I. Here r is the most recent request, and I is a subset of S of size k representing the set of pages that are in memory immediately before servicing the request r.

When the system is in the state (r, I), and $r \notin I$, then there are k actions that can be taken by the on-line algorithm: for each $x \in I$, x can be evicted. The effect of the actions is described as follows. Suppose that p_{ij} are the transition probabilities of the underlying Markov chain. For each $r' \in S$,

$$Pr(Y_{t+1} = (r', I') \mid H_t, Y_t = (r, I), A_t = x) = p_{rr'}$$

where $I' = I \setminus \{x\} \cup \{r\}$.

When the system is in state (r, I), and $r \in I$, only the trivial action can be taken. The algorithm A is simply a policy for this Markov decision problem.

We set

$$w_{(r,I),a} = \begin{cases} 1 & \text{if } r \notin I \\ 0 & \text{otherwise} \end{cases}$$

and define the expected fault rate $f_A(M, k)$ to be the expected average cost per unit time, as defined in (1) above.

The following two theorems instantiate basic results in Markov decision theory ([7], Chapter 3, Theorem 2, and Chapter 6, Lemma 4) to the case of Markov paging.

Theorem 1 *For a given Markov chain there is an on-line page replacement policy that has minimum fault rate and is memoryless, time-invariant and deterministic.*

Thus for the optimum algorithm, the decision of which page to evict on a fault depends only on the current request and the k pages in memory, and is deterministic.

Theorem 2 *The problem of computing the optimal on-line paging strategy on Markov chain M can be expressed as a linear programming problem in $n\binom{n}{k}$ variables.*

Note that this linear programming formulation gives us an algorithm for computing the optimal paging strategy on M whose running time may be exponential in k. We know of no technique to improve this upper bound, and this leads us to the study of efficiently computable strategies that approximate $f^*(M)$ well on every M. Note that when k is close to zero or to n, the linear program gives an efficient algorithm for computing the optimal on-line paging strategy. In fact, it is particularly instructive to consider the case $n = k + 1$, both for its own sake and because it has traditionally given good insights in paging [2, 8].

Theorem 3 *When $n = k + 1$, for every M, there is an efficiently computable deterministic paging strategy that evicts only one of two fixed nodes ($k - 1$ pages are never evicted) whose fault-rate is at most $2f^*(M, k)$.*

Proof: Let G be the complete directed graph on n states, weighted by expected hitting times in M (the weight of edge (i, j) being $H_{i,j}$). For any memoryless, time-invariant and deterministic algorithm, in any execution the "hole" (the page that isn't in the memory) must eventually get into some cycle in G. Once in a cycle, the expected fault rate is just the reciprocal of the average edge weight around the cycle. Thus the optimal paging algorithm must use a *max mean cycle*. (See [12] for an algorithm to find a max mean cycle.) It follows that there is a cycle of length 2 that has page fault rate at most twice optimal: just pick a pair (i, j) of adjacent vertices on a max mean cycle, such that $H_{i,j}$ is at least the mean. Note that in the case that the chain is reversible, the mean hitting time round a cycle is independent of direction around the cycle, so there must be a cycle of length 2 with optimal page fault rate. □

The following theorem compares the optimal online fault rate to the optimal off-line fault rate.

Theorem 4 *The expected page fault rate of OPT, the optimal online algorithm, is at most $O(\log k)$ times the expected page fault rate of the optimal off-line algorithm (that sees the entire request string output by the Markov chain, then serves it optimally).*

This follows from the existence of a $O(\log k)$-competitive randomized paging algorithm [8], combined with the von Neumann minimax principle [15]. The bound is the best possible: when the Markov chain is the random walk on the $k+1$ node complete graph K_{k+1}, any on-line algorithm has fault rate $\Omega(\log k)$ times that of the optimal off-line algorithm [15].

3 Negative results

We begin this section by establishing that two algorithms that have been proposed and found to fare poorly in practice are far from optimal in Markov paging. Following this, we begin our quest for a simple, efficiently computable paging algorithm that has page-fault rate within a constant multiple of the best possible, on every Markov chain. We show in this section that a number of intuitively "obvious" algorithms for Markov paging fail to achieve this goal, and pave the way for our optimal algorithm in Section 4. In all our negative results, $n = k + 1$; however, this is for simplicity of presentation only and can be generalized.

Remember that the hitting time $h_{x,y}$ from a state x in a Markov chain to another state y is defined to be the expected number of steps taken to first reach y starting from x, while the commute time $C_{x,y}$ is the expected number of steps to reach y from x and then return to x. We mention also that the commute time is related to the resistance [4], and thus the hitting times in most examples used here can be easily computed by the electrically literate reader.

The *Random Replacement* (*RR*) paging algorithm will, on a fault, evict a random page in memory. Although optimal in the competitive setting against an adaptive on-line adversary [16], it performs relatively poorly in practice.

Theorem 5 *There is a constant c such that for every $k > 2$, there is a Markov chain M for which $f_{RR}(M,k) \geq ckf^*(M,k)$.*

Definition 1 *For positive integers a, b, the* lollipop graph $L(a,b)$ *is formed by attaching one end of a path through b nodes to a complete graph K_a on a nodes.*

Proof of Theorem 5: Let $n = k+1$, and let M be the Markov chain representing the simple random walk on $L(k-1,2)$. The maximum hitting time is $\Omega(k^2)$, say between nodes x and y. The algorithm that alternates its hole between x and y (as in Theorem 3) thus has a fault rate of $O(1/k^2)$, since the expected fault rate is $1/C_{x,y}$, by a result of renewal theory (see for example [7], page 147). Therefore $f^*(M,k)$ is $O(1/k^2)$. On the other hand, the expected time between faults incurred by *RR* is the average over all pairs of nodes x, y in the graph of the hitting time from x to y. This average is $O(k)$.

□

The *Frequency Count* (*FC*) algorithm maintains, for each of the n pages, a count of the number of times that page has been accessed. On a fault, it evicts the the page in memory that has been accessed least often. When every request is drawn independently from a probability distribution, *FC* converges to the optimal algorithm. However, it performs poorly in practice, since it ignores locality of reference. This is reflected by the fact that it is far from optimal in Markov paging (Theorem 6).

Theorem 6 *There is a constant c such that for every $k > 2$, there is a Markov chain M for which $f_{FC}(M,k) \geq ckf^*(M,k)$.*

Definition 2 *For positive integers a, b, the* forked lollipop graph $FL(a,b)$ *is formed from $L(a, b-1)$ by connecting two new nodes to the external end-node of the $(b-1)$-path.*

Proof of Theorem 6: Let $n = k+1$, and let M be the Markov chain representing the simple random walk on $FL(k/2, k/2)$. The two prongs have lowest stationary probability on this chain, so will eventually have the smallest frequency count in any request sequence. Thus FC will eventually alternate the "hole" (the node not in memory) between these two nodes. The expected time between faults incurred by FC is $O(k^2)$ (which is the expected hitting time from one prong to the other), whereas $f^*(M,k)$ is $O(1/k^3)$ (obtained by alternating the hole between one prong and a node in the clique). □

Next, we show that many algorithms that intuitively should perform well in Markov paging will in fact perform poorly on some Markov chains.

A class of on-line algorithms that one may expect to perform well are *marking algorithms*. Marking algorithms use a notion of *phases*. A *new page* is one that wasn't requested in the previous phase. A new phase begins with a request to a new page. When a page is requested, it is marked. As soon as k distinct pages are marked, the phase ends, and all pages become unmarked. A marking algorithm has the property that it never evicts a marked page.

It has been shown that there are optimal marking algorithms under competitive analysis for paging with locality of reference [2], as well as randomized paging algorithms (arbitrary request sequences) [8, 10]. Therefore, one might think that the same would hold when reference strings are generated from a Markov chain. The following theorem shows that our search for a good algorithm should exclude marking algorithms. The lower bound in the theorem cannot be improved, since there is a marking algorithm A which is $O(\log k)$-competitive [8], and therefore there is a constant c such that $f_A(M,k) \leq c(\log k) f^*(M,k)$ for any Markov chain M.

Theorem 7 *There is a constant c such that for any $k > 2$, there is a Markov chain M for which $f_A(M,k) \geq c(\log k) f^*(M,k)$ for any marking algorithm A.*

Proof: Let M be the Markov chain corresponding to a simple random walk on the lollipop graph $L(k/2+1, k/2)$. Let y denote the node at the end of the path. A phase starts with a request at some vertex of the graph, and ends just before all nodes in the graph have been requested. The last node to be requested starts a new phase. The important property of this Markov chain that defeats any marking algorithm is that once a node in the clique is requested, with high probability all the nodes in the clique will be requested before the node y is requested. Hence on average, almost half the phases will begin on y. By a standard argument, if a phase begins on y, any marking algorithm will incur an expected $\Omega(\log k)$ faults in the clique before the phase ends. Thus any marking algorithm will incur $\Omega(\log k)$ faults per phase on average. On the other hand, the on-line algorithm that alternately evicts a node in the clique and y will incur an average of 1 fault per phase. □

The optimal offline algorithm for any reference string is commonly called MIN. MIN always replaces the page that will be requested furthest in the future. We now consider various on-line algorithms that mimic MIN on a Markov chain. The *maximum hitting time* (MHT) algorithm replaces, on a fault, that page in memory for which the *expected time* to the next request is the largest. Indeed, when the requests are drawn independently from a probability distribution, this algorithm performs well [9]; again, the locality of reference captured by Markov paging proves to be the undoing of this algorithm.

Theorem 8 *For every $k > 10$, there is a Markov chain M on $k+1$ nodes and a constant c such that $f_{MHT}(M,k) \geq ckf^*(M,k)$.*

Proof: Consider the forked lollipop $G = FL(2k/3, k/3)$. Suppose that the Markov chain generating the reference string is the simple random walk on G.

If MHT is run on the reference string generated by this Markov chain, eventually one of the two prongs at the end of the path will be evicted. From either, the maximum hitting time is $4n^2/9$, to the other. On the other hand, the maximum hitting time from a prong to any node in the clique is $2n^2/9 + O(n)$. Thus the page not in memory (the "hole") will thereafter oscillate forever between the two prongs, so that $f_{MHT}(M,k)$ is $\Omega(1/k^2)$. The optimal algorithm will alternately evict a node in the clique and a prong, with fault-rate $O(1/k^3)$. □

Let $LAST$ be the algorithm that on a fault evicts the page that has the highest probability of being the last of the k pages in memory to be requested. An attractive variation on $LAST$ is an algorithm we call Max Rank, (MR) defined as follows. Suppose that at the time of a fault, S is the set of pages in memory. Then there is some permutation on S that describes the order in which these pages will subsequently be seen. For each page $i \in S$, and $1 \leq j \leq k$, let $p_i(j)$ be the probability that page i is the jth page in S that will be seen. Define the expected rank of page i, R_i to be $\sum_j jp_i(j)$. Then MR evicts the page $p \in S$ such that R_p is maximum.

Theorem 9 *There is a constant c such that for any k: (i) there is a Markov chain M such that $f_{LAST}(M,k) \geq ckf^*(M,k)$, (ii) there is a Markov chain M such that $f_{MR}(M,k) \geq ckf^*(M,k)$*

Proof: (i) Consider the Markov chain corresponding to the standard random walk on an undirected $k+1$ node cycle: all nodes are equally likely to be visited last. The following proof of this fact is credited by Broder [3], without further reference, to Avrim Blum, Ernesto Ramos, and Jim Saxe, independently: Consider a point a on the cycle. Let its neighbors be b and c. Before visiting a, the walk will visit first one of its neighbors, say b. Given this fact, the probability that a is last, is the probability that the walk will visit c starting from b before it visits a. This is clearly independent of the position of a.

Consequently, on a fault, $LAST$ might always evict the neighbor of the faulted node. Therefore, $LAST$ can have expected fault rate $O(1/k)$ (since the expected

time to hit a neighbor in a cycle is $O(k)$). On the other hand, the algorithm which alternates the hole between two antipodal points faults has a expected fault rate of $O(1/k^2)$.

(ii) Consider a directed cycle on k nodes, with an extra node z that has edges to and from two antipodal nodes x and y. Let $p_{xz} = p_{yz} = p = 5/k$, and let $p_{zx} = p_{zy} = 1/2$. We show that starting from any node w on the cycle, the node of maximum rank is the node w^- that precedes w on the cycle. Indeed, the probability of avoiding half the cycle before hitting w^-, by going through z, is $p/2$, since the walk from w must reach z at the first chance (probability p), then can stay within $\{x, y, z\}$ for a while, and the last time it leaves z before venturing outside $\{x, y, z\}$ it must move to whichever of x and y is closer to w^- (probability 1/2). The probability of avoiding half the cycle before hitting z is just p. Hence $R_{w^-} \geq k - 1 - \frac{p}{2}(\frac{k}{2} - 1)$, while $R_z \leq k - p(\frac{k}{2} - 1)$, and the former is larger for $k > 20$.

In contrast, the optimal on-line algorithm will evict z whenever there is a fault on the cycle. In this case, Max Rank incurs $\Omega(k)$ times as many faults as the optimal on-line algorithm. □

Finally, an algorithm which is very close in spirit to our nearly optimal on-line algorithm of Section 4 is the Maximum Commute Time (MCT) Algorithm. On a fault for page v, MCT evicts the page w in memory that maximizes the *commute time between v and w*.

Theorem 10 *For every k, there is a Markov chain M for which $f_{MCT}(M, k) \geq kf^*(M, k)$.*

Proof: Consider the Markov chain corresponding to a directed cycle $k + 1$ nodes. Then every pair of nodes in the graph has the same commute time $(k+1)$. Therefore, on a fault at some node v Maximum Commute Time might always evict the successor of v, incurring a fault on every request. Since the optimal on-line algorithm has a fault rate of $1/k$, the claim is proven. □

4 A provably good algorithm

4.1 Description of the algorithm

The Commute Algorithm (CA) operates in phases. It keeps a window of the last $k+i$ requested pages, for some $0 \leq i \leq k$. (We assume $n \geq 2k$. For the case $n \leq 2k$, a simpler algorithm in the same spirit can be shown to have page fault rate that is within a constant factor of optimal.) At the beginning of a phase the window is just the k most recently requested pages; these pages are resident in memory. When a "new" page p (one that hasn't been requested in the current or last phase) is requested, it is added to the window. The phase ends (and the window shrinks back to size k) when the $k + 1$'st distinct page is requested in the current phase. At that time, CA

performs the minimum number of swaps necessary to ensure that the k most recently requested pages are again resident in memory, and the window again becomes the k most recently requested pages.

When the window contains $k+i$ pages, *CA* maintains a partial matching of i disjoint pairs of pages $\{(u_1, v_1) \ldots (u_i, v_i)\}$. The commute algorithm maintains the following invariant:

> For each j, $1 \leq j \leq i$, exactly one of u_j and v_j is in memory.

On a fault at u_j the page v_j is evicted, and vice versa. (Observe that *CA* is not a marking algorithm.) When a new page p is added to the window, it is paired with a page q that is in the window, but not in the matching, such that the commute time $C_{p,q}$ is maximized. The new pair is added to the partial matching, and may be involved in a single "switch", described below.

To describe the notion of a switch, we need some notation. For states a, b, u, v, we define the *relative distance* of the pair (a, b) to u to be

$$d[(a,b),u] = \frac{\min\{C_{a,u}, C_{b,u}\}}{C_{a,b}}.$$

Define also the relative distance of (a, b) to (u, v) to be

$$d[(a,b),(u,v)] = \min\{d[(a,b),u], d[(a,b),v]\} = \frac{\min\{C_{a,u}, C_{b,u}, C_{a,v}, C_{b,v}\}}{C_{a,b}}.$$

Intuitively, if $d[(a, b), u]$ is large, then both a and b are much further (in the commute time metric) from u than they are from each other. Similarly, if $d[(a, b), (u, v)]$ is large, then both a and b are much further from both u and v than they are from each other.

When a new pair (p, q) is added to the matching, *CA* does the following:

Case 1: If $d[(p, q), (u_j, v_j)] \leq c_s$ for all j, then there is no switch: we service the fault at p by evicting q. Here c_s, the "constant for switching", is a suitably chosen constant; the reader can verify during the proof below that by choosing $c_s = 2$, all inequalities involving c_s hold.

Case 2 (Switch): Otherwise, choose j so that $d[(p, q), (u_j, v_j)]$ is maximized, and replace the matched pairs (p, q) and (u_j, v_j) by (u_j, p) and (q, v_j). Service the fault at p by evicting whichever of u_j or v_j is in memory. This restores the invariant mentioned above.

In the upcoming proofs, we will be distinguishing one node in each pair, and therefore we may need to reverse the roles of u_ℓ and v_ℓ for some pairs, whether or not a switch has been done. The analysis below shows how this should be done.

As for the running time of *CA*, the complexity of computing the commute times in a Markov chain is polynomial in n. Therefore, with an initial preprocessing step that constructs the matrix of commute times, the complexity of running the algorithm *CA* is $O(k)$ per fault.

4.2 Analysis of the algorithm

Theorem 11 *There is a constant c such that for any Markov chain M and any k,*

$$f_{CA}(M,k) \leq c f^*(M,k).$$

We prove the theorem by establishing a strict relationship between the pairs of CA's matching.

Consider one phase of the algorithm: let the r'th subphase be the time when the window size is $k+r$. Let M_r and W_r be the matching and the set of pages in the window, respectively, during the r'th subphase. Lastly, let V_{M_r} be the pages involved in M_r. Notice that $W_1 \subset W_2 \subset \cdots \subset W_k$ and $V_{M_1} \subset V_{M_2} \subset \cdots \subset V_{M_k}$. We will refer to one node in each pair (u_i, v_i) in M_r as *distinguished*. Without loss of generality the distinguished node will always be u_i.

We say that M_r is a *good* matching if:

1. For all pairs (u_i, v_i) and $(u_j, v_j) \in M_r$, $i \neq j$, $d[(u_i, v_i), u_j] \leq c_d$. Here c_d is the maximum allowable distance to distinguished nodes. The fact that u_j is involved in this condition, rather than v_j, is what makes u_j distinguished.

2. For each pair $(u_i, v_i) \in M_r$ and $a \in W_r \setminus V_{M_r}$, $d[(u_i, v_i), a] \leq c_u$. Here c_u is the maximum allowable distance to unmatched nodes.

The reader can verify during the proof that by choosing $c_u = 1$ and $c_d = 6$, all inequalities involving these constants hold.

We will be using the following facts about commute times.

Lemma 12 *1. Commute times satisfy the triangle inequality.*

2. $d[(a,b),c] \leq \frac{C_{a,c}}{C_{a,b}}$ *and* $d[(a,b),c] \leq \frac{C_{b,c}}{C_{a,b}}$.

3. $\frac{C_{a,c}}{C_{a,b}} \leq d[(a,b),c] + 1$ *and* $\frac{C_{b,c}}{C_{a,b}} \leq d[(a,b),c] + 1$.

4. $d[(a,b),c] \leq d[(a,b),d] + \frac{C_{c,d}}{C_{a,b}}$.

5. If $d[(x,y),z] \geq k_0$, $d[(a,b),x] \leq k_1$ *and* $d[(a,b),z] \leq k_2$, *then*

$$C_{x,y} \leq \frac{(k_1 + k_2 + 1)}{k_0} C_{a,b}.$$

Proof: Parts 1 and 2 follow immediately from the definition of commute time and relative distance. For part 3,

$$\frac{C_{a,c}}{C_{a,b}} \leq \frac{\min(C_{a,c}, C_{a,c} - C_{a,b}) + C_{a,b}}{C_{a,b}} \leq \frac{\min(C_{a,c}, C_{b,c})}{C_{a,b}} + 1 \leq d[(a,b),c] + 1,$$

where the second inequality follows from part 1. The other case is similar. For part 4,

$$d[(a,b),c] = \frac{\min(C_{a,c}, C_{b,c})}{C_{a,b}} \leq \frac{\min(C_{a,d}, C_{b,d}) + C_{d,c}}{C_{a,b}} = d[(a,b),d] + \frac{C_{d,c}}{C_{a,b}}.$$

Finally, part 5 follows from the application of part 2, part 1 and part 3 to give

$$C_{x,y} \leq \frac{C_{x,z}}{k_0} \leq \frac{(\min(C_{x,a}, C_{x,b}) + \max(C_{a,z}, C_{b,z}))}{k_0} \leq \frac{(d[(a,b),x] + d[(a,b),z] + 1)}{k_0} C_{a,b}.$$

Finally, use the fact that $d[(a,b),x] \leq k_1$ and $d[(a,b),z] \leq k_2$. $\square$

Lemma 13 *The Commute Algorithm's matching is always good.*

Proof: The proof is by induction. Consider first the matching M_1 at the start of a phase: there is a single matched pair (p,q), consisting of a page p whose request started the phase, and a page q that was chosen (out of the shrunk window of the k most recently requested pages) to maximize $C_{p,q}$. In this base case, the distinguished node can be chosen arbitrarily. Clearly, for $a \in W_r \setminus V_{M_1}$, we have $d[(p,q),a] \leq 1$.

Now suppose that M_r is a good matching: we will show that M_{r+1} is also good. We take q to be the distinguished node in the new pair. By the choice of q (maximizing $C_{p,q}$) and the assumption on M_r, M_{r+1} satisfies the following goodness conditions:

1. $d[(p,q),a] \leq c_u$ for any $a \in W_{r+1} \setminus V_{M_{r+1}}$.
2. For any pairs $(u_i,v_i),(u_j,v_j) \in M_r$, $d[(u_i,v_i),u_j] \leq c_d.$, and for any $(u_i,v_i) \in M_r$ and $a \notin V_{M_{r+1}}$, $d[(u_i,v_i),a] \leq c_u$.
3. For each pair $(u_i,v_i) \in M_r$, $d[(u_i,v_i),q] \leq c_u$ since q was in the window but not in the matching.

There are two cases to consider:

Case 1: We did not switch. $(d[(p,q),(u_j,v_j)] \leq c_s, \forall j.)$

Remember that $d[(p,q),(u_i,v_i)] = \min(d[(p,q),u_i], d[(p,q),v_i])$. It might be that $d[(p,q),u_i] > c_d$. But in this case, $d[(p,q),v_i] \leq c_s$, otherwise we would have switched. We show that for all pairs (u_j,v_j) in M_r, $d[(u_j,v_j),v_i] \leq c_d$, so v_i can play the former distinguished role of u_i.

Let $(u_j,v_j) \in M_r$, $j \neq i$. We have:

$$d[(u_j,v_j),v_i] \leq d[(u_j,v_j),q] + \frac{C_{q,v_i}}{C_{u_j,v_j}} \leq c_u + \frac{(c_s+1)C_{p,q}}{C_{u_j,v_j}},$$

where both inequalities follow from Lemma 12, the first from part 4 and the second from part 3. However, from part 5 of Lemma 12, since $d[(p,q),u_i] \geq c_d$, $d[(u_j,v_j),q] \leq c_u$, and $d[(u_j,v_j),u_i] \leq c_d$, it follows that

$$C_{p,q} \leq \frac{c_u + c_d + 1}{c_d} C_{u_j,v_j}.$$

Substituting into the previous equation gives

$$d[(u_j,v_j),v_i] < c_u + \frac{(c_s+1)(c_u+c_d+1)}{c_d} \leq c_d.$$

Case 2: We did switch:

Consider the new edges (p,u_j) and (q,v_j) (chosen so that $d[(p,q),(u_j,v_j)]$ is maximized). We show that p and q become distinguished vertices, and that for some other pairs (u_k,v_k), v_k must become the distinguished vertex. The detailed proof that M_{r+1} is good follows:

We will need to use the following three inequalities

- Since $d[(p,q),(u_j,v_j)] > c_s$

$$C_{p,q} < \frac{\min\left\{C_{p,u_j}, C_{p,v_j}, C_{q,u_j}, C_{q,v_j}\right\}}{c_s}. \tag{2}$$

- From Lemma 12, part 5, since $d[(p,q),u_j] > c_s$ (because we switched), $d[(u_k,v_k),q] \leq c_u$, and $d[(u_k,v_k),u_j] \leq c_d$ (both because M_r was good), it follows that for any pair $(u_k,v_k) \in M_r$, $k \neq j$,

$$C_{p,q} < C_{u_k,v_k} \frac{c_u + c_d + 1}{c_s} \tag{3}$$

- Lastly,

$$d[(p,q),(u_k,v_k)] \leq d[(p,q),(u_j,v_j)]. \tag{4}$$

Many of the goodness conditions follow immediately from the goodness of M_r. The ones that require proof are described below, each with a derivation.

1. $d[(p,u_j),q] \leq c_d$, and $d[(q,v_j),p] \leq c_d$. This follows from (2).
2. $d[(p,u_j),a] \leq c_u$ and $d[(q,v_j),a] \leq c_u$ for any unmatched $a \in W_{r+1}$. For the first inequality, applying Lemma 12 part 2, the fact that $C_{p,q} \geq C_{p,a}$ for all unmatched a, and equation (2) gives:

$$d[(p,u_j),a] \leq \frac{C_{p,a}}{C_{p,u_j}} \leq \frac{C_{p,q}}{C_{p,u_j}} \leq \frac{1}{c_s}.$$

Similarly, for the second inequality:

$$d[(q,v_j),a] \leq \frac{C_{q,a}}{C_{q,v_j}} \leq \frac{C_{q,p} + C_{p,a}}{C_{q,v_j}} \leq \frac{2}{c_s}.$$

3. For each pair $(u_k, v_k) \in M_r$, $k \neq j$, $d[(u_k, v_k), q] \leq c_d$ and $d[(u_k, v_k), p] \leq c_d$. The first inequality follows from the fact that M_r was good, so $d[(u_k, v_k), q] \leq c_u$. As for the second inequality:

$$\begin{aligned} d[(u_k, v_k), p] &\leq d[(u_k, v_k), q] + \frac{C_{p,q}}{C_{u_k, v_k}} \quad \text{by Lemma 12, part 4} \\ &\leq c_u + \frac{C_{p,q}}{C_{u_k, v_k}} \quad \text{since } d[(u_k, v_k), q] \leq c_u \\ &\leq c_u + \frac{c_u + c_d + 1}{c_s} \quad \text{by equation (3)} \\ &\leq c_d. \end{aligned}$$

4. For each pair $(u_k, v_k) \in M_r$, either both $d[(p, u_j), u_k] \leq c_d$ and $d[(q, v_j), u_k] \leq c_d$, or v_k can become distinguished.
   To prove this, assume that $d[(p, u_j), u_k] > c_d$. By assumption

$$C_{p,u_k} > c_d C_{p,u_j}, \tag{5}$$

   and from (2) above,

$$C_{p,q} < C_{p,u_j}/c_s. \tag{6}$$

   Combining these two facts with lemma 12, part 3, we obtain

$$\begin{aligned} d[(p, q), u_k] &\geq \frac{C_{p,u_k}}{C_{p,q}} - \frac{C_{p,q}}{C_{p,q}} \\ &> \frac{c_d C_{p,u_j}}{C_{p,q}} - \frac{C_{p,u_j}}{c_s C_{p,q}} \\ &> (c_d - 1/c_s)\frac{C_{p,u_j}}{C_{p,q}}, \end{aligned}$$

   and so

$$d[(p, q), u_k] > d[(p, q), u_j]. \tag{7}$$

   Therefore, by the choice of j,

$$d[(p, q), v_k] < d[(p, q), (u_j, v_j)]. \tag{8}$$

   Therefore

$$\begin{aligned} C_{p,v_k} &\leq C_{p,q}(1 + d[(p, q), v_k]) \quad \text{by Lemma 12, part 4} \\ &\leq C_{p,q}(1 + d[(p, q), u_j]) \quad \text{by (8)} \\ &\leq C_{p,q} + C_{p,u_j} \\ &\leq C_{p,u_j}(1 + 1/c_2) \quad \text{by (2)} \end{aligned}$$

so $d[(p,u_j),v_k] \leq c_d$. The previous four lines of equations also establish that $d[(q,v_j),v_k] \leq c_d$, by substituting q for p and v_j for u_j.
Now let $(u_l,v_l) \in M_r$. From (5) and (6) we have

$$\begin{aligned} C_{q,u_k} &\geq C_{p,u_k} - C_{pq} \\ &> (c_d - 1/c_s)C_{p,u_j} \\ &> (c_d - c_s/2)C_{p,u_j}. \end{aligned}$$

But,

$$\begin{aligned} C_{q,u_k} &\leq C_{u_l,v_l}(1 + d[(u_l,v_l),q] + d[(u_l,v_l),u_k]) \\ &\leq C_{u_l,v_l}(1 + c_u + c_d) \end{aligned}$$

and so

$$C_{p,u_j} \leq \frac{1 + c_u + c_d}{c_d - c_s/2} C_{u_l,v_l}. \tag{9}$$

We have

$$\begin{aligned} d[(u_l,v_l),v_k] &\leq d[(u_l,v_l),q] + \frac{C_{p,q}(1 + d[(p,q),v_k])}{C_{u_l,v_l}} \\ &\leq c_u + \frac{C_{p,q}(1 + d[(p,q),u_j])}{C_{u_l,v_l}} \quad \text{(by 8)} \\ &\leq c_u + \frac{3C_{p,u_j}}{2C_{u_l,v_l}} \quad \text{(by 2)} \\ &\leq c_d \quad \text{(by 9)}. \end{aligned}$$

□

The following lemma shows why we are interested in relative distances:

Lemma 14 *Let M be a Markov chain. Let u,v,h be three states in M, and let $\gamma = Pr(h$ visited during an (u,v) commute$)$. Then*

$$\frac{C_{uv}}{C_{uh}} \geq \gamma \geq \frac{C_{uv}}{C_{uv} + C_{uh}}$$

Proof:

We use the following well-known proposition from renewal theory (see for example [7], page 147): Consider a Markov chain started in state i. Let $0 < S < \infty$ be a stopping time such that $X_S = i$. Let j be an arbitrary state. Then

$$E_i(\text{number of visits to } j \text{ before time } S) = \pi_j E_i(S),$$

where π_j is the stationary probability of state j, and $E_i(X)$ is the expected value of random variable X when the chain is started in state i.

Let u, v, h be three states in M. Using the proposition, we obtain that

$$E_u(\#\text{ of visits to } h \text{ during a } (u,v) \text{ commute}) = \pi_h(E_u(T_v) + E_v(T_u)) = \pi_h C_{uv}. \quad (10)$$

Consider a random walk starting at u. Let p_1 be the probability that h is visited before v and let p_2 be the probability that h is first visited after v, but before a (u,v) commute has completed. Clearly $\gamma = p_1 + p_2$. Furthermore,

$E_u(\#$ of visits to h during a (u,v) commute$) =$

$p_1 E_h(\#$ of visits to h during the time it takes to go from h to v to u back to $h)$

$+p_2 E_h(\#$ of visits to h during the time it takes to go from h to u and back to $h.)$

Once again using the proposition, we obtain

$$E_u(\#\text{ of visits to } h \text{ during a } (u,v) \text{ commute}) =$$
$$p_1\pi_h(E_h(T_v) + E_v(T_u) + E_u(T_h)) + p_2\pi_h(E_h(T_u) + E_u(T_h)).$$

Combining this with equation 10, we obtain

$$C_{uv} = p_1(E_h(T_v) + E_v(T_u) + E_u(T_h)) + p_2 C_{uh}.$$

Since $(E_h(T_v) + E_v(T_u) + E_u(T_h)) \geq C_{uh}$, we obtain

$$\gamma \leq \frac{C_{uv}}{C_{uh}}.$$

On the other hand, $(E_h(T_v) + E_v(T_u) + E_u(T_h)) \leq C_{uv} + C_{uh}$, and so

$$\gamma \geq \frac{C_{uv}}{C_{uv} + C_{uh}},$$

completing the proof of the theorem.

□

We are ready to prove the main theorem.

Proof of Theorem 11: We show that *CA* incurs an expected number of faults which is at most a constant times the expected number incurred by the optimal on-line algorithm *OPT*.

Let a *hole* of *OPT* be any page in W_r that *OPT* does not have in memory. We maintain a 1:1 mapping from pairs in the matching to holes of *OPT*, satisfying the following two properties:

- If h_i is the *OPT* hole associated to pair (u_i, v_i) for some i, then $d[(u_i, v_i), h_i] = O(1)$.

- The mapping of pairs to holes is changed only when there is either an OPT-fault, or a new pair is added to the matching. In both cases, the association changes for $O(1)$ pairs.

Say that a (x, y) *commute begins at time* t if (x, y) is a pair in the matching at time t (i.e. $(x, y) = (u_i, v_i)$ or $(x, y) = (v_i, u_i)$ for some i), CA had a fault on the most recent request, and the most recent request was at x. Let $T(x, y)$ be the set of times t such that a (x, y) commute begins at time t. Define a *new* node to be a node that is visited in the current phase, but wasn't one of the nodes visited in the previous phase, and let G be the total number of new nodes seen in all phases in the request sequence.

The number of faults incurred by CA during a page request sequence, denoted $C(CA)$, satisfies

$$C(CA) = \left(\sum_{(x,y)} \sum_{t \in T(x,y)} 1 \right) + O(G),$$

where the last term comes from ensuring that the last k requests are in memory at the end of each phase.

Let $h(x, y, t)$ be the OPT hole associated to pair (x, y) at time t, for $t \in T(x, y)$. Let $X_{x,y,t}$ be the indicator random variable that is 1 if $t \in T(x, y)$ and $h(x, y, t)$ is requested during the (x, y) commute beginning at time t.

For $t \in T(x, y)$, let $t \in Q(x, y)$ if either $h(x, y, t)$ gets swapped to another pair or the phase containing t ends before the (x, y)-commute starting at time t completes. Otherwise let $t \in R(x, y)$. Let $C(OPT)$ be the number of faults incurred by OPT. For any page request sequence, we have

$$\sum_{(x,y)} \sum_{t \in R(x,y)} X_{x,y,t} \leq 2C(OPT),$$

since each OPT fault can be accounted for at most twice, namely by a (x, y) commute and a (y, x) commute, for some pair (u_i, v_i).

The mapping of pairs to holes only changes if OPT incurs a fault or if a new pair is added to the matching, and then only $O(1)$ pairs are affected. A new pair is added to the matching only when a new node is visited. At most $2g$ commutes are in progress at the end of a phase, where g is the number of new nodes visited during the phase. Thus we have

$$\sum_{(x,y)} \sum_{t \in Q(x,y)} X_{x,y,t} = O(C(OPT) + G).$$

(Note that in fact $\sum_{(x,y)} \sum_{t \in Q(x,y)} 1 = O(C(OPT) + G)$.) Therefore,

$$\sum_{(x,y)} \sum_{t \in T(x,y)} X_{x,y,t} \quad = \quad O(C(OPT) + G)$$

$$= \quad O(C(OPT) + k),$$

where k is the number of pages that can be held in memory, since it is known [8] that any paging algorithm incurs $\Omega(G - k)$ faults.

Lastly, since $d[(x, y), h(x, y, t)] = O(1))$, by Lemma 14 there is a constant $p > 0$ such that for all u, v, t such that $t \in T(u, v)$,

$$E[X_{x,y,t}] \geq p.$$

Putting all this together, we obtain by linearity of expectations that

$$E[C(CA)] = O(E[C(OPT)] + k),$$

so the fault rate of CA is at most a constant factor greater than that of OPT.

It remains only to describe the mapping of pairs to holes.

At the beginning of a phase, the single matched pair is associated with any OPT-hole in the window — note that there is at least one. In general, if h_j is the hole associated with (u_j, v_j), we maintain the following invariant:

$$h_j \text{ is unmatched, or is } u_j \text{ or } v_j, \text{ or } \{h_j, h_k\} = \{u_k, v_k\} \text{ for some } k \neq j.$$

Notice that the invariant remains unchanged when the distinguished node in a pair changes. (In the case where $\{h_j, h_k\} = \{u_k, v_k\}$ for some $k \neq j$, and the designated node for pair $\{u_k, v_k\}$ changes, swapping h_j and h_k ensures that the invariant remains true.) The following procedure ensures that at any time h_j is changed, it is set to an unmatched node, or u_j or v_j, or u_k for some $k \neq j$, so $d[(u_j, v_j), h_j] = O(1)$, as required.

The first case to consider is when no switch was performed, and the pair (p, q) needs to find a hole $h(p, q)$. We have the following cases.

1. If $p = h_j$ for some j and $q = h_k$ for some k, then p becomes $h(p, q)$, h_k remains q and we continue with (u_j, v_j) in place of (p, q).

2. If only one of p or q is associated with a pair, say $q = h(u_k, v_k)$, then q becomes $h(p, q)$, and we continue with (u_k, v_k).

3. At this point, there must be an OPT-hole that is unmatched because there are at least i holes total and only $i-1$ pairs are associated with a hole. We consider three cases depending on what this unassociated OPT-hole is.

   (a) If the OPT-hole is unmatched, is p or is q, set $h(p, q)$ to this OPT-hole.

   (b) If the OPT-hole is u_k or v_k such that $h_k \in \{u_k, v_k\}$, set $h(p, q)$ to u_k and h_k to v_k.

   (c) If the OPT-hole is u_k or v_k such that $h_k \notin \{u_k, v_k\}$, set $h(p, q)$ to h_k and h_k to the unassociated OPT-hole .

The second case is when a switch was performed, producing pairs (p, u_j) and (q, v_j). In this case, we first unmatch h_j from (u_j, v_j), and then apply two steps according to the directions in the no-switch case above: first for (u_j, p) and then for (q, v_j).

When OPT incurs a fault at a hole h_j, the page is loaded into memory, so is no longer a hole. The pair (u_j, v_j) then finds an unassociated OPT-hole as in the no-switch case above. This scheme satisfies the two required properties of the mapping from pairs to holes.

□

5 Acknowledgements

We would like to thank the reviewers for extensive comments and excellent suggestions.

References

[1] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. In *Proc. 22nd Annual ACM Symposium on Theory of Computing*, pages 379–388, 1990.

[2] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 249–259, 1991.

[3] A.Z. Broder. Generating random spanning trees. In *Proc. 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 442–447, 1989.

[4] A.K. Chandra, P. Raghavan, W.L. Ruzzo, R.Smolensky, and P. Tiwari. The electrical resistance of a graph captures its commute and cover times. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 574–586, 1989.

[5] D. Cheriton and K. Harty. Application-controlled physical memory using external page-cache management. Technical report, Department of Computer Science, Stanford University, 1991.

[6] P.J. Denning. Working Sets Past and Present. In *IEEE Trans. Software Eng.* SE-6:64-84, 1980.

[7] C. Derman. *Finite State Markov Decision Processes.* Academic Press, New York, 1970.

[8] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. On competitive algorithms for paging problems. To appear in *Journal of Algorithms*, 1990.

[9] P.A. Franaszek and T.J. Wagner. Some distribution-free aspects of paging performance. *Journal of the ACM*, 21:31–39, 1974.

[10] S.S. Irani, A.R. Karlin, and S.J. Phillips. Strongly competitive algorithms for paging with locality of reference. In *Third Annual ACM-SIAM Symposium on Discrete Algorithms*, 1992.

[11] A.R. Karlin, M.S. Manasse, L. Rudolph, and D.D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):70–119, 1988.

[12] R.M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[13] P.A.W. Lewis and G.S. Shedler. Empirically derived models for sequences of page exceptions. *IBM J. Res. and Develop.*, 17:86–100, 1973.

[14] D. McNamee and K. Armstrong. Extending the mach external pager interface to accommodate user-level page replacement policies. Technical Report 90-09-05, Department of Computer Science and Engineering, University of Washington, 1990.

[15] P. Raghavan. Lecture Notes on Randomized Algorithms. Technical Report RC 15340, IBM Research, 1990.

[16] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. In *16th International Colloquium on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 687–703. Springer-Verlag, July 1989. Revised version available as IBM Research Report RC15840, June 1990.

[17] G.S. Shedler and C. Tung. Locality in page reference strings. *SIAM Journal on Computing*, 1:218–241, 1972.

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

[19] J.R. Spirn. *Program Behavior: Models and Measurements* Elsevier Computer Science Library. Elsevier, Amsterdam. 1977.

[20] J.S. Vitter and P. Krishnan. Optimal Prefetching via Data Compression In *Thirty-Second Annual IEEE Symposium on Foundations of Computer Science*, 1991.