Page Replacement for General Caching Problems

Susanne Albers* Sanjeev Arora † Sanjeev Khanna‡

Abstract

Caching (paging) is a well-studied problem in online algorithms, usually studied under the assumption that all pages have a uniform size and a uniform fault cost (*uniform caching*). However, recent applications related to the web involve situations in which pages can be of different sizes and costs. This *general caching problem* seems more intricate than the uniform version. In particular, the offline case itself is NP-hard. Only a few results exist for the general caching problem [8, 17]. This paper seeks to develop good offline page replacement policies for the general caching problem, with the hope that any insight gained here may lead to good online algorithms. Our first main result is that by using only a small amount of additional memory, say $O(1)$ times the largest page size, we can obtain an $O(1)$-approximation to the general caching problem. Note that the largest page size is typically a very small fraction of the total cache size, say 1%. Our second result is that when no additional memory is allowed, one can obtain an $O(\log(M + C))$-approximation where M and C denote the cache size and the largest page fault cost, respectively. Our results use a new rounding technique for linear programs which may be of independent interest. We also present a randomized online algorithm for the Bit Model [8] which achieves a competitive ratio of $O(\ln(1 + 1/c))$ while using $M(1 + c)$ memory.

1 Introduction

When a sequence of memory objects ("pages") are to be retrieved from a slow or distant memory, one often uses a *cache* —i.e., a fast memory of some small size, say M— to retain some "frequently-used" pages. Since the slow memory needs to be accessed only for pages that are not in the cache at the moment they are requested (i.e., when a *page fault* occurs), the overall access time for the sequence may be dramatically reduced even with a fairly small cache. This basic and obvious principle of system design has proved itself in many situations. The reduction in total access time depends upon the *page replacement* policy, in other words, the method of deciding upon which cached page(s) to *flush* whenever space is needed in the cache for a new page. The *online* version of this problem is of particular interest, and has been extensively studied (see [9], for instance).

When all pages have the same size and *cost* (the *cost* refers to the time delay in bringing the page from the slow memory), then the optimal *offline* policy is Belady's Rule [2]: always flush the cached page whose next request is furthest in the future. The popular online strategy is LRU: flush the cached page that was *least recently used*. The heuristic justification for LRU is that real-life request sequences often exhibit the property that the "past predicts the future." Thus a page that has not been accessed in the recent past is unlikely to be accessed in the near future. The *competitive ratio* of LRU is M [14], where M is the the cache size, and no deterministic online strategy can do any better. Randomized strategies can achieve much better performance. The *randomized marking algorithm* by Fiat *et al.* [5] has a competitive ratio $2 \log M$ against oblivious adversaries. Algorithms achieving an optimal ratio of $\log M$ were given in [12, 1]. Fur further work on the uniform paging problem see, e.g., [3, 6, 10, 11, 15, 16].

This paper studies page replacement policies for the *General Caching Problem*, when the pages either have varying sizes, or varying costs, or vary in both size and costs. This problem arises, among other places, in cache design for networked file systems or the world-wide web. For example, HTTP, the current protocol for handling page requests on the web, treats a web page —be it an image file or a text file— as an indivisible object. Similarly, the transmission time for a web page depends on whether it is on a server in Asia or in America, and also on transient conditions such as server load or network congestion. The current paper will make the simplifying assumption that the cost of obtaining a page, though arbitrary, is a known quantity.

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. Part of this work was done while visiting the Freie Universität Berlin. Email: albers@mpi-sb.mpg.de.

†CS Department, Princeton University, Princeton, NJ 08544. Email: arora@cs.princeton.edu. Supported by NSF CAREER award NSF CCR-9502747, an Alfred Sloan Fellowship, and a Packard Fellowship. Part of this work was done during a visit to MPI Informatik.

‡Department of Fundamental Mathematics Research, Bell Labs, 700 Mountain Avenue, Murray Hill, NJ 07974. E-mail: sanjeev@research.bell-labs.com. URL: http://cm.bell-labs.com/who/sanjeev.

Irani [8] recently studied special cases of this problem — the Bit Model and the Fault Model (defined below) — and Young [17] studied deterministic online algorithms for the general problem. Irani points out that Belady's rule is not optimal if page sizes and costs differ, and gives $O(\log m)$-approximation algorithms for the offline case of the Bit and the Fault models, where m is the ratio of the cache size to the size of the smallest page. Building on the insight obtained from the offline algorithms, Irani designs $O(\log^2 m)$-competitive randomized online algorithms for these problems. Young [17] extends this work to design *loosely competitive algorithms*, whereby the memory used by the algorithm, M, is somewhat more than M_{off}, the memory used by the offline optimum. Young's deterministic online algorithm for the general caching algorithm incurs a cost $M/(M - M_{\text{off}} + 1)$ times the cost of the offline optimum.

Although the online cases of the general caching problem are of greatest interest, as a first step one may try to gain insight by designing a good approximation algorithm for the offline case. Our first main result here is an $O(1)$-approximation to the general caching problem that uses only a small amount of additional memory—$O(1)$ times the size of the largest page. (Note that in practice, the largest page will be a tiny part of the cache; less than 1%.) This is to be contrasted with previous results of this flavor which require $\Omega(M)$ additional memory, but achieve a constant factor competitiveness even working in the online setting [17]. Our second main result here is a technique for efficiently translating any solution using only a small amount of additional memory to one that uses no extra memory. We use this technique to obtain an $O(\log(M + C))$-approximation to the general caching problem when no additional memory is allowed, here C denotes the largest page cost. To our knowledge, prior to our work no nontrivial guarantees were known for the general caching problem in the offline setting.

1.1 Problem Statement and Results

In what follows, we formally state our problem and give an overview of our results. The setup for a *general caching problem* as follows. We are given a cache capacity M, a sequence σ of page requests, and for each page p in this sequence a cost, denoted by $\text{COST}(p)$, and a size, denoted by $\text{SIZE}(p)$ are associated. The goal is to decide on which pages to retain in the cache at every time step so as to minimize the cost of servicing the page faults. We denote by $\text{OPT}(\sigma)$ the optimal paging solution as well the cost of the optimal solution, and use just OPT when the sequence σ is clear by the context. Finally, let S be the largest page size and C be the largest page fault cost.

The Caching Models: There are four models of caching which have been studied before.

1. **The Bit Model[8]:** In the Bit Model, for each page p, we have $\text{COST}(p) = \text{SIZE}(p)$. (The delay in bringing the page into memory depends only upon its size.)

2. **The Fault Model[8]:** In the Fault Model, for each page p, we have $\text{COST}(p) = 1$ while the sizes can be arbitrary.

3. **The Cost Model:** In the Cost Model, for each page p, we have $\text{SIZE}(p) = 1$ while the costs can be arbitrary. This problem variant, also known as *weighted caching*, is a special instance of the k-server problem and the offline version can be optimally solved in polynomial time [4].

4. **The General Model:** Finally, in the General Model, for each page p, both the cost and size can be arbitrary. A loosely competitive online algorithm with competitive ratio $M/(M - M_{\text{off}} + 1)$ is known [17].

Our first main result is for the case when we are allowed a small amount of additional memory—say, $O(1)$ times the size of the largest page. Our approach is to formulate the caching problems as integer linear programs and then solve a relaxation to obtain a fractional optimal solution (see Section 2.1). The *integrality gap* of the linear programs is unbounded, but nevertheless we can show the following result. We state the Theorem in a very general way using two parameters, ϵ and δ. Parameter ϵ allows for varying the approximation ratio whereas δ allows for varying the amount of additional memory.

THEOREM 1.1. *For each of the above problems there is a polynomial-time algorithm that, given any request sequence, finds a solution of cost $c_1 \cdot \text{OPT}_{\text{LP}}$, where OPT_{LP} is the cost of the fractional solution (with memory M). The solution uses $M + c_2 \cdot S$ memory, where S is the size of the largest page in the sequence. The values of c_1 and c_2 are as follows for the various models. Let ϵ and δ be real numbers with $\epsilon > 0$ and $0 < \delta \leq 1$.*

1. *$c_1 = 1/\delta$ and $c_2 = \delta$ for the Bit Model,*

2. *$c_1 = (1 + \epsilon)/\delta$ and $c_2 = \delta(1 + 1/(\sqrt{1+\epsilon} - 1))$ for the Fault Model,*

3. *$c_1 = (4 + \epsilon)/\delta$ and $c_2 = 2\delta(1 + 6/\epsilon)$ for the General Model.*

The c_1, c_2 values in the above theorem express trade-offs between the approximation ratio and the additional memory needed. For example, in the Bit Model, we can get a solution with cost $\mathrm{OPT_{LP}}$ using at most S additional memory. In the Fault model, we can get a solution with cost $4\mathrm{OPT_{LP}}$ using at most $2S$ additional memory. The approximation ratio can be made arbitrarily close to 1 by using c_2S additional memory for a large enough c_2. In the General Model we obtain a solution of $20\mathrm{OPT_{LP}}$ using $2S$ additional memory, but we can achieve approximation ratios arbitrarily close to 4.

Our next main result is for the case when no additional memory is allowed.

THEOREM 1.2. *The caching problem in the General Model has an $O(\log(M + C))$-approximation.*

Finally in Section 4 we present a randomized online algorithm for the Bit model that achieves a competitive ratio of $O(\ln(1 + 1/c))$ while using $M(1 + c)$ memory.

1.2 Ideas and Techniques

We mention here two central ideas of our paper. Theorem 1.1 relies on a method to rounding fractional solutions with only a small amount of additional memory. Some ideas from our rounding procedure may be applicable to other sequencing problems. Theorem 1.2 is based on a transformation of the caching problem into a variant of the set cover problem. This transformation gives a canonical procedure to convert any caching algorithm that uses some extra memory into one that obeys the memory constraint strictly, with some degradation in performance.

1.3 Organization

The remainder of the paper is organized as follows. Section 2 describes our techniques for rounding fractional solution to the caching problem and establishes Theorem 1.1. Section 3 describes our paradigm for memory reduction and establishes Theorem 1.2. Section 4 describes the randomized online algorithm for the Bit model. We conclude with some remarks in Section 5.

2 Rounding of Fractional Solutions

2.1 The LP Formulation

Let the request sequence be $\sigma = \sigma_1, \sigma_2, ..., \sigma_n$. For each page p and time t when it is accessed (i.e. $\sigma_t = p$), we define $J_{p,t} = \{t + 1, t + 2, ..., t' - 1\}$ where t' is the first time the page p is accessed after time t; if such a time t' does not exist then the set $J_{p,t}$ is empty. It is now easy to see that the following integer linear program gives us the optimal paging strategy:

$$\text{Minimize} \quad \sum_{t=1}^{n} \mathrm{COST}(\sigma_t)(1 - x_{\sigma_t,t-1})$$

Subject to:

$$\begin{array}{rcll} x_{p,t} & = & 1 & \forall\, p,t \text{ such that } \sigma_t = p \\ x_{p,t+1} & = & x_{p,\bar{t}} & \forall\, p \text{ where } \bar{t} \in J_{p,t} \\ \sum_p \mathrm{SIZE}(p) x_{p,t} & \leq & M & \forall\, t \\ x_{p,t} & \in & \{0,1\} & \forall\, p,t \\ x_{p,0} & = & 0 & \forall\, p \end{array}$$

Replacing the constraint $x_{p,t} \in \{0,1\}$ by the constraint $0 \leq x_{p,t} \leq 1$, we obtain a linear programming relaxation that can be solved in polynomial time. From here on, we denote an optimal LP relaxation solution by $\mathrm{OPT_{LP}}$.

The integrality gap, i.e., $\mathrm{OPT}/\mathrm{OPT_{LP}}$, can be $\Omega(M)$. For example, in the Bit model, suppose the sequence consists of repeated requests to 10 pages each of size $M/10 + 1$. OPT would be able to fit at most 9 pages into cache and $\mathrm{OPT_{LP}}$ would be able to fit all but 10 units. In this case, note that allowing our (integral) algorithm just 10 extra units of space would allow it to lower the cost to $\mathrm{OPT_{LP}}$. Our rounding scheme suggests that things never get much worse: for any instance of the general caching problem, allowing $O(S)$ extra space allows us to get an integral solution of cost $O(\mathrm{OPT_{LP}})$.

The rounding will only change the value of fractional variables; it will never modify variables that are 0/1. A *fractional page* is a shorthand for "a page p and a time interval $J_{p,t}$ during which $x_{p,t}$ is neither 0 nor 1." When we say that a fractional page is rounded up (resp., rounded down) we mean that $x_{p,t}$ is set to 1 (resp., 0) for the entire interval.

2.2 Rounding for the Bit Model

In this section we describe the rounding algorithm for the Bit Model. The algorithm uses the fact that the LP formulation of page replacement has a simple interpretation in the Bit Model. Specifically, if we imagine the pages being split into pieces of size ϵ where ϵ is infinitesimally small, then the LP can be viewed as the offline page replacement problem for equal-cost pages of size ϵ. It can be solved by using Belady's rule on these small pages. From now on we assume the fractional optimum is computed this way, whence the following Lemma is immediate.

LEMMA 2.1. *Suppose pages p and q are requested at times t and s respectively, and $J_{q,s} \subseteq J_{p,t}$. If $x_{p,t} > 0$ in the optimum solution, then $x_{q,s+1} = 1$.*

Proof. During the time interval $J_{q,s}$, page p has a higher priority of being evicted than q since its next request is further in the future. Therefore the optimum would

```
1.  Extra := 0;
2.  For t = 1 to n do
3.      If page p is accessed at time (t − 1) and 0 < x_{p,t} < 1 then
4.          If Extra + SIZE(p)(1 − x_{p,t}) ≤ δS then
5.              Extra := Extra + SIZE(p) · (1 − x_{p,t});
6.              x_{p,t̄} := 1 for all t̄ ∈ J_{p,t−1};   /* Round up p and credit cost savings to Extra */
7.          else
8.              Extra := Extra − SIZE(p) · x_{p,t};
9.              x_{p,t̄} := 0 for all t̄ ∈ J_{p,t−1};   /* Round down p and charge cost increase to Extra */
```

Figure 1: The rounding procedure

decrease the portion of p that is in cache before it decreases the portion of q. □

Our rounding algorithm produces a solution that may need up to $M + \delta S$ memory but incurs cost $\frac{1}{\delta}\mathrm{OPT}_{\mathrm{LP}}$, for any $0 < \delta \leq 1$. Consider a fixed δ, $0 < \delta \leq 1$. In a first step, the algorithm rounds down all fractional variables $x_{p,t}$ with $x_{p,t} < 1 - \delta$, i.e., each of these variable is set to 0. This increases the cost of the solution by at most a factor of $1/\delta$ since at the next access of any such page p, $\mathrm{OPT}_{\mathrm{LP}}$ pays at least a cost of $\delta \cdot \textsc{cost}(p)$. Let $\mathrm{OPT}^{\delta}_{\mathrm{LP}}$ be this new solution.

Given $\mathrm{OPT}^{\delta}_{\mathrm{LP}}$, we apply a rounding procedure described in Figure 1. The procedure sweeps over the fractional $x_{p,t}$'s from time $t = 1$ to $t = n$. At each step, it rounds up a fractional page p if, after the rounding, the rounded up pages occupy extra memory of no more than $\delta \cdot S$. Otherwise, the page is rounded down.

We first observe that $0 \leq \texttt{Extra} \leq \delta \cdot S$ always. The second inequality follows from line 4 of the algorithm. Note that a page is only rounded down if $\texttt{Extra} + \textsc{size}(p)(1 - x_{p,t}) > \delta \cdot S$, which is equivalent to $\texttt{Extra} > \delta \cdot S - \textsc{size}(p)(1 - x_{p,t})$. Since $1 - x_{p,t} \leq \delta$ and $\textsc{size}(p) \leq S$, the first inequality follows.

Theorem 2.1. *For any $0 < \delta \leq 1$, we can construct a rounded solution that incurs a cost of at most $(1/\delta)\mathrm{OPT}_{\mathrm{LP}}$, and that uses an additional memory of at most δS.*

Proof. Consider the rounded solution obtained by applying the above rounding procedure to $\mathrm{OPT}^{\delta}_{\mathrm{LP}}$. We first show that at any time the total extra space used by the rounded solution is at most the value of `Extra`. We prove this invariant by induction on time t. Suppose it is true for up to time $t-1$. There are two possibilities to consider at time t.

`Extra` is incremented at time t: Then for some page p that was accessed at time $t-1$, we must have set the fractional variable $x_{p,t}$ to 1. This requires an extra space of $(1 - x_{p,t}) \cdot \textsc{size}(p)$, which is equal to the amount by which we increment `Extra`. The invariant is maintained.

`Extra` is decremented at time t: Then for some page q that was accessed at time $t-1$, the fractional variable $x_{q,t}$ is set to 0. This frees up a space equal to $x_{q,t} \cdot \textsc{size}(q)$, and furthermore, this space is available for the rest of the time interval $J_{q,t}$. Note that at time t there may already by rounded-up pages in memory. However, Lemma 2.1 implies that *all* those pages will next be requested *during* $J_{q,t}$. Thus the space freed up by q can be used by all those pages, and the available space truly becomes at least $\texttt{Extra} + x_{q,t} \cdot \textsc{size}(q)$. Also, when q is requested next, the rounded-up pages free the required space again so that q can be loaded into cache. This completes the induction.

It remains to analyze the cost of the rounded solution. Let E_u be the total value by which `Extra` is ever incremented, i.e, E_u is the sum of the values $\textsc{size}(p)(1 - x_{p,t})$ considering all executions of line 5. Similarly, let E_d be the total value by which `Extra` is ever decremented. Compared to $\mathrm{OPT}^{\delta}_{\mathrm{LP}}$, our rounded solution saves a cost of E_u and incurs an extra cost of E_d. Since $E_u - E_d$ is equal to the final value of `Extra`, which is non-negative, the total cost of the rounded solution is bounded by

$$\mathrm{OPT}^{\delta}_{\mathrm{LP}} - E_u + E_d \leq (1/\delta)\mathrm{OPT}_{\mathrm{LP}}. \ \Box$$

2.3 Rounding for the Fault model

We partition the pages into $\lfloor \log_d S \rfloor + 1$ classes, for some real number $d > 1$, such that class C_i, $0 \leq i \leq \lfloor \log_d S \rfloor$, contains pages p with $Sd^{-i} \geq \textsc{size}(p) > Sd^{-(i+1)}$. Let S_i be the size of the largest page and s_i be the size of the smallest page in C_i.

We first modify the optimal LP relaxation solution so that Lemma 2.1 holds for pages from the same class. For every class C_i we sweep over the fractional variables $x_{t,p}$ of pages from C_i. Whenever we encounter a variable $x_{p,t+1}$, $0 < x_{p,t+1} < 1$, such that p was requested at time t, we check if there are variables $x_{q,s+1}$, with

$q \in C_i$ and $0 < x_{q,s+1} < 1$, such that q was requested at time s and $J_{q,s} \subseteq J_{p,t}$. If so, we increase the value $x_{q,s+1}$ and decrease the value of $x_{p,t+1}$ until $x_{q,s+1} = 1$ or $x_{p.t+1} = 0$. To maintain the second constraint of our linear program we also increase the other $x_{q,\overline{s}}$, with $\overline{s} \in J_{q,s}$, and decrease the other $x_{p,\overline{t}}$, with $\overline{t} \in J_{p,t}$.

More specifically, let $\epsilon = \min\{(1 - x_{q,s+1})\text{SIZE}(q), x_{p,t+1}\text{SIZE}(p)\}$. We increase $x_{q,\overline{s}}$ by $\epsilon/\text{SIZE}(q)$, for all $\overline{s} \in J_{q,s}$, and decrease $x_{p,\overline{t}}$ by $\epsilon/\text{SIZE}(p)$, for all $\overline{t} \in J_{p,t}$. Clearly, this modification does not need any additional space because $J_{q,s} \subseteq J_{p,t}$. The cost of the solution decreases by $\epsilon/\text{SIZE}(q)$ and increases by $\epsilon/\text{SIZE}(p) < d\epsilon \cdot \text{SIZE}(q)$. The net increase is at most $(d-1)\epsilon/\text{SIZE}(q)$. Note that the optimal LP relaxation solution incurs a cost of $\epsilon/\text{SIZE}(q)$ in loading an amount of ϵ of page q into cache on request $\sigma(s')$, where s' is the time of the next request to q after s. We conclude that overall modified solution is feasible and incurs a cost of at most $d\text{OPT}_{\text{LP}}$.

Given this modified solution, we apply the rounding algorithm described in the previous section separately for each class C_i. First we round down all $x_{p,t}$ with $x_{p,t} < 1 - \delta$, for a fixed $0 < \delta \leq 1$. Let $\text{OPT}^{\delta}_{\text{LP}}$ be the resulting solution. Then we execute the rounding procedure in Figure 1 for each class C_i, where in line 4 of the code δS is replaced by δS_i.

THEOREM 2.2. *For any $\epsilon > 0$ and $0 < \delta \leq 1$, we can construct a rounded solution that incurs a cost of at most $\frac{1+\epsilon}{\delta}\text{OPT}_{\text{LP}}$ and uses an additional memory of at most $\delta(1 + 1/(\sqrt{1+\epsilon} - 1))S$.*

Proof. Consider the rounded solution obtained by applying the rounding algorithm to $\text{OPT}^{\delta}_{\text{LP}}$ for each class C_i. As in the previous section we can show that, for each C_i, $0 \leq \texttt{Extra} \leq \delta \cdot S_i$ always and the total extra space needed by the algorithm in rounding pages from class C_i is no more than `Extra`. Thus, the total extra space required is no more than $\delta \sum_{i \geq 0} S_i \leq \delta \cdot S \sum_{i\geq 0} d^{-i} < \delta d S/(d-1)$.

We show that the total cost of the rounded solution is at most $d\text{OPT}^{\delta}_{\text{LP}}$. Since $\text{OPT}^{\delta}_{\text{LP}} \leq (d/\delta)\text{OPT}_{\text{LP}}$, the theorem then follows by setting $d = \sqrt{1+\epsilon}$. Consider a fixed class C_i. Let $\text{OPT}^{\delta,i}_{\text{LP}}$ be the cost the solution $\text{OPT}^{\delta}_{\text{LP}}$ incurs in serving requests to pages from class C_i. Let E^i_u be the total value by which `Extra` is ever incremented i.e, E^i_u is the sum of the values $\text{SIZE}(p)(1 - x_{p,t})$ considering all executions of line 5 for class C_i. Similarly, let E^i_d be the total value by which `Extra` is ever decremented. The total cost incurred by the rounded solution in serving requests to pages from C_i is at most

$$\text{OPT}^{\delta,i}_{\text{LP}} - E^i_u/S_i + E^i_d/s_i \tag{2.1}$$

$$\begin{aligned} &\leq \text{OPT}^{\delta,i}_{\text{LP}} - E^i_u/S_i + dE^i_d/S_i \\ &\leq \text{OPT}^{\delta,i}_{\text{LP}} + (d-1)E^i_u/S_i \\ &\leq d\text{OPT}^{\delta,i}_{\text{LP}}. \end{aligned}$$

The second inequality follows because $E^i_u \geq E^i_d$. The third inequality holds because $\text{OPT}^{\delta,i}_{\text{LP}} \geq E^i_u/S_i$. The desired bound follows by summing (2.1) for all classes. □

2.4 Rounding for the General Model

Let OPT_{LP} denote the fractional optimum (using memory M). We show how to round the optimum fractional solution so that, for any $\epsilon > 0$ and $0 < \delta \leq 1$, the resulting integral solution uses at most $2\delta S(1+6/\epsilon)$ extra space, where S is the size of the largest page, and incurs a cost of at most $\frac{4+\epsilon}{\delta}\text{OPT}_{\text{LP}}$. Thus we can achieve an approximation factor arbitrarily close to 4. Fix a $0 < \delta \leq 1$. We first modify the fractional solution so that all fractional $x_{p,t}$'s are at least $1-\delta$; any variable that is at less than $1-\delta$ is rounded down to 0. This increases the cost of the solution by at most a factor $1/\delta$

We note that Lemma 2.1 is false for the general case, and so the rounding relies on a more global view of the optimum solution, namely an updown sequence.

DEFINITION 2.1. *For any pair of positive reals c_1, c_2, with $c_1 < c_2$, an (c_1, c_2)*-updown sequence *in the fractional solution is a pair of disjoint subsequences (A, B) of fractional pages such that if every page in A is rounded up and every page in B is rounded down, then for some fixed c, $c_1 < c < c_2$, and every instant t, one of the following is true.*

1. *The total extra space needed by A is between c_1 and c, and the total extra space created by B is at least c and at most c_2.*
2. *The total extra space needed by A is less than c_1, and that created by B zero.*

LEMMA 2.2. *Let (A, B) be any (c_1, c_2)-updownsequence in the fractional optimum. Let u_A be the cost saved by rounding up A and d_B be the cost incurred by rounding down B. Then $d_B \leq (c_2/c_1)u_A$.*

Proof. We use the fact that if the fractional solution is optimum, then any "perturbations" to it will never decrease the cost. Imagine the following perturbation, where $\epsilon > 0$ is infinitesimally small. For each page in A, if x is the fraction allocated to any page in A, then reduce x by $x\epsilon/c_1$. The additional cost incurred is $u_A\epsilon/c_1$. For each page in B, if y is its allocation then increase y by $y\epsilon/c_2$. The cost saved is $d_B\epsilon/c_2$. By the definition of an updown sequence, at every time step one of the two must happen: (i) The space requirement for A reduces by between ϵ and $c\epsilon/c_1 > \epsilon$. The space requirement for B increases by between $c\epsilon/c_2 < \epsilon$ and ϵ. (ii) The space requirement for A reduces by some $\gamma \leq \epsilon$ but the space requirement for B does not increase.

In both cases, the perturbed solution does not need any new space, so it is feasible. Hence we have

$$(\text{additional cost incurred}) - (\text{cost saved}) \geq 0,$$

thus implying

$$\frac{u_A \epsilon}{c_1} - \frac{d_B \epsilon}{c_2} \geq 0.$$

Hence $d_B/c_2 \leq u_A/c_1$. □

Now we are ready to describe our Rounding Algorithm. Our algorithm is based on the following lemma whose proof appears later.

LEMMA 2.3. *Given any fractional solution and for any $c > 0$, we can in polynomial time decompose the set of fractional pages into disjoint $(c, 4c + 6\delta S)$ updownsequences $(A_1, B_1), (A_2, B_2), \ldots$, where S is the size of the largest page. These sequences also have the following property. Suppose for some time instant t there is a j such that the space provided by rounding down pages in B_j is less than the space required by rounding up the pages in A_j. Then at that instant t, there are no pages in $A_{j+1}, A_{j+2}, \ldots$ and $B_{j+1}, B_{j+2}, \ldots$, and the total space needed by rounding up pages in $A_j, A_{j+1}, A_{j+2}, \ldots$, is at most $2c + 2\delta S$.*

The Rounding Algorithm: We use the algorithm of Lemma 2.3 to find the disjoint $(c, 4c + 6\delta S)$ updownsequences $(A_1, B_1), (A_2, B_2), \ldots$. We round up all pages in the A_i's and round down all pages in the B_i's.

We claim that we finish with an integral solution of cost at most $4 + 6\delta S/c$ times the fractional cost we started with. This is because the cost saved by rounding up all A_i's is a lower bound on the fractional cost, and the additional cost incurred by rounding down all the B_i's is at most $4 + 6\delta S/c$ times larger, see Lemma 2.2. Since there was a factor $1/\delta$ increase in the cost of our solution due to rounding down of all variables with value less than $1 - \delta$, the final cost is $\frac{4+\epsilon}{\delta}\mathrm{OPT_{LP}}$, where $\epsilon = 6\delta S/c$.

We have to analyze the extra space needed by the new solution. Consider a time instant t and an index j. If the space provided by rounding down the pages in B_j is at least as large as the space required by rounding up the pages in A_j, then no additional memory is required in rounding the pages in A_j and B_j. Lemma 2.3 assures that if the space provided by pages in B_j is not sufficient, then the total space needed by the pages in $A_j, A_{j+1}, A_{j+2}, \ldots$, is also at most $2c + 2\delta S$. Substituting c by $6\delta S/c$, we obtain an additional space requirement of $2\delta(1 + 6/\epsilon)S$.

Proof of Lemma 2.3. In order to establish Lemma 2.3, we need to describe a procedure for constructing the sequences (A_i, B_i). We need the following two complementary procedures.

Algorithm Round-up(c): Given a set of fractional pages, the algorithm tries to produce a subsequence of pages which, if rounded up, require between c and $2c + 2\delta S$ space. We sweep over the pages from time $t = 0$ to $t = n$. Whenever the space needed by the pages in the current subsequence is less than c, we round up the page whose next request is farthest in the future.

Algorithm Round-down(c): Given a set of fractional pages, the algorithm tries to produce a subsequence of pages which, if rounded down, create between c and $2c + 2\delta S$ space. The algorithm is the same as above, but pages are rounded down.

Do for $i = 1, 2, \ldots$ until there are no more fractional pages: Run Round-up(c) and let A_i be the subsequence produced by it. Remove all pages in A_i from the set. Then run Round-down($2c + 2\delta S$) and let B_i be the subsequence produced by it. Remove all pages in B_i from the set. The next lemma shows a useful property of the above two procedures.

LEMMA 2.4. *In a sequence A constructed by Round-up(c), at any time the space needed by the pages rounded up is at most $2c + 2\delta S$. (A similar statement holds for a sequence B constructed by Round-down(c).)*

Proof. While A is constructed, we imagine that a pointer moves from left to right along the request sequence. The pointer is equal to the time step where *Round-up(c)* is currently located. Consider any time t. While the pointer is still to the left of t, the space needed at time t by pages rounded up is at most $c + \delta S$. The additive term δS is due to the fact that initially, we round down all fractional pages that are present to an extent of less than $1 - \delta$; thus for each page rounded up, the space needed is at most δS. We will show that while the pointer is to the right of t, an additional space of at most $c + \delta S$ can be added at time t. Assume, on the contrary, that an additional space of more than $c + \delta S$ is added and let t', $t' > t$, be the first pointer position when this happens. At time t', *Round-up(c)* rounds up a page p that also needs space at time t, i.e., p is not requested between t and t' and, hence, is available for rounding throughout the interval $[t, t']$. After the rounding of p, the extra space needed by the rounded up pages at time t' is no more than $c + \delta S$. Since at time t an additional space of more than $c + \delta S$ is needed, there must exist a page p' that was rounded up at some t'', $t < t'' < t'$, such that p' needs extra space at time t but not at time t'. Thus p' is requested during $[t', t']$. This is a contradiction because *Round-up(c)* always rounds up the pages whose next request is farthest in the future. That is, at time t'' the algorithm would round p instead of p'. $\square$

To finish the proof of Lemma 2.3, we need to prove the remaining property of the sequences. By Lemma 2.4, for any j, the pages rounded up in A_j need an additional space of at most $2c + 2\delta S$. Suppose j is such that for some time interval $[t, t']$, pages in B_j provide less than $2c + 2\delta S$ space if rounded down. Then by the definition of the algorithm *Round-down*, which is invoked with parameter $2c + 2\delta S$, the total space needed by the remaining fractional pages throughout $[t, t']$ is 0, i.e., there are no fractional pages in $A_{j+1}, A_{j+2}, \ldots$ during $[t, t']$. Hence $2c + 2\delta S$ is an upper bound on the total space required in $[t, t']$ when rounding up $A_j, A_{j+1}, \ldots,$. $\square$

We summarize the main result of this section.

THEOREM 2.3. *For any $\epsilon > 0$ and $0 < \delta \leq 1$, we can construct a rounded solution that incurs a cost of at most $\frac{4+\epsilon}{\delta}\mathrm{OPT_{LP}}$ and that uses an additional memory of at most $2\delta(1 + 6/\epsilon)S$.*

3 A Paradigm for Memory Reduction

In this section we design approximation algorithms that do not exceed the stated memory. We do this by pruning our earlier solutions. Note that because of the unbounded integrality gap mentioned earlier, we have to compare the cost of the final solution not to $\mathrm{OPT_{LP}}$ (as we did before) but to OPT. In the general model, we can convert any solution that uses excess memory to a solution using no extra memory such that the performance degrades by at most a factor of $O(\log(M + C))$ (in the General Model). We can improve the guarantee in the special case when the excess memory used at any time is bounded by a constant number of memory units; then the cost increase is $O(\mathrm{OPT})$.

3.1 The Basic Framework

Consider an input sequence $\sigma = \sigma_1, \sigma_2, ..., \sigma_n$. Let P be a paging solution for σ obtained via LP rounding (or in some other manner) that uses some extra pages. For our purposes, it will be convenient to view the solution P as a collection of labeled intervals, say $\{J_1, J_2, ...\}$, where each labeled interval J_i is specified as a 3-tuple of the form $J_i = \langle p, t_1, t_2\rangle$ indicating that

- page p resides in the memory from time t_1 through time t_2,
- $\sigma_{t_1 - 1} = \sigma_{t_2+1} = p$, and
- $\sigma_t \neq p$ for $t_1 \leq t \leq t_2$.

We refer to page p as the label of interval J_i, and define the *cost* of J_i to be $\mathrm{COST}(p)$. Let $M[t]$ denote the memory used by the solution P at time $t \in [1..n]$; then $M[t] = \mathrm{SIZE}(\sigma_t) + \sum_{\langle p,t_1,t_2\rangle \in P,\ t_1 \leq t \leq t_2} \mathrm{SIZE}(p)$. Let $E[t] = \max\{M[t] - M, 0\}$ denote the excess memory used by P at time t. We now formally state the *memory reduction problem*:

DEFINITION 3.1. (Memory Reduction Problem) *Given a solution P and an excess memory sequence $\{E[t]\}_{t=1}^{n}$, the* memory reduction problem *is to find a set $Y \subseteq P$ of labeled intervals such that $\sum_{\langle p,t_1,t_2\rangle \in Y,\ t_1 \leq t \leq t_2} \mathrm{SIZE}(p) \geq E[t]$ for all $t \in [1..n]$, and $\sum_{\langle p,t_1,t_2\rangle \in Y} \mathrm{COST}(p)$ is minimized.*

The memory reduction problem above can be viewed as a set covering problem.

DEFINITION 3.2. (Multiset Multicover Problem) *An instance of the* multiset multicover problem *comprises of a ground set U, a family of multisets $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ where each S_i is a multiset of U, and two functions $f : U \to \mathcal{N}$ and $g : \mathcal{S} \to \mathcal{N}$. The goal is to find a subset $X \subseteq \mathcal{S}$ such that every element $u \in U$ is covered at least $f(u)$ times by the sets in X, and the cost $\sum_{S \in X} g(S)$ is minimized.*

PROPOSITION 3.1. [13] *The multiset multicover problem has an $O(\log|U|)$-approximation.*

Given our paging solution P, we construct an instance I_P of the multiset multicover problem as follows:

- The ground set U corresponds to time steps $1, 2, ..., n$; define $f(t) = E[t]$ for $1 \leq t \leq n$.
- For each interval $J = \langle p, t_1, t_2\rangle$ in P, there is a set S_J that comprises of $\mathrm{SIZE}(p)$ copies of each element t that occurs in the set $\{t \mid (t_1 \leq t \leq t_2) \wedge E[t] > 0\}$; define $g(S_J) = \mathrm{COST}(p)$.

LEMMA 3.1. *Let X be a solution to the instance I_P of the multiset multicover problem. Then removing all labeled intervals J such that $S_J \in X$ gives a solution of identical cost to the memory reduction problem. Conversely, let Y be a solution to the memory reduction problem, then the collection of sets S_J such that $J \in Y$, gives a solution of identical cost to the multiset multicover instance I_P. Thus the cost of optimal solutions for the two instances is identical.*

LEMMA 3.2. *Let P be a paging solution for a request sequence σ which may violate the memory constraints at each time step in an arbitrary manner. Then the cost of an optimal solution to the memory reduction problem defined by P is bounded by* $\mathrm{OPT}(\sigma)$, *the cost of the optimal paging solution on σ.*

Combining Proposition 3.1 with Lemmas 3.1 and 3.2, we can conclude the following theorem.

THEOREM 3.1. *Let P be a paging solution for a request sequence σ which may arbitrarily violate the memory constraints. Then in polynomial time we can transform P into a solution P' which never violates the memory constraint and has cost $C_{P'} \leq C_P + O(\mathrm{OPT}(\sigma)\log n)$ where n is the length of the request sequence.*

While the above theorem yields a nice tool for relating the cost of memory reduction on an arbitrary solution to the optimal paging cost, unfortunately, the $\log n$ factor depends on the length of the paging sequence. We next show how this factor can be improved to an $O(1)$ factor when the total amount of excess memory used at any step is $O(1)$, and to an $O(\log(M + C))$ factor in general.

3.2 $O(1)$ Excess Units of Memory

We now examine the case when at any time step the total amount of excess memory used is at most α units, for some constant α. We will show that a simple dynamic programming approach can solve the memory reduction problem at a total cost of $\alpha\mathrm{OPT}(\sigma)$.

We first describe an optimal procedure to perform memory reduction when the excess memory used at any step is at most 1 unit. We maintain an array $S[1..n]$ where $S[t]$ stores the optimal cost of memory reduction on solution P restricted to time steps t through n. Initialize $S[n] = 0$ and suppose we have already computed the array entries $S[t+1]$ through $S[n]$. To compute $S[t]$, we proceed as follows. Let I_t denote all intervals $J = \langle p, t_1, t_2\rangle \in P$ such that $t_1 \leq t \leq t_2$. If $E[t] = 0$ then $S[t] = S[t+1]$, otherwise

$$S[t] = \min_{\langle p,t_1,t_2\rangle \in I_t} \{\textsc{cost}(p) + S[t_2 + 1]\}.$$

Thus $S[1]$ contains the optimal cost of memory reduction on the entire sequence σ. This computation can be performed in time $O(n^2)$. Finally, in order to obtain the actual set of pages evicted in the optimal memory reduction, we maintain an auxiliary array $R[1..n]$. If $S[t] = S[t + 1]$, then $R[t] = \emptyset$, otherwise $R[t]$ stores the page p chosen for eviction at time t; this is the page that yields the least value in the computation of $S[t]$.

Finally, in order to deal with an excess memory of upto α units, we can simply invoke the above procedure α times. This gives us the following theorem.

THEOREM 3.2. *Let P be a paging solution for a request sequence σ which may violate the memory constraints at each time step by at most α units of memory. Then in polynomial time we can transform P into another solution P' which never violates the memory constraint and has cost $C_{P'} \leq C_P + \alpha\mathrm{OPT}(\sigma)$.*

3.3 The General Model

The $\log n$ factor in Theorem 3.1 is a consequence of the large universe size (an element for every time step) that results due to the global nature of our construction. In order to get around this obstacle, we create a more "localized" reduction where we transform the memory reduction problem into several instances of multiset multicover problem, each with a small universe. In particular, we will partition our input sequence into blocks of length $O((MC)^{O(1)})$ and create an instance of the multiset multicover problem for each such block. In doing so we explicitly use the structure underlying the LP solution and our rounding scheme, described earlier.

Consider an LP solution; it is specified as a sequence $\{x_{p,t}\}_{t=1}^n$ for each page p in the input sequence. To begin with, we make the simplifying assumption that each variable $x_{p,t}$ satisfies the condition that $x_{p,t} \notin (1 - 1/M, 1)$, that is, no fractional variable takes a value "too close" to 1. We will later show how to handle the situation when this is not the case. Now observe that in our rounding algorithms described in the previous section, the only time t when a memory violation might be introduced is when we round up some positive fractional variable $x_{p,t}$ (implicitly rounding up $x_{p,t+1}, ..., x_{p,t'-1}$ where t' is the first time that p is accessed after time $t - 1$). Thus we know that the cost incurred by the LP solution at time t' is $\textsc{cost}(p)(1 - x_{p,t}) \geq 1/M$. It will be convenient for our purposes to imagine this cost being paid at time t itself. Now to transform our memory reduction problem to a multiset multicover problem, we only consider all such times t at which our rounding algorithm rounds upwards and creates a demand for excess memory. Let T denote the set of all such time steps. It is easy to verify that a sequence of page evictions makes P satisfy the memory constraints for each $t \in [1..n]$ if and only if it does so for each $t \in T$. Let n' denote $|T|$ and assume w.l.o.g. that $T = \{1, 2, ..., n'\}$. As before, we define $M[t]$ and $E[t]$ for $t \in T$, and proceed as follows:

1. Partition T into contiguous blocks of length M^2C, except possibly the last one.

2. Create a multiset multicover instance for each block as described earlier in Section 3.1.

3. Solve each instance created above and take the intervals corresponding to the union of the chosen sets as the final solution.

Since the universe size is bounded by M^2C for each block, we get an $O(\log(M+C))$ approximation within each block. On the other hand, observe that solving the multiset multicover instances independently may lead to a solution that is potentially much more expensive than a global optimal solution. This is due to the fact that a set may cut across many blocks. In particular, the extra cost incurred due to this partitioning scheme could be as large as the total cost of pages in the memory at the moment when a block ends, summed over all the blocks. However notice that since the total memory used at any time by our rounding of the LP is $O(M)$, the extra cost incurred per block due to the partitioning can be bounded by $O(MC)$. On the other hand, the LP solution itself incurs a cost of $\Omega(MC)$ in each block. Thus this extra cost incurred can be amortized over the cost of the LP solution (and hence $\mathrm{OPT}(\sigma)$) itself. This gives us an algorithm that will perform the desired memory reduction at a total cost of $O(\mathrm{OPT}(\sigma)\log(MC))$.

Finally, we need to show how to handle the case when some fractional variables take values in the open interval $(1-1/M, 1)$. We start by rounding up all such variables, and in addition, round down any variables that have value less than $1/2$. The rounding up step creates a violation of the memory constraint by at most 1 unit of memory at any time t. While the rounding down increases the cost of our solution by at most a factor of two. Next we use the dynamic programming algorithm described in Section 3.2 to create a new solution that satisfies the memory constraints exactly and any fractional variables remaining satisfy the desired property of not taking a value in the interval $(1-1/M, 1)$. The cost incurred in this step is $O(\mathrm{OPT}(\sigma))$. We now proceed as described above. Putting it all together,

THEOREM 3.3. *Let P be a paging solution for a request sequence σ which may violate the memory constraints at each time step in an arbitrary manner. Then in polynomial time we can transform P into another solution P' which never violates the memory constraint and has cost $C_{P'} \leq C_P + O(\mathrm{OPT}(\sigma)\log(M+C))$.*

Combining the above theorem with the rounding procedure of Section 2.4, we get:

THEOREM 3.4. *The caching problem in the General Model has an $O(\log(M+C))$-approximation.*

4 A Randomized Online Algorithm for the Bit Model

In this section we use the intuition derived from our offline algorithms to derive an online algorithm for the Bit Model. If OPT denotes the optimum offline cost using memory M_{off}, then our online algorithm achieves cost $O(\log(1+1/c))\mathrm{OPT}$ using excess memory $c \cdot M$, as long as $c \cdot M > S$, where S is the size of the largest page. This matches the best such result known for the uniform case (Young [15], modifying an analysis of Fiat *et al.* [5]).

Our online algorithm for the Bit Model is a modification of the *Marking* algorithm due to Fiat *et al.* [5], which we review briefly. The original *Marking* algorithm works for pages of uniform size and operates in a series of phases. At the beginning of each phase, all pages are unmarked. Whenever a page is requested, it is marked. If there is a request to a page not in cache, the algorithm chooses a page uniformly at random for among the unmarked pages in cache and evicts it. A phase ends when there is a request to a page not in cache and there are no unmarked pages in cache. At that point all marks are erased and a new phase is started.

The online algorithm for the Bit Model makes use of an auxiliary algorithm. This algorithm assumes that pages consist of infinitesimally small pages of size ϵ that can be loaded an evicted independently. We refer to these subpages as *ϵ-pages.*

Algorithm Fractional: Given a request sequence to the original pages, replace each request to a page p by a sequence of requests to the corresponding ϵ-pages. Run the *Marking* algorithm on that request sequence but with the following modification: A phase ends when, for the first time, there is a request to an ϵ-page not in cache and the size of the page the ϵ-page belongs to, plus the total size of the marked pages in cache exceeds the cache capacity. At that time evict pages that were not requested in the phase and use the free space to load the first missing pages in the next phase.

The analysis given by Young [15] for the *Marking* algorithm implies that if $M - S > M_{\mathrm{off}}$, then *Fractional* achieves a competitive ratio of $2\ln\frac{M-S}{M-S-M_{\mathrm{off}}}+1$ when $\frac{M-S}{M-S-M_{\mathrm{off}}} > e$ and 2 otherwise. Here M and M_{off} denote the cache size of the online and offline algorithms, respectively.

The integral algorithm for the Bit model divides the pages into $\lfloor \log_d S \rfloor + 1$ classes, for some $d > 1$, such that class C_i, $0 \leq i \leq \lfloor \log_d S \rfloor$, contains pages p with $Sd^{-i} \geq \mathrm{SIZE}(p) > Sd^{-(i+1)}$. Let S_i be the size of the largest page and s_i be the size of the smallest page in C_i. In the following we call a page *old* if it was requested in the previous phase but has not been requested in the current phase.

Algorithm Integral: Given a request sequence, the algorithm executes a marking strategy similar to *Fractional*. At any point in time the algorithm keeps track of the moves *Fractional* would do assuming a cache of capacity $M - \frac{d}{d-1}S$. If there is a request to a page p not in cache, the algorithm determines the total amount O_i^F of old pages from C_i that *Fractional* does not have in cache when all requests to the ϵ-pages of p are served. Let O_i^I be the total amount of old pages that *Integral* does not have in cache. *Integral* chooses $\lfloor (O_i^F - O_i^I)/S_i \rfloor$ pages uniformly at random from among the unmarked (i.e. old) pages in C_i that are in cache and loads p.

We can show that the expected cost incurred by *Integral* is at most d times the cost incurred by *Fractional*. Algorithm *Integral* may need more space than *Fractional* run on a cache of size $M - \frac{d}{d-1}S$. This is because, for each class C_i, *Integral* might evict less than *Fractional*. However, for each C_i, the difference is no more than S_i. Thus, *Integral* needs at most $\sum_{i\geq 0} S_i = S \sum_{i \geq 0} d^{-i} < \frac{d}{d-1}S$ more space than *Fractional* run on a cache of size $M - \frac{d}{d-1}S$.

THEOREM 4.1. *Let $d > 1$ and $M' = M - (1 + \frac{d}{d-1})S$ If $M_{\text{off}} < M'$, then Integral achieves a competitive ratio of $2d \ln \frac{M'}{M' - M_{\text{off}}} + d$ when $\frac{M'}{M'-M_{\text{off}}} > e$ and $2d$ otherwise.*

5 Concluding Remarks

The hardness results for caching problems are very inconclusive. The NP-hardness result for the Bit model uses a reduction from PARTITION, which has pseudopolynomial algorithms. Thus a similar algorithm may well exist for the Bit model. We do not know whether the Fault model is NP-hard.

We imagine that the requirement for extra memory in our $O(1)$-approximation algorithm for General Model could be removed. We showed in Section 3 a paradigm for pruning a solution so as to reduce its memory requirement. This relies on a special kind of set cover problem that may well have an $O(1)$-approximation. If so, we could start with the trivial solution (keep all pages in memory as long as they still have a future reference; possibly exceeding the cache capacity), and apply the memory reduction technique to get an $O(1)$-approximation.

Finally, the structure exposed by our rounding technique –specifically, the updown sequences — may provide insight into how to design a good online algorithm.

References

[1] D. Achlioptas, M. Chrobak and J. Noga. Competitive analysis of randomized paging algorithms. *Proc. Fourth Annual European Symp. on Algorithms (ESA)*, Springer LNCS, Vol. 1136, 419–430, 1996.

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

[3] A. Borodin, S. Irani, P. Raghavan and B. Schieber. Competitive paging with locality of reference. *Journal on Computer and System Sciences*, 50:244–258, 1995.

[4] M. Chrobak, H. Karloff, T. Paye and S. Vishwanathan. New results on the server problem. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991.

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

[6] A. Fiat and Z. Rosen. Experimental studies of access graph based heuristics: Beating the LRU standard. *Proc. 8th Annual ACM-SIAM Symp. on Discrete Algorithms*, 63–72, 1997.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, 1979.

[8] S. Irani. Page replacement with multi-size pages and applications to Web caching. *Proc. 29th Annual ACM Symp. on Theory of Computing*, 701–710, 1997.

[9] S. Irani and A.R. Karlin. Online computation. In *Approximation Algorithms for NP-hard Problems.* D. S. Hochbaum (Editor). PWS, Boston, MA, 1996.

[10] S. Irani, A.R. Karlin and S. Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing*, 25:477-497, 1996.

[11] A. Karlin, S. Phillips and P. Raghavan. Markov paging. *Proc. 33rd Annual Symp. on Foundations of Computer Science*, 24–27, 1992.

[12] L.A. McGeoch and D.D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6:816-825, 1991.

[13] S. Rajagopalan and V. Vazirani. Primal-dual RNC approximation algorithms for (multi)-set (multi)-cover and covering integer programs. *Proc. 34th Annual Symp. on Foundations of Computer Science*, 322–331, 1993.

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

[15] N. Young. On-line caching as cache size varies. *Proc. 2nd Annual ACM-SIAM Symp. on Discrete Algorithms*, 241–250, 1991.

[16] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11:525–541, 1994.

[17] N.E. Young. Online file caching. *Proc. 9th Annual ACM-SIAM Symp. on Discrete Algorithms*, 82–86, 1998.