ELSEVIER Theoretical Computer Science 181 (1997) 75–90

Theoretical Computer Science

Paging more than one page[1]

Esteban Feuerstein[a,b,*]

[a] *Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina*
[b] *Instituto de Ciencias, Universidad de General Sarmiento, Argentina*

Abstract

In this paper we extend the Paging Problem to the case in which each request specifies a set of pages that must be present in fast memory to serve it. The interest on this extension is motivated by many applications in which the execution of each task may require the presence of more than one page in fast memory. We introduce three different cost models that can be applied in this framework, namely the *Full*, *Uniform* and *Constant* cost models, and study lower and upper bounds for each one of them, using competitive analysis techniques.

1. Introduction

The *Paging Problem* arose as a theoretical model for a concrete problem present in the implementation of operating systems offering virtual memory [5]. It is the problem of managing a memory consisting in two levels, one of which with limited capacity and fast access (the cache) and the other one with slow access time but potentially unlimited capacity. Memory is divided in pages of fixed size, of which the cache may contain at most k. An operation consists in the request for a certain page, if that page is present in the cache the cost to serve the request is 0, otherwise it is necessary to bring it to the cache with cost 1. In this case we say that a *page fault* has occurred. As future requests are not known, we are interested in replacing policies that minimize the total cost required to serve a sequence of requests, i.e. that minimize the total number of page faults. All useful strategies are based on a notion of "locality" present in the sequence of requests that reflects the fact that normally programs tend to request pages from a subset of all possible pages (the "working set") and not completely uniformly.

* Correspondence address: Depto. de Computación-FCEyN_UBA. Pabellón I, Ciudad Universitaria, 1414 Buenos Aires, Argentina. E-mail: efeuerst@dc.uba.ar.

[1] This work was partially done while the author was at Dipartimento di Informatica e Sistemistica-Università di Roma "La Sapienza", partly supported by ESPRIT BRA Alcom II under contract No. 7141, and by Italian Ministry of Scientific Research Project 40% "Algoritmi, Modelli di Calcolo e Strutture Informative". Supported by EC project DYNDATA under program KIT and by UBA's "Programación para investigadores Jóvenes", project " Algoritmos efficientes para problemas on-line con aplicaciones".

The working set may vary along time, and a good heuristic is supposed to "learn" the present set, so as to be able to have a small number of faults. This type of problem belongs to a wide family of problems, called *on-line* problems, that arise in many different areas of computer science. Roughly speaking, algorithms for on-line problems try to learn from the past how the future will be. A whole research area has been developed in recent years devoted to the study of on-line problems and algorithms. The most widely accepted way of measuring the efficiency of on-line strategies is called *competitive analysis*. Among the wide literature about the subject we may cite as examples [7, 11, 15].

The enormous amount of theoretical and practical studies done on efficient paging strategies gave as a result that all operating systems provide hierarchical memory mechanisms devoted to smooth the effect on the performance of programs of slow access to some memory devices. This fact has encouraged research on efficient blocking techniques for data structures that are stored in secondary memory. This research has led to proposals of how to block or pack data in secondary memory pages so as to minimize the quantity of accesses to secondary memory done on solving some problem. Examples of this can be found in [1–3, 13]. All the previously cited works share the philosophy of blocking and/or replicating data in such a way that when the data will be requested some accesses to secondary memory will be avoided. In some cases an improvement is obtained using the fact that the exact order in which data will be accessed is known a priori (as in off-line applications like, for example, matrix multiplication), in other cases [6] that order is not completely known, but instead it is known that the requests for data will follow a specified pattern.

In this paper we apply the philosophy of Paging to the case in which to serve a request we need to bring into a fast memory of size k not a single page but a set of pages. At first sight this problem may seem a particular case of Paging, viewing each request for a set as a sequence of individual requests for the pages belonging to it. However, the cardinalities of the sets involved in each query are not fixed, and hence the advantage that could be obtained considering requests for more than one page as a look-ahead (like in [4]) cannot be considered as granted. Moreover, the more complex structure of the requests gives an extra degree of freedom concerning the cost that may be charged when a request cannot be satisfied with the contents of the cache.

1.1. *Motivations*

In many applications the execution of each task may require the presence of more than one page in fast memory to be executed. The following examples motivate the interest on studying our extension of the Paging Problem.

1. Given a graph stored in secondary memory a system must repeatedly answer queries asking for a path between pairs of nodes. The graph may be stored in such a way that a path of length l could be split among as much as l pages of secondary memory, and to perform some action on it *all* those pages need to be cached into fast memory. This problem has been studied in [8].

2. The transitive closure of a graph is stored in secondary memory as a set of pairs $\langle$*node, label of its connected component*$\rangle$. Each query specifies a set of nodes and the answer to a query is "yes" if all the nodes are in the same connected component, "no" otherwise.
3. A partially ordered set $(O, \sqsubseteq)$ of bounded dimension d is stored in secondary memory, and queries are of the form $x \sqsubseteq y$?. In this case it is known that elements of O may be labeled by d labels $l_1(x) \cdots l_d(x)$ in a way that for every pair x, y of elements of O it is verified that $x \sqsubseteq y$ if and only if for all $i = 1 \ldots d$, $l_i(x) \leqslant l_i(y)$. In this application, a subset of elements of O together with their d labels may be maintained in the cache, each query $x \sqsubseteq y$? requiring the presence of both x and y and their labels to be answered.
4. Consider the managing of a data base that is stored in secondary memory. To perform a **join** of two relations *both* involved relations must be present *simultaneously* in fast memory.

In some of the applications described above, the answer to each query is given by a set of items, each of which may occupy less than one page. Our definition allows also to treat those cases, just by replacing the cache by a portion of main memory dedicated to store a subset of the information that is stored in secondary memory. Therefore, our algorithms could be useful to reduce the number of secondary memory accesses done to answer a sequence of requests about a data structure that is stored in secondary memory, just by maintaining a small subset of it in main memory. At least, they provide a framework to analyze the possible strategies to use in practice for reducing the number of secondary memory accesses in those cases.

All our results may be also applied in a distributed framework where each processor holds a subset of pages of a virtual shared (read-only) memory. In this case, a processor may contain all the pages needed to process a particular task, but it is also possible that it will have to communicate with other processors to get the required pages. If we assume constant cost for each page transmitted through the network, maintaining an extra subset of the global memory in the private working memory will help to reduce the communication cost and improve the overall performance of the system, as in a distributed network communication plays the same bottle-neck role as secondary memory accesses in a centralized system.

1.2. Summary of results

We extend the paging problem to the case in which each query specifies a set R of pages of secondary memory that must be in a cache of size k. If R is contained in the cache no cost is charged, otherwise R must be brought into the cache and a cost is charged. We define three different cost measures, namely the Full, Uniform and Constant cost models, that are different ways of charging costs to algorithms for this problem. For each cost model, we establish lower bounds for the competitiveness of deterministic and randomized on-line algorithms and study the competitive ratios achieved by our algorithms in each case, against different types of adversaries, namely

Table 1
Summary of results – competitiveness of on-line algorithms

Problem	Lower bound	Upper bound
Full cost, deterministic	$\frac{k^2+k}{2}$	$\frac{k^2+k}{2}$
Full cost, randomized, lazy	$\frac{k}{2}\ln(k)$	$2kH_k$
Full cost, randomized, non-lazy	$\frac{k^2+k}{2}$	$\frac{k^2+k}{2}$
Uniform cost, deterministic	k	k
Uniform cost, randomized	H_k	$2H_k$
Constant cost, deterministic	k	k
Constant cost, randomized, lazy	$\lfloor \frac{k}{e} \rfloor + 1$	k
Constant cost, randomized, non-lazy	k	k

lazy and *non-lazy* adversaries (the results for randomized algorithms refer to an oblivious adversary [14]). An on-line algorithm is said to be *optimal* or *strongly competitive* if it achieves the lowest possible competitive ratio (and not an algorithm that serves the request sequence optimally). We show that our algorithms for each case are optimal or at most a constant factor away from optimal. Table 1 summarizes the results presented in this paper.

All the deterministic upper bounds presented in this paper are tight. In the case of non-lazy adversaries and for all cost models, our deterministic algorithm obtains the best possible competitive ratios achievable even by randomized algorithms, but without using randomization. This implies that against non-lazy adversaries randomization provides no help at all, a situation that appeared very seldom in the literature about on-line algorithms.

A particular case of this problem, in which each page represents the edges of a tree and requests are paths of the tree has been presented in [8]. In this paper we extend those results to the general case, and improve most of the lower bounds therein.

2. Cost models and lazy algorithms

In the traditional Paging problem a constant cost is charged to an algorithm each time it has a page fault. For our problem that may be a reasonable assumption only in some cases, while other cost models may be more suitable in different applications. Each cost model is characterized by *when* a cost is charged to an algorithm, as well as *which amount* will be charged. As we will see later, different cost models will give different results (lower and upper bounds on the competitive ratios of on-line algorithms). We consider the following cost models:

1. *Full cost model*: When a requested set R is not present in cache, charge the algorithm $|R|$, the cardinality of R.

2. *Uniform cost model*: Charge the algorithm the number of pages it brings into the cache.

3. *Constant cost model*: A unit cost is charged every time the algorithm changes its configuration, i.e. the contents of its cache.

As we said before, for each particular application one cost model may be more appropriate than the others. In the first example of the previous section the absence of a path joining two nodes may imply that a search must be done for it in the whole graph, making it reasonable to charge a cost proportional to the length of the requested path, and hence to apply the full cost model. In the fourth example it is necessary to fetch just the missing relation or relations, and hence it is reasonable to apply the uniform cost model, while in every case the constant cost model can be applied if we just want to count the number of times the computation is interrupted to do secondary memory accesses.

The uniform cost model is the one that resembles traditional Paging more. In fact we will see that lower and upper bounds in this model coincide with the values obtainable for that problem, while for the other two cost models things become different.

At this point a discussion is needed regarding the power of the adversaries that we will use in each cost model. Consider the following two restrictions that can be imposed to an algorithm for this problem:

1. Changes of configurations (that is, changes in the contents of the cache) are done only when a request produces a fault.
2. In case of a fault the request can be served only by bringing into fast memory the minimum number of pages sufficient to answer the query.

The second of the restrictions above coincides with requiring that the only pages that can be brought in the case of a fault are those belonging to the requested set.

An adversary obeying these restrictions is called a *lazy* adversary. The notion of laziness has been used in the literature on paging and the k-server problem: in [11] it has been proved that every non-lazy adversary can be transformed in a lazy one *without increasing the cost it incurs for serving any sequence of requests*. Therefore, for those two problems, it be can assumed that adversaries are lazy without loss of generality. However, the use of lazy or non-lazy adversaries for paging of sets does impose a difference. We now illustrate this.

Is it reasonable to force our adversaries to comply with the two restrictions above? We start by considering the first restriction. If the uniform or the constant cost models are considered, it can be easily seen that this constraint can be assumed to hold for every algorithm: by definition some cost is charged to an algorithm when it changes configuration, and hence there is no advantage in changing configuration between requests. In the full cost model the situation is different: as the cost is defined as a function of the faults and does not depend on the transitions between configurations, an adversary (being able to predict the following request) could move to a configuration that allows to serve the request *before* it is presented, and hence serve all requests with no cost. This is unnatural, and therefore we will always *impose* the first restriction to all algorithms.

As for the second restriction, it rules out the possibility that the adversary, being aware of which requests will be presented later, brings the necessary pages during the fault produced by a different request. In the uniform cost model an algorithm *pays* for each page that is brought, and therefore it is the same to allow this kind of behavior or not. In the other two cost models this is not true anymore: we will see that in some cases lazy adversaries are strictly less powerful than non-lazy ones. However, in some situations allowing non-lazy behavior to the adversary can be "too much", and therefore we will study the competitive ratios achievable by on-line algorithms against adversaries obeying both restrictions (lazy adversaries) and obeying only the first restriction (non-lazy adversaries).

Notice that we speak about imposing the restrictions only to adversaries. On-line algorithms will naturally obey both of them it is easy to see that otherwise their competitive ratios could only increase.

3. The algorithms

In [15] it has been shown that FIFO is optimal for the paging problem. In fact, if both the algorithm and the adversary have the same memory of size k then FIFO achieves a competitive ratio of k and this ratio is optimal since k is also a lower bound. Another k-competitive algorithm for paging called Flush-When-Full (FWF) has been presented in [10]. FWF maintains a set of marked pages. Initially, the marked pages are exactly those that are present in the cache. After each request, the marks are updated, then one page is evicted to make place for the requested page if necessary. The randomized version of FWF is called the Marking algorithm, or simply M [9]. In the latter, the choice of which unmarked page to evict is done at random uniformly among the unmarked pages. It has been proved that M is $2H_k$-competitive (where H_x denotes the xth harmonic number) and that H_k is a lower bound on the competitive ratio that can be achieved against an oblivious adversary [9].

The behavior of both FWF and M on a request for page p can be schematized as shown in Fig. 1. The two algorithms differ only in how the subroutine *choose* is implemented, that is, in the way in which the unmarked page to evict is chosen. FWF does it deterministically, following any predetermined order, while M does it randomly. Both algorithms work in *phases*, the first phase starting with the first request of the sequence and each new phase starting with the request that causes more than k pages to be marked (when the marks are deleted).

```
Mark $p$;
If more that $k$ pages are marked /* this starts a new phase */
then erase all the marks except that in $p$;
If $p$ is present in the cache
then do nothing
else choose a page $p'$ among the unmarked pages of the cache;
     evict $p'$ and bring $p$.
```

Fig. 1. Algorithms Flush-When-Full and Marking.

It is immediate to see that the lower bounds for the competitiveness of on-line strategies for the Paging problem hold also for our problem, independently of the cost model. However, we will derive higher lower bounds in some cases. As for the upper bounds, the algorithms that we propose for this problem are very simple. They are natural generalizations of FWF and M, which we call FWF-s and M-s, respectively (the 's' stands for 'set'), and their behavior on a request R is depicted in Fig. 2. Again, the difference between the algorithms is just the way in which the pages to evict are chosen. It is worthwhile to note that similar generalizations of FIFO and LRU achieve the same competitive ratios that we will prove for FWF-s.

3.1. General techniques for proofs

As their counterparts for paging, our algorithms work in *phases*. We use this concept in all our upper-bound proofs; therefore, it is useful to state this simple concept carefully: the first phase of FWF-s or M-s starts with the first request of the sequence, and a new phase starts each time the cardinality of the union of the sets requested during the current phase exceeds k; the first request of the new phase is the one that causes the marks to be deleted. For example, if $k = 5$, the cache holds pages $\{a,b,c,d,e\}$, and the sequence of requests is $q_1 = \{a,f\}$, $q_2 = \{b,g\}$, $q_3 = \{a,g\}$, $q_4 = \{a,h,i\}$, then q_4 is the first request of the second phase.

The concept of phase is important because it allows to partition any input sequence into finite subsequences for which every algorithm must have at least one fault. In general, to prove an upper bound (like in Theorems 4.2, 4.6, 5.1, 5.4 and 6.1) we show that the cost of the algorithm during a phase does not exceed some value, while the cost of any adversary is at least some other value.

In the case of lower bounds, they are proved by showing in each case how an input sequence can be constructed for every algorithm A in such a way that the ratio between the costs paid, respectively, by A and the adversary is at least some value. In general, these input sequences will be constructed by repeating an arbitrary number of times some pattern of requests for which the ratio between the costs satisfies the desired property. In some cases we name explicitly each occurrence of such pattern in

```
Mark all the pages in $R$;
If more that $k$ pages are marked /* this starts a new phase */
then erase all the marks except those in $R$;
If $R$ is present in the cache
then do nothing
else if $|R - CACHE| = x$ /* there are $x$ elements of $R$ missing */
     then *choose* $x$ pages $p_1 \dots p_x$ among
          the unmarked pages of the cache;
          evict $p_1 \dots p_x$;
          bring the pages in $R - CACHE$
```

Fig. 2. Algorithms FWF-s and M-s.

the sequence as an *epoch*, and sometimes an epoch is formed by *subepochs*. The way in which these concepts are defined depends on the characteristics of the problem at hand (cost model, lazy or non-lazy adversary, etc.).

The usual definition of competitiveness for randomized algorithms [14] states that on-line algorithm A is c-competitive if there exists a constant d such that for every finite sequence of requests

$$E[C_{\mathrm{A}} - c * C_{\mathrm{ADV}}] < d, \tag{1}$$

where C_{A} is the random variable denoting the cost charged to algorithm A, and C_{ADV} is the cost charged to the adversary. Generally, deterministic adversaries are considered, but we consider a randomized oblivious adversary (that is an adversary that generates a whole random sequence of queries before starting to serve it). Hence, C_{ADV} is also a random variable, and therefore inequality (1) becomes

$$E[C_{\mathrm{A}}] - c * E[C_{\mathrm{ADV}}] < d. \tag{2}$$

This approach is similar in essence to the approach developed in [9] of constructing a nemesis sequence for the on-line algorithm based on the possibility that an oblivious adversary has of knowing, not the exact contents of the on-line algorithm's cache but a probability distribution on it.

4. Full cost model

Theorem 4.1. *No deterministic on-line algorithm is c-competitive under the full cost model with* $c < \frac{1}{2}(k^2 + k)$, *even if lazy adversaries are considered.*

Proof. Consider a set of pages U such that $|U| \geqslant k + 1$, and suppose the adversary ADV and on-line algorithm A start with the same initial configuration, consisting of a set P of pages, $|P| = k$. The first request is for a one-page set $\{p_1\}$, $p_1 \notin P$, so both A and ADV have a fault. Making some abuse of notation, we denote as A and ADV the contents of the caches of A and ADV respectively, and hence we initially have $\mathrm{A} = \mathrm{ADV} = P$. Whatever A does to serve that request, next request will be for set $\{p_1, p_2\}$, where $p_2 \in P \cup \{p_1\}, p_2 \notin A$ (that is, p_2 is the page evicted by A). In general, the ith request, $i = 1, \ldots, k$ will be to set $\{p_1, \ldots, p_i\}$, where $p_i \in P \cup \{p_1\}$, $p_i \notin A$ (in other words, p_i is the page just evicted by A). A will fault on each request, with a cost equal to $\sum_{i=1}^{k} i = \frac{1}{2}(k^2 + k)$, while ADV can serve the first request moving to configuration $\{p_1, \ldots, p_k\}$ (pages $p_2, \ldots, p_k$ were originally in ADV's cache, and therefore the behavior of ADV is lazy), serving all the requests with cost 1. After the kth request both algorithms are in the same configuration, so the same pattern of requests can be repeated. This leads to an arbitrarily long sequence in which the cost of A is at least $\frac{1}{2}(k^2 + k)$ times the cost of the adversary. □

Next theorem shows that FWF-s is optimal for this problem.

Theorem 4.2. *FWF-s is $\frac{1}{2}(k^2+k)$-competitive under the full cost model.*

Proof. It is obvious that the adversary's cost is at least 1 for each phase. We will bound the cost of FWF-s during a phase using the following potential function:

$$\Phi = \frac{s^2+s}{2},$$

where s is the cardinality of the union of all the sets that were requested so far in the phase. $\Phi = 0$ before the first request of the phase (or, in other words, after the last request of the previous phase), and $\Phi \leqslant \frac{1}{2}(k^2+k)$ after the last request of the phase.

Consider the first request of the phase, namely R_1, that is served by FWF-s with a cost of $|R_1|$. As $|R_1| \geqslant 1$, $|R_1| \leqslant \frac{1}{2}(|R_1|^2 + |R_1|) = \Delta\Phi$; that is, the cost of the first request of a phase is not more than the variation in the potential.

Let us now consider the other requests of the phase for which FWF-s pays a positive cost: if FWF-s brings i new pages as a result of a fault on request R, we have that

$$\Delta\Phi = \frac{(s+i)^2+s+i}{2} - \frac{s^2+s}{2} = \frac{2is+i^2+i}{2} \geqslant \frac{2is+2i}{2} = is+i \geqslant s+i.$$

But $s+i \geqslant |R|$, because R is included in the union of all the sets requested in the phase and the i pages that are brought. Then the increment in the potential is at least the cost of the operation.

The total cost of a phase is at most the difference between the final and the initial potential, and hence the total cost charged to FWF-s is upper-bounded by $\frac{1}{2}(k^2+k)$. □

To prove a lower bound for randomized algorithms, we first need the following lemma:

Lemma 4.3. *After sufficiently many requests for subsets of a set Q, $|Q| \leqslant k$, every competitive on-line algorithm must have Q in its own cache with probability one.*

Proof. If algorithm A never has all the pages of Q in its own cache with probability one, then there will always be a request for a subset of Q for which A's expected cost is positive. As an adversary could serve all these requests for free, A would not be competitive. □

Theorem 4.4. *No randomized on-line strategy is c-competitive with $c < \frac{1}{2}k \ln k$ against a lazy oblivious adversary under the full cost model, for sufficiently large k.*

Proof. Consider the same initial configuration of both the cache of ADV and A, consisting on a set P of pages, $|P| = k$. P may be partitioned in two subsets X and Y of cardinalities l and $k-l$, respectively, $Y = y_1 \ldots y_{k-l}$ (l will be determined later).

Consider a random sequence of queries made of an arbitrarily large number of epochs, each epoch done in this way: the adversary asks for set $\{p\}$, $p \notin P$, and evicts the page y_j, for some $j \in \{1, \ldots, k-l\}$ uniformly chosen at random. Then the epoch continues with $k-l-1$ subepochs, where each subepoch consists of

- zero or more *Type* 1 requests, that are requests for sets $X \cup \{y_i\}$, $i \neq j$ already requested in the epoch, followed by
- a *Type* 2 request, that is a request for a set $X \cup \{y_{i'}\}$, $i' \neq j$ not yet requested in the epoch.

Type 1 requests will be repeated till A has (with probability one) the pages needed to answer *all* the requests so far in the epoch. This can be done by Lemma 4.3, and all these requests cost nothing to the adversary. After the epoch page p is "renamed" y_j so as to have again the initial configuration.

The expected cost charged to the adversary in each epoch is 1 (in fact 1 is the exact cost). The expected cost for A during the epoch is at least 1 plus $(l+1)$ times the expected number of faults of the epoch. If A ever evicts a page $x \in X$ during the epoch, it will surely fault for the following request. Hence, without loss of generality we may suppose that the pages in X will never be evicted by the on-line algorithm, as the expected number of faults in this case is not greater than the number of faults it would have otherwise. Therefore, l slots of the cache will be always occupied by the pages in X, and then the expected number of faults on the epoch depends on the probability that the pages y_j are present in the other part of the cache, of size $k-l$. The expected number of faults for this sequence is one less than the expected number of faults for an epoch of the Paging problem with cache of size $k-l$. It has been proved [9] that this number is greater than H_{k-l-1}. Hence, we have that the expected cost for the epoch is greater than $1+(l+1)(H_{k-l-1}-1)$.

Turning to inequality (2), it is obvious that c, the competitiveness coefficient, must necessarily be greater or equal than the maximum possible value of the previous expression for any algorithm to satisfy the inequality.

That maximum value is asymptotically $O(k\ln(k))$. For values of k around some hundreds, it achieves a value strictly greater than $\frac{1}{2}k\ln(k)$. In any case, letting $l = \frac{1}{2}k$ it is greater than $1+\frac{1}{2}k(\ln(\frac{1}{2}k)-1)$. □

In the following we will prove that M-s is nearly optimal under the Full cost model. We need some preliminary definitions.

During a phase, requests may involve three different kinds of pages:

- *marked* pages, that are pages already used during the current phase,
- *clean* pages, that are pages that where not used during the current phase nor in the previous one, and
- *stale* pages, that are pages that where used in the previous phase but not during the current phase.

In a similar way, we can divide the requests into four types:

- *clean* requests, that use clean and eventually marked pages,
- *stale* requests, that use stale and eventually marked pages,
- *mixed* requests, that use clean and stale pages, and eventually marked pages, and
- *marked* requests, that use only marked pages.

Without loss of generality, we can suppose that there will not be requests of the last kind, as by definition M-s would answer them with no cost. It is easy to see that each phase starts with a clean or a mixed request, that is, with a request that involves at least one clean page.

Lemma 4.5. *The expected number of faults of M-s during a phase is maximized if each query involves at most one clean or stale page.*

Proof. Suppose a query q involves $r > 1$ stale or clean pages. We will prove the claim of the lemma for $r = 2$, it can be easily extended for any other value by induction. Let a and b denote the pages. The expected number of faults F_q for answering q is equal to the probability that at least one of a and b is not present in the cache. Hence, we can write $F_q = P(\bar{a} \wedge \bar{b}) + P(\bar{a} \wedge b) + P(a \wedge \bar{b})$.

If the query q is divided into two queries q_a, q_b, each of them including, respectively, a and b, the expected number F_{q_a,q_b} of faults for both queries is $F_{q_a,q_b} = P(\bar{a} \wedge \bar{b}) + P(\bar{a} \wedge b) + P'(\bar{b})$ where P' denotes the probability *after* request q_a. We have that $P'(\bar{b}) \geqslant P(a \wedge \bar{b}) + P(\bar{a} \wedge \bar{b})$, and hence $F_{q_a,q_b} \geqslant F_q$. □

Theorem 4.6. *M-s is $2kH_k$ competitive against lazy adversaries under the full cost model.*

Proof. We will first bound the expected number of faults of M-s during a phase. By Lemma 4.5 we can suppose that each request involves only one clean or stale page, and eventually some marked pages. This proof is similar to the proof of the competitiveness of the Marking algorithm in [9]. The expected number of faults of M-s during the phase is smaller than the number of faults it would have if all the clean requests of the phase were done before any stale request. In such a case M-s has one fault for each clean request and an expectation of fault for each stale request equal to the probability that the involved stale page is not present in the cache. This probability is c/s, where c is the number of clean pages requested so far and s is the current number of stale pages. Hence, the expected number of faults of M-s during the phase is less than

$$c + \frac{c}{k} + \frac{c}{k-1} + \cdots + \frac{c}{c+1} = c\left(1 + \frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{c+1}\right)$$
$$= c(1 + H_k - H_c) \leqslant cH_k.$$

As for the cost charged to the adversary, it can be proved in the same way as in [9] that it is at least $\frac{1}{2}c$, half of the number of clean pages requested in the phase. In this part of the proof the lazy behavior of the adversary is crucial, as otherwise we could only prove that the cost incurred by the adversary during a phase is at least 1.

We have proved that the ratio between the expected number of faults of M-s and the adversary is smaller or equal than $2H_k$. In the full cost model, the maximum cost charged to M-s for a fault is k, while the minimum cost for a fault of the adversary is 1. Hence, if we consider the cost instead of the number of faults, we have that the cost of M-s is less than $2kH_k$ times the cost of the adversary, that is, M-s is $2kH_k$-competitive. □

The two previous theorems assert that M-s is at most a factor of 4 away from optimality.

We will now see that randomization is of no use if non-lazy adversaries are allowed.

Theorem 4.7. *No randomized on-line strategy is c-competitive under the full cost model against an oblivious adversary with* $c < \frac{1}{2}(k^2 + k)$.

Proof. The proof is based on a sequence of requests similar to that of Theorem 4.1, i.e. consisting in an arbitrary number of epochs, each epoch starting with both algorithms in the same configuration and a request for a page p_1 not present in that configuration; then to a set $\{p_1, p_2\}$, etc., p_i is chosen so that it minimizes (among all the pages) the probability of being present in A's cache after the request for $\{p_1, \ldots, p_{i-1}\}$. If the number of pages is sufficiently big, that minimum probability can be as small as desired, and hence the cost of A during the epoch is as close to $\frac{1}{2}(k^2 + k)$ as desired. When it faults on request $\{p_1\}$, the adversary can bring all the set $\{p_1, \ldots, p_k\}$ together, with a cost of 1. □

5. Uniform cost model

Theorem 5.1. *FWF-s is k-competitive under the uniform cost model.*

Proof. The total cost incurred by FWF-s during a phase is at most k, since each page that is requested during the phase is kept in cache till the end of the phase. The adversary has a cost of at least 1 per phase, and hence the thesis follows. □

In the reminder of this section we will show that M-s achieves a competitiveness factor of $2H_k$ under this cost model.

We will restrict ourselves to a particular kind of adversary. This adversary, before every query R with $|R| > 1$ (from now on these kind of queries will be referred to as *big* queries), requests all the pages that form R individually in any order. It is easy to see that this is not a real restriction, as the cost charged to this adversary is exactly the same that it would be charged to a general adversary, while the cost charged to M-s is at least the same it would have to pay in the general sequence (because in the uniform cost model an algorithm is charged exactly the number of missing pages, and in this case it will be charged at least that number). Moreover, we shall consider a lazy adversary (as we have already mentioned, for the uniform cost model this can be done without loss of generality).

As in the previous section, pages may be divided in marked, clean or stale, and requests in clean, stale, mixed and marked. We need two preliminary lemmas, the proof of the first of them is trivial and is omitted.

Lemma 5.2. *Each phase starts with a clean request.*

Lemma 5.3. *Every big query but the first one of the phase is served by M-s with no cost.*

Proof. By definition of M-s and the kind of adversary we are considering, all big requests but the first one of the phase find all the pages involved in the query already marked, and hence present. □

Theorem 5.4. *M-s is $2H_k$ competitive under the uniform cost model.*

Proof. By Lemma 5.3 we can eliminate from the sequence of requests all big queries except the first one of each phase, and hence the sequence of requests of a phase can be seen in the following way:

$$c_1 \dots c_i sc_1 \dots sc_j B c_1^1 c_2^1 \dots c_{x_1}^1 s_1^1 s_2^1 \dots s_{y_1}^1 c_1^2 c_2^2 \dots c_{x_2}^2 s_1^2 s_2^2 \dots s_{y_2}^2 \dots,$$

where $c_1 \dots c_i$ are requests for clean pages, $sc_1 \dots sc_j$ are requests for clean or stale pages, and B is either empty or a big query involving pages $c_1, \dots, c_i, sc_1, \dots, sc_j$ and eventually some other stale pages (denoted $s', s'', s''', \dots$) requested in the final part of the previous phase. Queries c_m^n and s_m^n are requests for one-page sets. Note that all other big queries eventually present in the phase have been omitted, due to Lemma 5.3, as well as requests for marked pages, that induce no cost to M-s.

Since a marked page is not evicted it follows that the cost charged to M-s for this sequence is less that the cost it would be charged for the sequence

$$c_1 \dots c_i sc_1 \dots sc_j s' s'' s''' \dots c_1^1 c_2^1 \dots c_{x_1}^1 s_1^1 s_2^1 \dots s_{y_1}^1 c_1^2 \dots c_{x_2}^2 s_1^2 \dots s_{y_2}^2 \dots .$$

The rest of this proof is similar to the proof of the competitiveness of M-s for the full cost model. The expected cost charged to M-s for this sequence is smaller than the cost it would be charged if all the clean pages of the sequence where requested before any request for a stale page. In such a sequence the expected cost charged to M-s during each phase is less than

$$c + \frac{c}{k} + \frac{c}{k-1} + \dots + \frac{c}{c+1} = c\left(1 + \frac{1}{k} + \frac{1}{k-1} + \dots + \frac{1}{c+1}\right)$$
$$= c(1 + H_k - H_c) \leqslant cH_k.$$

As for the cost charged to the adversary, we can prove in the same way as in [9] that it is at least $\frac{1}{2}c$, half of the number of clean pages requested in the phase.

This completes the proof of the $2H_k$-competitiveness of M-s. □

We recall that in [9] it has been proved that H_k is a lower bound for the competitiveness of any algorithm for the Paging problem, and hence it is also a lower bound for our problem under this cost model. Therefore, M-s is at most a factor of 2 away from optimality.

6. Constant cost model

Theorem 6.1. *FWF-s is k-competitive under the constant cost model, even against non-lazy adversaries.*

Proof. Under this cost model the cost incurred by FWF-s corresponds to the number of faults it has. The maximum number of faults FWF-s may have during a phase is k, while the adversary faults at least once. □

Theorem 6.2. *No randomized on-line strategy is c-competitive against an oblivious adversary with* $c < (\lfloor k/e \rfloor + 1)$ *(e is the base of the natural logarithm) under the constant cost model, even if lazy adversaries are considered.*

Proof. Consider an initial configuration in which ADV and A have the same k pages in their caches. Consider a random sequence of queries made of an arbitrarily large number of epochs, each epoch done in this way: the first request of the epoch consists of a set of size l (the value of l will be determined later) disjoint from the set of pages present in both caches. To serve this request, the adversary evicts l pages randomly chosen. The epoch continues with $k-l$ subepochs, each subepoch done in the following way: first, all the requests of the preceding subepochs of the epoch are repeated as in the proof of Theorem 4.4, secondly, a request is done for a one-page set formed by one of the $k-l$ pages not evicted by the adversary. The cost paid by the adversary during the epoch is 1, and after an epoch the configuration of the on-line algorithm will coincide with that of the adversary. Hence, after the end of an epoch a new epoch can start, and the proof of the theorem reduces to showing that the expected cost charged to any on-line algorithm during an epoch is at least $k/e+1$.

Every on-line algorithm will fault on the first request of the epoch, with a cost of 1. As for the second part of the sequence, the expected cost depends on the probability that, for each of the $k-l$ subepochs, the requested page is present at the moment it is requested for the first time in the epoch. The expected cost for this part of the sequence will hence be greater than

$$
\begin{aligned}
1+\frac{l}{k}+\frac{l}{k-1}+\cdots+\frac{l}{l+1} &= 1+l\left(\frac{1}{k}+\frac{1}{k-1}+\cdots+\frac{1}{l+1}\right)\\
&= 1+l(H_k-H_l).
\end{aligned}
$$

The maximum value for this expression is for $l=k/e$, when it assumes the value $1+k/e$. This completes the proof of the theorem. □

If we allow non-lazy adversaries, we can prove the following result:

Theorem 6.3. *No randomized on-line strategy is c-competitive against a non-lazy oblivious adversary with* $c<k$ *under the constant cost model.*

Proof. Starting with the same initial configuration, each of the first k requests may be to the page that minimizes the probability of being present in the on-line algorithm's cache. If the universe is sufficiently big, such probability can be as close to 0 as desired, and hence the cost incurred by the on-line algorithm is as close to k as desired. On the other hand, the adversary may bring that set of pages when it faults on the first request. Repeating this pattern an arbitrary number of times we get a sequence in which the ratio between the costs is k. $\square$

The previous theorems tell us that for this problem the use of randomization provides no help against non-lazy adversaries, while against lazy adversaries it may only allow to improve the performance of on-line algorithms at most by a factor of e. We can only show that M-s is k-competitive under this cost model: with the same arguments as in the proofs of competitiveness of M-s for the other cost models, we can show that the expected cost for a phase in which l clean pages are requested is not more than $l + l(H_k - H_l)$. For this cost model we can only assume that the adversary's cost for the phase is at least 1, and hence the worst-case ratio among the costs is when $l = k$. In that case we have that the competitiveness of M-s is k, and hence at most a factor of e away from optimality. Note however that a competitive ratio of k is achieved also by the deterministic algorithm FWF-s.

Acknowledgements

Some of the ideas present in this work arose while studying a particular case with Alberto Marchetti-Spaccamela, to whom I am grateful. I would like to thank the two anonymous referees for their useful suggestions on how to improve the preliminary version of this work.

References

[1] A. Aggarwal, B. Alpern, A.K. Chandra and M. Snir, A model for hierarchical memory, in: *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 305–314.

[2] A. Aggarwal and A.K. Chandra, Virtual memory algorithms, in: *Proc. 20th Ann. ACM Symp. on Theory of Computing* (1988) 173–185.

[3] A. Aggarwal, A.K. Chandra and M. Snir, Hierarchical memory with block transfer, in: *Proc. 28th Ann. Symp. on Foundations of Computer Science* (1987) 204–216.

[4] S. Albers, The influence of lookahead in competitive paging algorithms, in: *Proc. 1st Ann. European Symp. on Algorithms*, Lecture Notes in Computer Science, Vol. 726 (Springer, Berlin, 1993) 1–12.

[5] L.A. Belady, A study of replacement algorithms for virtual storage computers, *IBM System J.* **5** (1966) 78–101.

[6] A. Borodin, S. Irani, P. Raghavan and B. Schieber, Competitive paging with locality of reference, in: *Proc. 23rd Ann. ACM Symp. on Theory of Computing* (1991) 249–259.

[7] A. Borodin, N. Linial and M. Saks, An optimal online algorithm for metrical task systems, in: *Proc. 19th Ann. ACM Symp. on Theory of Computing* (1987) 373–382.

[8] E. Feuerstein and A. Marchetti-Spaccamela, Memory paging for connectivity and path problems in graphs, in: *Proc. 4th Ann. Symp. on Algorithms and Computation*, Lecture Notes in Computer Science Vol. 762 (Springer, Berlin, 1993) 416–425.

[9] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator and N.E. Young, Competitive paging algorithms, *J. Algorithms* **12** (1991) 685–699.

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

[11] M.S. Manasse, L.A. McGeoch and D. Sleator, Competitive algorithms for server problems, *J. Algorithms* **11** (1990) 208–230.

[12] L.A. McGeoch and D. Sleator, A strongly competitive randomized paging algorithm, *Algorithmica* **6** (1989) 816–825.

[13] M. Nodine, M. Goodrich and J.S. Vitter, Blocking for external Graph Searching, Technical Report CS-92-44, Brown University, 1992.

[14] P. Raghavan and M. Snir, Memory versus randomization in on-line algorithms, IBM Research Report RC 15622, 1990.

[15] D. Sleator and R.E. Tarjan, Amortized efficiency of list update and paging algorithms, *Comm. ACM* **28** (1985) 202–208.