DIMACS Technical Report 97-69
October 1997

Uniform Multipaging Reduces to Paging

by

Vincenzo Liberatore[1,2]
Department of Computer Science,
Hill Center, Rutgers University,
New Brunswick, NJ 08903.
E-mail: `liberato@cs.rutgers.edu`.
URL: `http://www.cs.rutgers.edu/~liberato/`.

[1]Affiliated Graduate Student Member
[2]Research supported by a DIMACS Graduate Fellowship.

ABSTRACT

Multipaging is the paging problem when more than one page can be requested at each step [3]. In the uniform cost model, a paging algorithm is charged for the number of pages loaded from disk to fast memory. In this note, we establish a general reduction from uniform multipaging to paging. As a consequence, we obtain the first strongly competitive randomized algorithm for uniform multipaging.

1 Introduction

The *paging problem* is defined as follows. A set of m pages is used by a program. However, only $k < m$ pages can be kept in fast memory at any given step. The remaining pages are stored on disk. At each step, a page is requested by the program. If that page is in fast memory, it can be supplied to the program with no further ado. However, if the requested page is not in fast memory, a *page fault* occurs and the missing page has to be loaded from disk. In turn, a page might have to be evicted from fast memory in order to accommodate the requested page. Since page faults involve disk accesses, page faults are extremely expensive. The objective of a paging strategy is to select the evicted page in order to minimize the number of page faults.

The paging problem is an *on-line problem* because at each step the paging strategy does not know which requests will be generated in the future. The performance of on-line algorithms is often measured in terms of their *competitive ratio*, which is defined as the ratio of the algorithm cost over the cost of the optimum off-line strategy [5, 9]. An on-line strategy is said to be *strongly competitive* if it achieves the best possible competitive ratio [5, 6]. On-line paging and its competitiveness analysis have attracted substantial attention in the literature [4, 5, 8, 9].

Multipaging is a version of the paging problem that allows for more than one page to be requested in one step. All the pages referenced in one request must be simultaneously present in fast memory in order for the request to be satisfied. Multipaging has been recently proposed to model paging of graph algorithms and database applications [3]. The cost of a multipaging algorithm is, in the *uniform* cost model, equal to the number of pages brought into fast memory. The off-line version of uniform multipaging is a simplification of local register allocation [2].

Uniform multipaging was observed to be similar to traditional paging. However, paging and multipaging were not tied by a formal equivalence statement. As a consequence, multipaging algorithms were analyzed from first principles, and without using known facts from paging. Furthermore, paging has a strongly competitive randomized algorithm [8], whereas no such algorithm was known for multipaging.

In this note, we prove that

Theorem 1 *If there is a deterministic (randomized) c-competitive paging algorithm, then there is a deterministic (randomized) c-competitive multipaging algorithm.*

Consequently, competitiveness results for multipaging can be deduced from competitiveness results for paging. In fact, the proof of Theorem 1 is constructive, and allows us to transform any competitive algorithm for paging into a competitive algorithm for multipaging. As a result, all known results on uniform multipaging [3] can be deduced as a corollary of Theorem 1. Furthermore, we obtain the first strongly competitive randomized algorithm for multipaging.

The gist of the proof of Theorem 1 is to transform a multipage request s into a paging request sequence by demanding all the pages in s twice. In the next section, we will prove Theorem 1.

2 From Multipaging to Paging

In this section, we will show how to reduce uniform multipaging to the traditional paging problem.

Paging and multipaging algorithms can be assumed without loss of generality to be *lazy* [7, 3], that is,

- Fast memory changes only in response to page faults, and
- No unrequested page is loaded into fast memory.

If G is a paging or multipaging algorithm, $c(G)$ will denote the total cost of G.

We will use the following notation. Given the sequences $\sigma_1, \sigma_2, \ldots, \sigma_h$, the sequence $\sigma = \sigma_1\sigma_2 \ldots \sigma_m \stackrel{\mathrm{def}}{=} \circ_{i=1}^{h} \sigma_h$ is the concatenation of $\sigma_1, \sigma_2, \ldots, \sigma_h$. If σ is a sequence, then $\sigma^h \stackrel{\mathrm{def}}{=} \circ_{i=1}^{h} \sigma$. Let $\sigma = (s_1, s_2, \ldots, s_m)$ be a multipaging reference sequence with $s_i = \{r_{i1}, r_{i2}, \ldots, r_{il_i}\}$. It will always be understood that $l_i \leq k$ for all i's, where k is the fast memory size. Define $s_i'' = (\circ_{j=1}^{l_i} r_{ij})$ and $s_i' = (s_i'')^2$. The sequence s_i' repeats twice the requests in s_i in any arbitrary order. Define $\sigma' = \circ_{i=1}^{m} s_i'$. The sequence σ' is an ordinary paging request sequence.

The rest of this section is organized as follows: first, we will obtain an optimum off-line algorithm for multipaging. Then, we will show how to transform an on-line paging algorithm into an on-line multipaging algorithm with no greater cost.

Let OPT$'$ be Beladi's optimum off-line algorithm for paging: in response to a page fault, evict the page that will be referenced furthest in the future [1]. Define the multipaging algorithm OPT as the algorithm that, in response to s_i, brings into fast memory exactly the pages that are in the fast memory of OPT$'$ at the end of s_i'.

Proposition 2 OPT *is an optimum multipaging algorithm and* $c(\mathrm{OPT}) = c(\mathrm{OPT}')$.

Proof. First, we will prove that OPT is a correct multipaging algorithm, then that $c(\mathrm{OPT}) = c(\mathrm{OPT}')$, and finally that OPT is optimum. All claims are proved by induction on the length of σ.

Let F_i be the set of pages in s_i but not in the main memory of OPT$'$ immediately before s_i'. If $F_i = \emptyset$, the fast memory contents do not change for either algorithm and the result follows by the induction hypothesis. Suppose $F_i \neq \emptyset$. The algorithm OPT$'$ does not evict any page in s_i during the first repetition of s_i'' because there are other pages requested further in the future. Therefore, OPT$'$ holds all the pages in s_i in fast memory at the beginning of the second repetition of s_i''. Moreover, OPT$'$ keeps in main memory all the pages in F_i throughout the second repetition of s_i'' in s_i' because OPT$'$ is lazy. Therefore, OPT$'$ has all the requested pages in the fast memory at the end of s_i', and OPT is a correct multipaging algorithm.

The cost of OPT$'$ during s_i' is equal to $|F_i|$. But the fast memory contents of OPT$'$ at the beginning of s_i' are exactly the same as those of OPT immediately before s_i, so that the same cost $|F_i|$ is charged to OPT for s_i. Then, $c(\mathrm{OPT}) = c(\mathrm{OPT}')$ by summing over the whole sequence.

Suppose now that there is a multipaging algorithm G that is cheaper than OPT, and assume without loss of generality that G is lazy. Then, we will construct a paging algorithm G' that is cheaper than OPT$'$ on σ' as follows. Let E_i be the set of pages evicted by G in response to s_i. If $E_i = \emptyset$, the result follows by the induction hypothesis. Suppose $E_i \neq \emptyset$. Then, G' evicts the pages in E_i at the beginning of s'_i and loads the requested pages that are not present in fast memory. G' can now satisfy s'_i with no extra cost. G' is a correct paging algorithm and costs as much as G, which contradicts the optimality of OPT$'$. □

We will now show how to transform any lazy on-line paging algorithm G' into an on-line multipaging algorithm G. The tricky part of the construction arises when G' evicts a page in s_i during the first repetition of s''_i. The transformation of OPT$'$ into OPT did not present such a problem. Indeed, the optimum algorithm OPT$'$ would not evict a page only to reload it very soon. On the other hand, an on-line algorithm G' might perform such evictions because it does not know the very next few requests.

Let M_i be the set of pages in the fast memory of G' immediately before s'_i. We will maintain the invariant that M_i are also the memory contents of G immediately before s_i. Let ρ_i be the number of pages requested in s_i, but not present in M_i. Let A_i be the set of all pages evicted by G' during s'_i.

Let B_i be the set of pages in $A_i \cap s_i$ evicted by G' during the first repetition of s''_i. Every page in B_i is reloaded during the second repetition of s''_i, so that the total cost of G' on s'_i is at least $\rho_i + |B_i|$.

Lemma 3 *The following hold:*

1. $M_{i+1} \subseteq (M_i \cup s_i) - (A_i - s_i)$.

2. $|B_i| \geq \rho_i - |A_i - s_i|$

Proof. Both results follows from the fact that G' is lazy and does not reload any evicted unrequested page. This observation immediately yields $M_{i+1} \subseteq (M_i \cup s_i) - (A_i - s_i)$.

G' incurs exactly ρ_i faults during the first repetition of s''_i. G' can satisfy at most $|A_i - s_i|$ of those faults by evicting a page not in s_i, so that $|B_i| \geq \rho_i - |A_i - s_i|$. □

We define G as follows. In response to s_i, G evicts $A_i - s_i$ plus a set C_i of $\max\{0, \rho_i - |A_i - s_i|\} \leq |B_i|$ pages chosen arbitrarily in $M_i - s_i$. Immediately after s_i, G evicts all the pages not in M_{i+1} and reloads $C_i \cap M_{i+1}$, which restores the invariant that the two memory configurations be equal. The cost of G on s_i is at most $\rho_i + |C_i| \leq \rho_i + |B_i|$, which is no more than the cost of G' on s'_i. Then, $c(G) \leq c(G')$ by summing over the whole sequence.

Remark. Although G is not lazy, it is routine to modify it into a lazy algorithm without increasing its cost [3].

Proof of Theorem 1. Let G' be a c-competitive paging algorithm, and G the corresponding multipaging algorithm defined above. Then, for some constant b,

$$c(G) \leq c(G') \leq c \cdot c(\mathrm{OPT}') + b = c \cdot c(\mathrm{OPT}) + b \; , \tag{1}$$

where the first inequality follows from the discussion above, the second by the definition of competitiveness and the third from Proposition 2. Inequality (1) establishes that G is c-competitive, and the theorem is proved. □

Theorem 1, the algorithm in [8], and the lower bound in [4] yield the following

Corollary 4 *There is a randomized algorithm for the multipaging problem that is strongly competitive against an oblivious adversary.*

Acknowledgements

We thank Martin Farach for proofreading this paper.

References

[1] L. A. Belady. A study of replacement algorithms for a virtual storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[2] M. Farach and V. Liberatore. On local register allocation. Technical Report TR97-33, DIMACS, 1997.

[3] E. Feuerstein. Paging more than one page. *Theoretical Computer Science*, 181(1):75–90, July 1997.

[4] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, Dec. 1991.

[5] S. Irani and A. R. Karlin. Online computation. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-hard problems*, chapter 13, pages 521–564. PWS Publishing Company, Boston, 1997.

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

[7] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[8] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

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