Thomas Lengauer (Ed.)

Algorithms – ESA '93

First Annual European Symposium
Bad Honnef, Germany
September 30 - October 2, 1993
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, Germany

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editor

Thomas Lengauer
Gesellschaft für Mathematik und Datenverarbeitung mbH
Institut für methodische Grundlagen (I1)
Schloß Birlinghoven, D-53732 Sankt Augustin, Germany

CR Subject Classification (1991): F.2, G, I.3.5

ISBN 3-540-57273-2 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57273-2 Springer-Verlag New York Berlin Heidelberg

Printed in Germany

Typesetting: Camera-ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Preface

The papers in this volume were presented at the First Annual European Symposium on Algorithms (ESA'93), held September 30–October 2, 1993, in Bad Honnef/Bonn, Germany. The symposium is intended to be an annual series of international conferences, held in early fall, that cover the field of algorithms. Within the scope of the symposium lies all research on algorithms, theoretical as well as applied, that is carried out in the fields of computer science and discrete applied mathematics. The symposium aims to cater to both of these research communities and to intensify the exchange between them.

The program committee met April 30, 1993, and selected the 35 contributed papers in this volume from 101 abstracts submitted in response to the call for papers. The selection was based on originality, quality, and relevance to the study of algorithms. It is anticipated that most of the submissions will appear in a more polished and complete form in scientific journals. The conference program also included invited lectures by Michael Paterson (Coventry): Evolution of an Algorithm, Alexander Schrijver (Amsterdam): Complexity of Disjoint Paths Problems in Planar Graphs, and Michael S. Waterman (Los Angeles): Sequence Comparison and Statistical Significance in Molecular Biology.

We wish to thank all members of the program committee, all those who submitted abstracts for consideration, our referees and colleagues who helped in the evaluations of the abstracts, and the many individuals who contributed to the success of the conference.

We would like to acknowledge the help of the following sponsoring institutions and corporations: Gesellschaft für Informatik (GI), Rheinische Friedrich-Wilhelms-Universität Bonn, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Association for Computing Machinery (ACM) Special Interest Group for Algorithms and Computation Theory, and European Association for Theoretical Computer Science (EATCS).

Finally we would like to thank the following people for their extraordinary engagement in organizing the conference: Christine Harms, Brigitte Hönig, Luzia Sassen-Heßeler, and Egon Wanke.

Bonn, September 1993 — Thomas Lengauer

Program Committee

Rainer Burkard (Graz)
Andras Frank (Budapest)
Gaston Gonnet (Zürich)
Mark Jerrum (Edinburgh)
Jan van Leeuwen (Utrecht)
Thomas Lengauer (Bonn, Chairman)
Fabrizio Luccio (Pisa)
Jiří Matoušek (Praha)
Kurt Mehlhorn (Saarbrücken)
Michel Minoux (Paris)
Miklos Santha (Orsay)
Paul Spirakis (Patras)
Esko Ukkonen (Helsinki)

Additional Referees

J. Almeida
G. Benson
A. Bertossi
S. Boucheron
L. Caudal
J.M. Couveignes
C. Crepeau
M. Crochemore
H. de Fraysseix
T. Elomaa
M. Farach
P. Ferragina
P. Fiorini
J.L. Fouquet
L. Gargano
D. Gpuyou-Beauchamps
R. Grossi
S. Irani
A. Joux
K. Kalorkoti
S. Kannan
C. Kenyon
B. Klinz
M. Leoncini
Y. Manoussakis
O. Nurmi
M. Nykänen
L. Pagli
M. Papatriantafillon
T. Pasanen
V. Paschos
U. Pferschy
P. Raghavan
S. Rao
M. Regnier
F. Rendl
G. Rote
R. Rudolf
R. Saad
A. Slissenko
M. Teillaud
P. Tsigas
J. van der Veen
J. Vilo
B. von Stengel
J. von zur Gathen
G.J. Wöginger
T. Wanrow
V. Zissimopoulos

Table of Contents

The Influence of Lookahead in Competitive Paging Algorithms

(Extended Abstract)

Susanne Albers*

Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany

Abstract. We introduce a new model of lookahead for on-line paging algorithms and study several algorithms using this model. A paging algorithm is *on-line with strong lookahead l* if it sees the present request and a sequence of future requests that contains l pairwise distinct pages. These pages also differ from the page requested by the present request. We show that strong lookahead has practical as well as theoretical importance and significantly improves the competitive factors of on-line paging algorithms. This is the first model of lookahead having such properties. In addition to lower bounds we present a number of deterministic and randomized on-line paging algorithms with strong lookahead which are optimal or nearly optimal.

1 Introduction

In recent years the competitive analysis of on-line algorithms has received much attention [ST85, KMRS88, MMS88, BBKTW90, G91]. Among on-line problems, the *paging problem* is of fundamental interest. Consider a two-level memory system which has a fast memory that can store k pages and a slow memory that can manage, basically, an unbounded number of pages. A sequence of requests to pages in the memory system must be served by a paging algorithm. A request is served if the corresponding page is in fast memory. If the requested page is not stored in fast memory, a *page fault* occurs. Then a page must be evicted from fast memory so that the requested page can be loaded into the vacated location. A paging algorithm specifies which page to evict on a fault. The cost incurred by a paging algorithm equals the number of page faults. A paging algorithm is *on-line* if it determines which page to evict on a fault without knowledge of future requests.

We analyze the performance of on-line paging algorithms using *competitive analysis* [ST85, KMRS88]. In a competitive analysis, the cost incurred by an on-line algorithm is compared to the cost incurred by an *optimal off-line* algorithm. An optimal off-line algorithm knows the entire request sequence in advance and can serve it with minimum cost. Let $C_A(\sigma)$ and $C_{OPT}(\sigma)$ be the cost of the on-line algorithm A and the optimal off-line algorithm OPT on request sequence σ. Then the algorithm A is c-competitive, if there exists a constant a such that

$$C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$$

for all request sequences σ. The *competitive factor* of A is the infimum of all c such that A is c-competitive. If A is a randomized algorithm, then $C_A(\sigma)$ is the

* *This work was done while the author was a student at the Graduiertenkolleg Informatik, Universität des Saarlandes, and was supported by a graduate fellowship of the Deutsche Forschungsgemeinschaft.*

expected cost incurred by A on request sequence σ. In this paper we evaluate the performance of randomized on-line algorithms only against the *oblivious adversary* (see [BBKTW90] for details). Belady [B66] has exhibited an optimal off-line paging algorithm which is also called the MIN algorithm. On a fault, MIN evicts the page whose next request occurs farthest in the future.

The paging problem (without lookahead) has been studied intensively. Sleator and Tarjan [ST85] have demonstrated that the well-known replacement algorithms LRU (Least Recently Used) and FIFO (First-In First-Out) are k-competitive. They have also proved that no on-line paging algorithm can be better than k-competitive; hence LRU and FIFO achieve the best competitive factor. Fiat *et al.* [FKLSY91] have shown that no randomized on-line paging algorithm can be better than $H(k)$-competitive against an oblivious adversary. Here $H(k) = \sum_{i=1}^{k} 1/i$ denotes the kth harmonic number. They have also given a simple replacement algorithm, called the MARKING algorithm, which is $2H(k)$-competitive. McGeoch and Sleator [MS91] have proposed a more complicated randomized paging algorithm which achieves a competitive factor of $H(k)$.

In this paper we study the problem of *lookahead* in on-line paging algorithms. An important question is, what improvement can be achieved in terms of competitiveness, if an on-line algorithm knows not only the present request to be served, but also some future requests. This issue is fundamental from the practical as well as the theoretical point of view. In paging systems some requests usually wait in line to be processed by a paging algorithm. One reason is that requests do not necessarily arrive one after the other, but rather in blocks of possibly variable size. Furthermore, if several processes run on a computer, it is likely that some of them incur page faults which then wait for service. Many memory systems are also equipped with prefetching mechanisms, i.e. on a request not only the currently accessed page but also some related pages which are expected to be asked next are demanded to be in fast memory. Thus each request generates a number of additional requests. In fact, some paging algorithms used in practice make use of lookahead [S77]. In the theoretical context a natural question is: What is it worth to know the future?

Previous research on lookahead in on-line algorithms has mostly addressed dynamic location problems and on-line graph problems [CGS89, I90, KT91, HS92]; only very little is known in the area of paging with lookahead. Consider the intuitive model of lookahead, which we call *weak lookahead*. Let $l \geq 1$ be an integer. We say that an on-line paging algorithm has a *weak lookahead of size* l, if it sees the present request to be served and the next l future requests. It is well known that this model cannot improve the competitive factors of on-line paging algorithms. If an on-line paging algorithm has a weak lookahead of size l, then an adversary that constructs a request sequence can simply replicate each request l times in order to make the lookahead useless. The only other result known on paging with lookahead has been developed by Young [Y91]. According to Young, a paging algorithm is *on-line with a resource-bounded lookahead of size* l if it sees the present request and the maximal sequence of future requests for which it will incur l faults. Young presents deterministic and randomized on-line paging algorithms with resource-bounded lookahead l which are $\max\{2k/l, 2\}$-competitive and $2(\ln(k/l) + 1)$-competitive, respectively. However, the model of resource-bounded lookahead is unrealistic in practice.

We now introduce a new model of lookahead which has practical as well as theoretical importance. As we shall see, this model can significantly improve the

competitive factors of on-line paging algorithms. Let $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ be a request sequence of length m. $\sigma(t)$ denotes the request at time t. For a given set S, $card(S)$ denotes the cardinality of S. Let $l \geq 1$ be an integer.

Strong lookahead of size l: The on-line algorithm sees the present request and a sequence of future requests. This sequence contains l pairwise distinct pages which also differ from the page requested by the present request. More precisely, when serving request $\sigma(t)$, the algorithm knows requests $\sigma(t+1), \sigma(t+2), \ldots, \sigma(t')$, where $t' = \min\{s > t | card(\{\sigma(t), \sigma(t+1), \ldots, \sigma(s)\}) = l+1\}$. The requests $\sigma(s)$, with $s \geq t'+1$, are not seen by the on-line algorithm at time t.

Strong lookahead is motivated by the observation that in request sequences generated by real programs, subsequences of consecutive requests generally contain a number of distinct pages. Furthermore, strong lookahead is of interest in the theoretical context when we ask how significant it is to know part of the future. An adversary may replicate requests in the lookahead, but nevertheless it has to reveal some really significant information on future requests.

In the following, we always assume that an on-line algorithm has a strong lookahead of fixed size $l \geq 1$. If a request sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ is given, then for all $t \geq 1$ we define a value $\lambda(t)$. If $card(\{\sigma(t), \sigma(t+1), \ldots, \sigma(m)\}) < l+1$ then let $\lambda(t) = m$; otherwise let $\lambda(t) = \min\{t' > t | card(\{\sigma(t), \sigma(t+1), \ldots, \sigma(t')\}) = l+1\}$. The *lookahead $L(t)$ at time t* is defined as

$$L(t) = \{\sigma(s) | s = t, t+1, \ldots, \lambda(t)\}.$$

We say that *a page x is in the lookahead at time t* if $x \in L(t)$.

The remainder of this paper is an in-depth study of paging with strong lookahead. Strong lookahead is the first model of lookahead that is significant from a practical and theoretical standpoint and also reduces the competitive factors of on-line paging algorithms. In Section 2 we consider deterministic on-line algorithms and present a variant of the algorithm LRU that, given a strong lookahead of size l, where $l \leq k-2$, achieves a competitive factor of $(k-l)$. We also show that no deterministic on-line paging algorithm with strong lookahead l, $l \leq k-2$, can be better than $(k-l)$-competitive. Thus our proposed algorithm is optimal. Furthermore, we give another variant of the algorithm LRU with strong lookahead l, $l \leq k-2$, which is $(k-l+1)$-competitive. Interestingly, this algorithm does not exploit full lookahead but rather serves the request sequence in a series of blocks. Section 3 addresses randomized on-line paging algorithms with strong lookahead. We prove that a modification of the MARKING algorithm with strong lookahead l, $l \leq k-2$, is $2H(k-l)$-competitive. This competitiveness is within a factor of 2 of optimal. In particular, we show that no randomized on-line paging algorithm with strong lookahead l, $l \leq k-2$, can be better than $H(k-l)$-competitive. Furthermore we present an extremely simple randomized on-line paging algorithm with strong lookahead l, $l \leq k-2$, which is $(k-l+1)$-competitive.

2 Deterministic paging with strong lookahead

Unless otherwise stated, we assume in the following that all our paging algorithms are *lazy* algorithms, i.e. they only evict a page on a fault.

Let $k \geq 3$. We consider the important case that an on-line paging algorithm has a strong lookahead of size $l \leq k-2$. The on-line paging algorithms we present are extensions of the algorithm LRU to our model of strong lookahead.

Algorithm LRU(l): On a fault execute the following steps. Among the pages in fast memory which are not contained in the present lookahead determine the page whose last request occurred least recently. Evict this page and load the requested page.

Theorem 1. *Let $l \leq k-2$. The algorithm LRU(l) with strong lookahead l is $(k-l)$-competitive.*

Now we prove this theorem. Let $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ be a request sequence of length m. We assume without loss of generality that LRU(l) and OPT start with an empty fast memory and that on the first k faults, both LRU(l) and OPT load the requested page into the fast memory. Furthermore we assume that σ contains at least $l+1$ distinct pages. The following proof consists of three main parts. First, we introduce the potential function we use to analyze LRU(l). In the second part, we partition the request sequence σ into a series of phases and then, in the third part, we bound LRU(l)'s amortized cost using that partition.

1. The potential function

We introduce some basic notations. For $t = 1, 2, \ldots, \lambda(1)-1$, let $\mu(t) = 1$ and for $t = \lambda(1), \lambda(1)+1, \ldots, m$, let

$$\mu(t) = \max\{t' < t | card(\{\sigma(t'), \sigma(t'+1), \ldots, \sigma(t)\}) = l+1\}.$$

Define

$$M(t) = \{\sigma(s) | s = \mu(t), \mu(t)+1, \ldots, t\}.$$

For a given time t, the set $M(t)$ contains the last $l+1$ requested pages.

For $t = 1, 2, \ldots, m$, let $S_{LRU(l)}(t)$ be the set of pages contained in LRU(l)'s fast memory after request t, and let $S_{OPT}(t)$ be the set of pages contained in OPT's fast memory after request t. $S_{LRU(l)}(0)$ and $S_{OPT}(0)$ denote the sets of pages which are initially in fast memory, i.e. $S_{LRU(l)}(0) = S_{OPT}(0) = \emptyset$. For the analysis of the algorithm we assign weights to all pages. These weights are updated after each request. Let $w(x,t)$ denote the weight of page x after request t, $1 \leq t \leq m$. The weights are set as follows. If $x \notin S_{LRU(l)}(t)$ or $x \in L(t)$, then

$$w(x,t) = 0.$$

Let $j = card(S_{LRU(l)}(t) \setminus L(t))$. Assign integer weights from the range $[1, j]$ to the pages in $S_{LRU(l)}(t) \setminus L(t)$ such that any two pages $x, y \in S_{LRU(l)}(t) \setminus L(t)$ satisfy

$$w(x,t) < w(y,t)$$

iff the last request to x occurred earlier than the last request to y. For $t = 1, 2, \ldots, m$, let

$$S(t) = S_{LRU(l)}(t) \setminus \{M(t) \cup L(t) \cup S_{OPT}(t)\}.$$

We now define the potential function:

$$\Phi(t) = \sum_{x \in S(t)} w(x,t).$$

Intuitively, $S(t)$ contains those pages which cause LRU(l) to have a higher cost than OPT. Instead of the pages $x \in S(t)$, OPT can store pages in its fast memory which

are not contained in $S_{LRU(l)}(t)$ but are requested in the future. The weight $w(x,t)$ of a page $x \in S(t)$ equals the number of faults that LRU(l) must incur before it can evict x.

2. The partitioning of the request sequence

We will partition the request sequence σ into phases, numbered from 0 to p for some p, such that phase 0 contains at most $l+1$ distinct pages and phase i, $i = 1, 2, \ldots, p$, has the following two properties. Let t_i^b and t_i^e denote the beginning and the end of phase i, respectively.

Property 1: Phase i contains exactly $l+1$ distinct pages, i.e.

$$card(\{\sigma(t_i^b), \sigma(t_i^b+1), \ldots, \sigma(t_i^e)\}) = l+1.$$

Property 2: For all $x \in S_{LRU(l)}(t_{i-1}^e) \setminus \{L(t_i^b) \cup S_{OPT}(t_{i-1}^e)\}$,

$$w(x, t_i^e) \leq k - l - 2.$$

In the following, we describe how to decompose σ. We partition the request sequence starting at the end of σ. Suppose that we have already constructed phases $P(i+1), P(i+2), \ldots, P(p)$. We show how to generate phase $P(i)$. Let $t_i^e = t_{i+1}^b - 1$. (We let $t_p^e = m$ at the beginning of the decomposition.) Now set $t = \mu(t_i^e)$ and compute $S_{LRU(l)}(t-1) \setminus L(t)$. If $S_{LRU(l)}(t-1) \setminus L(t) \neq \emptyset$, then let y be the most recently requested page in $S_{LRU(l)}(t-1) \setminus L(t)$. We consider two cases. If $S_{LRU(l)}(t-1) \setminus L(t) = \emptyset$ or if $S_{LRU(l)}(t-1) \setminus L(t) \neq \emptyset$ and $y \in S_{OPT}(t-1)$, then let $t_i^b = t$ and call the i-th phase $P(i) = \sigma(t_i^b), \sigma(t_i^b+1), \ldots, \sigma(t_i^e)$ a type 1 phase. Otherwise (if $S_{LRU(l)}(t-1) \setminus L(t) \neq \emptyset$ and $y \notin S_{OPT}(t-1)$) let t', $t' < t$, be the time when OPT evicted page y most recently. Let $t_i^b = t'$ and call the i-th phase $P(i) = \sigma(t_i^b), \sigma(t_i^b+1), \ldots, \sigma(t_i^e)$ a type 2 phase.

Lemma 2. *The partition generated above satisfies the following conditions.*

a) Phase $P(0)$ contains at most $l+1$ distinct pages.

b) Every phase $P(i)$, $1 \leq i \leq p$, has Property 1 and Property 2.

Proof. First we prove part a). We show that $P(0)$ is a type 1 phase. This immediately implies that $P(0)$ contains at most $l+1$ vertices. If $P(0)$ was a type 2 phase, then OPT would evict a page on the first request $\sigma(1)$. However, this is impossible because initially the fast memories are empty and on the first k faults both LRU(l) and OPT load the requested page into the fast memory.

Now we prove part b) of the lemma. Consider an arbitrary phase $P(i)$, $1 \leq i \leq p$. Let $t = \mu(t_i^e)$. If $S_{LRU(l)}(t-1) \setminus L(t) \neq \emptyset$, then let y be the most recently requested page in $S_{LRU(l)}(t-1) \setminus L(t)$ and let t'', $t'' < t$, be the time when y was requested most recently. If $P(i)$ is a type 2 phase, then let t', $t' \leq t-1$, be the time when OPT evicted y most recently. (Since $y \notin S_{OPT}(t-1)$, we have $t'' < t' \leq t-1$.)

We show that $P(i)$ contains exactly $l+1$ pages. For a type 1 phase there is nothing to show. Suppose $P(i)$ is a type 2 phase. Then $t_i^b = t'$. Let $s \in [t', t-1]$ be arbitrary and let x be the page requested at time s. We need to show $x \in L(t)$. So assume $x \notin L(t)$. Then by the definition of y, $x \notin S_{LRU(l)}(t-1)$, i.e. x was evicted by LRU(l) at some time $s' \in [s+1, t-1]$. Since y was not evicted by LRU(l) at time s' and y's most recent request was at time $t'' < s$, we must have $y \in L(s') \subseteq \{\sigma(s'), \ldots, \sigma(t-1), \sigma(t), \ldots, \sigma(t_i^e)\} = \{\sigma(s'), \ldots, \sigma(t-1)\} \cup L(t)$. But $y \notin \{\sigma(s'), \ldots, \sigma(t-1)\}$ and $y \notin L(t)$, by the definition of t'' and y. Thus $x \notin L(t)$ is impossible. We conclude that $P(i)$ contains exactly $l+1$ distinct pages.

It remains to prove that $P(i)$ has Property 2. Consider an arbitrary page $x \in S_{LRU(l)}(t_{i-1}^e) \setminus \{L(t_i^b) \cup S_{OPT}(t_{i-1}^e)\}$. If $w(x, t_i^e) = 0$, then the property clearly holds. Therefore assume $w(x, t_i^e) \geq 1$. By Property 1, $L(t_i^b)$ contains all pages which are requested in $P(i)$. Since $w(x, t_i^e) \geq 1$, we have $x \in S_{LRU(l)}(t_i^e) \setminus L(t_i^e)$ and hence $x \notin L(t_i^b) \cup L(t_i^e) \supseteq L(s)$ for all $s \in [t_i^b, t_i^e]$. Thus, x was a candidate for eviction by LRU(l) throughout $P(i)$, but was not evicted. This implies immediately that all pages requested in $P(i)$, i.e. all pages in $L(t_i^b)$, also belong to $S_{LRU(l)}(t_i^e)$. Using a very similar analysis we can show that $y \neq x$ and $y \in S_{LRU(l)}(t_i^e)$. Hence we have identified $l+2$ pages in $S_{LRU(l)}(t_i^e)$ which, at time t_i^e, were requested later than x. At time t_i^e, each of these pages has a weight of 0 or a weight which is greater than that of x. Thus, $w(x, t_i^e) \leq k - l - 2$. □

3. Bounding LRU(l)'s amortized cost

Using the partition of σ generated above, we will evaluate LRU(l)'s amortized cost on σ. First we will bound the increase in potential $\sum_{t=1}^{m} \Phi(t) - \Phi(t-1)$. Then we will estimate LRU(l)'s actual cost in each phase of σ. For $t = 1, 2, \ldots, m$, let

$$N(t) = S(t) \setminus S(t-1).$$

We set $M(0) = L(0) = \emptyset$ and $S(0) = S_{LRU(l)}(0) \setminus \{M(0) \cup L(0) \cup S_{OPT}(0)\}$, which is used in the definition of $N(1) = S(1) \setminus S(0)$.

We present two lemmas which are crucial in analyzing the change in potential $\Phi(t) - \Phi(t-1)$, $1 \leq t \leq m$. Note that

$$\begin{aligned}\Phi(t) - \Phi(t-1) &= \sum_{x \in S(t)} w(x,t) - \sum_{x \in S(t-1)} w(x, t-1) \\ &= \sum_{x \in N(t)} w(x,t) + \sum_{x \in S(t-1) \cap S(t)} (w(x,t) - w(x,t-1)) - \sum_{x \in S(t-1) \setminus S(t)} w(x, t-1).\end{aligned}$$

Lemma 3. *Let $1 \leq t \leq m$. If $x \in N(t)$, then $w(x,t) \leq k - l - 1$.*

Proof. By the definition of $N(t)$, we have $x \in S_{LRU(l)}(t) \setminus \{M(t) \cup L(t) \cup S_{OPT}(t)\}$. Since $x \notin M(t)$, page x is not requested in the interval $[\mu(t), t]$ and hence $x \in S_{LRU(l)}(\mu(t) - 1)$. We have $x \notin M(t) \cup L(t)$ which implies $x \notin L(s)$ for all s with $\mu(t) \leq s \leq t$. Thus, x has been a candidate for eviction by LRU(l) throughout the interval $[\mu(t), t]$, but was not evicted. It follows that all pages in $M(t)$ must be in $S_{LRU(l)}(t)$. Note that $M(t)$ contains $l+1$ pages because OPT does not evict a page before the $(k+1)$-st fault. At time t, all pages in $M(t)$ have a weight of 0 or a weight which is greater than $w(x,t)$. Thus $w(x,t) \leq k - l - 1$. □

Lemma 4. *Let $1 \leq t \leq m$ and $x \in S(t-1) \cap S(t)$. Then x's weight satisfies $w(x, t-1) \geq w(x,t)$. In particular, if LRU(l) incurs a fault at time t, then $w(x, t-1) > w(x,t)$.*

Proof. Note that by the definition of $S(t-1)$ and $S(t)$, we have $x \in S_{LRU(l)}(t-1) \setminus L(t-1)$ and $x \in S_{LRU(l)}(t) \setminus L(t)$. Hence $w(x, t-1) \geq 1$ and $w(x,t) \geq 1$. The inequality $w(x, t-1) \geq w(x,t)$, follows from the following two statements whose proofs we omit.

1) Let y, $y \neq x$, be a page which satisfies $w(y, t-1) = 0$ and $w(y,t) > 0$. Then $w(x,t) < w(y,t)$.
2) Let y, $y \neq x$, be a page which satisfies $w(y, t-1) > 0$ and $w(x, t-1) < w(y, t-1)$. Then $w(y,t) = 0$ or $w(x,t) < w(y,t)$.

Now suppose that LRU(l) incurs a fault at time t. Then, at time t, LRU(l) evicts a page z, $z \neq x$, whose last request occurred earlier than x's last request. Hence $1 \leq w(z,t-1) < w(x,t-1)$. Since the statements 1) and 2) hold, x's weight must decrease after z is evicted, i.e. $w(x,t-1) > w(x,t)$. □

Lemma 3 implies that at any time t, $1 \leq t \leq m$, a page $x \in N(t)$ can cause an increase in potential of at most $k-l-1$. Thus, for every t, $1 \leq t \leq m$, we have

$$\Phi(t) - \Phi(t-1) = (k-l-1)card(N(t)) - W(t), \tag{1}$$

where $W(t) = W^1(t) + W^2(t) + W^3(t)$ and

$$W^1(t) = \sum_{x \in N(t)} (k-l-1-w(x,t))$$

$$W^2(t) = \sum_{x \in S(t-1) \cap S(t)} (w(x,t-1) - w(x,t))$$

$$W^3(t) = \sum_{x \in S(t-1) \setminus S(t)} w(x,t-1).$$

For all $t = 1, 2, \ldots, m$, we have

$$W^1(t) \geq 0,\ W^2(t) \geq 0,\ W^3(t) \geq 0. \tag{2}$$

Clearly, $W^1(t) \geq 0$ and $W^3(t) \geq 0$. The inequality $W^2(t) \geq 0$ follows from Lemma 4.

Next we estimate $\sum_{t=1}^{m} card(N(t))$ and derive a bound on $\sum_{t=1}^{m} \Phi(t) - \Phi(t-1)$. To each element $x \in N(t)$ we assign the most recent eviction of x by OPT. More formally, let

$$X = \{(x,t) \in (\bigcup_{t=1}^{m} N(t)) \times [1,m] | x \in N(t)\}.$$

We define a function $f : X \longrightarrow [1,m]$. For $(x,t) \in X$ we define

$$f(x,t) = \max\{s \leq t | \text{OPT evicts page } x \text{ at time } s\}.$$

Note that f is well-defined. The next lemma presents two properties of the function f. Part b) will be useful when bounding LRU(l)'s actual cost in each phase of σ. The proof of the lemma is omitted.

Lemma 5. *a) The function f is injective.*
b) Let $(x,t) \in X$ and $f(x,t) = t'$. Let $t \in [t_i^b, t_i^e]$, $0 \leq i \leq p$. If $i = 0$, then $t' \in [t_0^b, t_0^e]$. If $i \geq 1$, then $t' \in [t_{i-1}^b, t_i^e]$.

Let T_{OPT} be the set of all $t \in [1,m]$ such that OPT evicts a page at time t. Note that $C_{OPT}(\sigma) = card(T_{OPT})$. Let $T_{OPT}^1 = \{f(x,t) | (x,t) \in X\}$. By Lemma 5, f is injective and hence

$$\sum_{t=1}^{m} card(N(t)) = card(X) = card(T_{OPT}^1).$$

Thus, by equation (1), we obtain

$$\sum_{t=1}^{m} \Phi(t) - \Phi(t-1) = (k-l-1)card(T_{OPT}^1) - \sum_{t=1}^{m} W(t). \tag{3}$$

Now we bound LRU(l)'s actual cost in each phase of σ. For $i = 0, 1, \ldots, p$, let $C_{LRU(l)}(i)$ be the actual cost LRU(l) incurs in serving phase $P(i)$, and let $C_{OPT}(i)$ be the cost OPT incurs in serving $P(i)$. Furthermore, let

$$T^2_{OPT} = T_{OPT} \setminus T^1_{OPT}$$

and, for $i = 0, 1, \ldots, p$, let

$$T^2_{OPT}(i) = \{t \in T^2_{OPT} | t^b_i \leq t \leq t^e_i\}.$$

Lemma 6. *a)* $C_{LRU(l)}(0) = C_{OPT}(0)$

b) For $i = 1, 2, \ldots, p$, $\quad C_{LRU(l)}(i) \leq C_{OPT}(i) + card(T^2_{OPT}(i-1)) + \sum_{t=t^b_i}^{t^e_i} W(t)$.

Proof. Part a) follows from the fact that phase $P(0)$ contains at most $l+1 < k$ distinct pages and that on the first k faults, both LRU(l) and OPT load the requested page into the fast memory.

In the proof of part b), we consider a fixed $i \in [1, p]$. If $C_{LRU(l)}(i) = 0$, then the inequality clearly holds because, according to line (2), $W(t) \geq 0$ for all $t \in [t^b_i, t^e_i]$. So suppose $C_{LRU(l)}(i) \geq 1$. Let $\tilde{C}(i) = card(S_{LRU(l)}(t^e_{i-1}) \setminus \{L(t^b_i) \cup S_{OPT}(t^e_{i-1})\})$. An easy exercise shows that $C_{LRU(l)}(i) \leq C_{OPT}(i) + \tilde{C}(i)$. In the following we sketch how to prove

$$\tilde{C}(i) \leq card(T^2_{OPT}(i-1)) + \sum_{t=t^b_i}^{t^e_i} W(t). \tag{4}$$

$C_{LRU(l)}(i) \leq C_{OPT}(i) + \tilde{C}(i)$ and inequality (4) imply part b).

We introduce some notations. Let $t \in [t^b_i, t^e_i]$. For $x \in N(t)$, let $W^1(x,t) = k - l - 1 - w(x,t)$. For $x \in S(t-1) \cap S(t)$, let $W^2(x,t) = w(x,t-1) - w(x,t)$ and for $x \in S(t-1) \setminus S(t)$, let $W^3(x,t) = w(x,t-1)$. Note that

$$W^1(t) = \sum_{x \in N(t)} W^1(x,t) \qquad W^2(t) = \sum_{x \in S(t-1) \cap S(t)} W^2(x,t)$$

$$W^3(t) = \sum_{x \in S(t-1) \setminus S(t)} W^3(x,t).$$

For any $x \in N(t)$ ($x \in S(t-1) \cap S(t)$, $x \in S(t-1) \setminus S(t)$) we have

$$W^1(x,t) \geq 0 \quad (W^2(x,t) \geq 0,\ W^3(x,t) \geq 1). \tag{5}$$

The inequality $W^1(x,t) \geq 0$ follows from Lemma 3. Lemma 4 implies $W^2(x,t) \geq 0$. If $x \in S(t-1) \setminus S(t)$, then $x \in S_{LRU(l)}(t-1) \setminus L(t-1)$ and hence $1 \leq w(x,t-1) = W^3(x,t)$.

We sketch the main idea of the proof of inequality (4). We show that for each page $x \in S_{LRU(l)}(t^e_{i-1}) \setminus \{L(t^b_i) \cup S_{OPT}(t^e_{i-1})\}$ one of the following two statements holds.

1) There exists a $t' \in T^2_{OPT}(i-1)$ such that OPT evicts page x at time t'.
2) There exists a time $t' \in [t^b_i, t^e_i]$ and a $j \in \{1, 2, 3\}$ such that $W^j(x,t') \geq 1$.

These statements, together with line (5), imply the correctness of inequality (4).

Consider a page $x \in S_{LRU(l)}(t^e_{i-1}) \setminus \{L(t^b_i) \cup S_{OPT}(t^e_{i-1})\}$. We distinguish between two main cases.

Case 1: For $t = t^e_{i-1}, t^b_i, t^b_i + 1, \ldots, t^e_i$, $x \notin S(t)$

We can prove that statement 1) holds. Since $x \in S_{LRU(l)}(t^e_{i-1}) \setminus \{L(t^b_i) \cup S_{OPT}(t^e_{i-1})\}$,

we have $x \notin S_{OPT}(t^e_{i-1})$. Let $t' = \max\{s \leq t^e_{i-1} | \text{OPT evicts page } x \text{ at time } s\}$. Using Property 1 and part b) of Lemma 5, we are able to show that $t' \geq t^b_{i-1}$ and $t' \notin T^1_{OPT}$. (A detailed proof is omitted here.) Thus $t' \in T^2_{OPT}(i-1)$.

Case 2: There exists a t, $t^e_{i-1} \leq t \leq t^e_i$, such that $x \in S(t)$

In this case we show that the above statement 2) holds. Let $t_{\min}$ be the smallest $t \in [t^e_{i-1}, t^e_i]$ such that $x \in S(t)$.

Case 2.1: $t_{\min} = t^e_{i-1}$

Let t'' be the time when LRU(l) incurs the first fault during phase $P(i)$. We consider $w(x, t'')$. If $w(x, t'') = 0$, then $x \notin S(t'')$. Hence there must exist a t', $t^b_i \leq t' \leq t''$, such that $x \in S(t'-1) \setminus S(t')$. Thus $W^3(x, t') \geq 1$. If $w(x, t'') \geq 1$, then we can prove that $x \in S(t''-1) \cap S(t'')$. Now Lemma 4 implies $W^2(x, t'') \geq 1$.

Case 2.2: $t_{\min} > t^e_{i-1}$

If $w(x, t_{\min}) < k-l-1$, then $W^1(x, t_{\min}) \geq 1$. Suppose $w(x, t_{\min}) = k-l-1$. By Property 2, $w(x, t^e_i) \leq k-l-2$. Now a simple argument shows that there must exist a time $t' \in [t_{\min}+1, t^e_i]$ such that $W^2(x, t') \geq 1$ or $W^3(x, t') \geq 1$.

The proof of Lemma 6 is complete.□

Now it is easy to finish the proof of Theorem 1. We estimate LRU(l)'s amortized cost. Applying equation (3) and Lemma 6 we can show

$$C_{LRU(l)}(\sigma) + \Phi(m) - \Phi(0) = \sum_{i=0}^{p} C_{LRU(l)}(i) + \sum_{t=1}^{m} \Phi(t) - \Phi(t-1) \leq$$

$$\sum_{i=0}^{p} C_{OPT}(i) + \sum_{i=0}^{p-1} card(T^2_{OPT}(i)) + \sum_{t=t^b_1}^{m} W(t) - \sum_{t=1}^{m} W(t) + (k-l-1)card(T^1_{OPT}).$$

Line (2) implies that $W(t) \geq 0$ for all $t \in [t^b_0, t^e_0]$. Hence

$$\begin{aligned} C_{LRU(l)}(\sigma) + \Phi(m) - \Phi(0) &\leq C_{OPT}(\sigma) + card(T^2_{OPT}) + (k-l-1)card(T^1_{OPT}) \\ &\leq (k-l)C_{OPT}(\sigma). \end{aligned}$$

The proof of Theorem 1 is complete.

Next we present another on-line algorithm with strong lookahead. This algorithm does not use full lookahead but rather serves the request sequence in a series of blocks.

Algorithm LRU(l)-blocked: Serve the request sequence in a series of blocks $B(1), B(2), \ldots$, where $B(1) = \sigma(1), \sigma(2), \ldots, \sigma(\lambda(1))$ and $B(i) = \sigma(t^e_{i-1}+1), \sigma(t^e_{i-1}+2), \ldots, \sigma(\lambda(t^e_{i-1}+1))$ for $i \geq 2$. Here t^e_{i-1} denotes the end of block $B(i-1)$. If there occurs a fault while $B(i)$ is processed, then the following rule applies. Among the pages in fast memory which are not contained in $B(i)$ determine the page whose last request occurred least recently. Evict that page.

LRU(l)-blocked has the advantage that it updates its information on future requests only once during each block. Thus it can respond to requests faster that LRU(l). Furthermore, LRU(l)-blocked takes into account that in practice requests often arrive in blocks. Interestingly, this simpler algorithm is only slightly weaker than LRU(l). Using a very similar analysis as in the proof of Theorem 1, we are able to show

Theorem 7. *Let $l \leq k-2$. The algorithm LRU(l)-blocked with strong lookahead l is $(k-l+1)$-competitive.*

The following theorem shows that LRU(l) and LRU(l)-blocked are optimal and nearly optimal, respectively.

Theorem 8. *Let A be a deterministic on-line paging algorithm with strong lookahead l, where $l \leq k-2$. If A is c-competitive, then $c \geq (k-l)$.*

Proof. Let $S = \{x_1, x_2, \ldots, x_{k+1}\}$ be a set of $k+1$ pages. We assume without loss of generality that $A's$ and OPT's fast memories initially contain $x_1, x_2, \ldots, x_k$. Let $SL = \{x_1, x_2, \ldots, x_l\}$. We construct a request sequence σ consisting of a series of phases. Each phase contains $l+1$ requests to $l+1$ distinct pages. The first phase $P(1)$ consists of requests to the pages in SL, followed by a request to the page x_{k+1} which is not in fast memory, i.e. $P(1) = x_1, x_2, \ldots, x_l, x_{k+1}$. Each of the following phases $P(i)$, $i \geq 2$, has the form $P(i) = x_1, x_2, \ldots, x_l, y_i$, where $y_i \in S \setminus SL$ is chosen as follows. Let $z_i \in S$ be the page which is not in A's fast memory after the last request of phase $i-1$. If $z_i \in S \setminus SL$, then set $y_i = z_i$. Otherwise, if $z_i \in SL$, y_i is an arbitrary page in $S \setminus SL$. The algorithm A incurs a cost of 1 in each phase. During $k-l$ successive phases, OPT's cost is at most 1. □

So far, we have assumed $k \geq 3$ and $l \leq k-2$, which, of course, is the interesting case. Note that if $l = k-1$ and the total number of different pages in the memory system equals $k+1$, then LRU(l) achieves a competitive factor of 1 because it behaves like Belady's optimal paging algorithm MIN [B66].

3 Randomized paging with strong lookahead

Suppose a randomized paging algorithm has a strong lookahead of size l. Again, we assume $k \geq 3$ and $l \leq k-2$. The first algorithm we propose is a slight modification of the MARKING algorithm due to Fiat *et al.* [FKLMSY91]. The MARKING algorithm proceeds in a series of phases. During each phase a set of marked pages is maintained. At the beginning of each phase all pages are unmarked. Whenever a page is requested, that page is marked. On a fault, a page is chosen uniformly at random from among the unmarked pages in fast memory, and that page is evicted. A phase ends immediately before a fault, when there are k marked pages in fast memory.

The modified algorithm with strong lookahead l uses lookahead once during each phase.

Algorithm MARKING(l): At the beginning of each phase execute an initial step: Determine the set S of pages which are in the present lookahead but not in fast memory. Choose $card(S)$ pages uniformly at random from among the pages in fast memory which are not contained in the current lookahead. Evict these pages and load the pages in S. After this initial step proceed with the MARKING algorithm.

Theorem 9. *Let $l \leq k-2$. The algorithm MARKING(l) with strong lookahead l is $2H(k-l)$-competitive.*

Proof. The idea of the proof is the same as the idea of the original proof of the MARKING algorithm [FKLMSY91]. We assume without loss of generality that MARKING(l)'s and OPT's fast memories initially contain the same k pages. During each phase we compare the cost incurred by MARKING(l) to the cost incurred by the optimal algorithm OPT. Consider an arbitrary phase. We use the same terminology as Fiat *et al.* A page is called *stale* if it is unmarked but was marked in the previous phase, and *clean* if it is neither stale nor marked.

Let c be the number of clean pages and s be the number of stale pages requested

in the phase. Note that $c + s = k$. Fiat *et al.* prove that OPT has an amortized cost of at least $c/2$ during the phase.

We evaluate MARKING(l)'s cost during the phase. Serving c requests to clean pages obviously costs c. It remains to bound the expected cost for serving the stale pages. Let s_1 be the number of stale pages contained in the lookahead at the beginning of the phase and let $s_2 = s - s_1$. Then $s_1 + c \geq l + 1$ because every page in the lookahead is either clean or counted in s_1. Thus $s_2 = s - s_1 \leq k - c - (l + 1 - c) = k - l - 1$. Note that serving the first s_1 stale requests does not incur any cost and that we just have to evaluate MARKING(l)'s cost on the following s_2 requests to stale pages. We are able to show that this expected cost is bounded by

$$\frac{c}{k - s_1} + \frac{c}{k - s_1 - 1} + \ldots + \frac{c}{k - s_1 - s_2 + 1} = \frac{c}{k - s_1} + \frac{c}{k - s_1 - 1} + \ldots + \frac{c}{k - s + 1}.$$

The above sum consists of $s_2 \leq k - l - 1$ terms and $\frac{c}{1}$ is missing. Hence the sum is bounded by $c(H(k - l) - 1)$, and we conclude that MARKING(l)'s cost during the phase is bounded from above by $cH(k - l)$. □

This following theorem implies that MARKING(l) is nearly optimal.

Theorem 10. *Let $l \leq k - 2$ and let A be a randomized on-line paging algorithm with strong lookahead l. If A is c-competitive, then $c \geq H(k - l)$.*

Proof. The proof is similar to Raghavan's proof that no randomized on-line paging algorithm without lookahead can be better than $H(k)$-competitive [R89]. So we just sketch the difference. Let $S = \{x_1, x_2, \ldots, x_{k+1}\}$ be a set of $k + 1$ pages and let $SL = \{x_1, x_2, \ldots, x_l\}$. We construct a request sequence which consists of a series of phases. The first phase is of the form $P(1) = x_1, x_2, \ldots, x_l, y_1$, where y_1 is chosen uniformly at random from all pages in $S \backslash SL$. The following phases $P(i)$ equal $P(i) = x_1, x_2, \ldots, x_l, y_i$, where y_i is chosen uniformly at random from $S \setminus \{SL \cup \{y_{i-1}\}\}$. It is possible to partition σ into rounds such that during each round OPT incurs a cost of 1 and any deterministic on-line algorithm with strong lookahead l incurs an expected cost of at least $H(k - l)$. Applying Yao's minimax principle [Y77], we obtain the theorem. □

We conclude this section by presenting another randomized algorithm, called RANDOM(l)-blocked. As the name suggests this algorithm is a variant of the algorithm RANDOM due to Raghavan and Snir [RS89]. On a fault RANDOM chooses a page uniformly at random from among the pages in fast memory and evicts that page. In terms of competitiveness RANDOM(l)-blocked represents no improvement upon the previously presented algorithms with strong lookahead. However, RANDOM(l)-blocked, as the original algorithm RANDOM, is very simple and uses no information on previous requests.

Algorithm RANDOM(l)-blocked: Serve the request sequence σ in a series of blocks. These blocks have the same structure as those in the algorithm LRU(l)-blocked. At the beginning of block $B(i)$ determine the set S_i of pages in $B(i)$ which are not in fast memory. Choose $card(S_i)$ pages uniformly at random from among the pages in fast memory which are not contained in $B(i)$. Evict these pages and load the pages in S_i. Then serve the requests in $B(i)$.

Theorem 11. *Let $l \leq k - 2$. The algorithm RANDOM(l)-blocked with strong lookahead l is $(k - l + 1)$-competitive.*

The proof of the theorem is omitted.