



ELSEVIER

Information Processing Letters 76 (2000) 155–162

Information
Processing
Letters

www.elsevier.com/locate/ipl

Limited bookmark randomized online algorithms for the paging problem

Wolfgang W. Bein^{a,*}, Rudolf Fleischer^b, Lawrence L. Larmore^{a,1}

^a Department of Computer Science, University of Nevada, Las Vegas, NV 89154, USA

^b Max-Planck-Institut für Informatik, Im Stadtwald, A-66123 Saarbrücken, Germany

Received 15 February 2000; received in revised form 1 August 2000

Communicated by S.E. Hambrusch

Abstract

An efficient randomized online algorithm for the paging problem for cache size 2 is given, which is $\frac{3}{2}$ -competitive against an oblivious adversary. The algorithm keeps track of at most one page in slow memory at any time. A lower bound of $\frac{37}{24} \approx 1.5416$ is given for the competitiveness of any *trackless* online algorithm for the same problem, i.e., an algorithm that keeps track of no page outside the cache. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Design of algorithms; Online algorithms; Randomized algorithms; Paging

1. Introduction

No randomized online algorithm for the paging problem can be less than H_k -competitive regardless of the available resources, where k is the number of cache locations.² Known H_k -competitive paging algorithms, such as Partition and Equitable [1,3], keep track of (i.e., mark) pages in slow memory. A mark to a page in slow memory must be implemented by some kind of identifier that permits an algorithm to determine whether a page fault is to that particular page. We refer to that identifier as a *bookmark*.

Recently [2] the concept of *tracklessness* was introduced for the server problem. This formalization of

the concept is new, but many known algorithms, e.g., HARMONIC [7,9] and BALANCE₂ [8], are trackless. Tracklessness is a property of an online algorithm for the server problem which is a restriction on what input data the algorithm uses in its computation, and on what outputs the algorithm gives. Reducing the paging problem to the server problem in a uniform space [5,10], the tracklessness restriction translates into the rule that no bookmarks are used. Among known algorithms for the paging problem, LRU (least recently used) is trackless. The best known general trackless randomized algorithm for the paging problem is MARK, which is known to be $(2H_k - 1)$ -competitive [1,4].

In this paper we show that the use of bookmarks is indeed necessary to achieve optimal randomized competitiveness for the paging problem. We also show that no trackless randomized algorithm for the paging problem for $k = 2$ can be 1.54-competitive, regardless of the availability of other resources. On the other

* Corresponding author.

E-mail addresses: bein@cs.unlv.edu (W.W. Bein), rudolf@mpi-sb.mpg.de (R. Fleischer), larmore@cs.unlv.edu (L.L. Larmore).

¹ Research supported by NSF grant CCR-9821009.

² $H_k = \sum_{i=1}^k i^{-1}$, the k th Harmonic number.

hand, we present a randomized algorithm which uses only one bookmark, and which achieves the optimal competitiveness of 1.5.

2. Paging with one bookmark

In this section we describe a randomized online algorithm \mathcal{A} for the paging problem for $k = 2$, which uses just one bookmark. Algorithm \mathcal{A} is barely random [4], in fact it uses only one random bit regardless of the length of the request sequence. We define two deterministic algorithms, \mathcal{A}_A and \mathcal{A}_B . \mathcal{A} simply emulates \mathcal{A}_A with probability $\frac{1}{2}$ and emulates \mathcal{A}_B with probability $\frac{1}{2}$. \mathcal{A}_A and \mathcal{A}_B are quite similar, in fact they behave exactly the same except in one situation.

At the outset, our algorithm is strikingly similar to a standard marking algorithm such as Equitable [1]. For $k = 2$, Equitable marks ejected pages as long as unrecognized requests are made. The page to be ejected from cache is determined by a probability distribution dependent on the number of bookmarks. Then, when a request is made to a bookmarked page, the algorithm will erase all bookmarks. The bookmarks can be erased because the algorithm can assume that the optimal cache is equal to its own cache once such a bookmarked page is requested. For $k > 2$, the algorithm is more complex, but regardless of k , Equitable consists of phases, where at the beginning of each phase the algorithm assumes that its cache matches the optimal cache.

Algorithm \mathcal{A} uses a maximum of one bookmark throughout. As an unrecognized page is requested, such a page is bookmarked and the algorithm switches from mode “satisfied” to “bookmarked”. Like Equitable, \mathcal{A} becomes “satisfied” if there is a request to a cache page or the bookmarked page. \mathcal{A} differs from Equitable in its behavior on a second consecutive unrecognized request. Rather than using a second bookmark, it instead moves into a third mode, which we call “unsatisfied” and erases the bookmark. It is surprising, yet it is the key for the motivation for our algorithm, that if we let the algorithm return to the “satisfied” mode on the very next request, the algorithm does not lose competitiveness.

Table 1 describes the details of the algorithm \mathcal{A} . Note that we refer to \mathcal{A}_A and \mathcal{A}_B as the algorithm

resulting from the choice in step 1. As discussed, algorithm \mathcal{A} has three *computational states*, SATISFIED, BOOKMARKED and UNSATISFIED. A page request can be to a page that is unrecognized, or kept as a bookmark, or currently in cache. We call the page in the cache that was most recently requested *junior*, the other *senior*. Initially, one of the two pages in cache is arbitrarily called *junior*. The bookmarked page is referred to as *book* and the current request is called r . We define the *ken* of an algorithm to be the set of those pages that are either in the cache or bookmarked. We say, that a page is *unrecognized* if it is not in the ken.

Note that the cardinality of the ken of both algorithms \mathcal{A}_A and \mathcal{A}_B is 2 if the state is SATISFIED or UNSATISFIED, and 3 if the state is BOOKMARKED. In fact, although algorithms \mathcal{A}_A and \mathcal{A}_B differ, they have same ken and state at every iteration. More specifically, let K_A^t, K_B^t to be the ken of $\mathcal{A}_A, \mathcal{A}_B$, respectively, at step t . Similarly, write S_A^t, S_B^t for the computational states of \mathcal{A}_A and \mathcal{A}_B at step t .

Lemma 1. *For any t , $S_A^t = S_B^t$ and $K_A^t = K_B^t$. Furthermore, if the state is BOOKMARKED, then the page that is bookmarked by \mathcal{A}_A is in the cache of \mathcal{A}_B , and the page that is bookmarked by \mathcal{A}_B is in the cache of \mathcal{A}_A .*

Proof. We assume inductively that the statement of the Lemma is true at the end of iteration $t - 1$, and show that the statement is an invariant for iteration t , i.e., after execution of step 4. The statement for the ken is trivial because \mathcal{A}_A and \mathcal{A}_B differ only when r is unrecognized and State = SATISFIED.

As for the statement about the computational states, note that because $K_A^{t-1} = K_B^{t-1}$, a requested page r is unrecognized by \mathcal{A}_A if and only if it is unrecognized by \mathcal{A}_B . In this case, an iteration of step 4 executes the else-case for both algorithms and therefore $S_A^t = S_B^t$. On the other hand, if r is recognized, then in cases $r = \textit{senior}$ and $r = \textit{book}$ the resulting state is always SATISFIED. \square

Theorem 1. *\mathcal{A} is $\frac{3}{2}$ -competitive against an oblivious adversary.*

Proof. It is sufficient to show that

$$\text{cost}_A(x) + \text{cost}_B(x) \leq 3 \text{cost}_{\text{opt}}(x) \quad (1)$$

Table 1
The 1-bookmark algorithm \mathcal{A}

```

1. Version is  $\mathcal{A}_A$  or  $\mathcal{A}_B$  with probability  $\frac{1}{2}$ ;
2. State = SATISFIED; book = EMPTY;
3. Read(r);
4. switch(r)
   case r = junior          /* request to junior cache page */
     no action
   case r = senior         /* request to senior cache page */
     book = EMPTY;
     State = SATISFIED;
   case r = book           /* request to bookmarked page */
     Eject senior; book = EMPTY;
     State = SATISFIED;
   else                       /* request is unrecognized */
     if State = SATISFIED
       Version =  $\mathcal{A}_A$ : book = junior; Eject junior;
       Version =  $\mathcal{A}_B$ : book = senior; Eject senior;
       State = BOOKMARKED
     if State = BOOKMARKED
       Eject senior; book = EMPTY;
       State = UNSATISFIED;
     if State = UNSATISFIED;
       Eject senior; book = EMPTY;
       State = SATISFIED;

5. goto 3;

```

if x is any request sequence. We may assume that the adversary chooses each service in such a way that its overall cost is optimal, i.e., the adversary will eject the page that will be requested furthest in the future. Let A^t be the contents of the adversary's cache at time t . Define $cost_{opt}^t$ to be the optimal cost at step t , and $cost_A^t$ and $cost_B^t$ the cost of \mathcal{A}_A and \mathcal{A}_B at step t , respectively.

We define a potential Φ^t at each time $t \geq 0$ as follows:

$$\Phi^t = \begin{cases} 0 & \text{if } S^t = \text{SATISFIED and } A^t = K^t, \\ 3 & \text{if } S^t = \text{SATISFIED and } A^t \neq K^t, \\ 1 & \text{if } S^t = \text{BOOKMARKED and } A^t \subseteq K^t, \\ 4 & \text{if } S^t = \text{BOOKMARKED and } A^t \not\subseteq K^t, \\ 2 & \text{if } S^t = \text{UNSATISFIED.} \end{cases} \quad (2)$$

We claim that

$$\Phi^t + cost_A^t + cost_B^t \leq \Phi^{t-1} + 3 cost_{opt}^t \quad (3)$$

for all t , which implies (1).

Our proof parallels the breakdown into cases in the iteration $(t-1)$ to t in Table 1. The first two cases deal with the part of the iteration before the “else”, i.e., r is recognized as $r = \textit{junior}$, or $r = \textit{senior}$, or $r = \textit{book}$.

Case 1: We consider the situation that $r = \textit{junior}$ or $r = \textit{senior}$, and State = SATISFIED or UNSATISFIED. Note that if the state is either SATISFIED or UNSATISFIED, then, by Lemma 1, \mathcal{A}_A and \mathcal{A}_B have the same junior and senior pages. Further note that we can immediately dispose of the case that $r = \textit{junior}$ as then there is no change in potential and all costs are 0. Assuming now $r = \textit{senior}$ the algorithm pays 0, and the new state is SATISFIED. We consider two sub-cases.

Case 1.1: $r \in A^{t-1}$. The left side of (3) is 0, since $A^t = K^t = A^{t-1}$.

Case 1.2: $r \notin A^{t-1}$. In this case $cost_{opt} = 1$, which implies that the right side of (3) is at least 3, while the left side of (3) is at most 3.

Case 2: We cover the remaining cases before the else-case. As in case 1, we assume $r \neq \text{junior}$. Therefore, these situations occur under the condition that State = BOOKMARKED and $r = \text{senior}$ or $r = \text{book}$. Equivalently, this condition can be expressed as: State = BOOKMARKED and $r \in K^{t-1}$. By Lemma 1, $r = \text{senior}$ for \mathcal{A}_A and $r = \text{book}$ for \mathcal{A}_B , or vice versa. Thus, $\text{cost}_A + \text{cost}_B = 1$, and the new state is SATISFIED. We consider two subcases.

Case 2.1: $r \in A^{t-1}$. $\Phi^t = 0$, since $A^t = K^t = A^{t-1}$, thus the left side of (3) is 1. The right side of (3) is at least 1.

Case 2.2: $r \notin A^{t-1}$. In this case $\text{cost}_{\text{opt}} = 1$, which implies that the right side of (3) is at least 4. Since $\Phi^t \leq 3$, the left side of (3) is at most 4.

The final three cases deal with situations where r is unrecognized in the iteration $(t - 1)$ to t in Table 1. For these cases we always have $\text{cost}_A + \text{cost}_B = 2$.

Case 3: r is unrecognized, State = SATISFIED.

Case 3.1: $r \in A^{t-1}$. In this case, $\Phi^{t-1} = 3$ and $\Phi^t = 1$. Both sides of (3) are equal to 3.

Case 3.2: $r \notin A^{t-1}$. In this case $\text{cost}_{\text{opt}} = 1$. The optimal algorithm ejects one member of A^{t-1} but not the other.

Case 3.2.1: $K^{t-1} = A^{t-1}$. In this case, $\Phi^{t-1} = 0$. Whichever page is ejected by the optimal algorithm is either retained by \mathcal{A}_A and bookmarked by \mathcal{A}_B or vice versa, thus $A^t \subseteq K^t$. Both sides of (3) are equal to 3.

Case 3.2.2: $K^{t-1} \neq A^{t-1}$. We have $\Phi^{t-1} = 3$. The right side (3) is 6, while the left side is at most 6.

Case 4: r is unrecognized, State = BOOKMARKED. The new state is UNSATISFIED, thus $\Phi^t = 2$. The left side (3) is 4.

Case 4.1: $r \in A^{t-1}$. Then $\Phi^{t-1} = 4$, and $\text{cost}_{\text{opt}} = 0$. Both sides of (3) are equal to 4.

Case 4.2: $r \notin A^{t-1}$. Then $\Phi^{t-1} \leq 1$, and $\text{cost}_{\text{opt}} = 1$. The right side of (3) is at least 4.

Case 5: r is unrecognized, State = UNSATISFIED. Then $\Phi^{t-1} = 2$.

Case 5.1: $r \in A^{t-1}$. Then $\Phi^t = 0$, and $\text{cost}_{\text{opt}} = 0$. Both sides of (3) are equal to 2.

Case 5.2: $r \notin A^{t-1}$. Then $\Phi^t \leq 3$, and $\text{cost}_{\text{opt}} = 1$. The right side (3) is 5, while the left side is at most 5.

This concludes the proof of (3), and therefore Theorem 1. \square

3. Paging without bookmarks

We now show that one bookmark is crucial: any randomized algorithm that does not use bookmarks cannot achieve optimal competitiveness.

3.1. Preliminaries

As before, we refer to the *optimal cache* as the set consisting of those two pages that would be in the cache if the optimal offline strategy were followed. We consider only request sequences consisting of lazy and fresh requests and make the usual assumption that no page is requested twice in a row. Offline strategies using only lazy and fresh requests can be fully described using symbols a, b, s . The symbol a denotes, “Fresh request; eject the newer page”. The symbol b denotes, “Fresh request; eject the senior page”. The symbol s denotes, “Hit” (always on the senior page, as we assume that no page is requested twice in a row). Conveniently, responses by a trackless algorithm can be described using these symbols as well, where a means “Fault; eject the junior page” the symbol b denotes, “Fault; eject the senior page”, and the symbol s denotes “Hit”.

For example, if the request sequence is $asass$, the set of possible algorithm response sequences is exactly $asass, asbab, asbbs, baaab, baabs, babab, babbs, bbass, bbbab, bbbbs$, where we assume that the optimal and algorithm cache are matched at the beginning and the end. (Without the assumption that caches be matched at the end of the sequence there are four more possible responses, $asbaa, baaaa, babaa, bbbaa$.) To see this, refer to the parse diagram in Fig. 1. The solid lines represent requests, and the shaded lines refer to the responses an algorithm can give. Nodes labeled 1 or 2 refer to situations where the optimal and algorithm’s cache are matched or unmatched, respectively. For the the request sequence

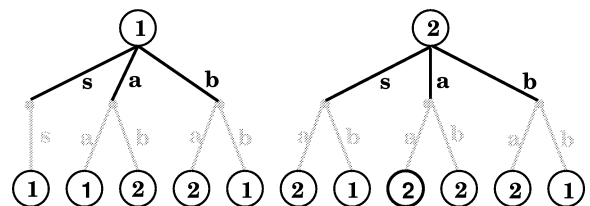


Fig. 1. The parse diagram.

Table 2
Fundamental strategies

i	x_i	c_i	$cost_{opt}(x_i)$	i	x_i	c_i	$cost_{opt}(x_i)$
1	<i>assss</i>	3/14	1	5	<i>bssss</i>	3/14	1
2	<i>aasss</i>	1/14	2	6	<i>basss</i>	1/14	2
3	<i>asass</i>	2/14	2	7	<i>bsass</i>	2/14	2
4	<i>aasas</i>	1/14	3	8	<i>basas</i>	1/14	3

asass, one starts parsing at the root node labeled 1 (of the left tree). If the algorithm responds with *a*, one arrives at a leaf labeled 1 and the caches are matched. Therefore, parsing continues at the root with label 1. The next request is *s*, which can only have *s* as a response which again leads to a leaf labeled 1. Continuing once again at the root labeled 1 with request *a*, the response can be either be *a* or *b*. If it is *b*, then one reaches a leaf labeled 2, thus the caches are unmatched and parsing continues at the root labeled 2. The final two requests are both *s*. If the first response is *b*, then one is at a node labeled 1 and the response for the final *s* request can only be *s*. We conclude that the request sequence *asass* can indeed have *asbbs* as a response, but not, say, *asbba*. Carefully parsing all possibilities in the the parse diagram, one sees that of the 3^5 possible strings of length 5, only the 14 strings mentioned above are feasible for *asass*. Note that the request sequence *asass* has cost 2, whereas the feasible responses for *asass* range in cost from 2 to 5.

For a randomized algorithm \mathcal{A} , let \mathcal{A}_y denote the conditional probability that algorithm \mathcal{A} responds with *y*, given that *y* is a feasible response. If the algorithm \mathcal{A} is *C* competitive, then

$$\sum_{\substack{y \text{ is feasible} \\ \text{for } x}} cost_{\mathcal{A}}(y)\mathcal{A}_y \leq C cost_{opt}(x) \quad \text{for all } x. \quad (4)$$

If there is a page fault for algorithm \mathcal{A} , then it is legal to respond with either *a* or *b*, i.e.,

$$\mathcal{A}_{ya} + \mathcal{A}_{yb} = \mathcal{A}_y \quad \text{for all } y. \quad (5)$$

If there is a hit, then, the algorithm will respond with *s*. A hit occurs if and only if *y* and *ys* are feasible responses, thus,

$$\mathcal{A}_{ys} = \mathcal{A}_y \quad \text{for all } y. \quad (6)$$

Finally,

$$\mathcal{A}_y \geq 0 \quad \text{for all } y \text{ with } \mathcal{A}_\varepsilon = 1, \quad (7)$$

where ε denotes the empty string.

We call conditions (7) *fundamental* constraints, conditions (5) and (6) *online* constraints, and conditions (4) *competitive* constraints.

3.2. A lower bound of $\frac{37}{24}$

We now prove that no trackless randomized algorithm can have competitiveness less than $\frac{37}{24}$. To this end we show that, for any $C < \frac{37}{24}$ and for any constant *K*, and for any trackless randomized online algorithm \mathcal{A} , there exists a request sequence ϱ for which

$$E cost_{\mathcal{A}}(\varrho) > K + C cost_{opt}(\varrho).^3$$

To construct that randomized request sequence ϱ we consider the eight length-limited request sequences $\mathcal{S} = x_1, x_2, x_3, x_4, x_5, x_6, x_8$ given in Table 2.

After any one of these eight length-limited request sequences has been requested we assume that the optimal and algorithm cache are matched. If that were not the case, we could simply lengthen the request sequence by appending sufficiently many lazy requests, without extra cost to the optimal offline strategy. With this stipulation, ϱ is simply obtained by starting with the empty string ε and then appending repeatedly, for some $i \in 1, \dots, n$, $x_i \in \mathcal{S}$ to ϱ with probability c_i .

Lemma 2. *Let \mathcal{A} be a randomized online algorithm for the paging problem with cache size 2. Then, for any constant $C < \frac{37}{24}$, and for any constant *K*, there*

³ As usual, *E* denotes the expected value.

exists a request sequence Q such that $E \text{ cost}_{\mathcal{A}}(Q) > K + C \text{ cost}_{\text{opt}}(Q)$.

Proof. To prove the lemma it is sufficient to show that

$$\sum_{i=1}^8 c_i E \text{ cost}_{\mathcal{A}}(x_i) \geq \frac{37}{24} \sum_{i=1}^8 c_i \text{ cost}_{\text{opt}}(x_i).$$

We first note that

$$\sum_{i=1}^8 c_i \text{ cost}_{\text{opt}}(x_i) = \frac{12}{7}.$$

We now show

$$\sum_{i=1}^8 c_i E \text{ cost}_{\mathcal{A}}(x_i) \geq \frac{37}{14}.$$

Since the algorithm and optimal cache are assumed to be matched at the end of the length-limited request sequence we may assume that \mathcal{A} will not give a as a response at the last step. Since the fifth service is not a , we conclude, by the online constraints, that $\mathcal{A}_y = \mathcal{A}_{ys} = \mathcal{A}_{yb}$ for any y of length 4.

We now make use of a symmetry to further reduce the complexity of the system. Define \mathcal{A}' , the *reflection* of \mathcal{A} , by $\mathcal{A}'_{az} = \mathcal{A}_{bz}$, $\mathcal{A}'_{bz} = \mathcal{A}_{az}$, and $\mathcal{A}'_{sz} = \mathcal{A}_{sz}$ for all $z \in \Gamma^*$. We say that \mathcal{A} is *symmetric* if $\mathcal{A} = \mathcal{A}'$, i.e., $\mathcal{A}_{az} = \mathcal{A}_{bz}$ for any $z \in \Gamma^*$. Without loss of generality, \mathcal{A} is symmetric, since

$$\sum_{i=1}^8 c_i E \text{ cost}_{\mathcal{A}}(x_i) = \sum_{i=1}^8 c_i E \text{ cost}'_{\mathcal{A}}(x_i),$$

and $\frac{1}{2}\mathcal{A} + \frac{1}{2}\mathcal{A}'$ is symmetric.

We now consider the online constraints associated with each x_i . Note that the possible response sequences for x_1, \dots, x_8 were parsed out using the process described in Section 3.1.

$$\begin{aligned} \text{cost}_{\mathcal{A}}(x_1) = & \mathcal{A}_{asss} + 5\mathcal{A}_{baaab} + 4\mathcal{A}_{baabs} \\ & + 3\mathcal{A}_{babss} + 2\mathcal{A}_{bbsss}, \end{aligned} \quad (8)$$

$$\begin{aligned} \text{cost}_{\mathcal{A}}(x_2) = & 2\mathcal{A}_{aass} + 5\mathcal{A}_{abaab} + 4\mathcal{A}_{ababs} \\ & + 3\mathcal{A}_{abbss} + 5\mathcal{A}_{baaab} + 4\mathcal{A}_{baabs} \\ & + 3\mathcal{A}_{babss} + 5\mathcal{A}_{bbaab} + 4\mathcal{A}_{bbabs} \\ & + 3\mathcal{A}_{bbsss}, \end{aligned} \quad (9)$$

$$\begin{aligned} \text{cost}_{\mathcal{A}}(x_3) = & 2\mathcal{A}_{asass} + 4\mathcal{A}_{asbab} + 3\mathcal{A}_{asbbs} \\ & + 5\mathcal{A}_{baaab} + 4\mathcal{A}_{baabs} + 5\mathcal{A}_{babab} \end{aligned}$$

$$\begin{aligned} & + 4\mathcal{A}_{babbs} + 3\mathcal{A}_{bbass} + 5\mathcal{A}_{bbbab} \\ & + 4\mathcal{A}_{bbbsb}, \end{aligned} \quad (10)$$

$$\begin{aligned} \text{cost}_{\mathcal{A}}(x_4) = & 3\mathcal{A}_{aasas} + 4\mathcal{A}_{aasbb} + 5\mathcal{A}_{abaab} \\ & + 5\mathcal{A}_{ababb} + 4\mathcal{A}_{abbas} + 5\mathcal{A}_{abbbb} \\ & + 5\mathcal{A}_{baaab} + 5\mathcal{A}_{baabb} + 4\mathcal{A}_{babas} \\ & + 5\mathcal{A}_{babbb} + 5\mathcal{A}_{bbaab} + 5\mathcal{A}_{bbabb} \\ & + 4\mathcal{A}_{bbbas} + 5\mathcal{A}_{bbbbb}, \end{aligned} \quad (11)$$

$$\begin{aligned} \text{cost}_{\mathcal{A}}(x_5) = & \mathcal{A}_{bssss} + 5\mathcal{A}_{aaaab} + 4\mathcal{A}_{aaabs} \\ & + 3\mathcal{A}_{aabss} + 2\mathcal{A}_{absss}, \end{aligned} \quad (12)$$

$$\begin{aligned} \text{cost}_{\mathcal{A}}(x_6) = & 2\mathcal{A}_{basss} + 5\mathcal{A}_{bbaab} + 4\mathcal{A}_{bbabs} \\ & + 3\mathcal{A}_{bbbsb} + 5\mathcal{A}_{aaaab} + 4\mathcal{A}_{aaabs} \\ & + 3\mathcal{A}_{aabss} + 5\mathcal{A}_{abaab} + 4\mathcal{A}_{ababs} \\ & + 3\mathcal{A}_{abbss}, \end{aligned} \quad (13)$$

$$\begin{aligned} \text{cost}_{\mathcal{A}}(x_7) = & 2\mathcal{A}_{bsass} + 4\mathcal{A}_{bsbab} + 3\mathcal{A}_{bsbbs} \\ & + 5\mathcal{A}_{aaaab} + \mathcal{A}_{4baabs} + 5\mathcal{A}_{aabab} \\ & + 4\mathcal{A}_{aabbs} + 3\mathcal{A}_{abass} + 5\mathcal{A}_{abbab} \\ & + 4\mathcal{A}_{abbbs}, \end{aligned} \quad (14)$$

$$\begin{aligned} \text{cost}_{\mathcal{A}}(x_8) = & 3\mathcal{A}_{basas} + 4\mathcal{A}_{basbb} + 5\mathcal{A}_{bbaab} \\ & + 5\mathcal{A}_{bbabb} + 4\mathcal{A}_{bbbas} + 5\mathcal{A}_{bbbbb} \\ & + 5\mathcal{A}_{aaaab} + 5\mathcal{A}_{aaabb} + 4\mathcal{A}_{aaabs} \\ & + 5\mathcal{A}_{aabbb} + 5\mathcal{A}_{abaab} + 5\mathcal{A}_{ababb} \\ & + 4\mathcal{A}_{abbas} + 5\mathcal{A}_{abbbb}. \end{aligned} \quad (15)$$

Consider now a weighted combination of the above eight equations, using weights proportional to the c_i , to form a new equation which will yield the desired bound. We add 3 times (8), (10), 2 times (9), (11), 3 times (12), (14), 2 times (13), and (15). We simplify the result by replacing \mathcal{A}_{bz} by \mathcal{A}_{az} for each z of length 4, by replacing \mathcal{A}_{yb} and \mathcal{A}_{ys} by \mathcal{A}_y for each y of length 4, and then combining terms. We then reduce the inequality by using the online constraints. As an example, we have $\mathcal{A}_{aass} = \mathcal{A}_{aasa} + \mathcal{A}_{aasb} = \mathcal{A}_{aas}$. Also, all values of \mathcal{A} are non-negative. Thus $4\mathcal{A}_{aass} + 6\mathcal{A}_{aasa} + 8\mathcal{A}_{aasb} = 10\mathcal{A}_{aas} + 2\mathcal{A}_{aasb} \geq 10\mathcal{A}_{aas}$. By the fundamental and online constraints,

$$\mathcal{A}_a + \mathcal{A}_b = \mathcal{A}_e = 1.$$

We assume \mathcal{A} is symmetric, hence $\mathcal{A}_a = \frac{1}{2}$. Altogether, we have

$$\begin{aligned}
& 14 \sum_{i=1}^8 c_i E \text{ cost}_{\mathcal{A}}(x_i) \\
&= 6\mathcal{A}_{asss} + 8\mathcal{A}_{asas} + 16\mathcal{A}_{asba} + 12\mathcal{A}_{asbb} \\
&\quad + 4\mathcal{A}_{aass} + 6\mathcal{A}_{aasa} + 8\mathcal{A}_{aasb} + 70\mathcal{A}_{aaaa} \\
&\quad + 58\mathcal{A}_{aaab} + 24\mathcal{A}_{aabs} + 28\mathcal{A}_{aaba} + 26\mathcal{A}_{aabb} \\
&\quad + 12\mathcal{A}_{abss} + 12\mathcal{A}_{abas} + 40\mathcal{A}_{abaa} \\
&\quad + 36\mathcal{A}_{abab} + 12\mathcal{A}_{abbs} + 36\mathcal{A}_{abba} + 36\mathcal{A}_{abbb} \\
&\geq 6\mathcal{A}_{ass} + 8\mathcal{A}_{asa} + 12\mathcal{A}_{asb} + 10\mathcal{A}_{aas} + 58\mathcal{A}_{aaa} \\
&\quad + 50\mathcal{A}_{aab} + 12\mathcal{A}_{abs} + 48\mathcal{A}_{aba} + 48\mathcal{A}_{abb} \\
&\geq 14\mathcal{A}_{as} + 60\mathcal{A}_{aa} + 60\mathcal{A}_{ab} \\
&= 74\mathcal{A}_a \\
&= 37
\end{aligned}$$

which completes the proof. \square

Lemma 2 implies our lower bound:

Theorem 2. $\frac{37}{24}$ is a lower bound on the competitiveness of any trackless randomized online algorithm for the paging problem for $k = 2$.

3.3. Additional computational results

Additional lower bounds were found by generating a linear program based on Eqs. (4)–(7) of Section 3.1. We considered request sequences of length n . To that end, in the online and fundamental constraints, we replaced the “for all y ” by “for all $|y| \leq n$ ”. For the competitive constraints, “for all x ” was replaced by “for all $|x| \leq n$ ”, and the individual constraints were generated using the parsing technique described in of Section 3.1. The objective function “min C ”, was added. We then wrote a program that takes as input n and generates the corresponding linear program. Using sequences of greater length increases (exponentially) the size of the linear program, as the number of variables and constraints grows exponentially with the allowed length of the adversary strategy. Using the CPLEX software package at the Max-Planck-Institut für Informatik, Saarbrücken, we were able to allow sequences of length up to 11, and ultimately obtained a lower bound of approximately 1.5487358. Our computational results are summarized in Table 3.

Table 3
Lower bound computational results

n	Fraction	Decimal approximation
3	3/2	1.5000000000
4	3/2	1.5000000000
5	37/24	1.5416666667
6	37/24	1.5416666667
7	303/196	1.5459183673
8	319/206	1.5485436893
9	319/206	1.5485436893
10	3723/2404	1.5486688852
11	8453/5458	1.5487358007

4. Notes and open questions

The minimum number of bookmarks needed for an optimally competitive (i.e., H_k -competitive) randomized online algorithm for the paging problem for $k > 2$ is unknown. It is bounded above by $\lceil 5k^2 H_k \rceil$ [1]. We conjecture that there is an optimally competitive algorithm with its number of bookmarks bounded by $\Omega(\log k)$ and $O(k)$. We also conjecture that for $k = 3$, only one bookmark suffices.

The optimal competitiveness of any trackless randomized algorithm for the paging problem for any $k \geq 2$ is unknown, but is definitely greater than $\frac{3}{2}$ and less than 2 for $k = 2$. The best-known such algorithm, MARK, is known not to be optimal, and for $k = 2$, MARK is 2-competitive. However, a trackless randomized algorithm for the same problem with competitiveness less than 2 is known [6].

References

- [1] D. Achlioptas, M. Chrobak, J. Noga, Competitive analysis of randomized paging algorithms, in: Proc. 4th European Symp. on Algorithms, Lecture Notes in Comput. Sci., Vol. 1136, Springer, Berlin, 1996, pp. 419–430.
- [2] W.W. Bein, L.L. Larmore, Trackless online algorithms for the server problem, Inform. Process. Lett. 74 (2000) 73–79.
- [3] A. Borodin, R. El-Yaniv, Call admission and circuit-routing, in: Online Computation and Competitive Analysis, Cambridge University Press, 1997.
- [4] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998. <http://www.cup.org/Titles/56/0521563925.html>.

- [5] A. Borodin, S. Irani, P. Raghavan, B. Schieber, Competitive paging with locality of reference, *J. Comput. Systems Sci.* 50 (1995) 244–258.
- [6] M. Chrobak, E. Koutsoupias, More on competitive analysis of online algorithms for caching, Unpublished manuscript.
- [7] M. Chrobak, L.L. Larmore, HARMONIC is three-competitive for two servers, *Theoret. Comput. Sci.* 98 (1992) 339–346.
- [8] S. Irani, R. Rubinfeld, A competitive 2-server algorithm, *Inform. Process. Lett.* 39 (1991) 85–91.
- [9] P. Raghavan, M. Snir, Memory versus randomization in on-line algorithms, *IBM J. Res. Development* 38 (1994) 683–707.
- [10] D. Sleator, R.E. Tarjan, Amortized efficiency of list update and paging rules, *Comm. ACM* 28 (1985) 202–208.