Paging with connections: FIFO strikes again

Leah Epstein [a] Yanir Kleiman [b] Jiří Sgall [c,1] Rob van Stee [d,2]

[a] *Department of Mathematics, University of Haifa, 31905 Haifa, Israel.*

[b] *The Academic College of Tel-Aviv Yaffo, Antokolski 4 61161 Tel-Aviv, Israel.*

[c] *Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic, and Dept. of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Praha.*

[d] *Department of Computer Science, University of Karlsruhe, P.O. Box 6980, D-76128 Karlsruhe, Germany. Tel. nr. +49-721-6086781, Fax +49-721-6083088*

Abstract

We continue the study of the integrated document and connection caching problem. We focus on the case where the connection cache has a size of one and show that this problem is not equivalent to standard paging, even if there are only two locations for the pages, by giving the first lower bound that is strictly higher than k for this problem. We then give the first upper bound below the trivial value of $2k$ for this problem. Our upper bound is $k + 4\ell$ where ℓ is the number of locations where the requested pages in a phase come from. This algorithm groups pages by location. In each phase, it evicts all pages from one location before moving on to the next location. In contrast, we show that the LRU algorithm is not better than $2k$-competitive.

We also examine the resource augmented model and show that the plain FIFO algorithm is optimal for the case $h = 2$ and all $k \geq 2$, where h is the size of the offline document cache. We show that also in this case FIFO is better than LRU, although this is not true for standard paging.

Keywords: paging, connection caching, online algorithms

Email addresses: `lea@math.haifa.ac.il` (Leah Epstein), `yanirk@gmail.com` (Yanir Kleiman), `sgall@math.cas.cz` (Jiří Sgall), `vanstee@ira.uka.de` (Rob van Stee).

[1] Partially supported by research project MSM0021620838 of MŠMT ČR.

[2] Corresponding author. Research supported by the Alexander von Humboldt Foundation.

1 Introduction

The caching problem is one of the major problems in the field of online algorithms. It has been studied considering many different models and aspects of the problem. The basic problem comes from the real world of computing. An operating system has a limited amount of high speed memory and needs to decide which pages of data will be kept in the high speed memory and which pages will be discarded. When a page that was recently discarded is requested again, the operating system pays a significant cost in order to retrieve that page from the low speed memory. This decision must be made with each new request, therefore the solution cannot be pre-computed and must be addressed with an online algorithm.

The abstract representation of the problem is as follows. The operating system has k pages of data in its cache. There is no a priori upper bound on the total number of pages that can be requested. An online algorithm is an algorithm that decides after each request which pages to keep in its cache and which pages to discard without knowledge of future requests. The goal is to minimize the number of page faults, i.e. requests to pages while they are not in cache.

Connection caching is a version of this problem that deals with a small cache of network connections that has to be maintained in order to access data from different locations. Recently, a combined model was suggested by Albers and van Stee [1]. The model deals with both document caching and connection caching. In this model we have two caches, one that holds documents (or *pages*) and one that holds *connections.*

The algorithms in [1] perform best if the size of the connection cache is equal to the size of the document cache. However, in practice, the connection cache is usually much smaller. In this article we focus on the case where we have a cache of k pages and only one slot in the connection cache. We call this problem "Paging with Connections". In such a model the connection that is stored in the connection cache can be referred to as the (single) current location. There are three types of requests. If the requested page is already in cache, the request costs nothing. If the requested page is not in cache but is from the current location, the request is noted as a *page fault* (or *regular fault*) and the cost is one. If the requested page is not in cache and is from a different location, the request is noted as a *connection fault* and it costs 2, one for switching the connection and another one for putting the requested page in cache.

The competitive ratio is the asymptotic worst case ratio between the cost of an online algorithm and the cost of an optimal offline algorithm (denoted by OPT) which knows all requests in advance. Throughout the paper we refer to the online algorithm as ALG and to the optimal offline algorithm as OPT wherever they are mentioned.

A generalization of the model is the *resource augmentation* model. In this model the adversary has less pages in its cache than the online algorithm. This model obviously gives an advantage to the online algorithm and may reduce the competitive ratio. This approach was for instance used by Sleator and Tarjan [12] to give more realistic results for the standard paging problem.

Previous work. *The classic paging problem.* Two common paging algorithms for the classical paging problem are LRU (Least Recently Used) and FIFO (First In First Out). LRU computes for each page which is present in the cache the last time it was used and evicts the page for which this time is minimum. FIFO acts like a queue, evicting the page that has spent the longest time in the cache. Variants of both are common in real systems. Although LRU outperforms FIFO in practice, LRU and FIFO are known to have the same competitive ratio of k. Further this ratio is known to be the best possible, see [12,10].

The bound k for the standard problem is actually derived directly from a general bound which Sleator and Tarjan [12] showed and which allows resource augmentation. That is a lower bound of $\frac{k}{k-h+1}$ for every online algorithm when the caches of OPT and ALG have sizes of h and k ($h \leq k$) respectively. They also showed a matching upper bound for LRU and FIFO.

Randomized algorithms. A randomized paging algorithm called MARK was introduced in [8] with a competitive ratio of $2H_k$, where H_k is the k-th harmonic number. Randomized algorithms that achieve a better competitive ratio of H_k were introduced in [11] and later in [3].

Arbitrary page sizes or costs. In practice not all requested pages are of the same size or cost. Research of models where every page has an arbitrary size was done by Irani [9]. She considered a model where every page has a cost of 1 and a model where every page's cost is equal to its size, and presented randomized online algorithms with a competitive ratio of $O(\log^2 k)$ for both models. Young [13] presented a deterministic k-competitive online algorithm for the general case where pages have arbitrary sizes and an arbitrary non-negative loading cost.

Connection caching. Cohen, Kaplan and Zwick [5,6] introduced the connection caching problem, where there is a sequence of requests for TCP connections. Whenever there is a request for a connection that is not open, it must be established. When the cost of opening a connection is uniform over all connections, they presented a k-competitive deterministic online algorithm and a $O(H_k)$-competitive randomized online algorithm, where k is the number of connections that can be simultaneously maintained. The k-competitive deterministic algorithm was developed independently by Cao and Irani [2].

Combined model. As stated above, an integrated document and connection caching model was introduced by Albers and van Stee [1]. Denote the document cache size by k and the connection cache size by k'. Their main result is a deterministic algorithm with competitive ratio $2k - k' + 4$. This result is then adapted for other models like randomized algorithms and Irani's models of pages of arbitrary size.

It can be seen that the performance guarantee of this algorithms is near-optimal for a large connection cache, but degrades as the connection cache becomes smaller and is only $2k$ for a connection cache of size 1. In this paper, we focus on the practically important case of a small connection cache. Specifically, we focus on the case $k' = 1$, i.e. there is only one slot in the connection cache. Albers and van Stee [1] show that LRU and FIFO perform poorly in this scenario, giving a lower bound of $2k - O(1)$ for these algorithms.

FIFO *versus* LRU. An issue in the analysis of paging algorithms is the relative performance of FIFO and LRU. Although LRU outperforms FIFO in practice, standard competitive analysis cannot distinguish between these two algorithms. This was finally accomplished by Chrobak and Noga [4] using a more refined model. Interestingly, there are several paging models in which the opposite can be shown. One such model was presented by Epstein, Imreh and van Stee [7]. We show in this paper that this role reversal also occurs in paging with connections.

New results. We show that this problem is different from the standard paging problem, by showing a lower bound of $k+\frac{k-1}{k+1}$ for $k > 2$ and $\frac{5}{2}$ for $k = 2$. Recall that the best competitive ratio for standard paging is k. Clearly, the k-competitive algorithms for the standard problem are at most $2k$-competitive for our model, since each fault costs an algorithm at most 2. We show that LRU is not better than this bound. However, FIFO achieves a better bound for $k = 2$, which is the best possible ratio for this case and equals $\frac{5}{2}$.

We further design an algorithm which performs much better than LRU. Its competitive ratio is $k + O(\ell)$, where ℓ is the number of different locations. This ratio is $k + 8$ if there are two locations.

We explore the resource augmented problem, where OPT has less pages in its cache. We show that in this model as well, our problem is different, since a lower bound we show is higher than $\frac{k}{k-h+1}$ for every $h < k$. We focus on the case $h = 2$ and show that the tight bound in this case is $\frac{k+3}{k}$, achieved by

FIFO. This is in contrast to LRU that has competitive ratio of exactly $\frac{k+2}{k-1}$.

Notation. *A* page is denoted by a lower case letter together with zero or more apostrophes, or an index. A location is denoted by a capital letter, which usually (unless stated otherwise) matches the letter of the pages that are at that location. For example, at location A we may have the pages $a, a', a_1 \dots$

We write a state of an algorithm as abC for $k = 2$, and generalize this in the obvious manner for larger k. When we compare the state of an online algorithm to the state of the offline algorithm, we write the online state first and separate them by a slash. For example abB/abA. If there is an order in which the pages are to be evicted (e.g. in the cache of LRU or FIFO), then the leftmost page in this writing is first to be evicted, and the rightmost is last.

2 Lower bounds

We prove the following theorem, showing that even the problem with $k' = 1$ is not equivalent to the standard paging problem, even if only two distinct locations are present.

Theorem 1 *Any online algorithm for the paging problem with connections has a competitive ratio at least* $k + \frac{k-1}{k+1}$.

Proof. We use two locations and construct a sequence which consists of superphases, where each superphase consists of phases as defined in the sequel.

At the time when a superphase starts, OPT has pages from a single location in its cache. During the superphase, OPT inserts into its cache pages from the other location. *A* new superphase starts when OPT has pages from the other location only. In the beginning of a superphase, both OPT and ALG are in the same location, the location from which OPT has all its pages at that time. We call this location A and the other location B.

We next define the phases of a superphase. The length of each phase is at least k. The first phase starts with a request for page $b \in B$. This is a page from B that ALG does not have in its cache. Let $a_1, \dots, a_k$ denote the pages from A that reside in the cache of OPT before the request for b. Every subsequent request of this phase is made for the page in $\{a_1, \dots, a_k, b\}$ that ALG currently does not have in its cache. These requests are made until the earliest time where exactly one page $a_j \in \{a_1, \dots, a_k\}$ was not yet requested, and this ends the first phase. OPT replaces this single page by b in the beginning of the phase, and therefore faults only on b. In this phase, OPT has a regular fault as well as a connection fault. In all other phases in this superphase OPT will only have one regular fault. Note that ALG is not at location B when the first phase ends, since the last request of the phase must be for a page from $\{a_1, \ldots, a_k\}$.

We repeat a similar process until after some phase OPT has only pages from B. In this process the first page of a phase is a new page (that is not present in any of the two caches) from B. The rest of the pages are a subset of $k-1$ pages out of the contents of the cache of OPT in the start of the phase. Therefore, except for the beginning of the very first phase, OPT has pages of both locations in its cache. At most one page is replaced in the cache of OPT in each phase, and thus a superphase lasts at least k phases. Note that at the end of the superphase, both algorithms are connected to B.

ALG faults on every request, therefore it has at least k regular faults per phase. It is left to count the number of connection faults of ALG. In the first phase, ALG has a connection fault twice: once at the first request, and another one at the second request, which must be for a page of A. All phases but the last one must have requests for pages of both locations, since once a phase has requests for pages of B only the superphase ends, and each phase has at least one request for a page of B. Let $s \geq k$ be the number of phases. We find a total of at least $1 + 2(s-1)$ connection faults for ALG, and a total cost of at least $sk + 2s - 1$, whereas the total cost of OPT is $s + 1$. We get the ratio

$$\frac{sk + 2s - 1}{s + 1} \geq k + 1 - \frac{2}{s+1} \geq k + 1 - \frac{2}{k+1} = k + \frac{k-1}{k+1}.$$

This concludes the proof. ■

For the cache size equal to 2, we can prove a slightly stronger lower bound.

Theorem 2 *Any online algorithm with $k = 2$ slots in the cache has a competitive ratio of at least $\frac{5}{2}$, even if only two locations are allowed.*

Proof. Denote the two locations by A and B. We start in state abB/abA or $abB/a'bA$. That is, OPT has one page from location A in its cache, which may or may not be the same page as the online algorithm has. We show how to return to one of these states in such a way that any online algorithm pays at least 5/2 times the optimal cost. The optimal cost in each such phase is non-zero and bounded by a constant. The cost of reaching such an initial state is also constant. Thus repeating the phase sufficiently many times proves that the competitive ratio is arbitrarily close to $\frac{5}{2}$.

The main line of the lower bound is as follows (only writing states of the online algorithm):

$$abB \xrightarrow{a'} ba'A \xrightarrow{a} a'aA \xrightarrow{b} a'bB \text{ or } abB.$$

OPT serves this request sequence by going to $a'bA$ and then back to abA, at cost at most 2 (the cost is 1 if it already had a' in its cache). The online algorithm pays 5. Note that the final states are equivalent to the allowed starting states.

We now check what happens if the online algorithm deviates from this path at the first or second request. In these cases, the request sequence continues differently. Note that no deviation is possible at the last request.

(1) The online state after the first request is $a'aA$. The next request is to b. We go back to a starting state: OPT goes to $a'bA$ for the first request and stays there. The ratio is $4:1$.

(2) The online state after the second request is baA. OPT goes to $a'aA$ for the first request and stays there. We now alternate between requests to a' and a until the online algorithm goes to $a'aA$ as well.
Then the next request is to b, followed by a request to the page that the online algorithm drops to serve this request (a or a'). OPT goes to location B to serve the request to b, keeping the page that the online algorithm drops in its cache. The final state is thus abA/abB or $a'bA/a'bB$, equivalent to second starting state. The overall ratio is at least $8:3$.
Note: if the online algorithm drops b to serve the last request (i.e. it goes back to $a'aA$), we repeat alternating requests to b and the page from A that OPT has until it goes to abA or $a'bA$. The ratio only increases.

■

We further show that in contrast to the standard paging problem, the performance of LRU is very poor in this model. The competitive ratio of LRU is $2k$, which can be achieved by any algorithm that uses a marking strategy for the pages and an arbitrary strategy for the connection cache.

In particular, LRU has a ratio of 4 for $k = 2$, even using only two locations. This is shown as follows. We start in a state baB/abA, where b is the least recently used page (i.e. the last request was to a and did not cause a fault). The lower bound then works as follows:

$$baB/abA \xrightarrow{a'} aa'A/a'bA \xrightarrow{b} a'bB/a'bA \xrightarrow{a'} ba'B/a'bA.$$

LRU has two connection faults and pays 4. The optimal offline algorithm only pays 1 (for the first request). The final state is equivalent to the starting state.

The generalization for general k is as follows. Let $a, b_1, \ldots, b_{k-1}$ be k pages from k distinct locations. We denote the location of a by A. Let $a' \in A$ as well. The initial state of a phase is

$$b_1 b_2 \ldots b_{k-1} aX / b_1 b_2 \ldots b_{k-1} aA,$$

where X is the location of b_{k-1}. The phase consists of requests for $a', b_1, \ldots, b_{k-1}, a'$. All requests but the last one are both regular and connection faults. Starting from the first request, each of the pages b_i, $i = 1, \ldots, k-1$ are evicted and reloaded in the next step. The last step changes the priority of a'. The phase ends at the equivalent state $b_1 b_2 \ldots b_{k-1} a' X / b_1 b_2 \ldots b_{k-1} a' A$. OPT has a single regular fault, whereas LRU has k regular faults and k connection faults. Therefore the competitive ratio of LRU is $2k$.

Note that this example can be implemented using only two locations for even k and three for odd k.

3 Algorithms

In this section we introduce the first algorithms for paging with connections with a competitive ratio strictly smaller than $2k$ for the case where the number of network locations is less than $k/4$. We first describe an algorithm for the case where all requests come from only two locations, and then show how to generalize this algorithm for more locations.

3.1 Algorithm TwoLocations

Denote the location of the very first request by A and the other location by B. On any page fault, first, if all pages are marked, unmark all pages and start a new phase. Second, evict an unmarked page from A if possible. Otherwise evict the unmarked page which has been in the cache the longest. Load the requested page and mark it.

Note that pages are only marked when a page fault occurs. Thus this algorithm is not a marking algorithm in the classical sense. In particular, a phase might last longer than that of a marking algorithm, but all we need is that a phase contains requests to k distinct pages, so that each algorithm (in particular the optimal one) must fault at least once per phase.

Definition *A new* page is a page which was not requested in the previous phase.

Claim 1 *In any phase, the number of connection faults is at most four times the number of new pages requested.*

Proof. Consider a request sequence. Since TwoLocations does not change its state in response to requests that do not cause faults for it, we begin by removing any requests that do not cause faults. This can only decrease the optimal cost to serve this sequence. Thus we may assume that each request causes a fault for the algorithm TwoLocations.

Pages from B are not evicted until all pages from A which were requested in the previous phase have been evicted. Say that the first eviction of a page at B happens at the rth fault. This implies that in the previous phase, $r-1$ pages from A were requested.

We are going to assign connection faults to new pages, depending on when the connection faults occur, as follows:

(1) To the first new page (that starts the phase), the connection fault that possibly occurs on this request, and the one that possibly occurs on the first request to an old page from B
(2) Up to request r, at most two connection faults to each new page from B
(3) After request r, at most two connection faults to each new page requested *up to and including request r* (from A or B), apart from the first new page
(4) After request r, at most two connection faults to each new page from A

Thus each new page is assigned at most four connection faults. We now explain that this covers all the connection faults that may occur in the phase.

The first request may obviously cause a connection fault, as may the first request to an old page from B. This is handled in Step 1. If no other new pages are requested, these are all the connection faults that occur, since in this case first all the old pages from A are requested and then all the old pages from B.

Consider a request no later than r. At this point, no old pages from B can be requested, because they are all still in the cache and each request causes a fault. Only old pages from A can be requested (and, of course, new pages from anywhere). If this page fault causes a connection fault, the request must be to a new page from B, or to a page from A (either old or new) immediately after a new page from B was requested. This is handled in Step 2.

Consider a request after the rth request. Suppose that in the first r requests, there were b requests to a (new) page from B. Then there were $r-b$ requests to pages from A. Denote the number of requests to new pages from A by a. We have $a \geq 1$ because the first request, that started this phase, must be to a new page.

In the first r requests, there were $r - b - a$ requests to old pages from A. Since there exist $r - 1$ old pages from A, this means that $b + a - 1$ old pages from A are not yet requested in the current phase at this point. In the worst case, these are requested alternatingly with old pages from B now, leading to at most $2(b + a - 1)$ additional connection faults. These are assigned to the $b + a - 1$ new pages in the requests $2, \ldots, r$ in Step 3.

Finally, connection faults can also occur if new pages from A are requested after the rth request. Each such page can cause two connection faults: one to switch to A and one to switch back (which can happen on an old page from B). This is handled in Step 4.

This completes the proof. ■

Theorem 3 *TwoLocations (TL) has a competitive ratio of at most $k + 8$.*

Proof. Denote the number of new pages in phase i by p_i. Denote the cost of algorithm $\mathcal{A}$ in phase i by $\mathcal{A}(i)$. In phases $i - 1$ and i, $k + p_i$ distinct pages are requested. Thus $\textsc{opt}(i - 1) + \textsc{opt}(i) \geq p_i$ for all $i > 1$. Summing over all even phases $i > 1$, we have $\textsc{opt} \geq \sum_{i \text{ even}} p_i$. We have $\textsc{opt}(1) \geq p_1$ since by assumption $\textsc{opt}$ and TL start with the same cache contents. So summing over the odd phases gives $\textsc{opt} \geq \sum_{i \text{ odd}} p_i$. This implies that $2\textsc{opt} \geq \sum_i p_i$. Additionally, we have as usual $\textsc{opt} \geq n$ where n is the number of phases.

On the other hand, by Claim 1 we have $TL(i) \leq k + 4p_i$. Thus overall, we have

$$TL \leq nk + 4\sum p_i \leq k\textsc{opt} + 8\textsc{opt} = (k + 8)\textsc{opt}.$$

This completes the proof. ■

3.2 A generalized algorithm: Evict By Location (EBL)

This algorithm works in phases. During a phase, a page is marked if a request to this page caused a fault. Only unmarked pages are evicted, and only if there is a page fault. If there is a page fault and all pages are marked, all pages are unmarked and a new phase begins.

At the start of a new phase, group the pages in the cache by their locations. Sort the locations in any order, but start with the location of the current request (which starts this phase). Denote the locations by $A_1, A_2, \ldots$ in this order.

During a phase, evict all pages from A_i before evicting any page from A_{i+1}, for any i.

Theorem 4 *The algorithm EBL has a competitive ratio of at most $k + 4\ell$ for an environment with ℓ locations.*

Proof. We divide the phase into subphases according to the grouping defined above. In subphase i, only pages from A_i are evicted. Denote the length of subphase i by k_i. We generalize the connection faults assignment from the proof of TwoLocations, and prove that the connection faults can be assigned so that any new page requested in a phase is assigned at most 4ℓ connection faults, where ℓ is the number of different locations requested in this phase.

(1) To the first new page in the phase, we assign the first connection fault from each subphase
(2) In subphase i, to new pages that are not from A_i we assign at most two connection faults in subphases $i, \ldots, \ell$
(3) In subphase i, to new pages from A_i we assign at most two connection faults in subphases $i + 1, \ldots, \ell$

For each i, the first request in subphase i is not to an old page of A_i, because all of them are still in the cache at this point. The second request in subphase i (or the first one to A_i) will cause a connection fault. This is handled in Step 1 (note that the first page requested in the phase is new).

Consider subphase 1. All old pages from $A_2, \ldots$ are still in cache. Some old pages in A_1 may be out. Any new page not from A_1 can cause a connection fault, plus another one if the next request is for A_1 again. See Step 2. Suppose there are a_1 such pages. Denote the number of new pages from A_1 by $a'_1 \geq 1$. (Note that the first page in this phase is new.) Then after subphase 1, there are still $a_1 + a'_1$ old pages from A_1 which have not been requested in the current phase. We call these pages *late*.

In subphase 2, all old pages from A_3 and later are still in the cache. Some old pages in A_1 and A_2 may be out. Denote the number of new pages requested in this subphase which are not from A_2 by a_2. Each such page can cause two connection faults in this subphase (Step 2). This also holds for the $a_1 + a'_1$ late pages from A_1, if they are requested in subphase 2. Since these correspond to new pages requested in subphase 1, the connection faults are assigned to those new pages (Step 3). Finally, there are a'_2 new pages from A_2, which do not cause connection faults in the current subphase, but instead cause late pages.

In fact, any request out of these three categories corresponds to one old page from A_2 which is **not** requested in this subphase, and which might still be requested later and cause extra connection faults (Step 2 and 3). Generally, it may be seen that in each subphase, each new page requested corresponds in a one-to-one fashion to an old page which is not requested in its proper subphase. This concludes the explanation of the assignment procedure.

Denoting the number of new pages in subphase j of phase i by n_{ij}, we find that

$$\text{EBL }(i) \le k + \sum_{j=1}^{\ell} 2n_{ij}(\ell + 1 - j) \le k + 2\ell \sum_{j=1}^{\ell} n_{ij}.$$

Here we have used that in any phase, $\sum_{j=1}^{\ell} n_{ij} \ge 1$. On the other hand, $\text{OPT}(i-1) + \text{OPT}(i) \ge \sum_{j=1}^{\ell} n_{ij}$, and $\text{OPT} \ge n$ where n is the number of phases. Thus

$$\text{EBL } \le kn + 2\ell \sum_{i=1}^{n}\sum_{j=1}^{\ell} n_{ij} \le k\text{OPT} + 4\ell\text{OPT} = (k+4\ell)\text{OPT}.$$

This completes the proof. ■

4 Resource Augmentation

In this section we consider the case where the document cache of OPT has size $2 \le h \le k$. Recall that the tight bound on the competitive ratio in the standard model is $\frac{k}{k-h+1}$.

We begin by giving a lower bound for the case $h < k$. A lower bound for $h = k$ is given by Theorem 1. An interesting question is whether the competitive ratio for the problem with locations tends to 1 when k grows. In this section we show that the tight ratio for the case $h = 2$ is $\frac{k+3}{k}$ and thus it is slightly worse but still tends to 1. We show that this ratio is achieved by FIFO, whereas the competitive ratio of LRU is exactly $\frac{k+2}{k-1}$.

Theorem 5 *The competitive ratio of any online algorithm, such that the algorithm has k slots in its cache and* OPT *has $2 \le h < k$ slots is at least*

$$\frac{k + 2 + \lceil \frac{h-1}{k-h+1} \rceil}{k-h+2}.$$

This number is strictly higher than $\frac{k-h+1}{k}$ for any $h < k$.

Proof. We use phases. The only requirement for the starting state is that ALG is in a different location than OPT. We denote the pages of OPT by $a_1, a_2, \ldots, a_h$ (though they may be from different locations) and the location of OPT by B.

We issue requests for the following sequence of pages:

$$b_1, b_2, \ldots, b_{k-h} \in B.$$

These pages have in common that none of them appear in either the cache of OPT or the cache of ALG.

After $k-h$ such requests there is a request for a page p which is from a different location than all the pages in the caches of OPT and ALG. Every next step after that, ALG has in its cache k pages, thus at least one of the pages $q_i \in \{a_1, a_2, \ldots, a_h, b_1, b_2, \ldots, b_{k-h}, p\}$ is not present in its cache. The next request will be issued for page q_i. We issue such requests until at least $h-1$ different pages, not including page p, are requested. OPT can choose the pages it discards, so after the first $k-h$ requests and the request for page p it will have in its cache page p and the $h-1$ unique pages that were requested after it. Thus it does not pay for any request after the request for p.

Since there are at least $h-1$ requests after the request for page p, there are at least k requests in the phase. ALG pays at least 1 for any of the requests we issued. It pays 2 for at least three requests: The request for page b_1 (the first request), the request for page p and the request for page q_1 which must be from a different location from page p. In total ALG pays at least $k+3$ for the whole sequence.

OPT pays only 1 for each of the first $k-h$ requests and another 2 for the request for page p, to a total of $k-h+2$.

For large values of h, we can improve the estimate for the cost of ALG by taking into consideration the maximum amount of pages from the same location that ALG has in its cache. Since OPT uses a sequence of $k-h$ pages from the location where it was at the start of the phase, throughout the scheme we need only a maximum of $k-h+1$ pages from each location. Therefore ALG has to move to a different location at least every $k-h+1$ pages. This move costs ALG 2 instead of 1 and it occurs $\lceil \frac{h-1}{k-h+1} \rceil$ times. In other words, the cache of ALG holds pages from at least $\lceil \frac{h-1}{k-h+1} \rceil$ different locations.

Hence ALG pays at least $k+2+\lceil \frac{h-1}{k-h+1} \rceil$ and the lower bound is proved. ■

This lower bound is slightly larger than $\frac{k}{k-h+1}$ for all values of $h < k$. Note that the proof fails for $k = h$ since in this case there are no initial requests in the phase.

4.1 The case $h = 2$

For this case, we show matching upper and lower bounds. The optimal upper bound is achieved by FIFO. Additionally, we prove that the algorithm LRU performs strictly worse than FIFO even for $h = 2$, and give tight bounds on its competitive ratio.

Theorem 6 *Any online algorithm with $k \geq 2$ slots in the cache has a competitive ratio of at least $\frac{k+3}{k}$ when compared to an offline algorithm with 2 slots* *in the cache.*

Proof. For $k > 3$, this follows directly from Theorem 5. For $k = 2$, this is Theorem 2. ■

Theorem 7 *The competitive ratio of* FIFO *for* $h = 2$ *is* $\frac{k+3}{k}$.

Proof. The lower bound was proved in Theorem 6. Consider now a sequence of requests. Since in FIFO only faults change the behavior, we are only interested in requests that were faults of FIFO, and remove all other requests form the sequence (this may only increase the cost of OPT and does not change the cost of FIFO).

We use the names "bad fault" for a fault of FIFO which is not a fault of OPT and "good fault" for a fault of both OPT and FIFO. A bad fault is a fault that raises the competitive ratio of FIFO and a good fault is a fault that helps FIFO maintain a low competitive ratio because OPT pays for it as well. We define a phase to be a sequence of faults which starts with a bad fault and ends one request before the next bad fault. In other words, the first request in the phase is a bad fault and the other faults in the same phase, if any, are all good faults.

We begin by giving a lower bound of k on the length of any phase. Let p be a request which causes a bad fault, and let q denote the previous request. Already before the request for page p, OPT must have in its cache page q that was just requested and page p. Just after p is requested, FIFO has in its cache pages p, q and $k-2$ older pages.

If the next bad fault will be on page q it has to occur after at least $k-1$ faults in order for page q to be discarded by FIFO. Similarly, for the next bad fault to be on page p it has to occur after at least k good faults. For the next bad fault to be on a different page, it must occur after at least $k+1$ good faults since one good fault is necessary to put the new page in the cache of OPT, and afterwards it takes k faults for FIFO to discard it. This shows that in all cases, there are at least $k-1$ good faults after the bad fault that starts the phase.

The bad faults cost a maximum of 2 for FIFO and nothing for OPT. The first good fault may cost 2 for FIFO and only 1 for OPT. After that both of the algorithms are in the same location, so the next $k-2$ faults costs the same for FIFO and OPT. The worst case occurs if all these faults cost 1. In this case FIFO pays $k+2$ and OPT pays $k-1$. We will call a phase with this exact ratio a "bad phase" for FIFO.

However, during a bad phase OPT does not change its location. Moreover, the next phase starts with a request for page q. This request must be a fault of OPT. This is true since for $k \geq 2$, by definition of FIFO, any three consecutive requests must be distinct, therefore, among two consecutive requests, at least one is a fault. Since the first request for page q was a fault of OPT (since the request just after it, which is for p is not a fault of OPT), OPT is still in the location of page q, and so is FIFO which is in the same location.

Therefore in any phase which immediately follows a bad phase, FIFO only pays 1 for the first request, and the same as OPT for the remaining requests since they are still in the same location. The ratio of the costs in this phase is at most $k : (k-1)$. This is a "good phase" for FIFO.

Every bad phase must be followed by at least one good phase, which yields an average ratio of $(2k+2) : (2k) = (k+1) : k$ or even better. To avoid good phases, OPT must either change the connection or prolong the phase so another page will be requested in the beginning of the next phase.

To change its connection OPT must at some point suffer a cost of 2 instead of 1, thus raising its total cost to k. Such a change of connection may cost FIFO 2 instead of 1 if done on one of the last $k-2$ requests, or nothing extra if done on the second request for which FIFO already pays 2. Therefore the maximum cost FIFO will suffer in this case will be $k+3$ instead of $k+2$ and the ratio of the phase is $(k+3) : k$. We will call such a phase a "normal phase".

If OPT does not change its location, the phase must be prolonged by at least one request. If the phase is prolonged by exactly one request the bad fault will again be the request for page p, and therefore the next phase may be a bad phase or a normal phase. In the prolonged phase, the cost of both OPT and FIFO is increased by one or two so the ratio of the phase is also $(k+3) : k$. This is also a "normal phase".

Since for every additional request OPT and FIFO pays the same, prolonging the phase by more than one request will yield a ratio lower than $(k+3) : k$.

Considering the entire request sequence, the average cost ratio is at most $(k+1) : k$ in good and bad phases (since there are at least as many good phases as bad phases) and at most $(k+3) : k$ in normal phases. This proves the upper bound. ■

Finally, we consider the performance of the algorithm LRU for the case $h = 2$. We start with an example where LRU achieves a ratio not better than $\frac{k+2}{k-1}$, and then show a matching upper bound. This is worse than the upper bound of FIFO. The example holds for $k > 2$, an example for $k = 2$ was given earlier.

Finally, we prove the following theorem.

Theorem 8 *The competitive ratio of* LRU *for $h = 2$ is exactly $\frac{k+2}{k-1}$, which is strictly worse than* FIFO*.*

Proof. We first show the lower bound. For simplicity, let $a_1, \ldots, a_{k-1}, a_k = a_0$ be pages from a location A, and a be a page of location A. The initial state of a phase is

$$a_1 a_2 \ldots a_{k-2} b a_{k-1} B / a_{k-1} b A.$$

The phase starts with requests for $a_k, a_1, \ldots, a_{k-2}$. At this time LRU does not have b in its cache anymore and this is the next request. The last request is for a_{k-2}. This is not a fault for any of the two algorithms, but it changes the state of LRU which is now $a_k a_1 \ldots a_{k-3} b a_{k-2} B$. OPT is in the state $a_{k-2} b A$, it had $k-1$ regular faults on the first $k-1$ requests. Note that OPT never changes its location. LRU has a connection fault on the first page of a phase and on b. It has regular faults on all requests but the last one. Therefore the costs of LRU are $k+2$ and the optimal cost is $k-1$. Note that the state we reached at the end of a phase is equivalent to the initial one.

We now prove the upper bound. We use terminology similar to the proof of Theorem 7, only here we cannot assume that all requests are faults for the algorithm, therefore there are "good faults", "bad faults" and "non-faults" which are requests that are not a fault for any of the two algorithms, and "excellent faults" which are faults of OPT but not of the algorithm.

A change that we do in the sequence without changing the operation of LRU or the cost is removal of all requests but one, in sub-sequences of consecutive identical requests.

A phase is the sub-sequence of requests, from the time that a bad fault occurs, until just before the next such time. Similarly to the proof for FIFO, we denote the first request in a phase by p and the request just before that by q. OPT must have had a fault on q. Otherwise the request just before q, r, must be identical to p or to q. Since we removed multiple consecutive instances of the same request, it must be p. However, if p appeared just before q, it is still in the cache of LRU, and cannot be a fault of LRU.

After the first good fault and until the end of the phase, LRU pays no more than OPT. This can be seen as follows. Immediately after the good fault, LRU and OPT are at the same location. This does not change as long as there are good faults. There can be no bad faults by definition. LRU and OPT pay the same as long as they remain at the same location. If OPT changes its location and LRU does not, this happens on a request that is not a fault for LRU. The cost that OPT has for this fault can be assigned to the first good fault that follows. Note that we showed above that the phase ends in a good fault. This shows that LRU does not pay more than OPT after its first good fault.

Between the bad fault that starts the phase and the first good fault, LRU pays nothing. However LRU possibly pays two more than OPT for the bad fault that starts the phase, and one more than OPT for the first good fault. Thus its overall cost for the phase is at most 3 more than OPT.

Denote the page that starts the next phase (i.e. causes a bad fault) by p'. After the last time that p' is requested, OPT keeps p' in its cache (otherwise it would not have a reason to load it again before the next request to p', and that means that would not be a bad fault). ¿From this point on, OPT can at most serve the very next request without a fault, but after that it has to fault on every request since it has only one free slot in the cache. There are at least k requests until LRU drops p', so OPT pays at least $k-1$ (more if there are also connection faults). Note that p' may be p or q.

This completes the proof. ■

5 Conclusions

We have shown the first algorithms for paging with connections that break the trivial performance bound of $2k$. An open question left by these results is the precise role that the locations play in this model. All our lower bounds require only two or three locations, and it is unclear whether using more locations can improve these results. However, it is possible that the competitive ratio of an adaptation of our algorithm EBL is smaller than $2k$ even for a large number of locations. The idea is to define the size of a location at the time when a phase starts as the number of pages from this location which are present in the cache of the algorithm. Then we use a sorted list of locations (largest location first).

References

[1] Susanne Albers and Rob van Stee. A study of integrated document and connection caching. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Proceedings*, pages 653-667, 2003.

[2] Pei Cao and Sandy Irani. Cost-aware www proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[3] Dimitris Achlioptas, Marek Chrobak and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234:203-218, 2000.

[4] Marek Chrobak and John Noga. LRU is better than FIFO. *Algorithmica*, 23:180–185, 1999.

[5] Edith Cohen, Haim Kaplan and Uri Zwick. Connection caching. In *Preceedings of the 31st ACM Symposium on the Theory of Computing*, pages 612-621. ACM, 1999.

[6] Edith Cohen, Haim Kaplan and Uri Zwick. Connection caching under various models of communication. In *Preceedings of the Twelfth ACM Symposium on Parallel Algorithms and Architectures*, pages 54-63. ACM, 2000.

[7] Leah Epstein, Csanad Imreh, and Rob van Stee. More on weighted servers or FIFO is better than LRU. *Theoretical Computer Science*, 306(1-3):305–317, 2003.

[8] Amos Fiat, Richard Karp, Michael Luby, Lyle A. McGeoch, Daniel Sleator and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685-699, Dec 1991.

[9] Sandy Irani. Page replacement with multi-size pages and applications to web caching. In *Preceedings of the 29th ACM Symposium on the Theory of Computing*, pages 701-710, 1997.

[10] Anna Karlin, Mark Manasse, Lyle Rudolph, and Daniel Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[11] Lyle McGeoch and Daniel Sleator. A strongly competitive randomized paging algorithm. *J. Algorithms*, 6:816-825, 1991.

[12] Daniel Sleator and Robert E. Tarjan. Amoritzed efficiency of list update and paging rules. *Communications of the ACM*, 28:202-208, 1985

[13] Neal E. Young. On-line file caching. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 82-86, 1998.