Combining Request Scheduling with Web Caching ⋆

Tomás Feder [a], Rajeev Motwani [b,1], Rina Panigrahy [c], Steve Seiden [2], Rob van Stee [d,3], An Zhu [b,4]

[a] *268 Waverley St., Palo Alto, CA 94301.*

[b] *Department of Computer Science, Stanford University, Stanford, CA 94305.*

[c] *Cisco Systems.*

[d] *Centre for Mathematics and Computer Science (CWI), P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands.*

Abstract

We extend the classic paging model by allowing reordering of requests under the constraint that a request is delayed by no longer than a predetermined number of time steps. We first give a dynamic programming algorithm to solve the offline case. Then we give tight bounds on competitive ratios for the online case. For caches of size k, we obtain bounds of $k + O(1)$ for deterministic algorithms and $\Theta(\log k)$ for randomized algorithms. We also give bounds for the case where either the online or the offline algorithm can reorder the requests, but not both. Finally, we extend our analysis to the case where pages have different sizes.

⋆ A preliminary version of part of this paper (coauthored Feder, Motwani, Panigrahy, and Zhu) appeared in *Proceedings of the* 13^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. ACM/SIAM, 2002.

Email addresses: tomas@theory.stanford.edu (Tomás Feder), rajeev@cs.stanford.edu (Rajeev Motwani), rinap@cisco.com (Rina Panigrahy), Rob.van.Stee@cwi.nl (Rob van Stee), anzhu@cs.stanford.edu (An Zhu).

[1] Supported in part by NSF Grant IIS-0118173, an Okawa Foundation Research Grant, and Veritas.

[2] Supported by AFOSR grant No. F49620-01-1-0264.

[3] Supported by the Deutsche Forschungsgemeinschaft, Project AL 464/3-1, and by the European Community, Projects APPOL and APPOL II. Work performed while this author was at Department of Computer Science, Freiburg University, Germany.

[4] Supported by a GRPW fellowship from Bell Labs, Lucent Technologies.

1 Introduction

Current web caching algorithms serve requests in the order of arrival, despite the fact that web requests are essentially independent and better hit rates may be achieved by reordering the requests. (Such a phenomenon is even more evident at high volume websites). A natural constraint is that no request should be delayed for long. We propose the following online r-reordering problem: given an HTTP request sequence $L_0 = (x_1, x_2, \ldots, x_n)$, the proxy server can reorder the sequence as long as in the new ordering $x_i \prec x_j$ only if $i - r < j$. We call r the reordering parameter. The goal is to minimize the total number of misses(faults) for a given cache of size k and reordering parameter r.

We study such a model in the context of online algorithms and competitive analysis. Our model extends the classical paging model by allowing reordering among nearby requests in the sequence. In our model, the online algorithm can see the page requests that can be served currently, while respecting the r-reordering rule. Thus, a virtual window w of length r is positioned at the first unserved page of the sequence. The window includes the current page request and $r-1$ subsequent pages in the sequence, regardless of whether some of the pages might have been served already. All the unserved pages in the window w are legal pages to be served next, hence visible to the online algorithm. Our work also extends the reordering model to l-lookahead ($l > r$), which allows an online algorithm to see more future requests. A virtual window w' of length l is positioned at the first unserved page of the sequence, and all pages within w' are visible to the algorithm, though only pages in the first r positions can be served next. As before, we require a page to be brought into cache before it can be served. The measurement is then the total number of cache misses. The competitive ratio is the ratio of the number of cache misses incurred by our online algorithm vs. that of the optimal offline algorithm.

We begin the study of the online algorithms by considering a cache of size 1, which is already interesting and non-trivial. We first show that the offline reordering problem can be solved in time $O(nr2^r)$ and $O(nr^{p+1})$, where p denotes the total number of pages. Thus the problem is polynomially solvable when r is at most logarithmic, or when r is arbitrary but p is constant.

We also consider a generalization for the case of cache size 1. We can translate the miss rate into a more familiar theoretical measure—distance. We assume serving (visiting) a page currently in the cache costs nothing, thus $d(i, i) = 0$. In the case of a miss, the new page j is fetched and replaces a page i currently in the cache. Thus in the unit cost case $d(i, j) = 1$ ($i \neq j$). In the weighted version, page i costs w_i to fetch. A naive distance function sets $d(i, j) = w_j$. We can transform this function into a metric one by introducing a new page O and forming a star structure with the rest of the pages, letting $d(O, j) = \frac{1}{2}w_j$,

consequently, $d(i,j) = \frac{1}{2}(w_i + w_j)$ for $i \neq j$. We augment the original request sequence by adding the new page request O to the beginning and end of the sequence. Thus the cost of fetching page j is split into two occasions: when j is first brought into the cache, and when j is replaced by another page. It is then sufficient to design online algorithms for the metric distance function $d(i,j) = \frac{1}{2}(w_i + w_j)$ in order to solve the weighted version, motivating the study for problem instances under general metric distances.

We then consider the online version of the r-reordering problem with $d(i,j) = 1$ or $d(i,j) = \frac{1}{2}(w_i + w_j)$ for $i \neq j$ and give a 2-competitive deterministic greedy online algorithm. We give a 3-competitive deterministic algorithm for a general metric distance function. We provide a lower bound of 1.5 for deterministic algorithms and 1.25 for randomized algorithms. For the addition of l-lookahead, we give a deterministic online algorithm that is $1 + O(r/l)$ competitive for the metric case. We also prove a $1 + \Omega(r/l)$ lower bound on the competitiveness of any algorithm.

We remark here that the unit size cache model is similar to the k-client problem [2]. In the k-client problem, requests form k different queues, the server can serve any of the first request of each queue. Alborzi et al. gave $(2k-1)$-competitive online algorithms, while $\frac{\log k}{2}$ is a lower bound on this ratio. Divakaran and Saks [4] studied an online scheduling problem with job set-ups. Their problem is different from ours in that jobs (not requests) arrive with release time, processing time, and sequence-independent but job-dependent set-up times. The goal is to minimize the maximum flow time. A $O(1)$-competitive online algorithm is presented [4].

For general cache sizes, we show that the competitive ratio is lower bounded by k and H_k for deterministic and randomized online algorithms, respectively, even if the algorithm has reordering and additional lookahead, while the optimal offline cannot reorder the requests. This then implies that we have tight bounds k and H_k for the following two possible comparisons: 1) Both the online and the offline algorithms cannot reorder the requests (the original model). 2) The online algorithm can have lookahead and/or reorder the requests, while the optimal offline algorithm cannot do so.

For the setting where both the online and the offline algorithms can reorder the requests, we provide deterministic and randomized r-reordering algorithms that have competitive ratios $k+2$ and $2H_k+2$, respectively, without additional lookahead (i.e., $r = l$). More generally, the deterministic algorithm is $(\frac{k}{k-s+1} + 2)$-competitive with respect to an optimal offline algorithm having cache size $s \leq k$, again within an additive factor 2 of our lower bound.

Since current caching algorithms serve the request sequence in order, it is also interesting to compare the performance of an online algorithm without

Table 1: Lower Bounds

	Optimal with r-reordering	
	Deterministic	Randomized
Online, no reordering but with l-lookahead	$\max(\frac{k}{k-s+1}, \frac{r}{2(2k-s)}, \frac{r}{k-s+l})$, $\max(k, \frac{r}{k}, \frac{r}{l})$ if $s = k$	$\max(H_k, \frac{r}{k})$
Online with reordering (w./w.o. lookahead)	k	H_k

Table 2: Upper Bounds

	Optimal with r-reordering	
	Deterministic	Randomized
Online, no reordering but with l-lookahead	$\frac{k}{k-s+1} + \frac{2r}{k-s+\min(k,l)} + 2$, $k + 2\lceil \frac{r}{\min(k,l)} \rceil$ if $s = k$	$4H_k + 9.01\frac{r}{k} + 7.01$ (without lookahead)
Online with reordering (w./w.o. lookahead)	$k + 2$	$2H_k + 2$

Fig. 1. Lower Bounds and Upper Bounds for the Reordering Model

reordering with cache size k to an optimal offline algorithm with r-reordering with cache size $s \leq k$. We also allow the online algorithm to have l-lookahead [5] but no reordering. We show a lower bound of $\max(\frac{k}{k-s+1}, \frac{r}{2(2k-s)}, \frac{r}{k-s+l})$ for any deterministic online algorithm. For the randomized situation, we show a lower bound of $\max(H_k, \frac{r}{k})$. We further show that a modified LRU (deterministic) and a modified marking algorithm (randomized) are within a constant factor 5 and 13.011 of our lower bound, respectively. Such results completely classify the possible comparisons between online and optimal offline algorithms with or without reordering/lookahead in the web caching model. Figure 1 summarizes our bounds. Most of the results can be extended to the multi-sized page model to allow different page sizes. We consider two specific models for the case of multi-sized page: *the bit model* and *the fault model*. In the bit model, the cost measure is the number of bits loaded into the cache; in the fault model, it is the number of pages.

The paging problem (without reordering) has been studied extensively. Sleator and Tarjan [10] have demonstrated that the well-known replacement algorithms LRU (Least Recently Used) and FIFO (First-in First-out) are k-competitive and no online deterministic paging algorithm can be better than k-

[5] Here l can take on any value, not necessarily greater than r, which is the parameter for the offline algorithm.

competitive. Fiat et al. [5] have shown that no randomized online algorithm can be better than H_k-competitive where $H_k = \sum_{i=1}^{k} \frac{1}{i}$ is the k^{th} harmonic number. McGeogh and Sleator [9] devised an H_k-competitive algorithm.

Several extensions of the standard paging model with lookahead have been proposed as well. Lookahead allows the online algorithm to see future page requests before making an eviction decision. Young [11] proposed the *resource-bounded* l-lookahead model, where the algorithm can see the current page request and the maximal future sequence of page requests for which it will incur l page faults. For this model, Young presented deterministic and randomized algorithms with competitive ratios $\max(2k/l, 2)$ and $2(\ln(k/l) + 1)$, respectively. Albers [1] proposed the *strong* l-lookahead model, where the algorithm can see the future l distinct pages which are different from the current page request. Albers presents an optimal $(k-l)$-competitive deterministic algorithm, and a $2H_{k-l}$-competitive randomized algorithm, which is within a factor 2 of optimal. Breslauer [3] proposed the *natural* l-lookahead model, where the algorithm has in the lookahead queue at most l distinct page requests that are currently not in the cache. Breslauer presented a tight bound of $\frac{k+l}{l+1}$ for deterministic algorithms for both natural and resource-bounded l-lookahead.

There are other extensions of the paging model in the literature. For example, a multi-threaded paging model was proposed by Feuerstein et. al. [6], where there are multiple servers and request queues.

The rest of the paper is organized as follows: Section 2 presents an offline algorithm, Section 3 is devoted to algorithms and analysis for caches of size 1, Section 4 analyzes deterministic algorithms with and without reordering/lookahead for caches of size k, and their performance is compared to the optimal offline reordering algorithm, Section 5 presents results under the same comparisons for randomized algorithms, finally Section 6 summarizes our results.

2 Offline r-reordering

We first show how to solve the offline r-reordering problem using dynamic programming. We denote by n the total length of the sequence, p the total number of distinct pages, and r the reordering parameter. From now on, we use $\mathcal{OPT}$ to denote the optimal offline strategy.

Theorem 1 *The offline r-reordering problem can be solved by dynamic programming in time $O(nr \sum_{s \leq p} \binom{r}{s})$. An extra factor of $\min(p, r)$ is incurred for general metric distance functions.*

Proof. Without loss of generality, we can assume that when $\mathcal{OPT}$ fetches one page into the cache, all occurrences of the same page requests in the current window is served, advancing the window as far as possible.

Let $w = \{p_1, \ldots, p_2, \ldots, p_m, \ldots\}$ denote the current window, where p_i is the first occurrence of the i^{th} different unserved page in w (so there are a total of m unserved distinct pages, with $m \leq p$). [6]

The key observation is that we can restrict $\mathcal{OPT}$ to serve either p_1 or p_2 next, i.e., $\mathcal{OPT}$ can exchange the first served page p_i $(i > 2)$ with either p_1 or p_2, whichever will be served first later. It's easy to verify that the optimality is maintained.

Next we use dynamic programming, with the following table entries. Each table entry corresponds to the current "state" of the sequence, and records the least number of misses needed to arrive at such "state":

(1) The current position of the window. There are n such choices.
(2) For each page in the current window, up to which occurrence has the page been served already. Since there are at most r pages in the window, and up to p difference pages, there are at most $\sum_{s \leq p} \binom{r}{s}$ such choices.

For each table entry T, we need to backtrack up to r previous table entries that could have resulted in T. For general metric distance functions, we augment each table entry to also include which page was served last. □

The complexity of the offline case for general r and p remains an interesting open problem.

3 Online r-reordering, l-lookahead for Unit Size Cache

The greedy online r-reordering algorithm always serves the first unserved page in the current window, followed by as many occurrences of the same page, while advancing the window as far as possible. For instance, let $w = \{x_1, x_2, \ldots, x_i, \ldots\}$ be the current window. Let x_i be the first occurrence of an unserved page different from x_1. Then add all occurrences of page x_1 from position 1 up to $i + r - 1$ to the sequence, advancing the window so that now it starts with x_i. Now we serve all occurrences of x_i, and so on. We call pages such as x_1 and x_i leading pages. Thus the number of misses is exactly the number of leading pages. We use L_{OPT} to denote the optimal sequence, and L_G the sequence created by the greedy algorithm.

[6] It is possible that some occurrences in the current window have already been served, however, by definition none of the p_i's has been served.

Theorem 2 *The greedy online r-reordering algorithm is 2-competitive, even in the case where pages may have weights. It is not α-competitive for any $\alpha < 2$.*

Proof. We show that if a page occurs t times in L_{OPT}, then it occurs at most $2t$ times in L_G (we count consecutive occurrences as one). Consider a maximal window $I = \{x_i, \ldots, x_j\}$ in the request sequence such that all the occurrences of page x in this window appear consecutively in L_{OPT}, with $x_i = x_j = x$. We show that in L_G there are at most two leading pages of x out of all occurrences in I. Let $x_{i'}$ be first such leading page, then $x_{i'} = x$ and $i' \geq i$. Similarly, let $x_{i''}$ be the second leading page, $x_{i''} = x$ and $i'' > i' + r$. Let $x_q \neq x_{i''}$ be the next leading page after serving $x_{i''}$ in L_G. We know that all occurrences of x up to x_{q+r-1} are included together right after $x_{i''}$ in L_G. We claim that $q+r > j$, i.e. any occurrences of x after position $q + r - 1$ cannot be included in I. If not, then $\mathcal{OPT}$ must have served x_q before serving x_i, but $q > i'' > i' + r \geq i + r$, a contradiction. This proves that the next lead page of x in L_G is beyond I.

For the lower bound, first consider $r = 2$ and a sequence of pages $(123)^{2l}$. The greedy algorithm will serve these pages in the order they occur, for a total cost of $6l$. $\mathcal{OPT}$ starts with 2, then serves two 1s, then two 3s, then two 2s, and so on, for a total cost of $3l + 1$; the ratio $(6l)/(3l + 1)$ approaches 2 as l grows. The result for even $r = 2q$ is obtained by the sequence $(1^q 2^q 3^q)^{2l}$. For odd $r = 2q + 1$, we consider instead the sequence $(1^q 2^{q+1} 3^q 1^{q+1} 2^q 3^{q+1})^l$. □

Theorem 3 *No deterministic online algorithm for the r-reordering problem can be better than 1.5-competitive. No randomized online algorithm can be better than 1.25-competitive.*

Proof. Let $r = 2$. We use pages 0, 1, $0'$ and $1'$. The sequence start with a pattern of either 0100 or 01011, then repeat the chosen pattern to continue the sequence with pages $0', 1'$ instead of $0, 1$, then switch back to $0, 1$, and so on. Both of the patterns can be done in 2 misses (1000 and 00111 respectively). But since a window of size $r = 2$ just sees 01, it cannot distinguish between the 2 sequences. Deterministically, it can be made to require 3 misses per pattern, for a factor of $3/2 = 1.5$.

For a randomized lower bound, assume a randomized algorithm will serve page 0 with probability p and serve page 1 with probability $1 - p$. If $p \geq 1/2$, then the adversary picks the pattern 0100, otherwise the adversary will pick 01011. In either construction, the expected number of misses is no less than $(3/2)1/2 + 1/2 = 1.25$.

The case of general r is obtained by replacing each 0 with 0^{r-1} in the sequence. □

3.1 Extension to l-lookahead

We now consider r-reordering with l-lookahead (with $l > r$). The upper bound obtained will be improved later in the study of the more general case for metric spaces.

Theorem 4 *There is a deterministic online r-reordering algorithm with l-lookahead that achieves competitiveness $1 + 16r/l$, for $(l/8r)$ integer.*

Proof. Partition the sequence into groups of size $l/2$, each of these consisting of t segments of size $4r$ (where $t = l/8r$). Within a group, let D_i be the number of distinct pages in the i^{th} segment. We assume that the middle set of size $2r$ within each segment has at least two distinct pages. Otherwise the sequence can be reduced into two separate sequences, with the single page as the dividing point. Once this condition is met, it is easy to show that a collection of consecutive occurrences of the same page in L_{OPT} must be contained within two consecutive segments (if we reach three consecutive segments for a page p_1, then a different page p_2 in the middle $2r$ of the middle segment could not be included before or after we serve p_1). This gives a lower bound of $\sum D_i/2$ for the optimal solution.

On the other hand, we can choose to split a group of size $l/2$ by selecting a segment out of the t segments as a split segment. We choose the segment j with the smallest D_j, and choose a random dividing point within this j^{th} segment. The lookahead l guarantees that we can decide the splitting segment for the next group of size $l/2$. Between the two chosen dividing points we can solve the problem optimally by dynamic programming. However, we may serve the pages in the split segment one more time compared to $\mathcal{OPT}$, which is upper bounded by D_j. The loss is at most $D_j \leq \sum D_i/t = (\sum D_i/2)(16r/l)$. □

Theorem 5 *No deterministic online r-reordering algorithm with l-lookahead can have competitive ratio better than $1 + 3r/[2(l + r + 1)]$, for $(l + r + 1)/3r$ integer. In the randomized case, the lower bound becomes $1 + 3r/[4(l + r + 1)]$.*

Proof. Let $r = 2$, and consider the two sequences $(01)^{3t+1}00$ and $(01)^{3t+1}01$. Let $l = 6t + 3$. Then with lookahead l the two sequences cannot be distinguished, and yet whether we start with 0 or with 1 makes a difference between costs $2t + 2$ and $2t + 3$. The sequence is then continued with pages $0'$ and $1'$ as in Theorem 3. The ratio is then $1 + 1/(2t + 2) = 1 + 3/(l + 3)$. Similar to Theorem 3, randomized algorithms can at best halve the additive term in the deterministic case, giving a lower bound of $1 + 3/[2(l + 3)]$.

The case of general r is obtained by replacing each 0 with 0^{r-1} in the sequence, and letting $l = r(3t + 1) + r - 1$. □

3.2 Extension to Metric Distance Functions

We first consider metric distance functions with reordering parameter r and no additional lookahead. We present the SP (Shortest Path) algorithm. Let the initial cache content be x_0, and the first window $w = \{x_1, x_2, \ldots, x_r\}$. The algorithm find a minimum travelling salesman path visiting all the pages starting with x_0 and ending with x_r. After serving page x_r, the current window becomes $w = \{x_{r+1}, \ldots, x_{2r}\}$. The algorithm then find a minimum travelling salesman path visiting all the pages from x_r to x_{2r}, with x_r and x_{2r} as the starting and ending point. In general, at the i^{th} stage, the algorithm follows the minimum length path visiting the pages $x_{(i-1)r}, \ldots, x_{ir}$, starting with $x_{(i-1)r}$ and ending with x_{ir}.

Theorem 6 *The SP algorithm for metric distance functions is* 3*-competitive, but not* α*-competitive for any* $\alpha < 3$*.*

Proof. Let p_i be the last page in L_{OPT} of an x_j with $j \leq ir$. The first observation is that p_i's must appear in order in $L_{OPT} = \{\ldots, p_1, \ldots, p_2, \ldots, p_i \ \ldots, \}$, because of the reordering constraint. By definition, the pages between p_{i-1} and p_i must contain x_{ir}, and only contain pages x_j with $(i-1)r < j < (i+1)r$.

We can construct a tour $\mathcal{R}$ visiting pages between p_{i-1} and p_i in the following manner. First we go from p_{i-1} to p_i, visiting only the pages x_j with $(i-1)r < j < ir$; then go from p_i back to p_{i-1}, visiting only x_{ir}; finally, we go from p_{i-1} back to p_i again, visiting the remaining pages x_j with $ir < j < (i+1)r$. Now that $\mathcal{R}$ visits pages in between $x_{(i-1)r}$ and x_{ir} all together. And $\mathcal{R}$ costs at most 3 times that of L_{OPT}. Since our algorithm find the optimal path visiting $x_{(i-1)r}$ up to x_{ir}, it follows that SP produces a sequence which is within a factor of 3 of L_{OPT}. Therefore the algorithm is 3-competitive. In the case of r too big to find the optimal path in polynomial time, we can use a 5/3-approximation algorithm to approximate the shortest travelling salesman path between two specified end points [7]. Thus in total, we get a 5-competitive online algorithm.

For the lower bound, let $r = 2$, and consider in the unweighted case the sequence $(01)^{3t+1}0$, which as we said before has optimal cost $2t + 2$. Each stage of the algorithm considers a segment 010 and visits these three pages in this order, i.e. no reordering is done. Therefore the cost for the algorithm is $6t + 3$, and the ratio $(6t + 3)/(2t + 2)$ approaches 3 as t grows. The case of even $r = 2q$ follows by replacing 0 and 1 with q 0's and 1's. The case of odd $r = 2q + 1$ uses the sequence $(0^q 1^{q+1})^{3t+1}0$. □

We now consider r-reordering with l-lookahead for metric distance functions.

Theorem 7 *There is a deterministic online* r*-reordering algorithm with* l*-lookahead for metric distance functions that achieves competitiveness* $1+4(3r-$

$2)/l$, *for* $l/(6r-4)$ *integer.*

Proof. Partition the sequence into groups of size $l/2$, each of these consisting of t segments of size $3r-2$ (where $t = l/[2(3r-2)]$ is an integer). Each segment has pages in positions $1, \ldots, 3r-2$; the first $2r-1$ pages are called special pages, and the page in position r is called the distinct page.

For each segment i out of the t segments in a group, consider the optimal cycle C_i that visits the $2r-1$ special pages, starting and ending with the distinct page. We use c_i to denote the total cost of C_i. Note that c_i is at most twice the cost incurred on these special pages by L_{OPT}. Furthermore, special pages in distinct segments are visited by L_{OPT} in the order they occur, since they are separated by $r-1$ non-special pages in a segment. Thus $\sum_i c_i/2$ is a lower bound on the total distances incurred by L_{OPT}.

Our algorithm works as follows. We call the segment with the minimum c_i value in each group the special segment. Using lookahead l, select the next special segment s_2 whose optimal cycle C_{s_2} has the minimum cost out of the t segments in the next group. Use dynamic programming, solve optimally how to start from the current distinct page in the current special segment s_1, visiting the last $r-1$ non-special pages remaining in s_1, up to the pages just before s_2, and ending with the distinct page in s_2. Our algorithm then attach the minimum cycle C_{s_2}, find the next special segment s_3 and so on.

The crucial observation is that in L_{OPT}, the distinct page in segment i must appear after all the pages in the segments $i-1$ or lower, and before all the non-special pages in segment i and all the pages in segments $i+1$ or higher, by the reordering constraint. If ignore the cost of the cycles we attached, the rest of the cost incurred by our algorithm is upper bounded by that of $\mathcal{OPT}$. The total cost of the attached cycles c_i's is at most $2/t = 4(3r-2)/l$ times the cost incurred by the optimum. The overall competitive ratio is thus at most $1 + 2/t = 1 + 4(3r-2)/l$. □

4 Deterministic Algorithms for General Cache Sizes

In this section we consider the online reordering problem for a cache of size k. We first establish a lower bound.

Theorem 8 *No deterministic online algorithm for the r-reordering problem with l-lookahead can have a competitive ratio better than k with respect to the optimal solution without reordering and cache size k, or $k/(k-s+1)$ with respect to the optimal solution without reordering and cache size $s \leq k$.*

Greedy LRU —
1 **WHILE** there is a page $p \in Cache$ in the current window w
2 Serve first such p
3 Shift w if possible
4 Evict the least recently used page in cache
5 Fetch the first page in the current window and serve
6 Shift the window w

Fig. 2. Pseudo-code for the Greedy LRU Algorithm

Proof. We consider sequences consisting of blocks of $l \geq r$ identical pages. For such sequences, reordering and lookahead are useless, and the result follows from Sleator and Tarjan's results on the standard paging model [10]. □

It is obvious that the above bound also applies to cases where optimal algorithms are allowed to reorder the requests as well. We now concentrate on online algorithms. Consider the *Greedy LRU* algorithm, which operates as follows: always serve the cached pages in the current window first, and shift the window if possible. If there are no cached pages in the window, serve the first "miss" request in the window by evicting the least recently used page in the cache. See Figure 2 for the pseudo-code.

This practical algorithm has competitive ratio exceeding our lower bound by at most 2. We denote the optimal off-line r-reordering for a cache of size s by $\mathcal{OPT}(s)$, with $s \leq k$.

Theorem 9 *The Greedy LRU algorithm is* $(k+2)$*-competitive with respect to* $\mathcal{OPT}(k)$*, and* $(\frac{k}{k-s+1} + \frac{2r}{k-s+r})$*-competitive with respect to* $\mathcal{OPT}(s)$ *for* $s \leq k$.

Proof. We divide the sequence into *steps* that contain two parts each. The first part of step i begins with the window starting at a position [7] $f(i)$. When the Greedy LRU algorithm is about to fault for the $(k+1)^{th}$ time in the first part, we begin the second part, with the window starting at a position $g(i)$. The next step $i+1$ begins with the window starting at position $f(i+1) = g(i)+r$.

The definition of the first part of step i implies that there are at least $k+1$ distinct pages between positions $f(i)$ and $g(i)$. These pages must be served by $\mathcal{OPT}(s)$ with its window starting at a position p between $f(i)-r+1$ and $g(i) = f(i+1) - r$, so $\mathcal{OPT}(s)$ will fault at least $k-s+1$ times while its window starts at such a position p. Thus the number of faults incurred in the first part of all steps i is at most $k/(k-s+1)$ times the number of faults incurred by $\mathcal{OPT}(s)$.

[7] Here and henceforth the word "position" refers to the position of the pages with respect to the original request sequence.

Suppose Greedy LRU faults q times in the second part of step i. The q pages on which it faults are distinct and also different from the k pages in the cache when the window starts at position $g(i)$. These k pages were served in the first part of step i, so there are at least $k+q$ distinct pages between positions $f(i)$ and $g(i)+r-1 = f(i+1)-1$. These pages must be served by $\mathcal{OPT}(s)$ with its window starting at a position p between $f(i)-r+1$ and $f(i+1)-1$, so the optimal algorithm will fault at least $k-s+q$ times while its window starts at such a position p. Since $q \leq r$, we have $q/(k-s+q) \leq r/(k-s+r)$, and the faults incurred in the second part of all even (odd) steps i are at most $r/(k-s+r)$ times the number of faults incurred by $\mathcal{OPT}(s)$.

The overall competitiveness is thus $k/(k-s+1)+2r/(k-s+r)$, in particular $k+2$ if $s=k$. □

We now study the performance of algorithms without reordering and cache size k with respect to an optimal solution with r-reordering and cache size $s \leq k$. We first show a lower bound.

Theorem 10 *No algorithm without reordering can have a competitive ratio better than* $\frac{r}{2(2k-s)}$ *for* $s \leq k$, *or better than* $\frac{r}{k}$ *if* $s=k$. *No deterministic algorithm without reordering but with lookahead* l *can have a competitive ratio better than* $\max(\frac{k}{k-s+1}, \frac{r}{2(2k-s)}, \frac{r}{k-s+l})$, *or better than* $\max(k, \frac{r}{k}, \frac{r}{l})$ *if* $s=k$.

Proof. Consider an instance with $2k$ distinct pages, which has the form $(123\cdots(2k))^n$. $\mathcal{OPT}(s)$ will fault on at most $2k-s$ pages out of each consecutive block of r pages. Any algorithm without reordering and cache size k will fault on k pages out of each consecutive block of $2k$ pages. The first result follows since $(\frac{k}{2k})/(\frac{2k-s}{r}) = \frac{r}{2(2k-s)}$.

When $s=k$, consider an instance with $k+1$ distinct pages of the form $(123\cdots(k+1))^n$. $\mathcal{OPT}(s)$ will fault on at most 1 page out of each consecutive block of r pages. Any algorithm without reordering will fault on at least 1 page out of each consecutive block of k pages. This shows a lower bound on the competitive ratio of $\frac{r}{k}$.

Consider now the case with lookahead l, and consider instances with $k+l$ distinct pages. An adversary can always choose the page in position l to be different from the preceding $l-1$ pages and from the k pages in the cache. Thus the algorithm without reordering will fault on every page. $\mathcal{OPT}(s)$ will fault on at most $k+l-s$ pages out of each consecutive block of r pages. This shows a lower bound on the competitive ratio of $\frac{r}{k-s+l}$. □

We now present algorithms without reordering that match our lower bounds on competitiveness up to constant factors, for all values of r, l, k, and s.

Theorem 11 *LRU (without reordering/lookahead) is* $\frac{k+r}{k-s+1}$*-competitive with*

respect to $\mathcal{OPT}(s)$, in particular $(k+r)$-competitive if $s = k$.

Proof. We divide the sequence into steps that contain two parts each, as in Theorem 9. Again, there are at least $k+1$ distinct pages between positions $f(i)$ and $g(i)$ that must be served by $\mathcal{OPT}(s)$ with its window starting at a position p between $f(i) - r + 1$ and $g(i) = f(i+1) - r$, and so $\mathcal{OPT}(s)$ will fault at least $k - s + 1$ times while its window starts at such a position p.

The total number of faults incurred by LRU during step i is at most $k + r$, giving competitiveness $(k+r)/(k-s+1)$, in particular $k + r$ if $s = k$. □

Note that the above algorithm gives a competitive ratio of $2k$ or better, if $r \leq k$. For $r > k$ we devise online algorithms with lookahead but without reordering. We consider the following *modified LRU* algorithm without reordering. Our algorithm has a lookahead no more than k, i.e., the algorithm uses lookahead $l' = \min(k, l)$. The modified LRU operates like LRU in the first part of each step i beginning with the window starting at a position $f(i)$, and the second part of step i begins with the window starting at a position $g(i) = f(i+1) - r$. For the second part of step i, the r pages at positions starting with $g(i)$ are divided into $\lceil r/l' \rceil$ blocks of size at most l'. We run LRU on the l' pages in each block, except that we never evict from the cache a page in the current block. (This is possible because $l' \leq k$.)

Theorem 12 *The* modified LRU *algorithm without reordering and with lookahead $l' = \min(k, l)$ is $(\frac{k}{k-s+1} + \frac{2r}{k-s+\min(k,l)} + 2)$-competitive with respect to $\mathcal{OPT}(s)$, and $(k + 2\lceil \frac{r}{\min(k,l)} \rceil)$-competitive if $s = k$.*

Proof. The faults incurred in the first part of all steps i are at most $k/(k-s+1)$ times the number of faults incurred by the optimal algorithm. During the second part, let $q \leq l'$ be the maximum number of faults incurred by the modified LRU algorithm in a single block out of all $\lceil r/l' \rceil$ blocks. Then we have q distinct pages different from the k pages in the cache when the window starts at the beginning of a block, thus at least $k + q$ distinct pages between positions $f(i)$ and $g(i) + r - 1 = f(i+1) - 1$. $\mathcal{OPT}(s)$ faults at least $k - s + q$ times while its window starts between $f(i) - r + 1$ and $f(i+1) - 1$. Adding even and odd steps i separately, since $q/(k-s+q) \leq l'/(k-s+l')$, we obtain a ratio for the second part of the steps i given by $2l'\lceil r/l' \rceil/(k - s + l') \leq 2(r + l' - 1)/(k - s + l') < 2r/(k - s + l') + 2$. □

Note that in particular we obtain an $O(k)$-competitive algorithm as before with respect to the optimal r-reordering, provided $r = O(k \cdot \min(k, l))$.

4.1 Multi-sized pages

It is possible to adapt our proofs for the situation where not all the pages have the same size. As an example, we adapt the proof of Theorem 9. We denote the smallest possible page size by ϵ and the size of the cache by K, and write $k = K/\epsilon$.

Theorem 13 *In both the bit and the fault model, the Greedy LRU r-reordering algorithm for cache size K is $(k+3)$-competitive with respect to the optimal r-reordering algorithm for cache size K, and $[(k+1)/(k-s+1)+2]$-competitive with respect to the optimal r-reordering algorithm for cache size $S \leq K$, where $s = S/\epsilon$.*

Proof. We first consider the bit model.

We divide the sequence into *steps* that contain two parts. The first part of step i begins with the window starting at a position $f(i)$. As soon as the Greedy LRU algorithm has served distinct requests totaling more than K bits, the next distinct request after this, with the window starting at a position $g(i)$, is the start of the second part. The next step $i+1$ begins with the window starting at position $f(i+1) = g(i) + r$.

The definition of the first part of step i implies that there are $K + \delta$ bits requested (counting bits from identical requests only once) between positions $f(i)$ and $g(i)$, for some $\delta > 0$. These pages must be served by the optimal algorithm with its window starting at a position p between $f(i) - r + 1$ and $g(i) = f(i+1) - r$, so the optimal algorithm will pay at least $K - S + \max(\delta, \epsilon)$ while its window starts at such a position p. (Any fault costs at least ϵ.) Thus the faults incurred in the first part of all steps i cost at most $(K + \delta)/(K - S + \max(\delta, \epsilon)) \leq (k+1)/(k-s+1)$ times the optimal cost.

Suppose Greedy LRU faults q times in the second part of step i, and the total size of the pages on which it faults is Q. The q pages on which it faults are distinct and also different from the pages in the cache when the window starts at position $g(i)$. These pages were served in the first part of step i, so there are at least $K + Q$ bits from distinct pages requested between positions $f(i)$ and $g(i) + r - 1 = f(i+1) - 1$. These pages must be served by the optimal algorithm with its window starting at a position p between $f(i) - r + 1$ and $f(i+1) - 1$, so the optimal algorithm will pay at least $K - S + Q$ while its window starts at such a position p. Therefore, the cost incurred in the second part of all even (odd) steps i is at most the cost incurred by the optimal algorithm.

The overall competitiveness is thus $(k+1)/(k-s+1)+2$, in particular $k+3$ if $s = k$.

Window Randomized Marking —
1 **WHILE** there is a page $p \in Cache$ in the current window w
2 Serve first such p and mark the page
3 Shift w if possible
4 **IF** all pages in the cache are marked
5 Unmark all the pages
6 Evict randomly an unmarked page in the cache
7 Fetch and mark the first page in the current window and serve
8 Shift the current window w

Fig. 3. Pseudo-code for the window randomized marking algorithm

For the fault model, the proof is similar: the cost function changes, but the ratios remain the same. □

5 Randomized Algorithms for General Cache Sizes

We now consider randomized algorithms for the online reordering problem with cache size $k \geq 2$. The general lower bound is again straightforward.

Theorem 14 *No randomized online algorithm for the r-reordering problem with l-lookahead can have a competitive ratio better than H_k with respect to the optimal solution without reordering and cache size k.*

Proof. We consider sequences consisting of blocks of $l \geq r$ identical pages. For such sequences, reordering and lookahead are useless, and the result follows from a result of Fiat et. al. [5] without reordering or lookahead. □

The *window randomized marking* algorithm maintains a set of pages which can be *marked* in the cache. Initially no pages in the cache are marked. The algorithm operates in *marking phases* where it behaves iteratively as follows.

The algorithm always serves and marks cached pages currently in the window first and always advances the window as far as possible. Otherwise, we serve the first page in the current window, which is not in the cache. We pick an unmarked page in the cache uniformly at random and replace it with the newly fetched page. We mark the new page and advance the window. At the end of the marking phase, all pages in the cache are marked and the first page in the window is unmarked, plus all the cached pages in the window are served already. We then unmark all pages and proceed to the next marking phase. See Figure 3 for the pseudo-code.

We first show that this natural modification of the original marking algorithm is at most a factor of 4 away from optimal.

Theorem 15 *The window randomized marking algorithm is* $(4H_k+2)$*-competitive with respect to* $\mathcal{OPT}(k)$*.*

Proof. Let $\mathcal{M}$ denote the window randomized marking algorithm. We divide the sequence into *steps* that contain two parts each. Step i begins at a position $f(i)$ that is the starting position of the window at the beginning of a marking phase. The second part of step i begins at a position $g(i)$ that is the starting position of the window at the beginning of the next marking phase. Furthermore, if step $i+1$ has beginning position $f(i+1)$, then we require $f(i) \leq f(i+1) - r \leq g(i)$. This can be achieved by fixing the starting positions $f(i)$ in *decreasing* order of i. We add the marking phases before $f(i+1)$ to the second part of step i. The first marking phase we encounter (traversing the request sequence in reverse order) with starting position before $f(i+1)-r$ will be the first part of step i. Note then step i may not have a second part, if a single marking phase begins with the window starting at $f(i) \leq f(i+1)-r$ (and ends at $f(i+1)-1$).

Let $M(i)$ denote the cache content of $\mathcal{M}$ when its window starts at position $f(i)$, and let $OPT(i)$ denote that of $\mathcal{OPT}$ when its window starts at position $f(i)-r$. Let $d(i) = |OPT(i) - M(i)| = |M(i) - OPT(i)|$. Let u denote the number of distinct pages served by $\mathcal{M}$ in the first part of step i that were not in $M(i)$. $\mathcal{OPT}$ incurs at least $u - d(i)$ faults with its window starting at positions between $f(i) - r + 1$ and $g(i) - 1$, since there are at least $u - d(i)$ distinct pages served by $\mathcal{M}$ in the first part of step i that were not in $OPT(i)$. From a different angle, $\mathcal{OPT}$ also incurs at least $d(i+1)$ faults with its window starting at positions between $f(i) - r + 1$ and $f(i+1) - 1$. Consider the k pages in $M(i+1)$, they must occur somewhere between $f(i)$ and $f(i+1)-1$. In particular, take the $d(i+1)$ pages not in $OPT(i+1)$, they are either served without a fault but evicted in a subsequent fault by $\mathcal{OPT}$ with its window starting between $f(i) - r + 1$ and $f(i+1) - r - 1$, or served with a fault by $\mathcal{OPT}$ with its window starting between $f(i+1) - r$ and $f(i+1) - 1$.

Thus $\mathcal{OPT}$ incurs at least $\max(u - d(i), d(i+1)) \geq (u - d(i) + d(i+1))/2$ faults with its window starting between $f(i)-r+1$ and $f(i+1)-1$. Summing over all steps, the $d(i)$ terms telescope, so we can charge $u/4$ to the number of faults for $\mathcal{OPT}$ in step i, since every page on which $\mathcal{OPT}$ faults is counted at most twice in this sum.

The above argument establishes a lower bound on $\mathcal{OPT}$, we now obtain an upper bound on $\mathcal{M}$. During the first part of step i, $\mathcal{M}$ faults on the u pages not in $M(i)$. Consider the remaining $k - u$ pages that $\mathcal{M}$ serves, which are in $M(i)$. When serving the first such pages, $\mathcal{M}$ faults only if this page was

replaced in the cache by one of the u pages, and thus with probability at most u/k. In general, when serving the $(j+1)^{th}$ such page, this page is among the $k-j$ pages in $M(i)$ that have not been marked, and at most u of them have been replaced, so $\mathcal{M}$ faults with probability at most $u/(k-j)$. Summing over all $k-u$ pages, the expected number of misses is bounded by

$$\frac{u}{k} + \frac{u}{k-1} + \frac{u}{k-2} + \cdots + \frac{u}{u+1} = u(H_k - H_u) \leq u(H_k - 1).$$

Thus $\mathcal{M}$ faults in expectation at most $u + u(H_k - 1) = uH_k$ times during the first part of step i, hence at most $4H_k$ times the number of faults for $\mathcal{OPT}$.

The analysis of the second part of step i is the same as in Theorem 9. Assume $\mathcal{M}$ faults on q distinct pages in the second part of step i, so there are at least $k+q$ distinct pages between positions $f(i)$ and $f(i+1)-1$. These pages are served by $\mathcal{OPT}$ with its window starting at a position between $f(i)-r+1$ and $f(i+1)-1$, causing at least q faults for $\mathcal{OPT}$. Adding even and odd steps separately gives ratio at most 2 between the number of faults incurred by $\mathcal{M}$ in the second part of the steps and the total number of faults incurred by $\mathcal{OPT}$. Combining the two parts of each step gives ratio $4H_k + 2$. □

We now show that with a twist in the algorithm, we can improve the ratio to $2H_k + 2$. The modified window randomized marking algorithm runs in steps, again divided into two parts each. The first part of step i again consists of a single marking phase. The second part of step i consists of marking phases as before, except that it ends when the second part of step i has caused the window to advance exactly r positions. At that point in time, all marks are erased and step $i+1$ begins.

Theorem 16 *The modified window randomized marking algorithm is* $(2H_k + 2)$*-competitive with respect to* $\mathcal{OPT}(k)$*.*

Proof. The effect of the modification is to enforce $g(i) = f(i+1) - r$. Again, $\mathcal{OPT}$ incurs at least $u - d(i)$ faults with its window starting at positions between $f(i) - r + 1$ and $g(i) - 1 = f(i+1) - r - 1$, on pages in neither $OPT(i)$ nor $M(i)$. Also $\mathcal{OPT}$ incurs at least $d(i+1)$ faults with its window starting at positions between $f(i) - r + 1$ and $f(i+1) - 1$. We are going to distinguish between these misses. Let $d(i+1) = d_1(i+1) + d_2(i+1)$. Let $d_1(i+1)$ refer to the number of pages that are served without a fault but evicted in one of the subsequent faults, with its window starting between $f(i) - r + 1$ and $f(i+1) - r - 1$. Let $d_2(i+1)$ refer to the number of pages served with a fault by $\mathcal{OPT}$, with its window starting between $f(i+1) - r$ and $f(i+1) - 1$.

Note that the $d_2(i)$ faults are different from the $u - d(i)$ faults, since the $u - d(i)$ pages are not in $M(i)$ whereas the $d_2(i)$ pages are. Therefore the total number of faults, with $\mathcal{OPT}$'s window starting at positions between $f(i) - r$ and $f(i+1) - r - 1$, is at least $\max[u - d(i) + d_2(i), d_1(i+1)] \geq$

$(u-d(i)+d_2(i)+d_1(i+1))/2$. Summing over all steps, again the $d(i), d_1(i), d_2(i)$ terms telescope, but now the starting window positions occurring in the sum do not overlap. We can then assume that the number of faults for $\mathcal{OPT}$ in step i is $u/2$ instead of $u/4$.

The analysis for the modified algorithm is identical to that of Theorem 15. The expected number of faults for the modified algorithm in the first part of step i at most uH_k, giving ratio $2H_k$. For the second part we obtain a ratio 2. Combining the two parts gives the desired bound $2H_k + 2$. □

We now study the performance of randomized algorithms without reordering and cache size k with respect to an optimal solution with r-reordering and cache size k. The following result can be obtained from Theorems 10 and 14.

Theorem 17 *No randomized algorithm without reordering and with lookahead l can have a competitive ratio better than* $\max(H_k, \frac{r}{k})$.

We now present a randomized algorithm without reordering that is a constant factor away from the lower bound, for all values of r, l, k, using only lookahead $l' = \min(k, l)$. The *lookahead-only* algorithm again consists of steps divided into two parts. The first part of step i is a randomized marking phase without reordering. The second part of step i operates on blocks of size l' as in the second part of the *modified LRU* algorithm without reordering in Section 4.

Theorem 18 *The* lookahead-only *algorithm with lookahead $l' = \min(k, l)$ is $(4H_k + 2\lceil r/\min(k, l)\rceil)$-competitive with respect to $\mathcal{OPT}(k)$.*

Proof. We obtain ratio $4H_k$ for the first part of step i as in Theorem 15, and ratio $2\lceil r/l'\rceil$ for the second part of step i as in Theorem 12. □

Even more interesting is the fact that there is an online algorithm without either lookahead or reordering that achieves a slightly worse bound, but is still only a constant factor away from our lower bound.

We remark first that the original marking algorithm [9] as is does not have such property. It is only $(H_k + H_k \cdot \frac{r}{k})$-competitive (up to constant factors), and not $(H_k + \frac{r}{k})$-competitive. Consider a sequence $(123\cdots(k+1))^t$, $\mathcal{OPT}$ incurs only one fault every r pages, while the original randomized marking algorithm incurs about H_k faults every k pages.

Consider the following *no-lookahead randomized marking* algorithm. It also has two parts for each step. The first part of step i is again a randomized marking phase without reordering. For the second part, we run randomized marking algorithm on blocks of size $l = \lceil \alpha k \rceil$ for $\lceil r/l \rceil$ blocks, where $0 < \alpha < 1$ is a parameter to be determined later. Once the block limit $\lceil \alpha k \rceil$ is reached, all marks are cleared (even if some pages are still unmarked) and the next

marking phase begins.

Theorem 19 *The* no-lookahead randomized marking *algorithm with cache size k has competitive ratio $4H_k + 9.011 \cdot r/k + 7.011$ with respect to $\mathcal{OPT}(k)$.*

Proof. The ratio $4H_k$ for the first part of step i is due to Theorem 15. The analysis for second part of step i is similar to that of Theorem 12. Let q be the maximum number of pages in a single block that are not in the online algorithm's cache when the window starts at the beginning of the block. Then there are again at least $k+q$ distinct pages between positions $f(i)$ and $f(i+1)-1$, giving q faults for $\mathcal{OPT}$ with its window starting between $f(i)-r+1$ and $f(i+1)-1$. We may thus attribute $q/2$ faults from $\mathcal{OPT}$ to step i. Our algorithm, on the other hand, faults in each block on the q' pages (with $q' \leq q$), and on the remaining $l - q'$ pages that are in the cache at the beginning of the block, which in expectation is at most

$$\frac{q'}{k} + \frac{q'}{k-1} + \frac{q'}{k-2} + \cdots + \frac{q'}{k-l+q'+1} = q'(H_k - H_{k-l+q'}).$$

If $l = \lceil \alpha k \rceil$, this quantity is at most $q(H_k - H_{k-l}) \leq q[1 + \ln k - \ln(k - \alpha \cdot k)] = q[1 - \ln(1-\alpha)]$. There are thus a total of $q[2 - \ln(1-\alpha)]$ faults in each of the $\lceil r/l \rceil \leq r/(\alpha k) + 1$ blocks, giving a ratio of $2[2 - \ln(1-\alpha)](r/(\alpha k) + 1)$ for the second part of step i. Setting $\alpha = 0.778$ gives the desired bound. □

6 Conclusions

In this paper we have studied the model of web caching with request reordering. We presented good deterministic and randomized online algorithms, as well as established matching lower bounds for the competitive ratios (up to a constant factor). We also classified the performance comparisons between online algorithms and optimal offline algorithms with or without reordering/lookahead under the proposed model. One interesting open problem is the complexity of the offline reordering web caching problem with arbitrary r (the reordering parameter) and p (total number of distinct pages).

References

[1] S. Albers. The influence of lookahead in competitive paging algorithms. *Algorithmica*, 18:283–305, 1997.

[2] H. Alborzi, E. Torng, P. Uthaisombut, and S. Wagner. The k-client problem. *Journal of Algorithms*, 41:115–173, 2001.

[3] D. Breslauer. On competitive on-line paging with lookahead. *Theoretical Computer Science*, 209:2:365–375, 1998.

[4] S. Divakaran and M. Saks. An Online Scheduling Problem with Job Set-ups. DIMACS Technical Report, 2000.

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

[6] E. Feuerstein and A. Strejilevich de Loma. On-line Multi-threaded Paging. *Algorithmica*, 32:36–60, 2002.

[7] J.A. Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10: 178–193, 1978.

[8] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal Assoc. Comput. Mach.* 25 (1977), 322–336.

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

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

[11] N. Young. Competitive Paging and Dual-Guided On-Line Weighted Caching and Matching Algorithms. Ph.D. thesis, Princeton University, 1991. Available as Computer Science Department Technical Report CS-TR-348-91.