Algorithmica (2018) 80:3563–3596
https://doi.org/10.1007/s00453-018-0406-9

Quantifying Competitiveness in Paging with Locality of Reference

Susanne Albers[1] · Dario Frascaria[1]

Received: 20 October 2016 / Accepted: 12 January 2018 / Published online: 23 January 2018

Abstract The classical paging problem is to maintain a two-level memory system so that a sequence of requests to memory pages can be served with a small number of faults. Standard competitive analysis gives overly pessimistic results as it ignores the fact that real-world input sequences exhibit locality of reference. Initiated by a paper of Borodin et al. (J Comput Syst Sci 50:244–258, 1995) there has been considerable research interest in paging with locality of reference. In this paper we study the paging problem using an intuitive and simple locality model that records inter-request distances in the input. A characteristic vector $\mathcal{C}$ defines a class of request sequences with certain properties on these distances. The concept was introduced by Panagiotou and Souza (In: Proceedings of 38th annual ACM symposium on theory of computing (STOC), 2006). As a main contribution we develop new and improved bounds on the performance of important paging algorithms. A strength and novelty of the results is that they express algorithm performance in terms of locality parameters. In a first step we develop a new lower bound on the number of page faults incurred by an optimal offline algorithm OPT. The bound is tight up to a small additive constant. Technically, the result relies on a new approach of relating the number of page faults to the number of memory hits and amortizing suitably. Based on these expressions for OPT's cost,

A preliminary version of this paper has appeared in *Proceedings of 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, 2015.

Susanne Albers: Work supported by the European Research Council, Grant Agreement No. 691672, and by the German Research Foundation, Project Al 464/9-1.

✉ Susanne Albers
albers@in.tum.de

Dario Frascaria
frascari@informatik.tu-muenchen.de

1 Department of Computer Science, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany

we obtain nearly tight upper and lower bounds on LRU's competitiveness, given any characteristic vector $\mathcal{C}$. Furthermore, we compare LRU to FIFO and FWF. For the first time we show bounds that quantify the difference between LRU's performance and that of the other two strategies. The results imply that LRU is strictly superior on inputs with a high degree of locality of reference. There exist general input families for which LRU achieves constant competitive ratios whereas the guarantees of FIFO and FWF tend to k, the size of the fast memory. Finally, we report on an experimental study that demonstrates that our theoretical bounds are very close to the experimentally observed ones. Hence our contributions bring competitive paging again closer to practice.

Keywords Online algorithm · Optimal offline algorithm · Analysis of algorithms · Experimental study

1 Introduction

Paging is a fundamental resource management problem in computer science. In algorithms research it has been studied extensively ever since Sleator and Tarjan published their seminal paper [21] on the competitive analysis of algorithms. In the *paging problem* we are given a two-level memory system consisting of a small fast memory and a large slow memory. At any time up to k pages, for some $k \in \mathbb{N}$, can reside in fast memory. A paging algorithm ALG is presented with a request sequence $\sigma = \sigma(1), \ldots, \sigma(m)$, where each request $\sigma(t)$ specifies a memory page. Consider any $\sigma(t)$, $1 \leq t \leq m$. If the referenced page is in fast memory, $\sigma(t)$ is a *memory hit*. Otherwise $\sigma(t)$ is a *page fault* and the missing page must be loaded from slow memory into fast memory. If the fast memory is full, ALG must evict a page from fast memory; in the *online* setting this decision must be made without knowledge of any future requests. The goal is to serve σ so as to minimize the total number of faults.

For an online algorithm ALG and a request sequence σ, let ALG(σ) denote the number of page faults incurred. Let OPT(σ) be the number of faults generated by an optimal offline algorithm OPT. Strategy ALG is *c-competitive* if, for every σ, ALG(σ) is at most c times OPT(σ). The optimal competitive ratio achieved by deterministic online algorithms is equal to k [21]. Classical algorithms such as LRU (Least-Recently-Used), FIFO (First-In First-out) and FWF (Flush-When-Full) are all k-competitive.

It was soon observed that the competitiveness of k is overly pessimistic. In practice algorithms such as LRU and FIFO attain constant performance ratios in the range $[1.5, 4]$, see also [22]. Furthermore, LRU outperforms FIFO, which does not show in competitive analysis. The deficiency of the competitive measure is that it considers arbitrary request sequences whereas input sequences generated by real programs have a special structure. They exhibit *locality of reference*, i.e. whenever a page is requested it is likely to be referenced again in the near future. In a cornerstone paper Borodin et al. [10] initiated the investigation of paging with locality of reference. Over the years various frameworks modeling locality of reference have been proposed. Moreover, new and alternative performance measures have been introduced. In this paper we revisit paging with locality of reference, considering again the competitive performance measure. Compared to previous studies we present for the first time strong guarantees that quantify competitiveness in terms of locality parameters of the input. We analyze individual algorithms and relate pairs of strategies.

Input model We use a model for locality of reference introduced by Panagiotou and Souza [20]. The framework is simple, yet captures the essentials of locality of reference: Whenever a page is requested, it is likely to be re-accessed soon. Hence locality can appropriately be modeled by inter-request distances. Specifically, feasible input is defined by a *characteristic vector* $\mathcal{C} = (c_0, \ldots, c_{p-1})$, where p denotes the total number of distinct pages ever referenced. Again let σ be a request sequence and $\sigma(t)$ be the request at time t. We refer to $\sigma(t)$ as a *distance-l request*, where $0 \leq l \leq p-1$, if the following two properties hold. (1) The page x referenced by $\sigma(t)$ has been requested before in σ and its most recent request was $\sigma(t')$. (2) The number of distinct pages requested between $\sigma(t')$ and $\sigma(t)$ is equal to l, i.e. $|\{\sigma(t'+1), \ldots, \sigma(t-1)\}| = l$. In a request sequence σ characterized by $\mathcal{C} = (c_0, \ldots, c_{p-1})$, there are exactly c_l distance-l requests, for $l = 0, \ldots, p-1$. The total number of requests in σ is $p + \sum_{l=0}^{p-1} c_l$.

The concept of characteristic vectors allows one to easily quantity the number of pages faults incurred by LRU. This was already observed by Panagiotou and Souza [20]. On a fault LRU evicts the page whose last reference is longest ago. Thus at any time LRU's fast memory stores the (up to) k pages that were referenced most recently. Consequently, LRU never incurs a page fault on a distance-l request with $0 \leq l \leq k-1$ as the referenced page is still in fast memory. Moreover, LRU has a fault on every distance-l request with $k \leq l \leq p-1$ since the accessed page has been evicted from fast memory since its last reference. It follows that for any σ specified by $\mathcal{C} = (c_0, \ldots, c_{p-1})$, there holds $\text{LRU}(\sigma) = p + \sum_{l=k}^{p-1} c_l$.

Given any characteristic vector $\mathcal{C}$, the competitive ratio of an algorithm ALG is defined as $R_{\text{ALG}}(\mathcal{C}) = \max_\sigma \text{ALG}(\sigma)/\text{OPT}(\sigma)$, where the maximum ranges over all request sequences characterized by $\mathcal{C}$. As this set of sequences is finite, the minimum is well-defined.

Previous work There exists a considerable body of literature on paging with locality of reference. Due to the wealth of results we can only present a selection. A good survey article is [12]. In their initial paper [10] Borodin et al. introduced *access graphs G*, representing the execution of programs, to model locality of reference. The vertices of G correspond to the memory pages. Page x may be requested after y if they are adjacent in G. Borodin et al. showed that, for any G, the competitiveness $R_{\text{LRU}}(G)$ of LRU depends on the number of articulation nodes whose removal separates G. They also developed an algorithm that achieves the best possible competitive ratio attainable for any given G, up to a constant factor [10, 19]. Chrobak and Noga [13] proved that LRU is always at least as good as FIFO, i.e. for any G, $R_{\text{LRU}}(G) \leq R_{\text{FIFO}}(G)$.

Articles [4, 17, 18] make probabilistic assumptions about the input. A diffuse adversary [18] generates a request sequence according to a probability distribution that belongs to a known family of distributions. In Markov paging [17] the input is generated by a Markov chain. Algorithms are evaluated in terms of the page fault rate. In [1] concave functions, which model the working set sizes of programs, are used to restrict the allowed input. Again page fault rates are evaluated.

Especially in recent years various alternative performance measures, in addition to the well-known page fault rate, have been proposed. These include (a) the max/max ratio [6], (b) bijective and average analysis [2,3], (c) the relative worst-order ratio [7,8], (e) relative interval analysis [9,14] and (f) parametrized analysis [11]. In a bijective analysis two algorithms ALG_1 and ALG_2 are compared on permutations of the same requests. Let $\mathcal{I}_n$ denote the request sequences of length n. ALG_1 is *no worse* than ALG_2, in signs $\text{ALG}_1 \preceq \text{ALG}_2$, if for all $n \geq n_0$ there is a bijection $b : \mathcal{I}_n \to \mathcal{I}_n$ such that $\text{ALG}_1(\sigma) \leq \text{ALG}_2(b(\sigma))$ for all $\sigma \in \mathcal{I}_n$. In this setting LRU is no worse than any other online algorithm ALG assuming that locality is modeled by a concave function [3]. However, LRU $\preceq$ FIFO and FIFO $\preceq$ LRU, see [2], so that they are equally good under bijective analysis.

The concept of characteristic vectors was defined by Panagiotou and Souza [20]. As a main result they lower bound the number of page faults incurred by OPT on a request sequence σ characterized by $\mathcal{C} = (c_0, \ldots, c_{p-1})$, i.e.

$$\text{OPT}(\sigma) \geq \frac{1}{1 + \frac{k-1}{k} - \frac{k-1}{p-1}} \sum_{l=k}^{p-1} \frac{l-k+1}{l} c_l. \tag{1}$$

They mention that the bound is tight for characteristic vectors in which $p = k + 1$. Panagiotou and Souza [20] also define an (α, β)-adversary that chooses vectors $\mathcal{C}$ satisfying $\sum_{l=k}^{\alpha k-1} c_l \leq \beta \sum_{l=\alpha k}^{p-1} c_l$. Against this adversary LRU achieves a competitive ratio of $2(1+\beta)\alpha/(\alpha-1)$. Panagiotou and Souza [20] also study a setting where an adversary may construct an arbitrary request sequence but the size of the fast memory is chosen uniformly at random from a given range.

Our contribution We investigate paging using classical competitive analysis and adopt the concept of characteristic vectors $\mathcal{C} = (c_0, \ldots, c_{p-1})$ to model locality of reference. It is intuitive to represent input characteristics by a fingerprint of the inter-request distances: If a request sequence exhibits a high degree of locality, then a large majority of the requests are distance-l requests, for small l, so that the corresponding vector entries c_l take large values. Given a real-world trace, the underlying $\mathcal{C}$ can be extracted easily by a single scan over the data.

We present new and significantly improved bounds on the performance of the most important paging strategies. A particular strength and novelty of the results is that they quantify algorithm performance in terms of locality parameters. Furthermore, the bounds very accurately predict the corresponding performance observed in practice. This finding results from an experimental study we conducted with traces from a benchmark library. These tests confirm the value of our theoretical bounds.

In Sect. 2, given any characteristic vector $\mathcal{C}$, we develop a new lower bound on the number of page faults incurred by OPT to serve any request sequence σ characterized by $\mathcal{C}$. Technically, the analysis relies on a new approach that relates the number of page faults to the number of memory hits and amortizes the values appropriately. Specifically, we show that

$$\text{OPT}(\sigma) \geq \max\left\{p, k + \sum_{l=k}^{\lambda-1} c_l \frac{l-k+1}{k-1} + c_\lambda^* \frac{\lambda-k+1}{k-1}\right\}. \tag{2}$$

Here λ and c_λ^* are solutions of an equation that matches faults and hits, assuming that page faults preferably occur on long-distance requests. We prove that our lower bound is tight, up to an additive term of at most $2(\lambda - k + 1)$, which in turn is upper bounded by $2(p - k)$. More precisely, we construct an input sequence that OPT can serve with the stated number of faults. The construction and cost analysis of the sequence are involved. A strength of our lower bound is that it is tight for every characteristic vector. The bound (1) by Panagiotou and Souza [20] is only tight for a restricted class of characteristic vectors, as indicated above. Additionally, we show that our lower bound (2) is always greater than that given in (1). In the experiments (2) considerably outperforms (1).

In Sect. 3 we evaluate the competitiveness of LRU. Given the analysis of $\text{OPT}(\sigma)$, we derive nearly tight upper and lower bounds on $R_{\text{LRU}}(\mathcal{C})$, for any $\mathcal{C}$. The resulting ratios range between 1 and k, depending on $\mathcal{C}$. The experiments show that these refined ratios are very close to LRU's experimentally observed competitiveness. For all the traces and all values of k, the theoretical bounds are usually at most 2.5 times the experimentally observed performance. In most cases the gap is much smaller. This is the first time that theoretical performance guarantees for paging match the experimental ones up to a constant factor, independently of k. We remark that our theoretical guarantees cannot exactly match the experimental ones because $R_{\text{LRU}}(\mathcal{C}) = \max_\sigma \text{LRU}(\sigma)/\text{OPT}(\sigma)$ is still a worst-case ratio. A real-world trace, in general, is not a worst-case input for the underlying $\mathcal{C}$.

In Sect. 4 we show that LRU is superior to other popular paging strategies. We focus on a comparison with FIFO and FWF, which have received considerable attention in the memory management literature. We first prove that LRU is always at least as good as the other two strategies, i.e. $R_{\text{LRU}}(\mathcal{C}) \leq R_{\text{FIFO}}(\mathcal{C})$ and $R_{\text{LRU}}(\mathcal{C}) \leq R_{\text{FWF}}(\mathcal{C})$ for any $\mathcal{C}$. This is not surprising; similar relations have been shown in other frameworks as well. In this paper we go one step further and quantify the performance difference between LRU and FIFO, respectively FWF. We make use of the fact that LRU's competitiveness can be expressed as $R_{\text{LRU}}(\mathcal{C}) = \text{LRU}(\mathcal{C})/\text{OPT}(\mathcal{C})$, where $\text{OPT}(\mathcal{C})$ denotes the minimum number of page faults required to serve any request sequence defined by $\mathcal{C}$ and $\text{LRU}(\mathcal{C})$ is LRU's fixed cost for every input specified by $\mathcal{C}$. We prove that

$$R_{\text{FIFO}}(\mathcal{C}) \geq \frac{\text{LRU}(\mathcal{C}) + c(k-1)}{\text{OPT}(\mathcal{C}) + c(1 - 1/k) + 1},$$

where c depends on the vector entries c_l, $1 \leq l \leq p-1$. If the number of distance-l requests with $l \geq k$ is not too small, then FIFO's competitiveness tends to k as the locality in the input (captured by entries c_l, for small l) increases. In particular, there exist input classes $\mathcal{C}$ for which LRU's competitiveness is constant while that of FIFO is close to k. The same results hold for FWF, except that slightly "weaker" assumptions on the input are made. Finally, in Sect. 5 we report on the results of our experimental study.

Algorithms We describe the classical paging algorithms analyzed in the paper. Suppose that there is a page fault and the fast memory is full. Among the pages residing in fast memory, LRU evicts the one whose last reference is longest ago. FIFO drops the page that was loaded earliest. FWF deletes all pages from fast memory. An optimal offline algorihm OPT was given by Belady [5]. On a fault, when the fast memory is full, it evicts the page whose next reference is farthest in the future.

Notation and conventions Throughout this paper we assume that the initial fast memory is empty. Furthermore we assume $p > k$ since otherwise a request sequence can be served without any faults. Moreover let $k \geq 2$. When constructing and analyzing a request sequence, a page is called *new* if it has not been referenced so far.

2 Analysis of OPT

Let $\mathcal{C} = (c_0, \ldots, c_{p-1})$ be an arbitrary characteristic vector. First we develop a lower bound on $\text{OPT}(\sigma)$, for any σ defined by $\mathcal{C}$. Then we prove that our bound is nearly tight.

2.1 A Lower Bound

The lower bound on $\text{OPT}(\sigma)$ given by Panagiotou and Souza [20], cf. (1), depends on multicycles. A multicycle repeatedly requests a sequence of, say l, distinct pages. On each repetition of the sequence OPT incurs at least $l - k$ page faults if $l \geq k$. Panagiotou and Souza prove that every request sequence can be viewed as a collection of multicycles. Instead, our new lower bound on $\text{OPT}(\sigma)$ is based on a novel approach that relates page faults to memory hits. If OPT has a hit on a distance-l request with $l \geq k$, then it must have incurred at least $l - (k - 1)$ faults since the last reference to the requested page. We assign tokens to the respective faults. This allows us to lower bound the number of page faults in terms of the number of hits, see Lemma 1 below. The subsequent analysis then lower bounds the expression on the number of hits, for any request sequence. It turns out that the expression is minimized if the hits (faults) occur on the distance-l requests with the smallest (largest) possible value of l.

Formally, given any σ, let f_l denote the total number of page faults incurred by OPT on distance-l requests, $0 \leq l \leq p - 1$, and let $h_l = c_l - f_l$ be the number of hits on this type of requests. We relate the total number of faults to the number of hits.

Lemma 1 *Let σ be any request sequence characterized by $\mathcal{C}$. There holds a)* $\text{OPT}(\sigma) = p + \sum_{l=k}^{p-1} f_l$ *and (b)* $p + \sum_{l=k}^{p-1} f_l \geq k + \sum_{l=k}^{p-1} h_l \frac{l-k+1}{k-1}$.

Proof We first prove part (a). There holds $\text{OPT}(\sigma) = p + \sum_{l=0}^{p-1} f_l$ because OPT incurs one page fault whenever any of the p distinct pages is requested for the first time. Moreover, by the definition of f_l, OPT has exactly f_l faults on the distance-l requests, for $l = 0, \ldots, p-1$. It remains to argue that $f_l = 0$, for $l = 0, \ldots, k-1$. Obviously, $f_0 = 0$. So assume $l \geq 1$. Consider a distance-l request $\sigma(t) = x$ and let $\sigma(t')$, where $t' < t$, be the most recent request when page x was referenced in σ. Immediately after OPT has served $\sigma(t')$, page x is in fast memory. Whenever OPT incurs a fault on a request $\sigma(s)$, $t' < s < t$, the set $\{\sigma(s), \ldots, \sigma(t)\}$ of pages referenced until and including $\sigma(t)$ contains at most $l + 1 \leq k$ pages. This holds true because $\sigma(t)$ is a distance-l request, where $l \leq k - 1$. Thus the set $\{\sigma(s), \ldots, \sigma(t)\}$ contains at most $k - 1$ pages different from $y = \sigma(s)$. Hence when OPT serves $\sigma(s)$, its fast memory must store a page not contained in $\{\sigma(s+1), \ldots, \sigma(t)\}$. OPT evicts a page whose next request is farthest in the future. Therefore it drops a page that is not referenced by $\sigma(s+1), \dots, \sigma(t)$.

We next prove part (b). To this end we assign tokens to page faults whenever OPT has a hit on a distance-l request, where $l \geq k$, in σ. Let $\sigma(t) = x$ be such a request and let $\sigma(t')$ be the most recent request to x. A total of l distinct pages are referenced in the subsequence $\sigma(t'+1), \dots, \sigma(t-1)$. Algorithm OPT incurs at least $l-(k-1)$ page faults in this subsequence because $l \geq k$ and $\sigma(t)$ is a hit. Now we select the last $l-(k-1)$ page faults occurring before $\sigma(t)$ and assign a token to each of these faults. By this process, exactly $\sum_{l=k}^{p-1} h_l(l-k+1)$ tokens are placed.

In the following we upper bound the number of tokens a page fault may be assigned. Let $\sigma(s)$ be any page fault. Suppose that $\sigma(s)$ receives a token when there is a hit on a request $\sigma(t)$ with $s < t$. Reference $\sigma(t)$ is a distance-l request where $l \geq k$. The page $x = \sigma(t)$ is not requested in $\sigma(s), \dots, \sigma(t-1)$. For, if x were requested in this subsequence, then the $l-(k-1)$ tokens would be assigned to page faults occuring between the most recent request to x and $\sigma(t)$. Since x is not requested by $\sigma(s), \dots, \sigma(t-1)$ and $\sigma(t)$ is a hit, x must reside in fast memory when OPT has served $\sigma(s)$. Also x is different from the page referenced by $\sigma(s)$. Hence when $\sigma(s)$ receives a token due to a hit on $\sigma(t)$, page $x = \sigma(t)$ resides in fast memory when OPT has served $\sigma(s)$ and is different from the page accessed by $\sigma(s)$. Since there exist at most $k-1$ such pages, $\sigma(s)$ can receive at most $k-1$ tokens.

We next argue that the first k page faults in σ do not receive any token. Let $\sigma(t_1), \dots, \sigma(t_k)$ be the requests where these first k page faults occur. Recall that the initial fast memory is empty. Hence each $\sigma(t_i)$, $1 \leq i \leq k$, requests a new page that has not been referenced before in σ. There holds $t_1 = 1$, the k pages referenced by $\sigma(t_1), \dots, \sigma(t_k)$ are pairwise distinct and the subsequence $\sigma(1), \dots, \sigma(t_k)$ only contains requests to these pages. Furthermore, the first hit on a distance-l request with $l \geq k$ occurs after $\sigma(t_k)$. Let $\sigma(t)$, $t > t_k$, be such a hit and assume that the referenced page $x = \sigma(t)$ was requested most recently by $\sigma(t')$, where $t' < t_k$, so that any of the faults $\sigma(t_1), \dots, \sigma(t_k)$ could potentially be assigned a token. The subsequence $\sigma(t'+1), \dots, \sigma(t-1)$ contains l pages, at most $k-1$ of which can be identical to those referenced by $\sigma(t_1), \dots, \sigma(t_k)$ because $\sigma(t)$ is a page from $\sigma(t_1), \dots, \sigma(t_k)$. Hence $\sigma(t'+1), \dots, \sigma(t-1)$ contains at least $l-(k-1)$ pages that are different from those requested by $\sigma(t_1), \dots, \sigma(t_k)$. These pages different from $\sigma(t_1), \dots, \sigma(t_k)$ are referenced after $\sigma(t_k)$ and the first request to each of these pages is a fault since, again, the initial fast memory is empty. Our token assignment scheme places $l-(k-1)$ tokens on the last $l-(k-1)$ page faults prior to $\sigma(t)$. Hence faults $\sigma(t_1), \dots, \sigma(t_k)$ do not receive any token.

In summary, each fault is assigned at most $k-1$ tokens, where the first k faults not receive any. We conclude that the total number of tokens is upper bounded by $(p-k)(k-1) + \sum_{l=k}^{p-1} f_l(k-1)$, i.e.

$$\sum_{l=k}^{p-1} h_l(l-k+1) \leq (p-k)(k-1) + \sum_{l=k}^{p-1} f_l(k-1).$$

This is equivalent to $k + \sum_{l=k}^{p-1} h_l \frac{l-k+1}{k-1} \leq p + \sum_{l=k}^{p-1} f_l$. □

For the further analysis, given a vector $\mathcal{C} = (c_0, \ldots, c_{p-1})$, we define two functions f and g as well as values λ and c^*_λ. For any integer j with $k \leq j \leq p-1$ and any real number γ with $0 \leq \gamma \leq c_j$, let

$$f(j,\gamma) = k + \sum_{l=k}^{j-1} c_l \frac{l-k+1}{k-1} + \gamma \frac{j-k+1}{k-1} \quad \text{and} \quad g(j,\gamma) = p + (c_j - \gamma) + \sum_{l=j+1}^{p-1} c_l.$$

Intuitively, $f(j,\gamma)$ is the number of page faults, as implied by Lemma 1, if memory hits occur on all the distance-l requests, for $l = k, \ldots, j-1$, and γ distance-j requests. The corresponding $g(j,\gamma)$ is the number of requests where these faults can occur. If $f(p-1, c_{p-1}) \leq g(p-1, c_{p-1})$, then let $\lambda = p-1$ and $c^*_\lambda = c_{p-1}$. Otherwise determine the largest λ and corresponding c^*_λ such that $f(\lambda, c^*_\lambda) = g(\lambda, c^*_\lambda)$.

Lemma 2 *(a) The values λ and c^*_λ are well-defined.*
(b) Let j' and γ' be a pair such that $f(j',\gamma') \leq g(j',\gamma')$. Then $f(j',\gamma') \leq f(\lambda, c^*_\lambda) \leq g(\lambda, c^*_\lambda) \leq g(j',\gamma')$. Moreover, $j' \leq \lambda$. If $j' = \lambda$, then $\gamma' \leq c^*_\lambda$.

Proof For any fixed j, $k \leq j \leq p-1$, and variable γ, $0 < \gamma < c_j$, the functions $f(j,\gamma)$ and $g(j,\gamma)$ are continuous. For any fixed j and increasing γ, function $f(j,\gamma)$ is strictly increasing while $g(j,\gamma)$ is strictly decreasing. For $j = k, \ldots, p-2$, there holds $f(j, c_j) = f(j+1, 0)$ and $g(j, c_j) = g(j+1, 0)$. Hence f and g are continuous, when considering the transitions from $f(j, c_j)$ to $f(j+1, 0)$ and from $g(j, c_j)$ to $g(j+1, 0)$. Furthermore the functions are monotone, i.e. f is increasing and g is decreasing.

We first prove part (a). If $f(p-1, c_{p-1}) \leq g(p-1, c_{p-1})$, there is nothing to show. Suppose that $f(p-1, c_{p-1}) > g(p-1, c_{p-1})$. Since $f(k, 0) = k < p \leq g(k, 0)$, the monotonicity of f and g ensures the existence of j^* and γ^* such that $f(j^*, \gamma^*) = g(j^*, \gamma^*)$. The first parameter j of f and g is an integer upper bounded by $p-1$. Hence part (a) holds.

For the proof of part (b), first suppose that $f(p-1, c_{p-1}) \leq g(p-1, c_{p-1})$, in which case $\lambda = p-1$ and $c^*_\lambda = c_{p-1}$. For any pair j', γ' there holds $f(j',\gamma') \leq f(p-1, c_{p-1})$ and $g(p-1, c_{p-1}) \leq g(j',\gamma')$, which establishes the desired inequality. Obviously, $j' \leq p-1$. The monotonicity of f and g implies $\gamma' \leq c^*_\lambda$. Next assume that $f(p-1, c_{p-1}) > g(p-1, c_{p-1})$. In this case $f(\lambda, c^*_\lambda) = g(\lambda, c^*_\lambda)$. Functions f and g are monotone as described in the last paragraph. Hence, for any pair j, γ with $f(j,\gamma) > f(\lambda, c^*_\lambda)$ there holds $g(\lambda, c^*_\lambda) \geq g(j,\gamma)$. For any pair j,γ with $g(j,\gamma) < g(\lambda, c^*_\lambda)$ there holds $f(\lambda, c^*_\lambda) \leq f(j,\gamma)$. Now let j' and γ' be a pair such that $f(j',\gamma') \leq g(j',\gamma')$. If $f(j',\gamma') > f(\lambda, c^*_\lambda)$ held true, then $g(\lambda, c^*_\lambda) \geq g(j',\gamma')$ and $f(j',\gamma') > f(\lambda, c^*_\lambda) = g(\lambda, c^*_\lambda) \geq g(j',\gamma')$. If $g(j',\gamma') < g(\lambda, c^*_\lambda)$ held true, then $f(\lambda, c^*_\lambda) \leq f(j',\gamma')$ and $g(j',\gamma') < g(\lambda, c^*_\lambda) = f(\lambda, c^*_\lambda) \leq f(j',\gamma')$. In both case we obtain a contradiction to the fact that $f(j',\gamma') \leq g(j',\gamma')$. Therefore, the desired inequality holds. Since λ is the largest integer such that f and g assume an equal value, the monotonicity of the functions implies $j' \leq \lambda$. Furthermore, $\gamma' \leq c^*_\lambda$ if $j' = \lambda$. □

Theorem 1 *Let σ be any request sequence characterized by $\mathcal{C}$. There holds*

$$\text{OPT}(\sigma) \geq \max\left\{p, k + \sum_{l=k}^{\lambda-1} c_l \frac{l-k+1}{k-1} + c^*_\lambda \frac{\lambda-k+1}{k-1}\right\}.$$

Proof Obviously, $\text{OPT}(\sigma) \geq p$. By Lemma 1

$$\text{OPT}(\sigma) = p + \sum_{l=k}^{p-1} f_l \geq k + \sum_{l=k}^{p-1} h_l \frac{l-k+1}{k-1}. \tag{3}$$

Determine the largest j', where $k \leq j' \leq p-1$, and corresponding γ', where $0 \leq \gamma' \leq c_{j'}$ such that $\sum_{l=k}^{p-1} h_l = \sum_{l=k}^{j'-1} c_l + \gamma'$. Intuitively, we express the total number of hits in terms of a prefix of the c_l-values, for increasing $l \geq k$. Of course, the hits do not necessarily occur on all the distance-l requests, where $l \leq j'$. Then

$$\begin{aligned}\text{OPT}(\sigma) &= p + \sum_{l=k}^{p-1} f_l = p + \sum_{l=k}^{p-1} c_l - \sum_{l=k}^{p-1} h_l = p + (c_{j'} - \gamma') \\ &\quad + \sum_{l=j'+1}^{p-1} c_l = g(j', \gamma').\end{aligned} \tag{4}$$

In (3) expression $k + \sum_{l=k}^{p-1} h_l \frac{l-k+1}{k-1}$ is minimized if the hits occur on distance-l requests with smallest possible l subject to the constraint that at most c_l distance-l requests occur in σ. Hence

$$k + \sum_{l=k}^{p-1} h_l \frac{l-k+1}{k-1} \geq k + \sum_{l=k}^{j'-1} c_l \frac{l-k+1}{k-1} + \gamma' \frac{j'-k+1}{k-1} = f(j', \gamma').$$

Combining (3) and (4) together with the last inequality we obtain $\text{OPT}(\sigma) = g(j', \gamma') \geq k + \sum_{l=k}^{p-1} h_l \frac{l-k+1}{k-1} \geq f(j', \gamma')$. Using Lemma 2, part (b), we conclude $\text{OPT}(\sigma) \geq f(\lambda, c^*_\lambda) = k + \sum_{l=k}^{\lambda-1} c_l \frac{l-k+1}{k-1} + c^*_\lambda \frac{\lambda-k+1}{k-1}$. □

Proposition 1 *The lower bound on* $\text{OPT}(\sigma)$ *stated in Theorem 1 is always greater than that in inequality (1).*

The proof is given in the "Appendix".

2.2 Tightness of the Lower Bound

The lower bound of Theorem 1 is essentially best possible. We present a strategy that, given an arbitrary $\mathcal{C} = (c_0, \ldots, c_{p-1})$, constructs a request sequence that can be served with the stated number of page faults, up to an additive constant of $2(\lambda - k + 1)$.

```
Strategy GenerateRequestSequence (GRS):
 1. Request $k$ new pages; $new := p - k$;
 2. while $\sum_{l=k}^{p-1} c_l > 0$ do
 3.     $l^*$ := smallest $l \geq k$ such that $c_l > 0$;
 4.     if $\sum_{l=l^*}^{p-1} c_l + new \geq l^* - k + 1$ then
 5.         $l^*$ := largest $j$ with $j \leq p - 1$ such that $\sum_{l=j}^{p-1} c_l + new \geq j - k + 1$;
 6.         if $\sum_{l=k}^{p-1} c_l + new \geq l^*$ then
 7.             $l^*$ := smallest $j$ such that $\sum_{l=k}^{j} c_l \geq k - 1$;
 8.     for $i := 1$ to $l^* - k + 1$ do LongDistanceRequest;
 9.     for $i := 1$ to $k - 1$ do ShortDistanceRequest;
10. while $\sum_{l=0}^{k-1} c_l > 0$ do Generate a distance-$l$ request, for any $l$ with $c_l > 0$; $c_l := c_l - 1$;
11. while $new > 0$ do Request a new page; $new := new - 1$;
```

Fig. 1 The strategy for generating a request sequence

The strategy is called *GenerateRequestSequence*, or *GRS* for short. It takes the original $\mathcal{C}$ and in a general step issues a distance-l request, $0 \leq l \leq p - 1$, according to a specific protocol. The corresponding value c_l is reduced by 1. The process stops when all vector entries c_l, $0 \leq l \leq p - 1$, are equal to 0 and all the p distinct pages have been requested.

On a high level the constructed request sequence consists of a series of phases. In each phase, first a certain number of distance-l requests, for the largest possible $l \geq k$, are issued. Then $k - 1$ distance-l requests, for the smallest possible $l \geq k$, may be generated. Finally, distance-l request with $l < k$ are generated. The request sequence can be served so that page faults only occur in the first part of the phases, i.e. on distance-l requests issued for the largest possible l. All other requests are memory hits. This allows us to analyze the number of page faults in terms of the functions f and g as well as values λ and c_λ^* defined in Sect. 2.1. Hence we derive a bound similar to that in Theorem 1.

Description of GRS Let $\mathcal{C} = (c_0, \ldots, c_{p-1})$ be an arbitrary characteristic vector. First, starting with an empty fast memory, *GRS* requests k new pages. Then *GRS* generates a sequence of phases in which requests to new pages or distance-l requests with $k \leq l \leq p - 1$ are issued. The goal is to reduce the vector entries $c_k, \ldots, c_{p-1}$ to 0 while generating subsequences of requests that can be served with low cost. Each phase, except for possibly the last one, consists of exactly l^* requests, for some properly chosen l^* that depends on the state of $\mathcal{C}$ at the beginning of the phase. Such a phase with l^* requests is *complete*. The last phase may contain fewer requests. Finally, *GRS* issues distance-l requests with $0 \leq l \leq k - 1$ and requests to the remaining new pages, if there are any. We remark that at any time t *GRS* can issue a distance-l request provided that at least $l + 1$ distinct pages have been requested so far. The algorithm just has to determine the most recent request $\sigma(t')$, where $t' < t$, such that $\sigma(t' + 1), \ldots, \sigma(t - 1)$ reference exactly l distinct pages. It then issues a request to the page specified by $\sigma(t')$. In Lemma 3 we show that *GRS* never fails, i.e. when it has to create a distance-l request, indeed at least $l + 1$ distinct pages have been referenced so far. Figure 1 gives a pseudo-code description of *GRS*. In line 1, k new page are requested. At any time a variable *new* stores the current number of new pages. The while-loop consisting of lines 2–9 generates the phases as described in the next two paragraphs. Each execution of lines 2–9 produces one phase.

The phases Each phase with l^* requests, for the calculated value l^*, contains $l^* - k + 1$ so-called *long-distance requests* followed by $k - 1$ *short-distance requests*. When generating a long-distance request, *GRS* either requests a new page or issues a distance-l request, for the largest index $l \geq k$ such that $c_l > 0$. In a short-distance request *GRS* poses a distance-l request, for the smallest possible $l \geq k$ such that $c_l > 0$. We will prove that each phase can be served so that page faults occur only on the long-distance requests; all short-distance requests are memory hits. As we shall see, the crucial property is that each short-distance request is to a page that was requested before in the phase or during the last k requests preceding the phase. The property holds if each short-distance request is a distance-l request, for some $l \leq l^*$.

Phase lengths An important component of *GRS* is the choice of l^*, for each phase. Loosely speaking, l^* is the smallest j such that (a) $\sum_{l=k}^{j} c_l \geq k-1$ and (b) $\sum_{l=k}^{p-1} c_l \geq j$, provided that such a value exists. Condition (a) ensures that $k - 1$ short-distance requests can be issued. Condition (b) guarantees that a complete phase can be generated. Condition (a) also implies that at the end of the phase the vector entries c_l with $l < j$ are equal to 0. Formally, in a first step *GRS* sets l^* to the smallest possible l such that $l \geq k$ and $c_l > 0$; cf. line 3 in the pseudo-code of *GRS*. A (weak) necessary condition for the generation of a complete phase is that the total number of requests still to be generated is at least $l^* - k + 1$, i.e. $\sum_{l=l^*}^{p-1} c_l + new \geq l^* - k + 1$. If this inequality does not hold, then a final phase consisting of less than $l^* - k + 1$ long distance requests is created. If the inequality does hold, then the value of l^* is refined. More precisely, it is first set to the largest j such that $\sum_{l=j}^{p-1} c_l + new \geq j - k + 1$; see line 5 in the pseudo-code. This is the right choice of l^* to ensure a proper termination of the phase generation in case no complete phase can be created anymore. Finally, *GRS* checks if $\sum_{l=k}^{p-1} c_l + new \geq l^*$. In this case a complete phase can be created and l^* is set to the smallest j such that $\sum_{l=k}^{j} c_l \geq k - 1$; cf. line 7 of the pseudo-code. In Lemma 4 below we show that a complete phase, consisting of l^* requests is generated, if and only if l^* is set in line 7. Otherwise an incomplete final phase is created. In order to distinguish the various setting of l^*, we introduce some notation. If l^* is set in line 7, we say that *Case C* holds, referring to the fact that a complete phase is generated. If l^* is set in line 3 or in line 5 and not refined further, then *Case I* or *Case I'* holds, respectively. In the latter cases, an incomplete phase is created.

Figure 2 depicts the procedures for issuing the short- and long-distance requests. In the generation of the long-distance requests, preference is given to new pages if there exist such. When the phase generation terminates, *GRS* generates distance-l requests, where $0 \leq l \leq k - 1$, and issues requests to new pages, in case there are any. These final requests are generated in lines 10 and 11 in the pseudo-code of *GRS*.

```
Procedure LongDistanceRequest:
1. if $new > 0$ then
2.    Request a new page; $new := new - 1$;
3. else if $\sum_{l=k}^{p-1} c_l > 0$ then
4.    $l$ := largest $j \geq k$ such that $c_j > 0$;
5.    Generate a distance-$l$ request; $c_l := c_l - 1$;
```

```
Procedure ShortDistanceRequest:
1. if $\sum_{l=k}^{p-1} c_l > 0$ then
2.    $l$ := smallest $j \geq k$ such that $c_j > 0$;
3.    Generate a distance-$l$ request; $c_l := c_l - 1$;
```

Fig. 2 The procedures *LongDistanceRequest* and *ShortDistanceRequest*

Theorem 2 *Let σ be the request sequence generated by GRS. There holds*

$$\text{OPT}(\sigma) \leq k + \sum_{l=k}^{\lambda-1} c_l \frac{l-k+1}{k-1} + c_\lambda^* \frac{\lambda-k+1}{k-1} + 2(\lambda-k+1).$$

Analysis of GRS In the remainder of this section we prove Theorem 2. We analyze the number of page faults needed to serve the sequence generated by *GRS*. Formally, a *phase* is a maximal subsequence of requests generated during an execution of the while-loop consisting of lines 2–9 of *GRS*. In such a phase the first up to $l^* - k + 1$ requests issued in line 8 are called *long-distance requests*. The remaining up to $k - 1$ requests issued in line 9 are *short-distance requests*. Here l^* is the value determined during the execution of lines 3–7. The phase is *complete* if it contains l^* requests. The following Lemma 3, part (b) proves that *GRS* constructs a request sequence characterized by $\mathcal{C}$. The proof needs part (a) of the lemma that identifies a property of short-distance requests. This property will also be needed in further lemmas.

Lemma 3 *(a) Consider an arbitrary execution of the while-loop consisting of lines 2–9 in GRS. Let l^* be the value determined by lines 3–7. If in a call to ShortDistanceRequest there holds $\sum_{l=k}^{p-1} c_l > 0$, then the smallest $j \geq k$ such that $c_j > 0$ satisfies $j \leq l^*$.*
(b) GRS never fails and generates a request sequence characterized by $\mathcal{C}$.

Proof a) Consider an arbitrary execution of the while-loop consisting of lines 2–9 in *GRS*, where l^* is the value determined in lines 3–7. A first observation is that if *ShortDistanceRequest* is called and $\sum_{l=k}^{p-1} c_l > 0$, then the value l^* must have been set in lines 5 or 7 (Case I' or Case C) of the current while-loop. For, if this were not the case, then $\sum_{l=l^*}^{p-1} c_l + new = \sum_{l=k}^{p-1} c_l + new < l^* - k + 1$ when line 3 of the loop was executed. Hence less than $l^* - k + 1$ long-distance requests could be generated before $\sum_{l=k}^{p-1} c_l = 0$ and no short-distance request would be issued.

We procede with the concrete proof of the statement of part (a). Suppose that l^* was set in line 5 but not reset in line 7 of *GRS*, i.e. Case I' holds. If $l^* = p-1$, there is nothing to show. If $l^* < p-1$, then by the choice of l^*, $\sum_{l=l^*}^{p-1} c_l + new \geq l^* - k + 1$ but $\sum_{l=l^*+1}^{p-1} c_l + new < l^* + 1 - k + 1$. This implies $\sum_{l=l^*+1}^{p-1} c_l + new \leq l^* - k + 1$ before line 8 is executed. Therefore, after the execution of this for-loop we have $c_l = 0$, for any $l > l^*$, because *LongDistanceRequest* issues distance-l requests, for the largest possible $l \geq k$. We conclude that the smallest possible $j \geq k$ with $c_j > 0$ always satisfies $j \leq l^*$.

Finally assume that l^* was set in line 7 of *GRS*, i.e. Case C holds. Then $\sum_{l=k}^{l^*} c_l \geq k-1$. Recall again that *LongDistanceRequest* issues distance-l requests, for the largest possible $l \geq k$. We conclude that during any of the $k-1$ calls of *ShortDistanceRequest* the smallest $j \geq k$ with $c_j > 0$ satisfies $j \leq l^*$.

(b) We show that the request generation in lines 8, 9 and 10 of *GRS* is always well-defined. First observe that in line 1 of *GRS* k distinct pages a requested. Hence in line 10, distance-l requests with $l \leq k-1$ can always be issued. Next consider an execution of *LongDistanceRequest*. If line 4 of the procedure is executed, then all the p distinct pages have already been referenced and a distance-l request can be generated for any l with $0 \leq l \leq p-1$.

We study a call to *ShortDistanceRequest* in line 9 of *GRS*. Consider the execution of the while-loop consisting of lines 2–9 in which the call is made. Let l^* be the value determined during lines 3–7. If prior to the call of *ShortDistanceRequest* all the p distinct pages have been referenced, a distance-l request, for any $0 \leq l \leq p-1$, can be generated. So suppose that less than p distinct pages have been requested so far. Then in the preceding execution of the for-loop in line 8, only new pages were requested by *LongDistanceRequest*. Hence the phase constructed so far contains $l^* - k + 1$ pairwise distinct pages. Together with the k new pages requested in line 1 of *GRS*, a total of at least $l^* + 1$ pairwise distinct pages have been referenced so far. If in the call to *ShortDistanceRequest* there holds $\sum_{l=k}^{p-1} c_l > 0$, then by part (b) of the lemma the smallest $j \geq k$ such that $c_j > 0$ satisfies $j \leq l^*$. Hence the request generation in line 3 of the procedure is well defined. □

In Lemma 4 below we prove that a complete phase is generated if and only if l^* is set in line 7 (Case C). For the proof we need the following auxiliary claim.

Claim 1 *Consider an arbitrary execution of the while-loop consisting of lines 2–9. If l^* is set in line 7, then the value is upper bounded by that of the previous setting in line 5.*

Proof Immediately before l^* is set in line 7 (Case C), with the prior choice of l^* in line 5, there holds $\sum_{l=l^*}^{p-1} c_l + new \geq l^* - k + 1$ and $\sum_{l=k}^{p-1} c_l + new \geq l^*$. We argue that $\sum_{l=k}^{l^*} c_l \geq k-1$. If $l^* = p-1$, there is nothing to show because $new \leq p-k$. Otherwise $\sum_{l=l^*+1}^{p-1} c_l + new < l^* + 1 - k + 1$ by the choice of l^*. Again this implies $\sum_{l=l^*+1}^{p-1} c_l + new \leq l^* - k + 1$ and the condition $\sum_{l=k}^{p-1} c_l + new \geq l^*$ ensures $\sum_{l=k}^{l^*} c_l \geq k-1$. Therefore the setting in line 7 cannot increase the value of l^*. □

Lemma 4 *Consider an arbitrary execution of the while-loop consisting of lines 2–9. If l^* is set in line 7, then a complete phase is generated. Otherwise this is the last execution of the while-loop and the phase contains less than l^* requests.*

Proof Suppose that l^* is set in line 7. Immediately before this setting, with the prior choice of l^* in line 5, there holds $\sum_{l=k}^{p-1} c_l + new \geq l^*$. By the above Claim 1, the setting in line 7 cannot increase the value of l^* so that the last inequality is maintained. Therefore, in the for-loops in lines 8 and 9 a total of l^* requests are issued. Next assume that l^* is not set in line 7. If in the execution of line 4 there holds $\sum_{l=l^*}^{p-1} c_l + new <$

$l^* - k + 1$, then less than $l^* - k + 1$ requests can be issued before $\sum_{l=k}^{p-1} c_l = 0$. If l^* is set in line 5 but not in line 7 (Case I'), then $\sum_{l=k}^{p-1} c_l + new < l^*$. In the subsequent execution of lines 8 and 9 less than l^* requests are issued before $\sum_{l=k}^{p-1} c_l = 0$. □

The next lemma identifies important properties of the requests in a phase. The long-distance requests reference distinct pages. The short-distance requests also reference distinct pages, having the property that they occured earlier in the phase or at the end of the previous phase. This ensures that the short-distance requests can be served without page faults.

Lemma 5 *Let P be an arbitrary phase and l^* be the value determined in lines 3–7 when the phase was generated.*

(a) The up to $l^* - k + 1$ long-distance requests reference pairwise distinct pages. These pages are also different from the last k distinct pages referenced before the beginning of P.
(b) The up to $k - 1$ short-distance requests reference pairwise distinct pages that are also different from the page referenced by the last long-distance request. Each of these short-distance requests references a page that was requested during the last k requests before P or by a long-distance request in P.

Proof (a) It suffices to show that when a distance-l request is generated by *LongDistanceRequest*, then $l \geq l^*$. Suppose that l^* was set in line 3 but not reset in lines 5 or 7, i.e. Case I holds. In this case there is nothing to show because in this case l^* is the smallest index l with $c_l \geq 0$. If l^* was set in line 5 or 7 (Case I' or Case C), then it satisfies $\sum_{l=l^*}^{p-1} c_l + new \geq l^* - k + 1$ because an adjustment in line 7 cannot increase the value determined in line 5. Since *LongDistanceRequest* issues distance-l requests, for the largest l with $c_l > 0$, the index l cannot drop below l^*.

(b) Whenever *ShortDistanceRequest* generates a distance-l request, $l \geq k$. Hence the up to $k - 1$ short-distance requests reference distinct pages that are also different from the last long-distance request. Observe that if short-distance requests are issued, they must be preceded by $l^* - k + 1$ long-distance requests. Recall that in line 1 of *GRS*, k new pages are requested. By Lemma 4 every phase except for possibly the last one is complete. It follows that before the beginning of P the last k requests reference distinct pages. By part (a) of this lemma they are also different from the long-distance requests in P. Hence the subsequence consisting of the last k requests before P and the first $l^* - k + 1$ requests in P reference a total of $l^* + 1$ distinct pages. Lemma 3, part (a), ensures that whenever *ShortDistanceRequest* generates a distance-l request, there holds $l \leq l^*$. Therefore, any such request references a page that was requested during the last k requests before P or by a long-distance request in P. □

Lemma 6 analyzes the service of the request sequence generated by *GRS*.

Lemma 6 *Suppose that the request sequence σ produced by GRS contains at least one phase generated in lines 2–9. Sequence σ produced by GRS can be served such that the following two properties hold. (1) No page faults occur on short distance requests, if there are any. (2) A the end of the last phase the last k distinct pages referenced are in fast memory.*

Proof Let ALG be an algorithm that serves σ as follows. The first k distinct pages referenced are loaded into fast memory without making any page evictions. Whenever there is a page fault in a phase P ALG evicts a page that is not referenced by any short-distance request in the phase. If there are several such pages, it evicts the page that was requested least recently. Note that ALG is well-defined because there exist at most $k-1$ short-distance requests in any phase.

At the beginning of the first phase the last k distinct pages referenced are in fast memory. We show that if at the beginning of a phase P the last k distinct pages referenced are in fast memory, then this property also holds at the end of P w.r.t. the pages requested before the end of P. Moreover, no page faults occur on short-distance requests in P. By Lemma 5 part (b) any short-distance request in P references a page that was either requested during the last k requests before P or by a long-distance request in P. In the first case, by assumption, the corresponding page is in fast memory at the beginning of P and will not be evicted by ALG until the end of the phase. In the second case the corresponding page will be loaded into fast memory when the long-distance request is served and not be evicted until the end of P. Thus the statement on the page faults holds. If P is a complete phase, then at the end of P the last k distinct pages referenced are all in fast memory. If P is not a complete phase, then the desired property also holds because ALG always keeps the most recently requested pages in fast memory in addition to those needed by the short-distance requests. □

In the next two lemmas let j^* be the value of l^* determined in lines 3–7 when the last phase is generated by *GRS*. The following lemma shows that the long-distance and short-distance requests are separated along j^*.

Lemma 7 *In the entire request sequence generated by GRS any distance-l request with $k \leq l < j^*$ is a short-distance request. Any distance-l request with $l > j^*$ is a long-distance request.*

Proof As always, let l^* be the value determined in an execution of lines 3–7 of GRS. We first prove that over the executions of the while-loop consisting of lines 2–9, these values form a non-decreasing sequence. To this end consider an arbitrary execution of the while-loop in lines 2–9. If l^* is set in line 7 (Case C), then l^* is the smallest j such that $\sum_{l=k}^{j} c_l \geq k-1$. By Lemma 4 the execution of the while-loop produces a complete phase so that exactly $k-1$ short-distance request are issued and $c_l = 0$, for any $l < l^*$. In a subsequent execution of the while-loop the chosen l^* value cannot be smaller because it is lower bounded by the smallest $l \geq k$ such that $c_l > 0$. On the other hand if l^* is set in lines 3 or 5 (Cases I or I'), then by Lemma 4 the current execution of the while-loop is the last one. Therefore, as claimed, the l^* values form a non-decreasing sequence.

By Lemma 3 part (a) and the just proven property that the l^* values form a non-decreasing sequence, any distance-l request with $l > j^*$ must be a long-distance request. We next show that no distance-l request with $l < j^*$ can be a long-distance request. Consider the generation of the last phase. If j^* is determined in lines 3 or 7 (Cases I or C), then $c_{j^*} > 0$ when the value j^* is set. If *LongDistanceRequest* has generated a distance-l request in any previous phase, then $l \geq j^*$ because the procedure always chooses the largest l such that $c_l > 0$. As for the last phase, if j^* is determined in line 3 (Case I), no distance-l request with $l < j^*$ can be issued. If j^* is determined in line 7 (Case C), then by Lemma 4 the last phase is complete. Since by the choice of j^* there holds $\sum_{l=k}^{j^*-1} c_l < k-1$, we obtain $\sum_{l=j^*}^{p-1} c_l \geq j^* - k + 1$. Therefore, in the last phase, no long-distance request can be a distance-l request with $l < j^*$.

Finally assume that j^* is set in line 5 but not reset in line 7, i.e. Case I' holds. If the variable *new* is still positive, then the long-distance requests issued in previous phases cannot be distance-l requests, for any $l \geq k$. If $new = 0$, then $\sum_{l=j^*}^{p-1} c_l \geq j^* - k + 1$. The latter inequality and the fact that there exist an $l \geq j^*$ such that $c_l > 0$ imply that neither in any previous phase nor in the last phase a distance-l request with $l < j^*$ can be issued. □

Let λ and c^*_λ be the values as defined in Sect. 2.1.

Lemma 8 *The value j^* is upper bounded by λ. Moreover the entire request sequence σ produced by GRS contains at most c^*_λ distance-λ requests that are short-distance requests.*

Proof Let γ^* be the number of distance-j^* requests that are issued as short-distance requests. By Lemma 7, the total number of short-distance requests is $\sum_{j=k}^{j^*-1} c_l + \gamma^*$. If at the end of the last phase all p distinct pages have been requested, the number of long-distance requests is $p - k + (c_{j^*} - \gamma^*) + \sum_{l=j^*+1}^{p-1} c_l$. Otherwise the number is upper bounded by $p - k$. In the following we relate these expressions. Intuitively, in any phase we distribute the number of long-distance requests among the short-distance requests.

Consider any distance-l request $l \leq j^*$ that is issued as a short-distance request. This request is contained in a phase with exactly $l^* - k + 1$ long-distance request where, as usual, l^* is the value determined in lines 3–7 of *GRS* when the phase is generated. By Lemma 3 part (a) there holds $l \leq l^*$. Now we split the number $l^* - k + 1$ of long-distance requests evenly among the short-distance requests of the phase. If the phase is complete, each short-distance request is assigned a request volume of $\frac{l^*-k+1}{k-1}$. If the phase is not complete, the assigned value is $\frac{l^*-k+1}{s}$, where $s < k - 1$ is the actual number of short-distance requests in the phase. Thereby, a short-distance request that is a distance-l request will be assigned a request volume of at least $\frac{l-k+1}{k-1}$. This implies that the number of long-distance requests is lower bounded by

$$\sum_{l=k}^{j^*-1} c_l \frac{l-k+1}{k-1} + \gamma^* \frac{j^*-k+1}{k-1}.$$

If at the end of the last phase all the p distinct pages have been requested, we obtain $p + (c_{j^*} - \gamma^*) + \sum_{l=j^*+1}^{p-1} c_l \geq k + \sum_{l=k}^{j^*-1} c_l \frac{l-k+1}{k-1} + \gamma^* \frac{j^*-k+1}{k-1}$. Recall the functions f and g defined in Sect. 2.1. The last inequality then reads as $f(j^*, \gamma^*) \leq g(j^*, \gamma^*)$. Lemma 2 part (b) implies $j^* \leq \lambda$. If $j^* = \lambda$, then $\gamma^* \leq c^*_\lambda$.

We finally study the case that at the end of the last phase some of the p pages have not yet been requested. In this case there are no distance-l requests, $k \leq l \leq p-1$, that are long-distance requests. Hence $p > \sum_{l=k}^{p-1} c_l \frac{l-k+1}{k-1}$. Using the functions f and g, we obtain $f(p-1, c_{p-1}) \leq g(p-1, c_{p-1})$. In this case $\lambda = p-1$ and $c^*_\lambda = c_{p-1}$. □

We finally prove the main result of this section.

Proof of Theorem 2 Given σ the service of the first k requests, which reference new pages, requires k page faults. These k pages then reside in fast memory. First assume that *GRS* does not generate any phases in lines 2–9, which implies $\sum_{l=k}^{p-1} c_l = 0$. Then any distance-l request with $0 \leq l \leq k-1$ is a memory hit. Thus $\text{OPT}(\sigma) = p$. We argue that the upper bound on $\text{OPT}(\sigma)$ given in Theorem 2 is at least p. If $\lambda = p-1$, this is obvious because $k + 2(\lambda - k + 1) > p$. If $\lambda < p-1$, then $f(\lambda, c^*_\lambda) = g(\lambda, c^*_\lambda)$, where $g(\lambda, c^*_\lambda) \geq p$. The upper bound on $\text{OPT}(\sigma)$ given in Theorem 2 is equal to $f(\lambda, c^*_\lambda) + 2(\lambda - k + 1) = g(\lambda, c^*_\lambda) + 2(\lambda - k + 1)$ and thus at least p.

In the following we assume that *GRS* generates at least one phase in lines 2–9. By Lemma 6 there exists an algorithm ALG that can serve the phases such that page faults occur only on the long-distance requests and at the end of the last phase the last k distinct pages referenced are in fast memory. Hence all the distance-l requests, where $0 \leq l \leq k-1$, can be served without any page faults. If after the service of these requests there still exist new pages, then all the long-distance requests must have referenced new pages. In this case $\text{OPT}(\sigma) = p$ and, as argued in the last paragraph, the theorem holds.

In the remainder of this proof we concentrate on the case that after the service of the distance-l requests, $0 \leq l \leq k-1$, all the p distinct pages have been referenced. We have to upper bound the number of long-distance requests, which will give us an upper bound on the number of page faults. Suppose that r phases $P(1), \ldots, P(r)$ have been generated by *GRS* in σ. The first $r-1$ phases are complete. Assume that phase $P(i)$, $1 \leq i \leq r-1$, consists of l^*_i requests, where l^*_i is the value determined for this phase in lines 3–7 of *GRS*. Let l'_i be the smallest $l \geq k$ such that $c_l > 0$ at the beginning of the phase. There holds $l'_i \leq l^*_i$. Algorithm ALG can serve $P(1), \ldots, P(r-1)$ so that at most $l^*_i - k + 1$ page faults are incurred on the long-distance requests in $P(i)$, for $i = 1, \ldots, r-1$. For any such phase $P(i)$ we charge a service cost of $l'_i - k + 1$, which is potentially smaller than $l^*_i - k + 1$, to the $k-1$ short-distance requests. Each such request is assigned a cost of $(l'_i - k + 1)/(k-1)$. Observe that each such request is a distance-l request with $l'_i \leq l$. Hence a short-distance request that is a distance-l request carries a cost of at most $(l-k+1)/(k-1)$.

Lemmas 7 and 8 ensure that, for any distance-l request that is issued as short-distance request, there holds $l \leq \lambda$. Moreover, there exist at most c^*_λ distance-λ requests that are short-distance requests. We obtain

$$\begin{aligned}\text{OPT}(\sigma) &\leq k + \sum_{i=1}^{r}(l^*_i - k + 1) = k + \sum_{i=1}^{r-1}(l'_i - k + 1) + \sum_{i=1}^{r-1}(l^*_i - l'_i) + l^*_r - k + 1 \\ &\leq k + \sum_{l=k}^{\lambda-1} c_l \frac{l-k+1}{k-1} + c^*_\lambda \frac{\lambda - k + 1}{k-1} + \sum_{i=1}^{r-1}(l^*_i - l'_i) + l^*_r - k + 1. \qquad (5)\end{aligned}$$

The last inequality follows from our charging scheme that assigns service cost for the long-distance requests to the short-distance requests. We next evaluate $\sum_{i=1}^{r-1}(l_i^* - l_i')$. We argue that for any $i = 1, \ldots, r-1$ there holds $l_i^* \leq l_{i+1}'$. Phase $P(i)$ is complete, i.e. l_i^* was set in line 7 of *GRS*, see Lemma 4. At the beginning of the phase, by the choice of l_i^*, there holds $\sum_{l=k}^{l_i^*} c_l \geq k-1$ but $\sum_{l=k}^{l_i^*-1} c_l < k-1$. Hence at the end of the phase $c_l = 0$, for any l with $k \leq l < l_i^*$.

We obtain $\sum_{i=1}^{r-1}(l_i^* - l_i') \leq l_{r-1}^* - l_1' < \lambda - k + 1$, where the last inequality follows from the facts that $l_{r-1}^* \leq \lambda$, cf. Lemma 8, and $l_1' \geq k$. Also, by Lemma 8, $l_r^* \leq \lambda$. Using these inequalities in (5), we obtain the desired upper bound on $\text{OPT}(\sigma)$. □

3 The Competitiveness of LRU

We present upper and lower bounds on the competitive ratio $R_{\text{LRU}}(\mathcal{C})$, for any $\mathcal{C}$. While the bounds involve a number of terms, we stress that they are nearly tight, up to an additive constant of $2(\lambda - k + 1)$ in the denominator of the ratios. Of course, one could simplify the expressions at the expense of weakening the bounds. After stating the corollary we show that our expressions for $R_{\text{LRU}}(\mathcal{C})$ range between 1 and k.

Corollary 1 *Let $\mathcal{C} = (c_0, \ldots, c_{p-1})$ be an arbitrary characteristic vector. Then*

$$R_{\text{LRU}}(\mathcal{C}) \leq \frac{p + \sum_{l=k}^{p-1} c_l}{\max\left\{p, k + \sum_{l=k}^{\lambda-1} c_l \frac{l-k+1}{k-1} + c_\lambda^* \frac{\lambda-k+1}{k-1}\right\}} \tag{6}$$

and

$$R_{\text{LRU}}(\mathcal{C}) \geq \frac{p + \sum_{l=k}^{p-1} c_l}{k + \sum_{l=k}^{\lambda-1} c_l \frac{l-k+1}{k-1} + c_\lambda^* \frac{\lambda-k+1}{k-1} + 2(\lambda - k + 1)}.$$

Proof For any σ, $\text{LRU}(\sigma) = p + \sum_{l=k}^{p-1} c_l$. This fact was already observed by Panagiotou and Souza [20] and also explained in the introduction of this paper. The corollary then follows from Theorems 1 and 2. □

We argue that the upper bound in (6) can be constant, and as low as 1, in particular when given vectors $\mathcal{C}$ modeling request sequences with a high degree of locality of reference. First consider the very simple case that $\mathcal{C} = (c_0, \ldots, c_{k-1}, 0, \ldots, 0)$. The ratio in (6) is equal to 1. A more interesting case is the scenario in which $\mathcal{C}$ has a small number of positive entries c_l with $l \geq k$. In the benchmark library we used there exist traces with this property, see Fig. 4 in Sect. 5. In order to keep the calculations simple we assume that there is a single positive entry c_l with $l \geq k$. W.l.o.g. $c_{p-1} > 0$, i.e. $\mathcal{C} = (c_0, \ldots, c_{k-1}, 0, \ldots, 0, c_{p-1})$. The entries $c_0, \ldots, c_{k-1}$ may take arbitrary values as they are irrelevant for LRU's and OPT's cost. If $f(p-1, c_{p-1}) \leq g(p-1, c_{p-1})$, then $c_{p-1} \leq k-1$ and the ratio in (6) is upper bounded by 2. So assume $f(p-1, c_{p-1}) > g(p-1, c_{p-1})$, in which case $\lambda = p-1$ and c_λ^* satisfies $k + c_\lambda^*(p-k)/(k-1) = p + c_{p-1} - c_\lambda^*$. This implies $c_\lambda^* \geq c_{p-1}(k-1)/(p-1)$. We obtain that the ratio in (6) is upper bounded by $(p + c_{p-1})/(k + c_{p-1}(p-k)/(p-1))$.

For increasing c_{p-1} the last ratio approaches $\frac{p-1}{p-k}$. If $p = k + 1$, then the latter expression is equal to k, which is consistent with the fact that LRU is k-competitive on sequences in which a total of $k + 1$ distinct pages are referenced. If $p = rk$, for some constant $r > 1$, then $\frac{p-1}{p-k}$ is smaller than $\frac{r}{r-1}$, i.e. we obtain constant competitive ratios if r is not too close to 1.

Finally, the upper bound for $R_{\text{LRU}}(\mathcal{C})$ is not greater than k: First assume that, for the given $\mathcal{C}$, there holds $f(p-1, c_{p-1}) \leq g(p-1, c_{p-1})$. In this case $k+\sum_{l=k}^{p-1} c_l \frac{l-k+1}{k-1} \leq p$, which implies $\sum_{l=k}^{p-1} c_l \leq (k-1)p$. Thus the numerator in (6) is upper bounded by kp. On the other hand, if $f(p-1, c_{p-1}) > g(p-1, c_{p-1})$, then $f(\lambda, c_\lambda^*) = g(\lambda, c_\lambda^*)$. In this case the numerator in (6) is

$$\begin{aligned} p + \sum_{l=k}^{p-1} c_l &= p + \sum_{l=k}^{\lambda-1} c_l + c_\lambda^* + (c_\lambda - c_\lambda^*) + \sum_{l=\lambda+1}^{p-1} c_l = \sum_{l=k}^{\lambda-1} c_l + c_\lambda^* + g(\lambda, c_{\lambda^*}) \\ &= \sum_{l=k}^{\lambda-1} c_l + c_\lambda^* + f(\lambda, c_\lambda^*) \; = \; k + \sum_{l=k}^{\lambda-1} c_l \frac{l}{k-1} + c_\lambda^* \frac{\lambda}{k-1}. \end{aligned}$$

The last expression is at most k times the denominator in (6) because $l/(l-k+1) \leq k$, for any $l \geq k$.

4 Separating LRU from FIFO and FWF

We compare LRU to FIFO and FWF and start with an analysis of FIFO. First we present Lemma 9 below, which specifies request sequences which OPT can serve with low cost. It is essential for all the results developed in this section. More precisely, Lemma 9 states that, for any characteristic vector, among the request sequences that OPT can serve with the smallest number of page faults, there exists one in which the distance-l requests with $l \leq k-1$ occur at the end of the sequence. We then show that, on such a sequence σ^*, FIFO incurs at least as many faults as LRU. This establishes Theorem 3, stating that the competitiveness of FIFO is at least as high as that of LRU. Furthermore, given σ^*, we can construct a nemesis sequence on which FIFO incurs strictly more faults than LRU. The main idea is to rearrange the suffix of distance-l requests with $l < k-1$ and some distance-l requests with $l \geq k$ and build a series of phases causing a high cost for FIFO. This is made precise in Theorem 4 that separates the performance of FIFO from that of LRU. Thereafter we show similar results for FWF.

Lemma 9 *Let $\mathcal{C} = (c_0, \ldots, c_{p-1})$ be an arbitrary characteristic vector. Consider the request sequences defined by $\mathcal{C}$ for which* OPT *incurs the smallest number of page faults. Among these sequences there exists one in which all distance-l requests with $0 \leq l \leq k-1$ occur at the end of the sequence.*

Proof Given an arbitrary request sequence characterized by $\mathcal{C}$ we perform two transformations. First, we repeatedly remove the distance-l requests with $0 \leq l \leq k-1$ so that the resulting sequence is characterized by $\mathcal{C}_0 = (0, \ldots, 0, c_k, \ldots, c_{p-1})$. Then, for $l = 0, \ldots, k-1$, we append c_l distance-l requests at the end of the sequence.

Neither of these transformations increases the number of page faults incurred by OPT. This proves the lemma.

Transformation 1 Consider an arbitrary request sequence σ characterized by $\mathcal{C} = (c_0, \ldots, c_{p-1})$. Let $\sigma(t)$ be the first request in σ that is a distance-l' request, for some l' with $0 \leq l' \leq k-1$. We modify σ so that $\sigma(t)$ is removed and, for all other distance-l requests in σ, the value of l does not change. Hence the resulting sequence σ' is characterized by a vector that differs from $\mathcal{C}$ only in that the l'-th component is equal to $c_{l'} - 1$. As we will see, the optimum service cost of σ' is not higher than that of σ. By repeating these modifications, for a total of $\sum_{l=0}^{k-1} c_l$ times, we obtain a sequence characterized by $\mathcal{C}_0 = (0, \ldots, 0, c_k, \ldots, c_{p-1})$ whose optimum service cost is not higher than that of σ.

Again, let σ be the original request sequence and $\sigma(t)$ be the first request in σ that forms a distance-l' request with $0 \leq l' \leq k-1$. Let $x_0 = \sigma(t)$ be the referenced page and $\sigma(t')$ with $t' < t$ be the most recent request to x_0. If $l' = 0$, then σ' is obtained from σ by simply deleting request $\sigma(t)$. This preserves the distances l in all remaining distance-l requests and σ' can be served in the same way as σ. In this case we are done.

In the following we concentrate on the case $l' > 0$. Since $\sigma(t)$ is the first distance-l request with $0 \leq l \leq k-1$, the pages requested by $\sigma(t'+1), \ldots, \sigma(t-1)$ are pairwise distinct and hence $t - 1 - t' = l'$. For $i = 1, \ldots, l'$, let x_i be the page requested by $\sigma(t'+i)$. Thus the subsequence $\sigma(t'), \ldots, \sigma(t)$ is equal to $x_0, x_1, \ldots, x_{l'}, x_0$. Also note that x_0 is different from the pages $x_1, \ldots, x_{l'}$. Now the sequence σ' is obtained from σ by deleting $\sigma(t)$ and renaming requests $\sigma(s)$ with $s > t$ that are made to pages in $\{x_0, \ldots, x_{l'}\}$ in a cyclic fashion. More specifically, requests to pages x_i are replaced by requests to x_{i-1}, where $0 < i \leq l'$, and requests to x_0 are replaced by $x_{l'}$. Formally, the first $t-1$ requests in σ' are identical to those in σ. Request $\sigma(t)$ does not occur in σ. Consider any $s > t$. If $\sigma(s)$ references x_i, $0 \leq i \leq l'$, then the corresponding request $\sigma'(s-1)$ references $x_{(i-1) \bmod (l'+1)}$. Finally, if $\sigma(s)$ references a page different from x_i, for all $0 \leq i \leq l'$, then $\sigma'(s-1)$ is identical to $\sigma(s)$.

We prove that σ' is a request sequence characterized by a vector differing from $\mathcal{C}$ only in that entry $c_{l'}$ is replaced by $c_{l'} - 1$. Recall that σ and σ' are identical on the first $t-1$ requests. Thus any distance-l request in this prefix of σ remains a distance-l request in σ'. Therefore, it suffices to consider an arbitrary request $\sigma(s)$ with $s > t$. We show that if $\sigma(s)$ is a distance-l request, $0 \leq l \leq p-1$, then the corresponding $\sigma'(s-1)$ is also a distance-l request. We focus on the number of distinct pages from $\{x_0, \ldots, x_{l'}\}$ referenced between $\sigma(s)$ (resp. $\sigma'(s-1)$) and the most recent request to the page accessed by $\sigma(s)$ (resp. $\sigma'(s-1)$). This is sufficient because requests to other pages do not change in the sequence modification described above.

We first study the case that the page requested by $\sigma(s)$ was last referenced during $\sigma(t'), \ldots, \sigma(t)$. In this case $\sigma(s) = x_i$, for some $0 \leq i \leq l'$. First assume that $i = 0$, i.e. $\sigma(s) = x_0$ and the most recent reference to x_0 is $\sigma(t)$. Moreover $\sigma'(s-1) = x_{l'}$. If no pages from $\{x_0, \ldots, x_{l'}\}$ are requested in the subsequence $\sigma(t+1), \ldots, \sigma(s-1)$, then by the above argument we are done. So let $x_{i_1}, \ldots, x_{i_m}$ with $i_1 < \ldots < i_m$ be the pages from $\{x_0, \ldots, x_{l'}\}$ requested in $\sigma(t+1), \ldots, \sigma(s-1)$. There holds $i_1 \geq 1$ because the most recent request to x_0 is $\sigma(t)$. In σ' pages $x_{i_1-1}, \ldots, x_{i_m-1}$ are referenced in the subsequence starting after $\sigma'(t-1) = x_{l'}$ and ending before $\sigma'(s-1)$.

Since $i_m - 1 < l'$, the number of distinct pages referenced between $\sigma(t)$ and $\sigma(s)$ is the same as the number of distinct pages between $\sigma'(t-1)$ and $\sigma'(s-1)$. Next assume that $0 < i \leq l'$. Here $\sigma(s) = x_i$ and $\sigma'(s-1) = x_{i-1}$. Between $\sigma(t'+i)$ and $\sigma(s)$ first there are requests to pages $x_{i+1}, \dots, x_{l'}$ and x_0. Furthermore, assume that pages $x_{i_1}, \dots, x_{i_m}$ with $i_1 < \cdots < i_m$ from the set $\{x_1, \dots, x_{i-1}\}$ are referenced in this subsequence. Note that after request $\sigma(t)$, a reference to a page from $\{x_0, x_{i+1}, \dots, x_{l'}\}$ turns into a page from $\{x_i, \dots, x_{l'}\}$ in σ'. Thus between $\sigma'(t'+i-1) = x_{i-1}$ and $\sigma'(s-1)$ we find pages $x_i, \dots, x_{l'}$ and $x_{i_1-1}, \dots, x_{i_m-1}$ when focusing on the set $\{x_0, \dots, x_{l'}\}$. Hence the total number of distinct pages referenced remains the same.

We next address the case that the page requested by $\sigma(s)$ has been referenced last by some $\sigma(s')$ with $t < s' < s$. In this case the number of distinct pages referenced in the subsequence $\sigma(s'+1), \dots, \sigma(s-1)$ is the same as in $\sigma'(s'), \dots, \sigma'(s-2)$. This holds true because the pages from $\{x_0, \dots, x_{l'}\}$ were just renamed cyclically. Finally assume that the page y requested by $\sigma(s)$ was last referenced by $\sigma(s')$ with $s' < t'$. In this case $y \neq x_i$, for all i with $0 \leq i \leq l'$. We observe that the subsequences $\sigma(s'+1), \dots, \sigma(s-1)$ and $\sigma'(s'+1), \dots, \sigma'(s-2)$ reference all the pages from $\{x_0, \dots, x_{l'}\}$. Again the number of distinct pages in $\sigma(s'+1), \dots, \sigma(s-1)$ is the same as that in $\sigma'(s'), \dots, \sigma'(s-2)$.

It remains to analyze service cost. When serving σ, on a page fault OPT always evicts a page whose next request is farthest in the future. Hence when processing a request $\sigma(t'+i)$, $0 \leq i \leq l'$, OPT does not evict any of the pages $x_{i+1}, \dots, x_{l'}$ or x_0 from fast memory because $l' \leq k-1$. In particular, $\sigma(t) = x_0$ is not a page fault. Consider the following algorithm ALG that serves σ' in the same way as OPT serves σ, with the following modification: After request $\sigma'(t') = \sigma(t')$, whenever OPT evicts x_i, ALG evicts $x_{(i-1) \bmod (l'+1)}$, for $i = 0, \dots, l'$. Recall that σ' is obtained from σ by replacing occurrences of x_i by $x_{(i-1) \bmod (l'+1)}$ after request $\sigma(t)$. This implies that after $\sigma(t-1) = \sigma'(t-1)$ the total number of page faults incurred by OPT on requests to x_i is equal to the number of page faults incurred by ALG on references to $x_{(i-1) \bmod (l'+1)}$, $0 \leq i \leq l'$. Hence both algorithms have identical service costs because up to request $\sigma(t-1) = \sigma'(t-1)$ they incur the same number of faults.

Transformation 2 Consider the sequence σ^* characterized by $\mathcal{C}_0 = (0, \dots, 0, c_k, \dots, c_{p-1})$. For $l = 0, \dots, k-1$, append c_l distance-l requests at the end of σ^*. We observe that the last k requests in σ^* reference pairwise distinct pages and that each of the newly appended requests is to one of these k pages. Hence the addition of the new requests does not generate extra service cost because when OPT serves the last k requests in σ^*, it can always evict a page not referenced in the remainder of the sequence. □

Theorem 3 *For any $\mathcal{C}$, there holds $R_{\text{FIFO}}(\mathcal{C}) \geq R_{\text{LRU}}(\mathcal{C})$.*

Proof Let $\mathcal{C} = (c_0, \dots, c_{p-1})$ be an arbitrary vector. Among the sequences characterized by $\mathcal{C}$, consider those for which OPT incurs the smallest number of page faults. By Lemma 9 there exists one in which all distance-l requests, $0 \leq l \leq k-1$, are issued at the end of the sequence. Fix a sequence σ^* with this property. There holds $R_{\text{LRU}}(\mathcal{C}) = \text{LRU}(\sigma^*)/\text{OPT}(\sigma^*)$ because on every sequence characterized by $\mathcal{C}$ LRU incurs the same number $p + \sum_{l=k}^{p-1} c_l$ of page faults. In the following we show $\text{FIFO}(\sigma^*) \geq \text{LRU}(\sigma^*)$. This establishes the theorem.

Let σ_1^* be the prefix of σ^* consisting of all distance-l requests, $k \le l \le p-1$, and the requests to new pages. The remaining requests of σ^* are distance-l requests with $0 \le l \le k-1$. We will show that FIFO incurs a page fault on each request of σ_1^*, which consists of $p + \sum_{l=k}^{p-1}$ requests. Hence $\text{FIFO}(\sigma^*) \ge p + \sum_{l=k}^{p-1} c_l = \text{LRU}(\sigma^*)$.

Any $k+1$ consecutive requests in σ_1^* reference distinct pages because, for any distance-l request, there holds $l \ge k$. On each of the first k requests in σ_1^* FIFO incurs a page fault because the initial fast memory is empty. After request $\sigma_1^*(k)$ FIFO has the pages referenced by $\sigma_1^*(1), \ldots, \sigma_1^*(k)$ in fast memory. We show inductively that, for any $t > k$, FIFO has a page fault on $\sigma_1^*(t)$ and after the service of this request the algorithm has the pages accessed by $\sigma_1^*(t-k+1), \ldots, \sigma_1^*(t)$ in its fast memory. So consider any $t > k$. If $t = k+1$, then before $\sigma_1^*(t)$ FIFO has pages $\sigma_1^*(1), \ldots, \sigma_1^*(k) = \sigma_1^*(t-k), \ldots, \sigma_1^*(t-1)$ in fast memory. If $t > k+1$, then by induction hypothesis FIFO has the pages $\sigma_1^*(t-k), \ldots, \sigma_1^*(t-1)$ in fast memory before reference $\sigma^*(t)$. Since any $k+1$ consecutive requests in σ_1^* reference distinct pages, the page requested by $\sigma_1^*(t)$ is not in FIFO's fast memory and a page fault occurs. As any prior request $\sigma_1^*(s)$ with $s < t$ has been a page fault, FIFO will evict the page requested by $\sigma_1^*(t-k)$ when serving $\sigma_1^*(t)$. □

The next theorem sharply separates LRU from FIFO. Observe that, for any $\mathcal{C} = (c_0, \ldots, c_{p-1})$, LRU's competitiveness can be expressed as $R_{\text{LRU}}(\mathcal{C}) = \text{LRU}(\mathcal{C})/\text{OPT}(\mathcal{C})$, where $\text{LRU}(\mathcal{C}) = p + \sum_{l=k}^{p-1} c_l$ is the number of faults incurred by LRU on every input characterized by $\mathcal{C}$ and $\text{OPT}(\mathcal{C})$ denotes the minimum number of page faults required to serve any request sequence defined by $\mathcal{C}$. We use this notation in the following. Theorem 4 presents a lower bound on $R_{\text{FIFO}}(\mathcal{C})$, given $R_{\text{LRU}}(\mathcal{C}) = \text{LRU}(\mathcal{C})/\text{OPT}(\mathcal{C})$, for any $\mathcal{C}$. In that lower bound c depends on the minimum c_l, where $1 \le l \le k-1$, and roughly $\sum_{l=k}^{p-1} c_l$. For increasing c, the competitiveness of FIFO can be made arbitrarily close to $(k-1)/(1-1/k) = k$. In Section 3 we analyzed vectors $\mathcal{C} = (c_0, \ldots, c_{k-1}, 0, \ldots, 0, c_{p-1})$ and showed that LRU's competitiveness is constant, for sufficiently large c_{p-1}, provided that p is not too close to k. Hence, for large $c_1, \ldots, c_{k-1}$ and c_{p-1}, the competitiveness of LRU is a small constant while that of FIFO is close to k. We remark that, in general, c cannot be larger than $\text{LRU}(\mathcal{C})$ but this is sufficient to establish a lower bound of at least $k/2$ on FIFO's competitiveness.

Theorem 4 *Let $\mathcal{C} = (c_0, \ldots, c_{p-1})$ be any vector. Let $c_{\min} = \min_{1 \le l \le k-1} c_l$ and $c = \min\{\lfloor c_{\min}/2 \rfloor, p - k + \sum_{l=k}^{p-1} c_l\}$. Then*

$$R_{\text{FIFO}}(\mathcal{C}) \ge \frac{\text{LRU}(\mathcal{C}) + c(k-1)}{\text{OPT}(\mathcal{C}) + c(1-1/k) + 1}.$$

Proof Among the request sequences characterized by $\mathcal{C}$, let σ^* be one for which OPT incurs the minimum number $\text{OPT}(\mathcal{C})$ of page faults and in which all distance-l requests, $0 \le l \le k-1$, occur at the end of the sequence. Lemma 9 ensures the existence of such a request sequence. Given σ^*, we construct a nemesis sequence σ for FIFO in three steps.

(1) First remove all distance-l requests with $0 \le l \le k-1$ from σ^*. In this truncated sequence remove the last c requests, which are requests to new pages or distance-l

requests with $k \le l \le p-1$. Let σ_1 denote the resulting request sequence. (2) Append to σ_1 a sequence of c phases $P(1), \ldots, P(c)$. Any $P(i)$ consists of two parts. In the first part, for $l = 1, \ldots, k-1$ and in this specific order, a distance-l request is issued. The second part of the phase starts with a request to a new page if less than p distinct pages have been referenced so far. Otherwise it starts with a distance-l request, where $l \ge k$ is an index such that the current request sequence contains less than c_l distance-l requests. Then again, for increasing $l = 1, \ldots, k-1$, a distance-l request is issued. Let σ_2 denote the request sequence obtained after this step. (3) Append the missing distance-l requests, where $0 \le l \le k-1$, at the end of σ_2. Specifically, while there exists an l with $0 \le l \le k-1$ such that the current request sequence contains less than c_l distance-l requests, issue such a request. Let $\sigma_3 = \sigma$ be the resulting request sequence.

We state some properties of the above construction. Sequence σ_1 consists of at least k requests because after the removal of the distance-l requests with $0 \le l \le k-1$ from σ^*, exactly $p + \sum_{l=k}^{p-1} c_l$ requests remain and $c \le p - k + \sum_{l=k}^{p-1} c_l$. In step (2) each $P(i)$ contains two distance-l requests, for any $1 \le l \le k-1$, as well as one request to a new page or a distance-l request with $k \le l \le p-1$. Thus the choice of c, where $c \le \lfloor c_{\min}/2 \rfloor$, as well as the request removals of Step (1) ensure that the construction of $P(1), \ldots, P(c)$ is well-defined. Sequence σ_2 contains all the $p + \sum_{l=k}^{p-1} c_l$ requests to new pages and distance-l requests with $k \le l \le p-1$. Hence the final request sequence $\sigma_3 = \sigma$ is an input characterized by $\mathcal{C}$.

In the following we first prove that $\text{FIFO}(\sigma) \ge \text{LRU}(\mathcal{C}) + c(k-1)$. Consider the prefix of σ^* consisting of the requests to new pages and the distance-l requests with $k \le l \le p-1$. In the proof of Theorem 3 we showed that FIFO incurs a page fault on each request of this prefix sequence. Hence FIFO has a page fault on each request of σ_1, which consists of $p + \sum_{l=k}^{p-1} c_l - c$ requests. We next prove that FIFO incurs k page faults in each $P(i)$, $1 \le i \le c$. This implies $\text{FIFO}(\sigma) \ge p + \sum_{l=k}^{p-1} c_l - c + c \cdot k = \text{LRU}(\mathcal{C}) + c(k-1)$.

As for FIFO's cost in $P(1), \ldots, P(c)$ we show the following statement: In the second part of each $P(i)$, $1 \le i \le c$, FIFO incurs a page fault on each of the k requests; at the end of $P(i)$ FIFO has the pages referenced by these k requests in fast memory. The proof is by induction on i. Consider any $P(i)$ and let $x_0, \ldots, x_{k-1}$ be the pages referenced by the last k requests preceding $P(i)$. We first observe that these pages are pairwise distinct. If $i = 1$, then $x_0, \ldots, x_{k-1}$ are the pages accessed by the last k requests of σ_1. All these requests are references to new pages or distance-l requests with $l \ge k$. Hence the desired property holds. If $i > 1$, then $x_0, \ldots, x_{k-1}$ are the pages referenced by the k requests in the second part of $P(i-1)$. The j-th of these references is a distance-$(j-1)$ request, for $j = 2, \ldots, k$, so that no page can occur twice. This implies that $P(i)$ has the form $x_{k-2}, \ldots, x_0, y, x_0, \ldots, x_{k-2}$; cf. the construction of the phases in Step (2). Here y is the page accessed by the first request in the second part of $P(i)$. As this is a request to a new request page or a distance-l request with $l \ge k$, page y is different from $x_0, \ldots, x_{k-1}$.

We next argue that at the beginning of $P(i)$ FIFO has pages $x_0, \ldots, x_{k-1}$ in fast memory. If $i = 1$, then the k requests before $P(1)$, which reference the sequence $x_0, \ldots, x_{k-1}$, are a suffix of σ_1. Recall that FIFO has a fault on each request in σ_1.

Hence when the fault on x_j occurs, FIFO evicts a page different from $x_0, \ldots, x_{j-1}$, for $j = 1, \ldots, k-1$. Thus $x_0, \ldots, x_{k-1}$ are in fast memory at the beginning of $P(i)$. If $i > 1$, then the property follows from the induction hypothesis. Given this fact about FIFO's fast memory content, the algorithm does not incur any page fault in the first part of $P(i)$. On the request to y FIFO has a page fault and evicts x_0. In the sequel, for $j = 0, \ldots, k-2$, FIFO has a page fault on the reference to x_j and evicts x_{j+1} so that at the end of $P(i)$, pages $y, x_0, \ldots, x_{k-2}$ reside in fast memory.

It remains to analyze OPT's cost on σ. Consider an algorithm ALG that serves σ as follows. On σ_1 it performs the same page replacements as OPT does when serving σ^*, except for the last k requests of σ_1. On these references, whenever there is page fault, ALG evicts a page not accessed during this suffix of k requests. Thus at the beginning of $P(1)$ the pages referenced by the last k requests prior $P(1)$ reside in ALG's fast memory. The algorithm can then serve $P(1)$ and any subsequent phase $P(i)$ so that a page fault occurs only on the first request of the second part of the phase. More specifically, when serving this request, ALG evicts the page referenced last before $P(i)$, which is not needed in the remainder of the phase. Hence at the end of a phase $P(i)$ ALG has the k pages referenced in the second part of $P(i)$ in fast memory. Using this fact for $i = c$, we obtain that all the distance-l requests with $0 \leq l \leq k-1$ issued after $P(c)$ are memory hits. A final observation is that on the first $p + \sum_{l=k}^{p-1} c_l$ requests of the initial sequence σ^*, OPT incurs at least one page fault on any k consecutive requests: After OPT has served any request $\sigma(t)$, it cannot have all the pages accessed by $\sigma(t+1), \ldots, \sigma(t+k)$ in fast memory because $\sigma(t), \ldots, \sigma(t+k)$ reference pairwise distinct pages. Thus on the c requests that are removed from the already truncated sequence in Step (1) OPT incurs at least $\lfloor c/k \rfloor$ page faults. We conclude that $\text{OPT}(\sigma) \leq \text{OPT}(\mathcal{C}) + c - \lfloor c/k \rfloor \leq \text{OPT}(\mathcal{C}) + c(1 - 1/k) + 1$.□

Next we address FWF and develop results corresponding to those for FIFO. In the separation bound of Theorem 6 the vector entries $c_1, \ldots, c_{k-1}$ may be by a factor of 2 smaller compared to those in Theorem 4.

Theorem 5 *For any $\mathcal{C}$, there holds $R_{\text{FWF}}(\mathcal{C}) \geq R_{\text{LRU}}(\mathcal{C})$.*

Proof The proof is very similar to that of Theorem 3. Given $\mathcal{C} = (c_0, \ldots, c_{p-1})$, let σ^* be the request sequence for which OPT incurs the minimum number of page faults, among sequences characterized by $\mathcal{C}$, and in which the distance-l requests with $0 \leq l \leq k-1$ all occur at the end of the sequence. We will show $\text{FWF}(\sigma^*) \geq \text{LRU}(\sigma^*)$. Let σ_1^* be the prefix of σ^* consisting of the requests to new pages and the distance-l requests, where $l \geq k$. It suffices to show that FWF has a page fault on each request of σ_1^*. The first k requests in σ_1^* are references to new pages. Thereafter FWF evicts all pages from fast memory on request $\sigma_1^*(ik+1)$, for any $i \geq 1$, because the referenced page is different from those requested by the k previous ones $\sigma_1^*((i-1)k+1), \ldots, \sigma_1^*(ik)$. Obviously, $\sigma_1^*(ik+1)$ as well as the $k-1$ subsequent requests are page faults. □

Theorem 6 *Let $\mathcal{C} = (c_0, \ldots, c_{p-1})$ be any vector. Let $c_{\min} = \min_{1 \leq l \leq k-1} c_l$ and $c = \min\{c_{\min}, p - k + \sum_{l=k}^{p-1} c_l\}$. Then*

$$R_{\text{FWF}}(\mathcal{C}) \geq \frac{\text{LRU}(\mathcal{C}) + c(k-1)}{\text{OPT}(\mathcal{C}) + c(1 - 1/k) + 1}.$$

Proof of Theorem 6 The basic structure of the proof is very similar to that of Theorem 4. Considering the sequences defined by $\mathcal{C}$, we fix an input σ^* for which OPT incurs the minimum number of page faults and in which the requests to new pages and the distance-l requests, $0 \leq l \leq k-1$, occur at the end of the sequence. We transform σ^* into a nemesis sequence σ for FWF. The transformation is similar to that described in the proof of Theorem 4, except that the construction of the phases in Step (2) is different.

(1) Remove all distance-l requests, $0 \leq l \leq k-1$, from σ^*. Moreover, remove the last c requests from this truncated sequence. Let σ_1 be the resulting sequence. (2) Append phases $P(1), \ldots, P(c)$ to σ_1. More specifically, suppose that at the end of σ_1 FWF has j pages in fast memory, $1 \leq j \leq k$. Then each $P(i)$, $1 \leq i \leq c$, is of the following form. First, for $l = j, \ldots, k-1$, a distance-l request is issued. Then a request to a new page or a distance-l request with $l \geq k$ is placed. Finally, for increasing $l = 1, \ldots, j-1$, a distance-l request is issued. Let σ_2 denote the request sequence obtained after this step. (3) Append the missing distance-l requests with $0 \leq l \leq k-1$ to σ_2 and let $\sigma_3 = \sigma$ be the final request sequence.

We analyze FWF's cost. The proof of Theorem 5 implies that FWF incurs a page fault on each request of σ_1. Let $z_1, \ldots, z_k$ be the pages referenced by the last k requests of σ_1. These pages are pairwise distinct. By assumption FWF has j pages in fast memory at the end of σ_1, where $1 \leq j \leq k$. These must be $z_{k-j+1}, \ldots, z_k$. Thus the first phase $P(1)$ starts with requests to pages $z_{k-j}, \ldots, z_1$ in this order as these are distance-l requests, for $l = j, \ldots, k-1$. Each of these requests is a page fault for FWF. In each phase $P(i)$, $1 \leq i \leq c$, there exists one request that is made to a new page or forms a distance-l request with $l \geq k$. Let y_i denote the page specified by this request in $P(i)$. Consider the first such page y_1. Before the respective request in $P(1)$

Table 1 The files of the test suite

File name	Application	Length	p
`espresso` (Linux)	Circuit simulator	326,938,361	77
`gcc-2.7.2` (Linux)	GNU C/C++ compiler	37,524,334	458
`gnuplot` (Linux)	GNU plotting utility	68,458,509	7718
`grobner` (Linux)	Grobner basis functions	7,787,835	67
`gs3.33` (Linux)	GhostScript	134,371,942	558
`lindsay` (Linux)	Hypercube simulator	123,690,749	521
`p2c` (Linux)	Pascal to C transformer	30,722,431	132
`acroread` (Windows NT)	Acrobat reader	94,794,501	1903
`cc1` (Windows NT)	Compiler core for gcc	263,765,501	716
`compress` (Windows NT)	Compression utility	129,116,176	396
`go` (Windows NT)	AI playing "Go"	106,790,719	267
`netscape` (Windows NT)	Netscape web browser	22,077,106	1037
`powerpoint` (Windows NT)	MS Powerpoint	37,384,786	1000
`winword` (Windows NT)	MS Word	114,359,299	983
`vortex` (Windows NT)	Database program	543,247,591	4275

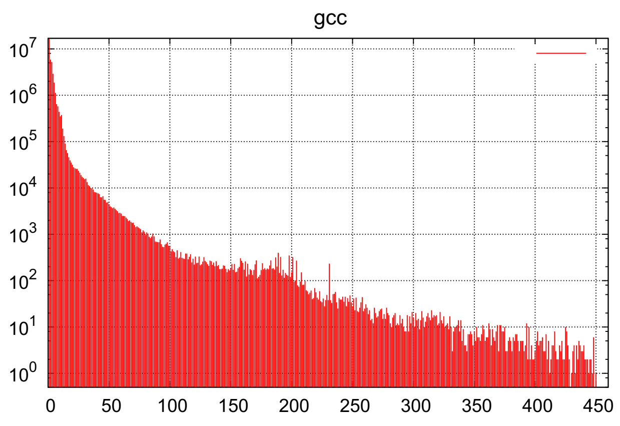
the page sequence $z_{k-j+1}, \ldots, z_k, z_{k-j}, \ldots, z_1$ is requested and all of these pages reside in FWF's fast memory when the request to y_1 has to be served. Again, the pages $z_{k-j+1}, \ldots, z_k, z_{k-j}, \ldots, z_1$ are pairwise distinct and y_1 differs from all of them. Thus on the request to y_1 FWF flushes its fast memory. Consider a pair of requests to y_i and y_{i+1} issued in $P(i)$ and $P(i+1)$, respectively, where $1 \le i < c$. Let $x_1, \ldots, x_{k-1}$ be the pages referenced by the $k-1$ requests between y_i and y_{i+1} in $P(i)$ and $P(i+1)$. These pages are pairwise distinct and differ from y_i as they are distance-l requests, for $l = 1, \ldots, k-1$. Moreover, y_{i+1} is different from these pages and also different from y_i. Thus, if FWF flushes its fast memory on the request to y_i, all requests to $x_1, \ldots, x_{k-1}$ are page faults and FWF again evicts all pages from fast memory on the request to y_{i+1}. As all requests of $P(c)$ issued after y_c are also page faults, the total number of faults incurred by FWF is $\text{FWF}(\sigma) \ge p + \sum_{l=k}^{p-1} c_l - c + c \cdot k = \text{LRU}(\mathcal{C}) + c(k-1)$.

We finally evaluate OPT's cost on σ. Let again $z_1, \ldots, z_k$ denote the k pages referenced at the end of σ_1. Consider the algorithm ALG that serves σ_1 as OPT serves this prefix of the original sequence σ^* with the following exception: On the last k requests of σ_1, whenever there is a fault on a request to a page z_i, $1 \le i \le k$, evict a page different from these k pages. Hence at the end of σ_1, ALG has the pages $z_1, \ldots, z_k$ in fast memory and the first $k-j$ requests of $P(1)$ can be served without any page fault. Algorithm ALG can then serve the c phases so that a page fault only occurs on the requests to y_i, $1 \le i \le c$. This holds true because if prior to the request to y_i in $P(i)$ the subsequence of the last $k-1$ references is $x_1, \ldots, x_{k-1}$, then the $k-1$ requests after y_i access $x_{k-1}, \ldots, x_1$ in this order. This implies that within the phases, apart from the pages $y_1, \ldots, y_c$, only pages $z_1, \ldots, z_{k-j}, z_{k-j+2}, \ldots, z_k$ are referenced, which ALG always keeps in fast memory. Hence on the request to y_1, page z_{k-j+1} is evicted. On the fault to y_{i+1} page y_i is deleted, $1 \le i < c$. This also implies that all the final distance-l requests, $0 \le l \le k-1$, can be served without any page fault. In conclusion, as in the proof of Theorem 4, $\text{OPT}(\sigma) \le \text{OPT}(\mathcal{C}) + c - \lfloor c/k \rfloor \le \text{OPT}(\mathcal{C}) + c(1 - 1/k) + 1$. □

5 Experiments

We report on an experimental study we have performed with reference traces from the benchmark library [15]. This test suite was specifically designed to evaluate the performance of memory systems. Details can be found in the SIGMETRICS paper [16]. The trace library consists of 15 files that contain sequential logs of memory locations used by various programs. Standard applications from the Linux and the Windows NT operating systems were executed. Table 1 shows a list of the files.

In a first step, for each trace, we have extracted the underlying characteristic vector, simply by counting the number of distance-l requests, for each $l \ge 0$, in it. Uniformly over all files, in each resulting vector, the entries basically form a non-increasing sequence, with a large majority of the requests representing distance-l requests, for small values of l. Once again this confirms the fact that real-world sequences exhibit a high degree of locality. Figures 3 and 4 depict the extracted characteristic vectors for four representative files, namely `gcc`, `netscape`, `lindsay` and `winword`. Due to space considerations we do not show the results for all the 15 files. Note that the values of the vector entries are shown in a logarithmic scale. Figure 4 gives the vector



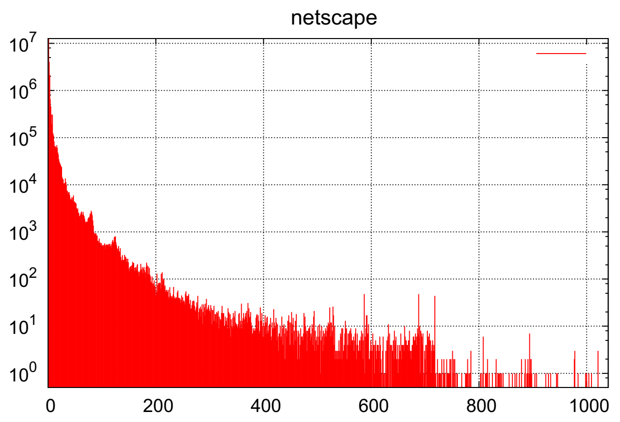
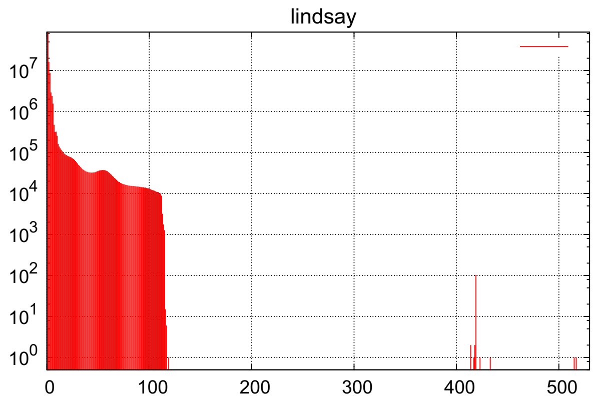


Fig. 3 The underlying characteristic vectors of `gcc` and `netscape`. The horizontal axis represents the values l, $0 \leq l \leq p-1$. The vertical axis shows the range of c_l-values

of `lindsay` that contains a few positive entries c_l, for large l, after a long preceding subsequence of zero-valued entries. This pattern was refered to in the calculations of Section 3.

In a second step we have compared, for each trace/request sequence σ, the optimum number of page faults $\text{OPT}(\sigma)$ to our bounds given in Theorems 1 and 2. For all the traces the difference is small. Hence our lower bound on $\text{OPT}(\sigma)$ in Theorem 1 quite accurately predicts the optimum cost. Furthermore, the additive expression of $2(\lambda - k + 1)$ in the bound of Theorem 2 is not critical. We point out that our lower bound on $\text{OPT}(\sigma)$ cannot match the true service cost because the bound holds for *every* request sequence specified by a characteristic vector $\mathcal{C}$. The given trace σ, in general, is not a sequence that can be served with the minimum number of faults, among inputs characterized by the underlying $\mathcal{C}$. Additionally, we have evaluated the lower bound on $\text{OPT}(\sigma)$ given by Panagiotou and Souza [20], cf. inequality (1). In the experiments our new lower bound developed in this paper is always significantly better. The gap



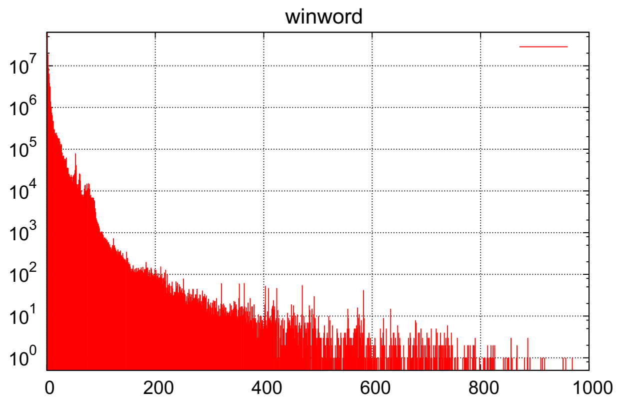


Fig. 4 The underlying characteristic vectors of `lindsay` and `winword`

increases as the fast memory size k increases. It turns out that, for larger values of k, the lower bound by Panagiotou and Souza is quite weak. One could slightly improve it by considering the maximum of p and the expression of (1). However, this only resolves cases where there is no need for a sophisticated lower bound. Figures 5 and 6 show the plots for the four sample traces. Even for small values of k, our new lower bound improves upon that of Panagiotou and Souza by at least 25– 100%.

Finally we have compared, for each trace σ and underlying $\mathcal{C}$, the upper and lower bounds on LRU's competitiveness $R_{\text{LRU}}(\mathcal{C})$ (see Corollary 1) to the experimentally observed competitiveness for σ. For all the files, our bounds give small constant competitive factors that are typically in the range $[1, 4]$. Our upper bound on $R_{\text{LRU}}(\mathcal{C})$ is usually at most 2.5 times the experimentally observed competitiveness. Again, our bounds cannot exactly match the latter competitiveness since $R_{\text{LRU}}(\mathcal{C})$ is the maximum ratio of $\text{LRU}(\sigma)/\text{OPT}(\sigma)$, considering σ characterized by $\mathcal{C}$. A trace at hand, in general, is not such a worst-case sequence. Interestingly, for varying k, our bounds exhibit the same overall behavior as the experimentally observed competitiveness. Thus they correctly describe the general qualitative behavior of $R_{\text{LRU}}(\mathcal{C})$, depending on k. We refer the reader to Figs. 7 and 8, which depict again the results for our four selected

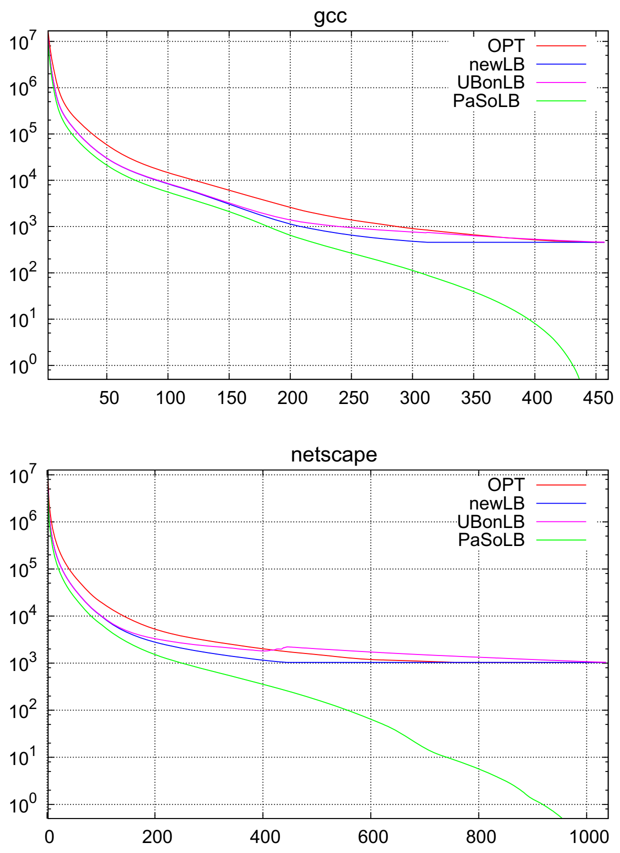
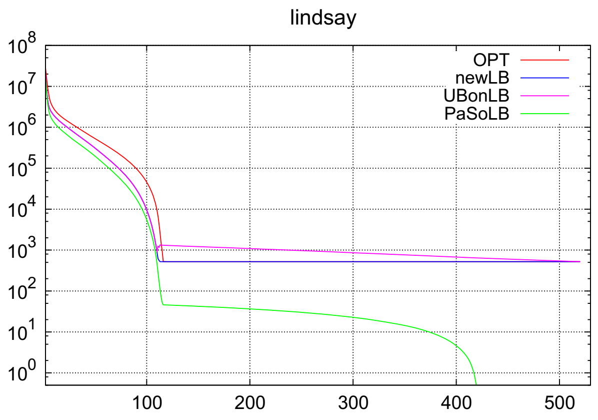


Fig. 5 The plots show the results on OPT's cost, for traces `gcc` and `netscape`. The values along the horizontal axis are the values of k, ranging between 2 and p, the total number of pages in the considered trace. The true optimum number of page faults is plotted in red. The blue and the pink lines depict the bounds of Theorems 1 and 2, respectively. The lower bound by Panagiotou and Souza, see (1), is shown in green (Color figure online)

samples traces. An exception in the trace library is the file `lindsay`. For a few values of k, the upper and lower bounds on LRU's competitiveness is as high as 40. For these k, there are only a few positive vector entries c_l, with $l \geq k$, in the characteristic vector $\mathcal{C}$. These outliers cause high competitive ratios in the theoretical bounds. Indeed there exist sequences characterized by $\mathcal{C}$ for which the performance factors depend linearly on k.



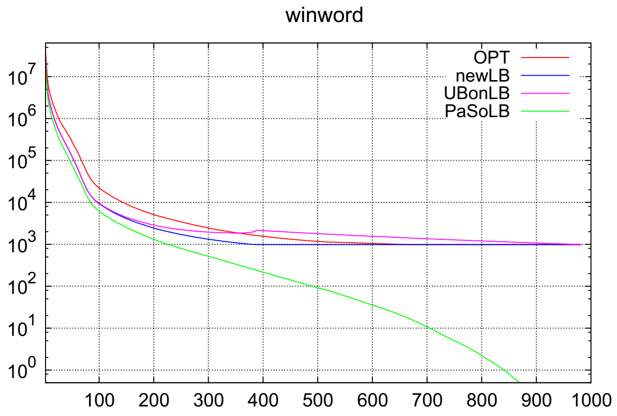


Fig. 6 The results on OPT's cost, for traces `lindsay` and `winword`

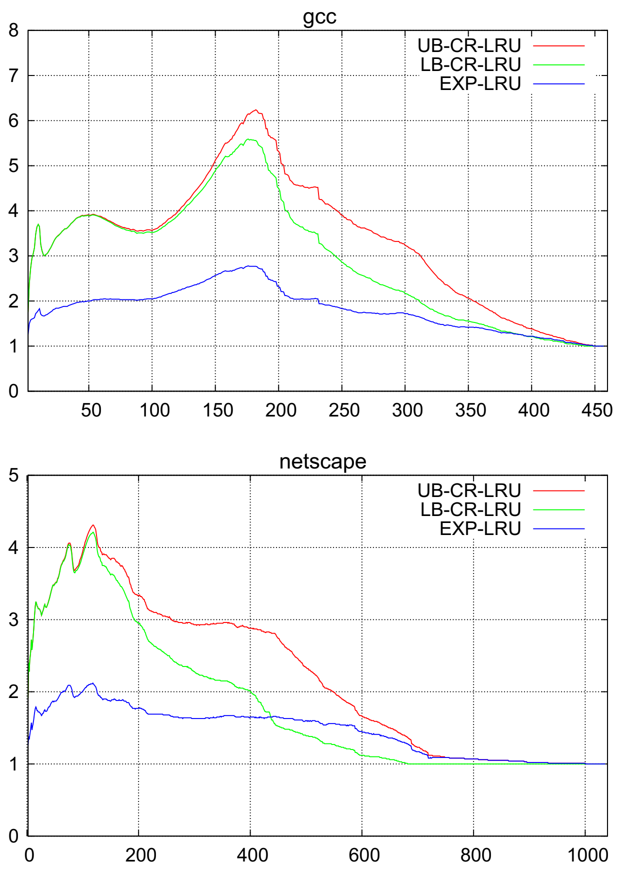
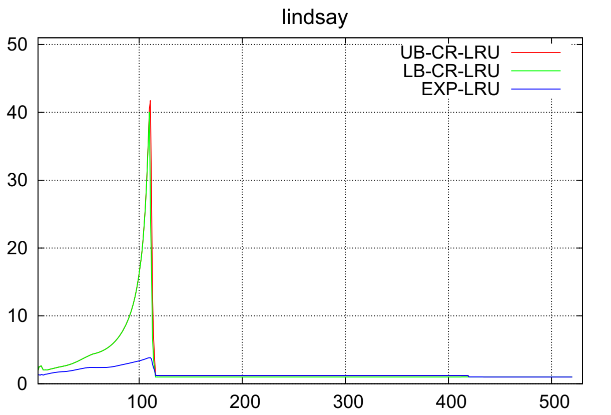


Fig. 7 The plots depict the results on LRU's competitiveness, for traces `gcc` and `netscape`. Again, the horizontal axis represents the possible values of k. The experimentally observed competitiveness of LRU is shown in blue. The upper and lower bounds on $R_{\text{LRU}}(\mathcal{C})$ are plotted in red and green, respectively (Color figure online)



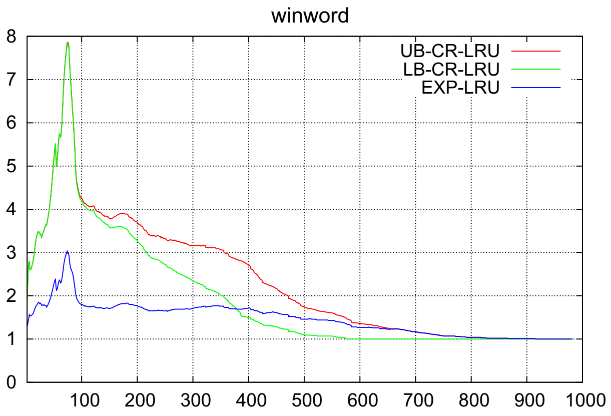


Fig. 8 The results on LRU's competitiveness, for traces `lindsay` and `winword`

Appendix

Proof of Proposition 1 Let $d = 1 + \frac{k-1}{k} - \frac{k-1}{p-1}$, which is the denominator in the first expression on the right-hand side of (1). Since $p > k$ there holds $d \geq 1$. If $f(p-1, c_{p-1}) \leq g(p-1, c_{p-1})$, then $\lambda = p - 1$ and $c^*_\lambda = c_{p-1}$. In this case the proof is obvious. Suppose that $f(p-1, c_{p-1}) > g(p-1, c_{p-1})$. Our goal is to prove that $\frac{1}{d} \sum_{l=k}^{p-1} c_l \frac{l-k+1}{l}$ is smaller than $f(\lambda, c^*_\lambda) = g(\lambda, c^*_\lambda)$. There holds

$$
\begin{aligned}
&\frac{1}{d}\sum_{l=k}^{p-1} c_l \frac{l-k+1}{l}\\
&= \frac{1}{d}\left(\sum_{l=k}^{\lambda-1} c_l \frac{l-k+1}{l} + c^*_\lambda\left(\frac{\lambda-k+1}{\lambda}\right) + (c_\lambda - c^*_\lambda)\left(\frac{\lambda-k+1}{\lambda}\right) + \sum_{l=\lambda+1}^{p-1} c_l \frac{l-k+1}{l}\right)\\
&\le \frac{1}{d}\left(\sum_{l=k}^{\lambda-1} c_l \frac{l-k+1}{k} + c^*_\lambda\left(\frac{\lambda-k+1}{k}\right) + (c_\lambda - c^*_\lambda)(\frac{\lambda-k+1}{\lambda}) + \sum_{l=\lambda+1}^{p-1} c_l \frac{l-k+1}{l}\right)\\
&= \frac{1}{d}\left(\frac{k-1}{k}\left(f(\lambda, c^*_\lambda) - k\right) + (c_\lambda - c^*_\lambda)\left(\frac{\lambda-k+1}{\lambda}\right) + \sum_{l=\lambda+1}^{p-1} c_l \frac{l-k+1}{l}\right)\\
&= \frac{1}{d}\left(\frac{k-1}{k}\left(g(\lambda, c^*_\lambda) - k\right) + (c_\lambda - c^*_\lambda)\left(\frac{\lambda-k+1}{\lambda}\right) + \sum_{l=\lambda+1}^{p-1} c_l \frac{l-k+1}{l}\right)\\
&= \frac{1}{d}\left(\frac{k-1}{k}\left((p-k) + c_\lambda - c^*_\lambda + \sum_{l=\lambda+1}^{p-1} c_l\right) + (c_\lambda - c^*_\lambda)\left(\frac{\lambda-k+1}{\lambda}\right) + \sum_{l=\lambda+1}^{p-1} c_l \frac{l-k+1}{l}\right)\\
&= \frac{1}{d}\left(\frac{k-1}{k}(p-k) + (c_\lambda - c^*_\lambda)\left(\frac{\lambda-k+1}{\lambda} + \frac{k-1}{k}\right) + \sum_{l=\lambda+1}^{p-1} c_l(\frac{l-k+1}{l} + \frac{k-1}{k})\right)\\
&< p + \frac{1}{d}\left((c_\lambda - c^*_\lambda)\left(\frac{\lambda-k+1}{\lambda} + \frac{k-1}{k}\right) + \sum_{l=\lambda+1}^{p-1} c_l(\frac{l-k+1}{l} + \frac{k-1}{k})\right) \qquad (7)\\
&\le p + (c_\lambda - c^*_\lambda) + \sum_{l=\lambda+1}^{p-1} c_l = g(\lambda, c^*_\lambda), \qquad (8)
\end{aligned}
$$

where (7) follows from the fact that $d \ge 1$ and $k \ge 2$. Finally, (8) follows from $\frac{1}{d}(\frac{l-k+1}{l} + \frac{k-1}{k}) \le 1$, for $l \le p-1$. □

References

1. Albers, S., Favrholdt, L.M., Giel, O.: On paging with locality of reference. J Comput Syst Sci **70**(2), 145–175 (2005)
2. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: Proceedings of 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 229–237 (2007)
3. Angelopoulos, S., Schweitzer, P.: Paging and list update under bijective analysis. J. ACM **60**(2), 7 (2013)
4. Becchetti, L.: Modeling locality: a probabilistic analysis of LRU and FWF. In: Proceedings of 12th Annual European Symposium of Algorithms (ESA), Springer LNCS, vol. 3221, pp. 98–109 (2004)
5. Belady, L.A.: A study of replacement algorithms for virtual storage computers. IBM Syst. J. **5**, 78–101 (1966)
6. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica **11**(1), 73–91 (1994)
7. Boyar, J., Favrholdt, L.M., Larsen, K.S.: The relative worst-order ratio applied to paging. J. Comput. Syst. Sci. **73**(5), 818–843 (2007)

8. Boyar, J., Gupta, S., Larsen, K.S.: Access graphs results for LRU versus FIFO under relative worst order analysis. In: Proceedings of 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), Springer LNCS, vol. 7357, pp. 328–339 (2012)
9. Boyar, J., Gupta, S., Larsen, K.S.: Relative interval analysis of paging algorithms on access graphs. In: Proceedings of 13th International Symposium on Algorithms and Data Structures (WADS), Springer LNCS, vol. 8037, pp. 195–206 (2013)
10. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. J. Comput. Syst. Sci. **50**, 244–258 (1995)
11. Dorrigiv, R., Ehmsen, M.R., López-Ortiz, A.: Parameterized analysis of paging and list update algorithms. Algorithmica **71**(2), 330–353 (2015)
12. Dorrigiv, R., López-Ortiz, A.: On developing new models, with paging as a case study. SIGACT News **40**(4), 98–123 (2009)
13. Chrobak, M., Noga, J.: LRU is better than FIFO. Algorithmica **23**(2), 180–185 (1999)
14. Dorrigiv, R., López-Ortiz, A., Munro, J.I.: On the relative dominance of paging algorithms. Theor. Comput. Sci. **410**(38–40), 3694–3701 (2009)
15. Kaplan, S.: Trace reduction for virtual memory simulation. Benchmark library. https://www3.amherst.edu/~sfkaplan/research/trace-reduction/index.html
16. Kaplan, S.F., Smaragdakis, Y., Wilson, P.R.: Trace reduction for virtual memory simulations. In: Proceedings of International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), pp. 47–58 (1999)
17. Karlin, A., Phillips, S., Raghavan, P.: Markov paging. SIAM J. Comput. **30**(3), 906–922 (2000)
18. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. SIAM J. Comput. **30**(1), 300–317 (2000)
19. Irani, S., Karlin, A.R., Phillips, S.: Strongly competitive algorithms for paging with locality of reference. SIAM J. Comput. **25**, 477–497 (1996)
20. Panagiotou, K., Souza, A.: On adequate performance measures for paging. In: Proceedings of 38th Annual ACM Symposium on Theory of Computing (STOC), pp. 487–496 (2006)
21. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Commun. ACM **28**, 202–208 (1985)
22. Young, N.E.: The k-server dual and loose competitiveness for paging. Algorithmica **11**, 525–541 (1994)