Parameterized Analysis of Paging and List Update Algorithms

Reza Dorrigiv[1], Martin R. Ehmsen[2], and Alejandro López-Ortiz[1⋆]

[1] Cheriton School of Computer Science, University of Waterloo, Canada, {rdorrigiv, alopez-o}@uwaterloo.ca
[2] Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark, ehmsen@imada.sdu.dk

Abstract. It is well-established that input sequences for paging and list update have locality of reference. In this paper we analyze the performance of algorithms for these problems in terms of the amount of locality in the input sequence. We define a measure for locality that is based on Denning's working set model and express the performance of well known algorithms in term of this parameter. This introduces parameterized-style analysis to online algorithms. The idea is to rather than normalizing the performance of an online algorithm by an (optimal) offline algorithm, we explicitly express the behavior of the algorithm in terms of two more natural parameters: the size of the cache and Denning's working set measure. This technique creates a performance hierarchy of paging algorithms which better reflects their intuitive relative strengths. Also it reflects the intuition that a larger cache leads to a better performance. We obtain similar separation for list update algorithms. Lastly, we show that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the parameterized case, which matches experimental results.

1 Introduction

The competitive ratio, first introduced formally by Sleator and Tarjan [35], has served as a practical measure for the study and classification of online algorithms. An algorithm (assuming a minimization problem) is said to be α-competitive if the cost of serving any specific request sequence never exceeds α times the cost of an optimal *offline* algorithm which knows the entire sequence. The competitive ratio is a relatively simple measure to apply yet powerful enough to quantify, to a large extent, the performance of many online algorithms. Notwithstanding the wide applicability of competitive analysis, it has been observed by numerous researchers (e.g. [9, 11, 29, 38, 15]) that in certain settings the competitive ratio produces results that are too pessimistic or otherwise found wanting. Indeed, the original paper by Sleator and Tarjan discusses the various drawbacks of the competitive ratio in the case of the paging problem and uses resource augmentation to address some of the observed drawbacks.

⋆ Part of this work took place while the third author was on sabbatical at the Max-Planck-Institut für Informatik in Saarbrücken, Germany.

A well known example for the shortcomings of competitive analysis is the paging problem. A paging algorithm mediates between a slower and a faster memory. Assuming a cache of size k, it decides which k memory pages to keep in the cache without the benefit of knowing in advance the *sequence* of upcoming page requests. After receiving the i^{th} page request the online algorithm must decide which page to evict, in the event the request results in a fault and the cache is full. The objective is to design an online algorithms that minimizes the total number of faults. Three well known paging algorithms are Least-Recently-Used (LRU), First-In-First-Out (FIFO), and Flush-When-Full (FWF) [10]. All these paging algorithms have competitive ratio k, which is the best among all deterministic online paging algorithms [10]. On the other hand, experimental studies show that LRU has a performance ratio at most four times the optimal offline [38]. Furthermore, it has been empirically well established that LRU (and/or variants thereof) are, in practice, preferable paging strategies to all other known paging algorithms [34].

Such anomalies have led to the introduction of many alternatives to competitive analysis of online algorithms (see [20] for a comprehensive survey). Some examples are *loose competitiveness* [38, 40], *diffuse adversary* [29, 39], the *Max/Max ratio* [9], the *relative worst order ratio* [15], and the *random order ratio* [28]. None of them fully resolve all the known issues with competitive analysis.

It is well known that input sequences for paging and several other problems show locality of reference. This means that when a page is requested it is more likely to be requested in the near future. Therefore several models for paging with locality of reference have been proposed. In the early days of computing, Denning recognized the locality of reference principle and modeled it using the well known *working set* model [17, 18]. He defined the working set of a process as the set of most recently used pages and addressed thrashing using this model. After the introduction of the working set model, the locality principle has been adopted in operating systems, databases, hardware architectures, compilers, and many other areas. Therefore it holds even more so today. Indeed, [19] states "locality of reference is one of the cornerstones of computer science."

One apparent reason for the drawbacks of competitive analysis of paging is that it does not incorporate the corresponding locality of reference. Several models incorporating locality have been proposed. The *access graph* model by Borodin et al. [12, 26, 16, 22] and its generalization by Karlin et al. [27] model the request sequences as a graph, possibly weighted by probabilistic transitions. Becchetti [8] refined the diffuse adversary model of Koutsoupias and Papadimitriou by considering only probabilistic distributions in which locality of reference is present. Albers, Favrholdt, and Giel [2] introduced a model in which input sequences are classified according to a measure of locality of reference.

Recently, Angelopoulos et al. introduced *Bijective Analysis* and *Average Analysis* [4] which combined with the locality model of Albers et al. [2], shows that LRU is the sole optimal paging algorithm on sequences with locality of reference. This resolved an important disparity between theory and practice of online paging algorithms, namely the superiority in practice of LRU. An analogous re-

sult for list update and MTF is shown in [5] and the separation of LRU was strengthened by Angelopoulos and Schweitzer in [6], earlier this year. These last separation results are based on heavy machinery specifically designed to resolve this singular long-standing question and leave open the question of how to efficiently characterize the full spectrum of performance of the various known paging and list update algorithms. We address this question in this paper.

In this paper we analyze the performance of paging and list update algorithms in terms of a measure of locality of reference. This measure is related to Denning's working set model [17], the locality of reference model of [2], and the working set theorem in the context of the splay trees and other self-organizing data structures [36, 25, 13]. We show that this new model produces the finest separation yet of list update algorithms while also being applicable to paging and other online algorithms. Paging and list update are the best testbeds for developing alternative measures, given our extensive understanding of these problems. We know why competitive analysis fails, what are typical sequences in practice and we can better evaluate whether a new technique indeed overcomes known shortcomings. It is important to note that even though well studied, most of the alternative models for these problems are only partially successful in resolving the issues posed by them and as such these problems are still challenging case studies against which to test a new model.

Summary of Contributions. We apply parameterized analysis to two fundamental online problems, paging and list update (defined in Section 3). We express the performance of well known paging and list update algorithms in terms of a measure of locality of reference. For paging, this leads to better separation than the competitive ratio. Furthermore, in contrast to competitive analysis it reflects the intuition that a larger cache leads to a better performance. We also provide experimental results that justify the applicability of our measure in practice. We obtain bounds on the parameterized performance of several list update algorithms and prove the superiority of MTF. We also apply our measures to randomized list update algorithms and show that, surprisingly, certain randomized algorithms which are superior to MTF in the classical model are not so in the parameterized case. Some of the proofs follow the general outline of standard competitive analysis proofs (e.g., those in [10]), yet in some cases provide finer separation of paging and list update algorithms.

For the paging problem, many studies have been presented that integrate the concept of locality, culminating with the LRU separation results in [4, 6]. These separation results are based on heavy machinery specifically designed to resolve this singular long-standing question. In contrast, the new measure we propose is easier to apply and creates a performance hierarchy of paging and list update algorithms which better reflects their intuitive relative strengths. Several previously observed experimental properties can be readily proven using the new model. This is a strength of the new model in that is effective, that is readily applicable to a variety of algorithms and with meaningful results.

2 Parameterized Analysis of Paging Algorithms

Recall that on a fault (with a full cache), LRU evicts the page that is least recently requested, FIFO evicts the page that is first brought to the cache, FWF empties the cache, Last-In-First-Out (LIFO) evicts the page that is most recently brought to the cache, and Least-Frequently-Used (LFU) evicts the page that has been requested the least. LFU and LIFO do not have a constant competitive ratio [10]. A paging algorithm is called *conservative* if it incurs at most k faults on any sequence that contains at most k distinct pages. A marking algorithm $\mathcal{A}$ works in phases: all the pages in the cache are unmarked at the beginning of each phase. We mark any page just after the first request to it in each phase. When an eviction is necessary, $\mathcal{A}$ should evict an unmarked page. LRU and FIFO are conservative algorithms, while LRU and FWF are marking algorithms.

As stated before input sequences for paging show locality of reference in practice. We want to express the performance of paging algorithms on a sequence in terms of the amount of the locality in that sequence. Therefore we need a measure that assigns a number proportional to the amount of locality in each sequence. None of the previously described models provide a unique numerical value as a measure of locality of reference[3]. We define a quantitative measure for non-locality of paging instances.

Definition 1. *For a sequence σ we define $d_\sigma[i]$ as either $k+1$ if this is the first request to page $\sigma[i]$, or otherwise, the number of distinct pages that are requested since the last request to $\sigma[i]$ (including $\sigma[i]$).*[4] *Now we define $\overline{\lambda}(\sigma)$, the "non-locality" of σ, as $\overline{\lambda}(\sigma) = \frac{1}{|\sigma|}\sum_{1\le i\le|\sigma|} d_\sigma[i]$. We denote the non-locality by $\overline{\lambda}$ if the choice of σ is clear from the context.*

If σ has high locality of reference, the number $d_\sigma[i]$ of distinct pages between two consecutive requests to a page is small for most values of i and thus σ has a low non-locality. Note that while this measure is related to the working set model [17] and the locality model of [2], it differs from both in several aspects. Alberst et al. [2] consider the maximum/average number of distinct pages in all windows of the same size, while we consider the number of distinct pages requested since the last access to each page. Also our analysis does not depend on a concave function f whose identification for a particular application might not be straightforward. Our measure is also closely related to the working set theorem in area of self-organizing data structures [36]. For binary search trees (like splay trees), the working set bound is defined as $\sum_{1\le i\le|\sigma|} \log(d_\sigma[i]+1)$. The logarithm can be explained by the logarithmic bounds on most operations in binary search trees. Thus our measure of locality of reference can be considered as variant of this measure in which we remove the logarithm.

[3] Formally a measure is a function that assigns a numerical non-negative value to an object, assigns the value of zero to the empty set and is additive over disjoint objects.

[4] Asymptotically, and assuming the number of requests is much larger than the number of distinct pages, any constant can replace $k+1$ for the $d_\sigma[i]$ of the first accesses.

	espresso	li	eqntott	compress	tomcatv	ear	sc	swm	gcc
Distinct	3913	3524	9	189	5260	1614	561	3635	2663
$\overline{\lambda}$	193.1	195.2	1.7	2.3	348.3	34.1	5.4	166.7	90.6
Ratio	4.9%	5.5%	19.3%	1.2%	6.6%	2.1%	1.0%	4.6%	3.4%

Table 1. Locality of address traces collected from SPARC processors running the SPEC92 benchmarks.

Experimental Evaluation of the Measure. In order to check validity of our measure we ran some experiments on traces of memory reference streams from the NMSU TraceBase [37]. Here we present the results of our experiments on address traces collected from SPARC processors running the SPEC92 benchmarks. We considered a page size of 2048 bytes and truncated them after 40000 references. The important thing to notice is that these are not special cases or artificially generated memory references, but are access patterns a real-life implementation of any paging algorithm might face. The results for the corresponding eleven program traces are shown in Table 1. The first row shows the number of distinct pages, the second row shows $\overline{\lambda}$, and finally the third row shows the ratio of the actual locality to the worst possible locality. The worst possible locality of a trace asymptotically equals the number of distinct pages in that trace. It is clear from the low ratios that in general these traces exhibit high locality of reference as defined by our measure.

Next we analyze several well known paging algorithms in terms of the non-locality parameter. We consider the fault rate, the measure usually used by practitioners. The fault rate of a paging algorithm $\mathcal{A}$ on a sequence σ is defined as $\mathcal{A}(\sigma)/|\sigma|$, i.e., the number of faults $\mathcal{A}$ incurs on σ normalized by the length of σ. The fault rate of $\mathcal{A}$, $FR(A)$, is defines as the asymptotic worst case fault rate of $\mathcal{A}$ on any sequence. The bounds are in the worst case sense, i.e., when we say $FR(\mathcal{A}) \geq f(\overline{\lambda})$ we mean that there is a sequence σ such that $\frac{\mathcal{A}(\sigma)}{|\sigma|} \geq f(\overline{\lambda}(\sigma))$ and when we say $FR(\mathcal{A}) \leq g(\overline{\lambda})$ we mean that for every sequence σ we have $\frac{\mathcal{A}(\sigma)}{|\sigma|} \leq g(\overline{\lambda}(\sigma))$. Also for simplicity, we ignore the details related to the special case of the first few requests (the first block or phase). Asymptotically and as the size of the sequences grow, this can only change the computation by additive lower order terms.

Lemma 1. *For any deterministic paging algorithm $\mathcal{A}$, $\frac{\overline{\lambda}}{k+1} \leq FR(\mathcal{A}) \leq \frac{\overline{\lambda}}{2}$.*

Proof. For the lower bound consider a slow memory containing $k+1$ pages. Let σ be a sequence of length n obtained by first requesting $p_1, p_2, \ldots, p_k, p_{k+1}$, and afterwards repeatedly requesting the page not currently in $\mathcal{A}$'s cache. Since $\frac{\mathcal{A}(\sigma)}{|\sigma|} = n/n = 1$, and $\overline{\lambda}$ is at most $k+1$ (there are $k+1$ distinct pages in σ), the lower bound follows.

For the upper bound, consider any request sequence σ of length n. If the i^{th} request is a fault charged to $\mathcal{A}$, then $d_\sigma[i] \geq 2$ (otherwise $\sigma[i]$ cannot have been evicted). Hence, $2\mathcal{A}(\sigma) \leq \sum_{i=1}^{n} d_\sigma[i]$ and the upper bound follows.

We now show that LRU gets the best possible performance in terms of $\overline{\lambda}$.

Theorem 1. $FR(\text{LRU}) = \frac{\overline{\lambda}}{k+1}$.

Proof. It follows from the observation that LRU faults on the request $\sigma[i]$ if and only if $d_\sigma[i] \geq k+1$, which implies $\text{LRU}(\sigma) \leq \frac{\overline{\lambda}}{k+1}$ and Lemma 1.

Next, we show a general upper bound for conservative and marking algorithms.

Lemma 2. *Let $\mathcal{A}$ be a conservative or marking algorithm, then $FR(\mathcal{A}) \leq \frac{2\overline{\lambda}}{k+3}$.*

Proof. Let σ be an arbitrary sequence and let φ be an arbitrary phase in the decomposition of σ. $\mathcal{A}$ incurs at most k faults on φ. For any phase except the first, the first request in φ, say $\sigma[i]$, is to a page that was not requested in the previous phase, which contained k distinct pages. Hence, $d_\sigma[i] \geq k+1$. There are at least $k-1$ other requests in φ to $k-1$ distinct pages, which all could have been present in the previous phase. But these pages contribute at least $\sum_{j=1}^{k-1}(j+1) = k-1+\frac{k^2-k}{2}$ to $\overline{\lambda}$. It follows that the contribution of this phase to $|\sigma|\overline{\lambda}$ is at least $k+1+k-1+\frac{k^2-k}{2} = \frac{k^2+3k}{2}$. Hence,

$$\frac{\mathcal{A}(\sigma)}{|\sigma|\overline{\lambda}} \leq \frac{k}{\frac{k^2+3k}{2}} = \frac{2}{k+3} \Rightarrow FR(\mathcal{A}) \leq \frac{2\overline{\lambda}}{k+3}.$$

There is a matching lower bound for FWF.

Lemma 3. $FR(\text{FWF}) \geq \frac{2\overline{\lambda}}{k+3}$.

Proof. Consider $\sigma = \{p_1 p_2 \ldots p_k p_{k+1} p_k p_{k-1} \ldots p_2\}^n$. $\text{FWF}(\sigma) = 2kn$, since FWF faults on all the requests. Now consider any block except the first. First, consider a page p_i, $2 \leq i \leq k$. The first and second request to p_i contribute i and $k+2-i$ to $|\sigma|\overline{\lambda}$, respectively, giving a total contribution of $k+2$. The requests to p_1 and p_{k+1} contribute $k+1$ each. Hence, the total contribution (for all phases except the first) is $(k-1)(k+2)+2(k+1) = k^2+3k$. Therefore

$$\frac{\text{FWF}(\sigma)}{|\sigma|\overline{\lambda}} = \frac{2k}{k^2+3k} = \frac{2}{k+3}.$$

Thus FWF has approximately twice as many faults as LRU on sequences with the same locality of reference, in the worst case. FIFO also has optimal performance in terms of $\overline{\lambda}$.

Lemma 4. $FR(\text{FIFO}) \leq \frac{\overline{\lambda}}{k+1}$.

Proof. See appendix.

Lemma 5. $FR(\text{LFU}) \geq \frac{2\overline{\lambda}}{k+3}$.

Proof. See appendix.

In contrast LIFO has much poorer performance than most other paging algorithms (the worst possible) in terms of $\overline{\lambda}$.

Lemma 6. $FR(\text{LIFO}) \geq \frac{\overline{\lambda}}{2}$.

Proof. Consider the sequence $\sigma = p_1 p_2 \dots p_k p_{k+1} \{p_k p_{k+1}\}^n$. We have $\text{LIFO}(\sigma) = k + 1 + 2n$ and $|\sigma|\overline{\lambda} = (k+1)(k+1) + 2 \cdot 2n$, and the bound follows.

LRU-2 is another paging algorithm proposed by O'Neil et al. for database disk buffering [30]. On a fault, LRU-2 evicts the page whose second to the last request is least recent. If there are pages in the cache that have been requested only once so far, LRU-2 evicts the least recently used among them. Boyar et al. proved that LRU-2 has competitive ratio $2k$, which is worse than FWF [14].

Lemma 7. $\frac{2k\overline{\lambda}}{(k+1)(k+2)} \leq FR(\text{LRU-2}) \leq \frac{2\overline{\lambda}}{k+1}$.

Proof. See appendix.

While no deterministic on-line paging algorithm can have competitive ratio better than k, there are randomized algorithms with better competitive ratio. The randomized marking algorithm MARK, introduced by Fiat et al. [21], is $2H_k$-competitive, where H_k is the k^{th} harmonic number. On a fault, MARK evicts a page chosen uniformly at random from among the unmarked pages. Let σ be a sequence and $\varphi_1, \varphi_2, \dots, \varphi_m$ be its phases. A page requested in phase φ_i is called *clean* if it was not requested in phase φ_{i-1} and *stale* otherwise. Let c_i be the number of clean pages requested in phase φ_i. Fiat et al. proved that the expected number of faults MARK incurs on phase φ_i is $c_i(H_k - H_{c_i} + 1)$.

Lemma 8. $FR(\text{MARK}) = \frac{2\overline{\lambda}}{3k+1}$.

Proof. Let σ be $\{p_1 p_2 \dots p_k p_{k+1} p_{k+2} \dots p_{2k} p_k p_{k-1} \dots p_1 p_{2k} \dots p_{k+1}\}^n$. This sequence has $4n$ phases. All pages of each phase are clean. Therefore we have $c_i = k$ for $1 \leq i \leq 4n$ and the expected number of faults MARK incurs on each phase is $k \times (H_k - H_k + 1) = k$. Thus $E(\text{MARK}(\sigma)) = 4nk$. We have $|\sigma|\overline{\lambda} = 4n(k + 1 + k + 2 + \dots + 2k) = 4n(k^2 + k(k+1)/2) = 2n(3k^2 + k)$. Hence $\frac{E(\text{MARK}(\sigma))}{|\sigma|\overline{\lambda}} = \frac{4nk}{2n(3k^2+k)} = \frac{2}{3k+1}$, which proves the lower bound.

For the upper bound, consider an arbitrary sequence σ and let $\varphi_1, \varphi_2, \dots, \varphi_m$ be its phases. Suppose that the i^{th} phase has c_i clean pages. Therefore the expected cost of MARK on phase i is at most $c_i(H_k - H_{c_i} + 1)$. The first request to the j^{th} clean page in a phase contributes at least $k + j$ to $|\sigma|\overline{\lambda}$ (k pages from previous phase and $j - 1$ clean pages that have been seen so far). The first request to the j^{th} stale page in a phase contributes at least $j + 1$. Therefore the contribution of phase i to $|\sigma|\overline{\lambda}$ is at least $\sum_{j=1}^{c_i}(k + j) + \sum_{j=1}^{k-c_i}(j + 1) = (2c_i^2 - 2c_i + k^2 + 3k)/2$, and $\frac{E(\text{MARK}(\sigma))}{|\sigma|\overline{\lambda}} \leq \frac{2c_i(H_k - H_{c_i} + 1)}{2c_i^2 - 2c_i + k^2 + 3k}$, where $1 \leq c_i \leq k$. This is an increasing function in terms of c_i and gets its maximum at $c_i = k$. Then we have

$$\frac{E(\text{MARK}(\sigma))}{|\sigma|\overline{\lambda}} \leq \frac{2k(H_k - H_k + 1)}{2k^2 - 2k + k^2 + 3k} = \frac{2}{3k+1}.$$

Finally we prove bounds on the performance of Longest-Forward-distance (LFD), an optimal offline algorithm. On a fault, LFD evicts the page whose next request is farthest in the future.

Lemma 9. $\frac{\overline{\lambda}}{3k+1} \leq FR(LFD) \leq \frac{2\overline{\lambda}}{3k+1}$.

Proof. Let σ be $\{p_1 p_2 \dots p_k p_{k+1} p_{k+2} \dots p_{2k} p_k p_{k-1} \dots p_1 p_{2k} \dots p_{k+1}\}^n$. This sequence has $4n$ phases. Each two consecutive phases of σ contain $2k$ distinct pages. LFD contains at most k pages in its cache before serving these phases and thus it would incur at least k faults on serving any two consecutive phases. Thus we have $\mathrm{LFD}(\sigma) \geq 2kn$. We have $|\sigma|\overline{\lambda} = 4n(k+1+k+2+\dots+2k) = 4n(k^2+k(k+1)/2) = 2n(3k^2+k)$. Hence

$$\frac{\mathrm{LFD}(\sigma)}{|\sigma|\overline{\lambda}} \geq \frac{2nk}{2n(3k^2+k)} = \frac{1}{3k+1}.$$

Any randomized algorithm can be viewed as a probability distribution on a set of deterministic algorithm. Since the performance of LFD on any sequence is at least as good as performance of any deterministic algorithm on that sequence, the performance of LFD is not worse than the expected performance of a randomized algorithm on any sequence. Thus the upper bound follows from Lemma 8.

This analysis can also be applied within the framework of Albers et al. [2]. One can readily obtain results similar but sharper to those of [2]. In them the cost is both in terms of f^{-1} for a concave function f and the non-locality parameter $\overline{\lambda}$. We omit these for lack of space. See appendix for details.

3 Parameterized Analysis of List Update Algorithms

In this section we study the parameterized complexity of list update algorithms in terms of locality of reference. In the list update problem, we have an unsorted list of m items. The input is a sequence of n requests that should be served in an online manner. Let $\mathcal{A}$ be an arbitrary online list update algorithm. To serve a request to an item x, $\mathcal{A}$ should linearly search the list until it finds x. If x is i^{th} item in the list, $\mathcal{A}$ incurs cost i to access x. Immediately after accessing x, $\mathcal{A}$ can move x to any position closer to the front of the list at no extra cost. This is called a *free exchange*. Also $\mathcal{A}$ can exchange any two consecutive items at a cost of 1. These are called *paid exchanges*. The idea is to use free and paid exchanges to minimize the overall cost of serving a sequence. Three well known deterministic online algorithms are *Move-To-Front* (MTF), *Transpose*, and *Frequency-Count* (FC). MTF moves the requested item to the front of the list and Transpose exchanges the requested item with the item that immediately precedes it. FC maintains a frequency count for each item, updates this count after each access, and makes necessary moves so that the list always contains items in non-increasing order of frequency count. Sleator and Tarjan showed that MTF is 2-competitive, while Transpose and FC do not have constant competitive ratios [35]. While list update algorithms can be more easily distinguished using

competitive analysis than in the paging case, the experimental study by Bachrach and El-Yaniv suggests that the relative performance hierarchy as computed by the competitive ratio does not correspond to the observed relative performance of the algorithms in practice [7]. Several authors have pointed out that input sequences of list update algorithms in practice show locality of reference [1, 33, 10] and indeed online list update algorithms try to take advantage of this property [23, 32]. Recently, Angelopoulos et al. [5] and Albers and Lauer [3] have studied list update with locality of reference. We define the non-locality of sequences for list update in an analogous way to the corresponding definition for paging (Definition 1) . The only differences are:

1. We do not normalize the non-locality by the length of the sequence, i.e., $\overline{\lambda}(\sigma) = \sum_{1 \le i \le |\sigma|} d_\sigma[i]$.
2. If $\sigma[i]$ is the first access to an item we assign the value m to $d_\sigma[i]$[5].

Theorem 2. *For any deterministic online list update algorithm $\mathcal{A}$ we have $\overline{\lambda} \le \mathcal{A}(\sigma) \le m \cdot \overline{\lambda}$.*

Proof. [Upper bound] Consider an arbitrary sequence σ of length n. Since the maximum cost that $\mathcal{A}$ incurs on a request is m, we have $\mathcal{A}(\sigma) \le nm$. We have $d_\sigma[i] \ge 1$ for all values of i. Thus $\overline{\lambda} \ge n$. Therefore $\frac{\mathcal{A}(\sigma)}{\overline{\lambda}} \le \frac{nm}{n} = m$.
[Lower bound] Consider a sequence σ of length n obtained by requesting the item that is in the last position of list maintained by $\mathcal{A}$ at each time. We have $\mathcal{A}(\sigma) = nm$. Also we have $d_\sigma[i] \le m$ because σ has at most m distinct items. Therefore $\overline{\lambda} \le nm$, and $\frac{\mathcal{A}(\sigma)}{\overline{\lambda}} \ge \frac{nm}{nm} = 1$.

Theorem 3. *MTF has the optimal performance in terms of $\overline{\lambda}$: $MTF(\sigma) \le \overline{\lambda}$.*

Proof. Consider the i^{th} request of σ. If this is the first request to item $\sigma[i]$, then $d_\sigma[i] = m$, while the cost of MTF on $\sigma[i]$ is at most m. Otherwise, the cost of MTF is $d_\sigma[i]$. Thus the cost of MTF on $\sigma[i]$ is at most $d_\sigma[i]$. Hence, $MTF(\sigma) \le \sum_{1 \le i \le n} d_\sigma[i] = \overline{\lambda}$, and the upper bound follows. Theorem 2 shows that this bound is tight.

The following lemmas show that other well known list update algorithms do not have the optimal performance in terms of $\overline{\lambda}$.

Lemma 10. $Transpose(\sigma) \ge \frac{m \cdot \overline{\lambda}}{2}$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$ be the initial list. Consider a sequence σ of length n obtained by several repetitions of pattern $a_m a_{m-1}$. We have $Transpose(\sigma) = n \cdot m$. Also we have $d_\sigma[i] = m$ for $1 \le i \le 2$ and $d_\sigma[i] = 2$ for $2 < i \le n$. Therefore $\overline{\lambda} = 2m + 2n - 4$, and $\frac{Transpose(\sigma)}{\overline{\lambda}} = \frac{n \cdot m}{2m + 2n - 4}$, which becomes arbitrarily close to $m/2$ as n grows.

[5] As for paging, asymptotically, and assuming the number of requests is much larger than m, any constant can replace m for the $d_\sigma[i]$ of the first accesses.

Lemma 11. $FC(\sigma) \geq \frac{(m+1)\overline{\lambda}}{2} \approx \frac{m \cdot \overline{\lambda}}{2}$.

Proof. See appendix.

Albers introduced the algorithm *Timestamp* (TS) and showed that it has competitive ratio 2 [1]. After accessing an item a, TS inserts a in front of the first item b that is before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorganize the list.

Lemma 12. $TS(\sigma) \geq \frac{2m \cdot \overline{\lambda}}{m+1} \approx 2\overline{\lambda}$.

Proof. See appendix.

Observe that parameterized analysis by virtue of its finer partition of the input space resulted in the separation of several of these strategies which are not separable under the classical model. This introduces a hierarchy of algorithms better reflecting the relative strengths of the strategies considered above. We can also apply the parameterized analysis to randomized list update algorithms by considering their expected cost.

In the next theorem we show that, surprisingly, certain randomized algorithms which are superior to MTF in the standard model are not so in the parameterized case. Observe that in the competitive ratio model a deterministic algorithm must serve a pathological, rare worst case even if at the expense of a more common but not critical case, while a randomized algorithm can hedge between the two cases, hence in the classical model the randomized algorithm is superior to the deterministic one. In contrast, in the parameterized model the rare worst case has a larger non-locality measure if it is pathological, leading to a larger denominator. Hence such a cases can safely be ignored, with a resulting overall increase in the measured quality of the algorithm.

The algorithm *Bit*, considers a bit $b(a)$ for each item a and initializes these bits uniformly and independently at random. Upon an access to a, it first complement $b(a)$, then if $b(a) = 0$ it moves a to the front, otherwise it does nothing. Bit has competitive ratio 1.75, thus beating any deterministic algorithm [31]. In the parameterized model this situation is reversed.

Theorem 4. $E(Bit(\sigma)) \geq \frac{(3m+1)\overline{\lambda}}{2m+2} \approx 3\overline{\lambda}/2$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$ be the initial list and n be an arbitrary integer. Consider the sequence $\sigma = \{a_m^2 a_{m-1}^2 \ldots a_1^2\}^n$. Let σ_i and σ_{i+1} be two consecutive accesses to a_j. After two consecutive accesses to each item, it will be moved to the front of the list with probability 1. Therefore a_j is in the last position of the list maintained by Bit at the time of request σ_i and Bit incurs cost m on this request. After this request, Bit moves a_j to the front of the list if and only if $b(a_j)$ is initialized to 1. Since $b(a_j)$ is initialized uniformly and independently at random, this will happen with probability 1/2. Therefore the expected cost of

Bit on σ_{i+1} is $\frac{1}{2}(m+1)$ and the expected cost of Bit on σ is $nm(m+\frac{m+1}{2})$. We have $\overline{\lambda} = m(m+1)n$. Therefore

$$\frac{E(Bit(\sigma))}{\overline{\lambda}} = \frac{n \cdot m(m+\frac{m+1}{2})}{m(m+1)n} = \frac{3m+1}{2m+2}.$$

4 Conclusions

We applied parameterized analysis in terms of locality of reference to paging and list update algorithms and showed that this model gives promising results. The plurality of results shows that this model is effective in that we can readily analyze well known strategies. Using a finer, more natural measure we separated paging and list update algorithms which were otherwise indistinguishable under the classical model. We showed that a randomized algorithm which is superior to MTF in the classical model is not so in the cooperative case, which matches experimental evidence. This confirms that the ability of the online adaptive algorithm to ignore pathological worst cases can lead to the selection of algorithms that are more efficient in practice.

References

1. S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, June 1998.
2. S. Albers, L. M. Favrholdt, and O. Giel. On paging with locality of reference. *JCSS*, 70(2):145–175, 2005.
3. S. Albers and S. Lauer. On list update with locality of reference. In *Proc. ICALP*, pages 96–107, 2008.
4. S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. On the separation and equivalence of paging strategies. In *Proc. SODA*, pages 229–237, 2007.
5. S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. List update with locality of reference. In *Proc. LATIN*, pages 399–410, 2008.
6. S. Angelopoulos and P. Schweitzer. Paging and list update under bijective analysis. In *Proc. SODA*, pages 1136–1145, 2009.
7. R. Bachrach and R. El-Yaniv. Online list accessing algorithms and their applications: Recent empirical evidence. In *Proc. SODA*, pages 53–62, 1997.
8. L. Becchetti. Modeling locality: A probabilistic analysis of LRU and FWF. In *Proc. ESA*, pages 98–109, 2004.
9. S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11:73–91, 1994.
10. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
11. A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *JCSS*, 50:244–258, 1995.
12. A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *JCSS*, 50:244–258, 1995.
13. P. Bose, K. Douïeb, and S. Langerman. Dynamic optimality for skip lists and B-trees. In *Proc. SODA*, pages 1106–1114, 2008.
14. J. Boyar, M. R. Ehmsen, and K. S. Larsen. Theoretical evidence for the superiority of LRU-2 over LRU for the paging problem. In *Proc. WAOA*, pages 95–107, 2006.

15. J. Boyar and L. M. Favrholdt. The relative worst order ratio for on-line algorithms. In *Proc. Italian Conf. on Algorithms and Complexity*, 2003.
16. M. Chrobak and J. Noga. LRU is better than FIFO. *Algorithmica*, 23(2):180–185, 1999.
17. P. J. Denning. The working set model for program behaviour. *CACM*, 11(5):323–333, 1968.
18. P. J. Denning. Working sets past and present. *IEEE Transactions on Software Engineering*, SE-6(1):64–84, 1980.
19. P. J. Denning. The locality principle. *CACM*, 48(7):19–24, 2005.
20. R. Dorrigiv and A. López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News*, 36(3):67–81, September 2005.
21. A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.
22. A. Fiat and Z. Rosen. Experimental studies of access graph based heuristics: Beating the LRU standard? In *Proc. SODA*, pages 63–72, 1997.
23. J. H. Hester and D. S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17(3):295, Sept. 1985.
24. J. Iacono. Improved upper bounds for pairing heaps. In *Proc. SWAT*, pages 32–45, 2000.
25. J. Iacono. Alternatives to splay trees with O(log n) worst-case access times. In *Proc. SODA*, pages 516–522, 2001.
26. S. Irani, A. R. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing*, 25:477–497, 1996.
27. A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov paging. *SIAM Journal on Computing*, 30(3):906–922, 2000.
28. C. Kenyon. Best-fit bin-packing with random order. In *Proc. SODA*, pages 359–364, 1996.
29. E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30:300–317, 2000.
30. E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proc. ACM SIGMOD Conf.*, pages 297–306, 1993.
31. N. Reingold and J. Westbrook. Randomized algorithms for the list update problem. Technical Report YALEU/DCS/TR-804, Yale University, June 1990.
32. N. Reingold, J. Westbrook, and D. D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11:15–32, 1994.
33. F. Schulz. Two new families of list update algorithms. In *Proc. ISAAC*, pages 99–108, 1998.
34. A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. John Wiley & Sons, 2002.
35. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *CACM*, 28:202–208, 1985.
36. D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *JACM*, 32(3):652–686, 1985.
37. N. M. S. University. Homepage of new mexico state university tracebase (online). Available at: http://tracebase.nmsu.edu/tracebase.html.
38. N. E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.
39. N. E. Young. On-line paging against adversarially biased random inputs. *Journal of Algorithms*, 37(1):218–235, 2000.
40. N. E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

A Omitted proofs

Lemma 4. $FR(\text{FIFO}) \leq \frac{\overline{\lambda}}{k+1}$.

Proof. Let σ be an arbitrary sequence. Consider a fault $\sigma[i]$ on a page p and let $\sigma[j_1], \sigma[j_2], \ldots, \sigma[j_m]$ be the requests to p since p last entered the cache. By definition all the requests $\sigma[j_1], \sigma[j_2], \ldots, \sigma[j_m]$ are hits and p is evicted between σ_{j_m} and σ_i. Observe that for p to get evicted at least k distinct pages other than p have to be requested since p entered the cache, hence $d_\sigma[i] + \sum_{h=1}^{m} d_\sigma[j_h] \geq k+1$. It follows that for each fault charged to FIFO we have at least a contribution of $k+1$ to $|\sigma|\overline{\lambda}$.

Lemma 5. $FR(\text{LFU}) \geq \frac{2\overline{\lambda}}{k+3}$.

Proof. Consider the (usual) sequence $\sigma = p_1^n p_2^n \ldots p_{k-1}^n \{p_k p_{k+1}\}^n$, where $\text{LFU}(\sigma) = k-1+2n$. For $|\sigma|\overline{\lambda}$, each of the pages $p_1, p_2, \ldots, p_{k-1}$ contributes $k+1+n-1 = k+n$, and the pages p_k and p_{k+1} contribute $(k+1)+2(n-1)$ each. Hence, $|\sigma|\overline{\lambda} = (k-1)(k+n) + 2(k+2n-1) = (k+3)n + k^2 + k - 2$, and therefore

$$\frac{\text{LFU}(\sigma)}{|\sigma|\overline{\lambda}} = \frac{2n+k-1}{(k+3)n + k^2 + k - 2},$$

which becomes arbitrarily close to $\frac{2}{k+3}$ as n grows.

Lemma 7. $\frac{2k\overline{\lambda}}{(k+1)(k+2)} \leq FR(\text{LRU-2}) \leq \frac{2\overline{\lambda}}{k+1}$.

Proof. Let σ be the sequence $\{p_1 p_2 \ldots p_{k-1} p_k p_k p_{k-1} \ldots p_1 p_{k+1} p_{k+1}\}^n$. Now, consider any repetition except the first. LRU-2 faults on all requests except the second request to p_k and the second request to p_{k+1}, giving a total of $2k$ faults. The first request to p_i, $1 \leq i \leq k-1$, contributes i to $|\sigma|\overline{\lambda}$ and the second request to p_i contributes $k+2-i$. Hence, each of these $k-1$ pages contributes $k+2$. For the pages p_k and p_{k+1}, their first request contributes $k+1$ and the second only 1 to $|\sigma|\overline{\lambda}$. This gives a total contribution of $(k+1)(k+2)$, and asymptotically the result follows.

For the upper bound, consider three consecutive faults on some page p. At least k other distinct pages should be requested since the first fault on p (at least $k-1$ other pages with at least 2 request and at least one other page).

Lemma 11. $FC(\sigma) \geq \frac{(m+1)\overline{\lambda}}{2} \approx \frac{m \cdot \overline{\lambda}}{2}$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$ be the initial list and n be an arbitrary integer. Consider the following sequence: $\sigma = a_1^n a_2^n a_3^n \ldots a_m^n$. On serving σ, FC does not change the order of items in its list and incurs cost $\sum_{i=1}^{m} ni = n \sum_{i=1}^{m} i = n\frac{m(m+1)}{2}$. We have $\overline{\lambda} = \sum_{i=1}^{m}(m + (n-1)) = m \cdot n + m^2 - m$. Therefore

$$\frac{FC(\sigma)}{\overline{\lambda}} = \frac{n\frac{m(m+1)}{2}}{m \cdot n + m^2 - m},$$

which approaches $\frac{m+1}{2}$ as n grows.

Lemma 12. $TS(\sigma) \geq \frac{2m \cdot \overline{\lambda}}{m+1} \approx 2\overline{\lambda}$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_m)$ be the initial list and n be an arbitrary integer. Consider the sequence σ obtained by the repetition of the block $a_m^2 a_{m-1}^2 \ldots a_1^2$ n times. Let B be an arbitrary block of σ. Each item a_i is accessed twice in B. TS does not move a_i after its first access in B, because each item has been accessed twice since the last access to a_i. After the second access, TS moves the item to the front of the list. Therefore each access is to the last item of the list and TS incurs a cost of m on each access. Thus, we have $TS(\sigma) = 2m^2 n$. Next we compute $\overline{\lambda}$. The first and second access to a in block B contribute m and 1 to $\overline{\lambda}$, respectively. Thus we have $\overline{\lambda} = m(m+1)n$. Therefore

$$\frac{TS(\sigma)}{\overline{\lambda}} = \frac{2m^2 n}{m(m+1)n} = \frac{2m}{m+1}.$$

B Restricting the set of legal sequences to those with locality of reference

We can further incorporate locality of reference assumption by restricting the input to those with high locality of reference in the max-model proposed by Albers et al. [2]. For a concave function f we say that a sequence is consistent with f if the number of distinct pages in any window of size n is at most $f(n)$ for any positive integer n. We model locality of reference by restricting the input to $\mathcal{I}_f$, sequences that are consistent with f. For any positive integer m, we define $f^{-1}(m)$ as the smallest size of a window in a sequence consistent with f that contains m distinct pages, i.e., $f^{-1}(m) = \min\{n \in \mathcal{N} | f(n) \geq m\}$. Table 2 shows the fault rate of paging algorithms in terms of $\overline{\lambda}$ when we restrict the input to $\mathcal{I}_f$.

Algorithm	Lower Bound	Upper Bound
General	$\frac{(k-1)\overline{\lambda}}{k(k-1)+f^{-1}(k+1)-2}$	$\frac{\overline{\lambda}}{2}$
Marking	$\frac{(k-1)\overline{\lambda}}{k(k-1)+f^{-1}(k+1)-2}$	$\frac{2k\overline{\lambda}}{k(k+1)+2(f^{-1}(k+1)-1)}$
LRU	$\frac{(k-1)\overline{\lambda}}{k(k-1)+f^{-1}(k+1)-2}$	$\frac{(k-1)\overline{\lambda}}{k(k-1)+f^{-1}(k+1)-2}$
FWF	$\frac{2k\overline{\lambda}}{k(k+1)+2(f^{-1}(k+1)-1)}$	$\frac{2k\overline{\lambda}}{k(k+1)+2(f^{-1}(k+1)-1)}$
FIFO	$\frac{(k-\frac{1}{k})\overline{\lambda}}{(k-1)(k+1)+f^{-1}(k+1)-1}$	$\frac{k\overline{\lambda}}{(k-1)(k+1)+f^{-1}(k+1)}$

Table 2. The fault rate of paging algorithms in terms of $\overline{\lambda}$ with respect to a concave function f.

Most of the proofs below uses constructions similar to the ones given by Albers et. al [2]. Let $FR^f(\mathcal{A})$ denote the fault rate of an algorithm $\mathcal{A}$ with respect to a function f.

Lemma 13. *For any deterministic online paging algorithm $\mathcal{A}$,*

$$\frac{(k-1)\overline{\lambda}}{k(k-1)+f^{-1}(k+1)-2} \leq FR^f(\mathcal{A}) \leq \frac{\overline{\lambda}}{2}.$$

Proof. The upper bound follows from the general upper bound proved in Lemma 1. For the lower bound, given the function f and algorithm $\mathcal{A}$, we construct a sequence σ as follows. We use $k+1$ distinct pages. The sequence σ is constructed in phases, each with length $f^{-1}(k+1)-2$, where each phase consists of $k-1$ blocks. Each block contains requests all to the page that was not in $\mathcal{A}$'s cache just before the first request of the block. Hence, $\mathcal{A}$ faults on the first request of each block and incurs $k-1$ faults on each phase. In each phase, block j, $1 \leq j \leq k-1$, starts with request $f^{-1}(j+1)-1$. The construction is well-defined and is consistent with f ([2, Theorem 1]). For an upper bound on the non-locality of a phase, observe that since there are only $k+1$ distinct pages, the first request of each block of the phase contributes at most $k+1$ to $|\sigma|\overline{\lambda}$. Each of the following requests in the block, contributes 1. Since there are $k-1$ block, the first requests of blocks contribute at most $(k+1)(k-1)$ to $|\sigma|\overline{\lambda}$ totally. Since there are $f^{-1}(k+1)-2-(k-1)$ other requests in a phase, each contributing 1 to $|\sigma|\overline{\lambda}$, we get a total contribution of at most $(k+1)(k-1)+f^{-1}(k+1)-2-(k-1) = k(k-1)+f^{-1}(k+1)-2$, and the result follows.

Lemma 14. $FR^f(\mathrm{LRU}) \leq \frac{(k-1)\overline{\lambda}}{k(k-1)+f^{-1}(k+1)-2}$.

Proof. Let σ be an arbitrary sequence consistent with f. Partition σ into phases such that each phase contains $k-1$ faults made by LRU, except possibly the last, and is maximal subject to that constraint. Hence, the request just before and just after a phase is a fault for all phases except the first and last phases. Let φ be any such phase. In [2, Theorem 2] it is shown that φ has length at least $f^{-1}(k+1)-2$. Since LRU faults on the i'th request if and only if $d_\sigma[i] \geq k+1$, each of the $k-1$ faults made by LRU in φ contributes at least $k+1$ to $|\sigma|\overline{\lambda}$. All other requests contribute at least 1. Hence, the total contribution to $|\sigma|\overline{\lambda}$ is at least $(k+1)(k-1)+f^{-1}(k+1)-2-(k-1) = k(k-1)+f^{-1}(k+1)-2$, and the upper bound follows.

Lemma 15. *For any marking algorithm $\mathcal{A}$, $FR^f(\mathcal{A}) \leq \frac{2k\cdot\overline{\lambda}}{k(k+1)+2(f^{-1}(k+1)-1)}$.*

Proof. Let σ be an arbitrary sequence and consider the decomposition of σ. $\mathcal{A}$ incurs at most k faults on each phase. For any phase φ except the last, the next phase begins with a request to a page not in φ. Hence, the subsequence consisting of φ and the first request of the next phase, contains $k+1$ distinct pages and has length at least $f^{-1}(k+1)$. It follows that the length of φ is at least $f^{-1}(k+1)-1$. For any phase φ except the first, the first request, say i, in φ is to a page that was not requested in the previous phase, which contained k distinct pages. Hence, $d_\sigma[i] \geq k+1$. Now, consider any phase φ except for the first and last phase. The first request contributes at least $k+1$ to $|\sigma|\overline{\lambda}$. There are at least $k-1$ other requests in φ to $k-1$ distinct pages, which all could

have been present in the previous phase. But these pages contribute at least $\sum_{j=1}^{k-1}(j+1) = k-1+\frac{k^2-k}{2}$ to $|\sigma|\overline{\lambda}$. The remaining $f^{-1}(k+1)-1-k$ requests in P (requests to pages already requested in φ) all contribute at least 1. It follows that the contribution to $|\sigma|\overline{\lambda}$ of φ is at least

$$k+1+k-1+\frac{k^2-k}{2}+f^{-1}(k+1)-1-k = \frac{k(k+1)+2(f^{-1}(k+1)-1)}{2}.$$

Hence,

$$\frac{\mathcal{A}(\sigma)}{|\sigma|\overline{\lambda}} \leq \frac{k}{\frac{k(k+1)+2(f^{-1}(k+1)-1)}{2}} = \frac{2k}{k(k+1)+2(f^{-1}(k+1)-1)}.$$

Lemma 16. $FR^f(\text{FWF}) \geq \frac{2k\cdot\overline{\lambda}}{k(k+1)+2(f^{-1}(k+1)-1)}$.

Proof. We construct a sequence σ as follows. We use $k+1$ pages, $p_1, p_2, \ldots, p_{k+1}$. σ consists of phases (corresponding to the phases of FWF) and each phase is composed of k blocks, where each block is a subsequence of requests to the same page. In each phase, block j, $1 \leq j \leq k$, has length $f^{-1}(j+1) - f^{-1}(j)$. By Proposition [2, Proposition 1], the block lengths are well-defined, i.e., they are non-zero, and non-decreasing in a phase, and the total length of a phase is $f^{-1}(k+1)-1$.

In the first phase, the jth block consists of requests to p_j ($1 \leq j \leq k$). Now we inductively define the $(i+1)$st phase. The first block consists of $f^{-1}(2)-f^{-1}(1) = 2-1 = 1$ requests to the unique page that was unmarked at the end of the ith phase. That causes FWF to fault and flush the cache (and all pages become unmarked). The second block consists of requests to the page that was requested in the last block of the ith phase. The third block consists of requests to the page that was requested in the second to last block of the ith phase. The following $k-2$ blocks are defined similarly. By [2, Theorem 4], the construction is consistent with f.

FWF faults k times in each phase. For the non-locality of the sequence consider any phase except the first or the least. The first request contributes $k+1$ to $|\sigma|\overline{\lambda}$. There are $k-1$ other distinct pages requested in the phase, and the first request to each of these contributes $\sum_{j=1}^{k-1}(j+1) = k-1+\frac{k^2-k}{2}$. The remaining $f^{-1}(k+1)-1-k$ requests (requests to pages already requested in the phase), contributes all 1 to $|\sigma|\overline{\lambda}$, and the result follows as in the previous lemma.

Lemma 17. $\frac{(k-\frac{1}{k})\overline{\lambda}}{(k-1)(k+1)+f^{-1}(k+1)-1} \leq FR^f(\text{FIFO}) \leq \frac{k\cdot\overline{\lambda}}{(k-1)(k+1)+f^{-1}(k+1)}$.

Proof. For the lower bound, follow the construction from [2, Theorem 5]. For the non-locality, observe that the first request in each block contributes $k+1$ to $|\sigma|\overline{\lambda}$. The following request to p_k contributes 2 and any following request to p_k contributes 1. Hence, each block contributes $k+1+|\text{block}|$, and since there are $k-1$ blocks in a phase and k phases in a super phase, the total contribution to $|\sigma|\overline{\lambda}$ of a super phase is

$$k(k-1)(k+1)+|\text{super phase}| = k(k-1)(k+1)+k(f^{-1}(k+1)-1),$$

and the lower bound follows.

For the upper bound, first consider a fault $\sigma[i]$ incurs on a page p and let $\sigma[j_1], \sigma[j_2], \ldots, \sigma[j_m]$ be the requests to p since p last entered the cache. By definition all the requests $\sigma[j_1], \sigma[j_2], \ldots, \sigma[j_m]$ are hits and p is evicted between request $\sigma[j_m]$ and $\sigma[i]$. Observe that for p to get evicted at least k distinct pages different from p has to be requested since it entered the cache, hence $d_\sigma[i] + \sum_{h=1}^{m} d_\sigma[j_h] \geq k+1$. It follows that for each fault charged to FIFO we have at least a contribution of $k+1$ to $|\sigma|\overline{\lambda}$. Now, partition the sequence into phases that contains exactly k faults by FIFO and starts with a fault. Since a subsequence consisting of a phase and the following request, contains $k+1$ faults it must have a length of at least $f^{-1}(k+1)$. Hence, a phase has length at least $f^{-1}(k+1)-1$. By the above observation the k faults in a phase contribute $k(k+1)$ to $|\sigma|\overline{\lambda}$. The remaining $f^{-1}(k+1)-1-k$ requests contribute at least 1. It follows that the total contribution of a phase is $k(k+1)+f^{-1}(k+1)-1-k = (k-1)(k+1)+f^{-1}(k+1)$. The upper bound follows.