Paging and List Update under Bijective Analysis

Spyros Angelopoulos and Pascal Schweitzer
Max-Planck-Institut für Informatik
Campus E1 4, 66123 Saarbrücken, Germany

July 3, 2008

Abstract

It has long been known that for the paging problem in its standard form, competitive analysis cannot adequately distinguish algorithms based on their performance: there exists a vast class of algorithms which achieve the same competitive ratio, ranging from extremely naive and inefficient strategies (such as Flush-When-Full), to strategies of excellent performance in practice (such as Least-Recently-Used and some of its variants). A similar situation arises in the list update problem: in particular, under the cost formulation studied by Martínez and Roura [TCS 2000] and Munro [ESA 2000] every list update algorithm has, asymptotically, the same competitive ratio. Several refinements of competitive analysis, as well as alternative performance measures have been introduced in the literature, with varying degrees of success in narrowing this disconnect between theoretical analysis and empirical evaluation.

In this paper we study these two fundamental online problems under the framework of *bijective analysis* [Angelopoulos, Dorrigiv and López-Ortiz, SODA 2007 and LATIN 2008]. This is an intuitive technique which is based on pairwise comparison of the costs incurred by two algorithms on sets of request sequences of the same size. Coupled with a well-established model of locality of reference due to Albers, Favrholdt and Giel [JCSS 2005], we show that Least-Recently-Used and Move-to-Front are the unique optimal algorithms for paging and list update, respectively. Prior to this work, only measures based on average-cost analysis have separated LRU and MTF from all other algorithms. Given that bijective analysis is a fairly stringent measure (and also subsumes average-cost analysis), we prove that in a strong sense LRU and MTF stand out as the best algorithms.

1 Introduction

In their seminal work, Sleator and Tarjan [30] introduced *competitive analysis* as a framework for analyzing the performance of online algorithms. Assuming a cost-minimization problem, an online algorithm is said to be *c-competitive* if its cost on any request sequence never exceeds c times the optimal offline cost for serving that sequence (up to an additive constant). The resulting measure (termed competitive ratio in [21]) has led to the development of a mathematical theory of online algorithms (see [7] and [17] for a comprehensive treatment of the area).

Notwithstanding its wide applicability, there exist online problems for which competitive analysis is not congruent with empirical evaluation. The most notable example is also one of the most well-studied problems in online computation, namely the *paging* problem. Consider, for instance, the paging strategies Flush-When-Full (FWF), First-In-First-Out (FIFO), and Least-Recently-Used (LRU). As shown in [30] and [21], all these strategies yield the same competitive ratio equal to the cache size (a result extended by [31] to the whole class of *marking* algorithms). However, in practice LRU (and some of its variants) are the preferred strategies for paging [29]. More specifically, experimental results suggest that the "empirical competitive ratio" of LRU is a small constant which does not depend on the cache size [32].

A similar situation arises in the context of another fundamental online problem, namely *list update*. In one of their central results in [30], Sleator and Tarjan showed that the Move-To-Front heuristic (MTF) is 2-competitive, and that this is the best competitive ratio that can be achieved (up to low-order terms). It must be emphasized that optimality of MTF is shown to hold for the particular definition of the cost incurred by a list-access algorithm as specified in [30] (to which we refer as the *standard cost model*). Some other algorithms, such as the Timestamp algorithm [2] are also 2-competitive in this model. In subsequent work, Martínez and Roura [25] and Munro [26] independently argued that the standard-cost model penalizes reasonable algorithms which in practice can be quite efficient. Instead, they proposed a different cost formulation (to which, following [4] we refer as the *modified-cost model*) and showed that *every* online algorithm has asymptotically the same, non-constant competitive ratio, namely $\Theta(l/\log l)$, where l is the size of the list. In other words, competitive analysis is met with the same, if not worse, inadequacy in evaluating the performance of algorithms. Martínez and Roura observed this by noting: "an important question is whether there exist alternative ways to define competitiveness such that MTF and other good online algorithms for the list update problem would be competitive even for the modified cost model".

Bijective Analysis. In this paper we study the performance of algorithms for paging and list update problems using *Bijective Analysis*, a technique introduced by Angelopoulos, Dorrigiv and López-Ortiz [3] for comparing the performance of online algorithms. Let $\mathcal{I}_n$ denote the set of all request sequences of size n, and for algorithm A and request sequence $\sigma \in \mathcal{I}_n$, let $A(\sigma)$ denote the cost of A on input σ. For concreteness, consider cost-minimization problems.

Definition 1 ([3]). *The online algorithm A is* no worse *than the online algorithm B on inputs of size n according to bijective analysis, if there exists a bijection $\pi : \mathcal{I}_n \to \mathcal{I}_n$ satisfying $A(\sigma) \leq B(\pi(\sigma))$ for each $\sigma \in \mathcal{I}_n$. We denote this by $A \preceq_{b,n} B$. We say that A is* no worse *than B if for every $n \geq n_0$ (with n_0 a constant) we have $A \preceq_{b,n} B$; we denote this by $A \preceq_b B$. A and B are* equivalent *according to bijective analysis (denoted by $A \equiv_b B$) if $A \preceq_b B$ and $B \preceq_b A$. Last, A is* strictly better *than B according to bijective analysis if $A \preceq_b B$ and $B \not\preceq_b A$.*

We adopt bijective analysis as our measure of choice since it has several advantages over the competitive ratio. More specifically, bijective analysis:

• *Is a simple and intuitive, yet powerful measure.* Showing that $A \preceq_b B$ guarantees that every "bad" request sequence for algorithm A corresponds to some request sequence which is at least as bad for algorithm B, if not worse. Thus the measure does not evaluate the performance of an algorithm on a single "worst-case" sequence, but instead takes into consideration the overall performance of the algorithm over *all* request sequences.

• *Allows direct comparison of two algorithms*, hence there is neither the need to appeal to an offline algorithm, nor to analyze the offline optimum.

• *Is consistent with some natural, "to-be-expected" properties of efficient online algorithms* which competitive analysis fails to yield. For instance [3] shows that LRU with lookahead is strictly better (according to bijective analysis) than LRU without lookahead. In contrast, as argued in [6], rather counterintuitively, no finite lookahead can improve the competitive ratio of any paging algorithm,.

• *Can carry over results to different cost formulations.* For instance, if $A \preceq_b B$ for two paging algorithms A and B under the standard cost definition (number of faults), then the same relation trivially carries over to the *fault rate* (number of faults over the sequence length), which is the measure favored by practitioners (see [1]).

• *Can incorporate assumptions concerning the space of request sequences.* Suppose that we restrict the set of sequences to those exhibiting certain natural or established properties (e.g., *locality of reference*). Applying bijective analysis over the restricted input set could lead to a better understanding of the performance of algorithms[1]. For instance, since the statement of LRU is tailored so that the algorithm exploits locality of reference to its benefit, one would aim to show that LRU is better than many an online algorithm, assuming an appropriate formal definition of what it means for a sequence to exhibit high locality.

Clearly, bijective analysis may be difficult or even not applicable to specific problems, since by definition it establishes a very strong relation among the compared algorithms (which may not exist). A substantially weaker measure (and thus easier to analyze) termed *Average Analysis* in [3] compares the average cost of two algorithms over requests of the same length. We say that A is *no worse* than B on inputs of size n according to average analysis, if $\sum_{\sigma \in \mathcal{I}_n} A(\sigma) \leq \sum_{\sigma \in \mathcal{I}_n} B(\sigma)$; we denote this by $A \preceq_{a,n} B$. Further, $A \preceq_a B$, $A \equiv_a B$ and $A \prec_a B$ are defined along the lines of Definition 1.

As shown in [3], all *lazy* paging algorithms are equivalent according to bijective analysis (a lazy algorithm is an algorithm which evicts a page from the cache only when a fault occurs). This equivalence shows that, for any two algorithms, the multisets of the cost-values incurred on all request sequences of the same length are the same. This implies that only performance measures which declare some sequences as "more important" can distinguish between paging algorithms. Indeed, [3] shows that LRU is shown to be strictly better than any online algorithm, according to average analysis on sequences which exhibit some locality of reference. Similarly, [4] shows that all list update algorithms are equivalent according to bijective analysis, in the modified-cost model, however, if the requests exhibit locality of reference, MTF is strictly better than any other algorithm under average analysis.

The question whether the same conclusions can be drawn under the substantially more stringent measure of bijective analysis was left open in [3] and [4]. In this paper we answer the question in the affirmative (c.f. Theorem 1 and Theorem 2). Given the strong relation that bijective analysis establishes between the compared algorithms, our results provide strong theoretical evidence to the superiority of the above strategies.

[1]This is not to be interpreted as in not allowing sequences outside the restricted set. Instead, it points out that the more refined our knowledge about the request sequences is, the more accurate analysis we should expect.

2 Preliminaries and related work

Problem definitions and cost models. We consider the *paging* problem in its standard version: Let $P = \{p_1, \ldots p_N\}$ denote the space of available pages. A paging algorithm mediates between a slow memory and a fast memory (cache) and must serve a sequence of requests to pages in P: if the current request can be found in the cache, a *hit* is incurred, and the algorithm pays no cost. Otherwise the request incurs a *fault*, and the paging algorithm must determine which page in the cache must be replaced by the requested page, at unit cost.

In the *list access* problem, given an unordered list of l items, an online algorithm A must serve a sequence of n requests to items in the list. In the *standard-cost model* [30], upon a request to an element x in the list, A performs first a linear search for item x: if x is the i-th element in the list, then A incurs a cost of i for accessing x. A can also move x closer to the front of the list at no additional cost (a "free" exchange), or can exchange any two consecutive items in the list at a unit cost (a "paid" exchange). In the more general *list update* problem, a request may refer to either the insertion of an element or its deletion from the list, however the important operation is the access operation, hence we only focus on request sequences that consist exclusively of access operations.

Martínez and Roura [25] and Munro [26] observed that in a realistic setting, rearranging the first i items of the list according to a permutation of $\{1, \ldots, i\}$ should require time proportional to i, in practice, whereas in the standard-cost model it requires time $\Omega(i^2)$. This motivated the *modified-cost model*, in which the cost of accessing the i-th element in the list, plus the cost of reorganizing the first i elements in any possible way (permutation) is linear in i. For definitiveness, in this model, we require that accessing the i-th item in the list costs i, and rearranging the first i items can be done at zero cost (i.e., by means of free exchanges).

In this work we address the performance of list-update algorithms under both models.

Alternatives to competitive ratio and locality of reference. The discrepancy between theoretical analysis and empirical performance was observed as early as in [30] and spawned an extensive line of research on refined definitions of the competitive ratio as well as on alternative measures for the analysis of online algorithms in general, and for the paging problem in particular. Among the known approaches are: the *Max-Max ratio* of Ben-David and Borodin [6]; the *diffuse adversary model* of Koutsoupias and Papadimitriou [24] and Young [33] [34]; *loose competitiveness* of Young [32] [35]; the *random order ratio* of Kenyon [23]; the *relative worst-order ratio* of Boyar, Favrholdt and Larsen [10] and Boyar, Ehmsen and Larsen [9]; and the *accommodation function model* of Boyar, Larsen and Nielsen [12].

Temporal locality of reference permeates memory access in computer systems, and is a well-established property of a typical request sequence in paging. Several models and techniques have been proposed that capture (and exploit) locality of reference, such as the *access graph model* due to Borodin, Irani, Raghavan and Schieber [8] (also in work by Chrobak and Noga [14]) and its generalization known as *the Markov paging model* due to Karlin, Phillips and Raghavan [22]; the *full access cost model* due to Torng [31]; the *random order ratio* of Kenyon [23]; the *adequate performance analysis* of Panagiotou and Souza [27] and; *probabilistic analysis* by Becchetti [5].

Due to space limitations, we refer the reader to the survey of Dorrigiv and López-Ortiz [16] for a detailed exposition. (See also Table 1 in the Appendix). Although previous work has shown separation results for certain specific algorithms, so far only average analysis has separated LRU and MTF from every other algorithm.

In our work we adopt an intuitive and well-established model of locality of reference due to Albers, Favrholdt and Giel [1], which in turn is based on Denning's concept of the *working set* [15]. Let f denote a function that represents the maximum (or average) number of distinct pages in a window of size w in the request sequence (i.e., a consecutive subsequence of length w). As shown by extensive experiments over real data, f is typically approximated by an increasing concave[2] function. There are two possible ways to capture locality in this model: In the *Max-model*, we say that a request sequence is *consistent with* f if the number of distinct pages in any window of size w is always at most $f(w)$, whereas in the *Average-model* we say that a request sequence is consistent with f if the *average* number of distinct pages over all windows of size w is always at most $f(w)$.

Let $\mathcal{I}_n^f$ denote the set of request sequences of size n. which are consistent with a given concave function f. As argued in the Introduction, we can still apply bijective and average analysis in comparing the performance of two algorithms over request sequences drawn from $\mathcal{I}_n^f$. We use upper-script f to denote the corresponding relations (e.g., $A \preceq_a^f B$, $A \preceq_b^f B$, etc.).

In list update, the notion of locality of reference is less self-evident than in paging. While not a de facto property of a typical request sequence, we know that in practice, applications of list-update algorithms encounter sequences with high locality [18] [28], and list update algorithms try to take advantage of this property. For instance, list update algorithms can be used in a very direct way for file compression: a standard practice then is to preprocess the input by means of the Burrows-Wheeler transform, which transforms a string to one of its permutations that has increased locality of reference, hence can be compressed more efficiently [13] [20]. Experiments over the Calgary Compression Corpus [19] which is a standard benchmark for file compression, show [4] that the maximum number of distinct characters in a window of size w can be approximated by a function concave in w: in other words, the model of Albers et al. can faithfully represent locality of reference in list accessing, at least in applications which arise from file compression.

In terms of the above notation, [3] and [4] show that $LRU \preceq_a^f A$ and $MTF \preceq_a^f B$, where A and B are any online paging and list access algorithm, respectively. In addition, they show that for every online algorithm A which is oblivious of f, in the sense that f is not provided as part of the input, there exists a concave function f such that $LRU \prec_a^f A$ and $MTF \prec_a^f A$. In words, this means that LRU and MTF are unique optimal algorithms according to average analysis. In the remainder of this paper we show that the same conclusions apply for bijective analysis over $\mathcal{I}^f$. Note that it suffices to prove that $LRU \preceq_b^f A$ and $MTF \preceq_b^f B$, since uniqueness is implied by average analysis. In the discussion of Section 3 and Section 4 we assume the Max-model for f. In Section 5 we argue that the optimality of LRU and MTF can also be derived in the Average-model, as well as in even broader and more permissive (but still natural) definitions of what constitutes locality of reference.

3 Bijective analysis for paging

We denote by $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ a sequence of n requests, with each σ_i an element of P. We use the notation $\sigma[i,j]$ to denote the (contiguous) subsequence $\sigma_i \ldots \sigma_j$.

Definition 2. *Let p_i, p_k denote two distinct pages in P. Let σ_j denote the j-th request in some request sequence σ. We define the* complement of σ_j with respect to i and k, *denoted by $\overline{\sigma_j}^{(i,k)}$ as the function that replaces p_i with p_k, and vice versa. Formally, $\overline{\sigma_j}^{(i,k)} = p_i$, if $\sigma_j = p_k$; $\overline{\sigma_j}^{(i,k)} = p_k$, if $\sigma_j = p_i$; and $\overline{\sigma_j}^{(i,k)} = \sigma_j$, otherwise.*

For convenience, we will use $\overline{\sigma_j}$ when i, k are implied from context. For any sequence $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ we also define $\overline{\sigma}$ as the sequence $\overline{\sigma}_1 \ldots \overline{\sigma}_n$.

The following lemma describes a property of sequences consistent with f. In very informal terms, it suggests that if the sequence $\ldots p_k \ldots p_i \ldots p_k \ldots p_i \ldots$ exhibits locality of reference, then so does the sequence $\ldots p_k \ldots p_i \ldots p_i \ldots p_k \ldots$.

[2] Albers *et al.* consider a slightly more restrictive class of functions called concave* functions.

Lemma 1. *Let $\sigma = \sigma_1 \dots \sigma_n$ be a sequence of n requests that is consistent with f. Let $t \leq n$ be an integer such that $\sigma[1,t]$ contains a request to p_i, and in addition, p_k does not appear in $\sigma[1,t]$ after the last request to p_i in $\sigma[1,t]$.*

Let σ' denote the sequence $\sigma[1,t] \cdot \overline{\sigma}[t+1,n]$, and suppose that σ' is not consistent with f. Then $\sigma[t+1,n]$ contains a request to p_i; furthermore, no request to p_k in $\sigma[t+1,n]$ occurs earlier than the first request to p_i in $\sigma[t+1,n]$.

Proof. Since σ' is not consistent with f, there must exist $j_1 < j_2 \leq n$ such that $\sigma'[j_1, j_2]$ contains more than $f(j_2 - j_1 + 1)$ distinct pages. Note that for any subsequence ω of $\sigma[t+1,n]$, $\overline{\omega}$ has the same number of distinct pages as ω. This implies that j_1 and j_2 must be such that $j_1 \leq t < j_2$.

Suffices then to argue that $\sigma[t+1, j_2]$ contains a request to p_i but no request to p_k. It is easy to see that $\sigma[j_1, t]$ cannot contain requests to both p_i and p_k, neither can it contain requests to none of these pages: if either of these cases occurred, then $\sigma[j_1, j_2]$ and $\sigma[j_1, t]\overline{\sigma}[t+1, j_2]$ would contain the same number of distinct pages, which would contradict that σ is consistent with f. From assumption, we further deduce that $\sigma[j_1, t]$ contains a request to p_i but no request to p_k.

Now $\sigma[j_1, t]\overline{\sigma}[t+1, j_2]$ must contain a request that does not appear in $\sigma[j_1, j_2]$, and p_k is the only such candidate. We conclude that $\sigma[t+1, j_2]$ contains p_i but does not contain p_k. □

Definition 3. *Let σ be a sequence of n requests. We say that an algorithm A is* LRU-like *on $\sigma[1, i-1] \cdot \sigma_i$ (or simply LRU-like on σ_i when σ is clear from context) if after serving $\sigma[1, i-1]$, it serves request σ_i as LRU would: namely if σ_i incurs a fault, given A's cache, A evicts the page requested the least recently in $\sigma[1, i]$ (or does nothing if σ_i incurs a hit).*

We aim to show that for every online algorithm A, we have $LRU \preceq^f_{b,n} A$. The straightforward approach would be to explicitly define an appropriate bijection, however this appears to be very complicated. Instead, we will show that starting with A, we can define a series of appropriate algorithms $B_1, B_2, \dots B_l$ such that $A \equiv B_1 \succeq^f_{b,n} \dots \succeq^f_{b,n} B_i \succeq^f_{b,n} \dots \succeq^f_{b,n} B_l \equiv LRU$. The result then follows from transitivity of the "$\preceq^f_{b,n}$" relation. We define each algorithm in the series in such a way that it comes "closer and closer" to the statement of the LRU algorithm itself (see Lemma 2).

Lemma 2. *Consider the set $\mathcal{I}^f_n$ of all request sequences of size n consistent with f, and a fixed integer $j \leq n$. Suppose that A is an online algorithm with the property that for every request sequence $\sigma = \sigma_1 \dots \sigma_n \in \mathcal{I}^f_n$, A is LRU-like on $\sigma[1,k] \cdot \sigma_{k+1}$, for all $k \geq j+1$. Then there exists an online algorithm B with the following properties:*

1. *For every input $\sigma \in \mathcal{I}^f_n$, B makes the same decisions as A on the first j requests of σ (i.e., A and B make the same eviction decisions for each fault due to requests in $\sigma[1,j]$).*
2. *For every input $\sigma \in \mathcal{I}^f_n$, B is LRU-like on $\sigma[1,j] \cdot \sigma_{j+1}$.*
3. *$B \preceq^f_{b,n} A$.*

Informally, Lemma 2 states that for any algorithm A which may make a non-LRU-like decision when considering the $(j+1)$-th request of some sequence in $\mathcal{I}^f_n$ (but will always make LRU-like decisions throughout the remaining $n - j - 1$ requests) one can define a new algorithm B which behaves as A up to the first j requests of any sequence in $\mathcal{I}^f_n$, and will always make an LRU-like decision on the $(j+1)$-th request; furthermore B is not worse than A in terms of bijective analysis.

Proof. We first describe the decisions of B on a request sequence $\sigma = \sigma_1 \dots \sigma_n \in \mathcal{I}^f_n$. For simplicity of notation, define $\sigma' = \sigma[1,j]$, $p = \sigma_{j+1}$ and $\sigma'' = \sigma[j+2, n]$ (σ' or σ'' may be the empty sequence). We introduce the concept of a *tag-based* algorithm to give a concise statement of B. Suppose that after serving $\sigma'p$, B assigns a set T of *tags* to each page in its cache (a tag is an integer). In particular, for every page p' in its cache, let $\mathrm{tag}_B[p', \sigma'p]$ denote the tag assigned to p', right after B has served $\sigma'p$. We say that B is tag-based wrt T on σ'' if it processes each request σ_l (with $l \geq j+2$) in σ'' according to the following rule: on the event of a fault, it will first evict from its cache the page of smallest tag; then it will update the tag of page σ_l to the index of its current request (namely l). Otherwise (on the event of a hit), B only updates the tag of σ_l to l.

We now define the actions of B. First, we require that B makes the same decisions as A on all requests in σ'. Next, if A makes an LRU-like decision on $\sigma' \cdot p$, then B makes the same LRU-like decision as A, as well as the same decisions on any request in σ'' as A. If this is not the case, however, then there must exist a pair of pages $p_i, p_k \in P$ with $p_i \neq p_k$ such that when p is requested, A evicts p_i from its cache, whereas p_k is the least recently requested page in σ'. We describe the subsequent actions of B by requiring first that B evicts p_k, then assigns tags to pages in its cache according to the following rules:

$$\mathrm{tag}_B[p_i, \sigma'p] \leftarrow \mathrm{last}[p_k, \sigma'] \quad \text{and} \quad \mathrm{tag}_B[p', \sigma'p] \leftarrow \mathrm{last}[p', \sigma'],$$

for all pages $p' \neq p_i$ in the cache of B after the request for p has been served. Here, we use the notation $\mathrm{last}[p_k, \sigma']$ to denote the index of the last access of page p_k in σ'. From that point onwards, we require that B is tag-based on σ'' for tags as defined above. It is worth pointing out that B is truly an online algorithm, since it does not use know the future when serving its requests.

Here is the intuition behind the statement of B. Since A is LRU-like on all σ_k, with $k \geq j+2$, it is also tag-based on σ'', for the set of tags $\mathrm{tag}_A[p', \sigma'p] = \mathrm{last}[p', \sigma p]$ (so the tag is the timestamp of the last request of the page). In view of this, B evicts p_k instead of p_i (thus making an LRU-like decision) and effectively demotes the timestamp of p_i so as to make p_i appear as if it was the least-recently used page at the time p is requested.

By construction, B satisfies properties (1) and (2) of the statement of the lemma. It remains then to show property (3). For this define the mapping π as follows:

$$\pi(\sigma) = \begin{cases} \sigma' p \overline{\sigma''} & \text{if } \sigma' p \overline{\sigma''} \text{ is consistent with } f \text{ and } A \text{ does not make an LRU-like decision on } \sigma' p, \\ \sigma & \text{otherwise.} \end{cases}$$

Here, $\overline{\sigma''}$ denotes the complement of σ'' with respect to i and k (note that i and k depend on $\sigma'p$). Observe that the composition $\pi \circ \pi$ is the identity map so π is a bijection (see Appendix B).

We need to show that for every $\sigma \in \mathcal{I}_n^f$, $A(\sigma) \geq B(\pi(\sigma))$. Suffices to consider only the case that A does not make an LRU-like decision on $\sigma'p$. Proposition 1 and Proposition 2 address the two possibilities (depending on whether or not $\pi(\sigma) = \sigma$). Since the proofs proceed by induction, we first require some formal definitions concerning the cache contents of A and B, as they serve σ and $\pi(\sigma)$, respectively.

We define by $C[A, \sigma'p]$ the contents of the cache configuration of algorithm A after it has served the sequence $\sigma'p$: this configuration consists of the set $P[A, \sigma'p]$ of pages in said cache, as well as assigned tags $\mathrm{tag}_A[p', \sigma'p]$ equal to $\mathrm{last}[p', \sigma'p]$ for all $p' \in P[A, \sigma'p]$. In line with an earlier observation, we choose these tags so as to make A behave as both an LRU-like algorithm and tag-based algorithm for all requests in σ'' (thus making it easy to compare A to B). This is because A updates the tags of pages in σ'' as a tag-based algorithm would do for pages in σ''. More generally, denote by $C[A, \sigma[1,t]]$ with $t \geq j+1$ the configuration which consists of pages $P[A, \sigma[1,t]]$ in A's cache, together with the corresponding tags, after A has served the sequence $\sigma[1,t]$. The complement of $C[A, \sigma[1,t]]$ with respect to i and k, denoted by $\overline{C}[A, \sigma[1,t]]$ is then defined as the cache configuration in which: i) the set of pages in the cache is the set $\overline{P[A, \sigma[1,t]]}$; and ii) tags are as in $C[A, \sigma[1,t]]$, with the exception that if $p_i \in \overline{P[A, \sigma[1,t]]}$ (resp. if $p_k \in \overline{P[A, \sigma[1,t]]}$) then its tag in $\overline{C}[A, \sigma[1,t]]$ is the tag of p_k in $C[A, \sigma[1,t]]$ (resp. the tag of p_i in $C[A, \sigma[1,t]]$).

Proposition 1. *If $\sigma' p \overline{\sigma''}$ is consistent with f then $B(\pi(\sigma)) = A(\sigma)$.*

Proof sketch. We need to show that for all $j+1 \leq l \leq n$, algorithm B satisfies the following properties: i) $C[A, \sigma[1,l]] = \overline{C}[B, \pi(\sigma)[1,l]]$, and ii) A incurs a fault on request σ_l if and only if B incurs a fault on request $\pi(\sigma)_l$. The proof proceeds by induction on l (see Appendix B). $\square$

Proposition 2. *If $\sigma' p \overline{\sigma''}$ is not consistent with f, then $B(\pi(\sigma)) \leq A(\sigma)$.*

Proof. From the construction of B, it is clear that $B(\pi(\sigma)[1, j+1]) = A(\sigma[1, j+1])$. Moreover, from the initial choice of tags, $C[A, \sigma[1, j+1]] = \overline{C}[B, \pi(\sigma)[1, j+1]]$. More precisely, $C[A, \sigma[1, j+1]]$ and $C[B, \pi(\sigma)[1, j+1]]$ have identical page sets, with the only difference that the former contains p_k whereas the latter contains p_i. Since $\sigma' p \overline{\sigma''}$ is not consistent with f, Lemma 1 dictates that p_i must appear in $\sigma[j+2, n]$, and if $\sigma_t = p_i$ for some $t \geq j+2$ is the earliest request of p_i in $\sigma[j+2, n]$, p_k is not requested in $\sigma[j+2, t]$. Suffices then to consider the following two cases:

- Case 1: A incurs no fault in $\sigma[j+2, t-1]$. In this case it follows that B will likewise incur no fault in $\pi(\sigma)[j+2, t-1]$. Upon request $\sigma_t = p_i$, A incurs a fault and makes an LRU-like eviction, namely it evicts page p_k, replaces it with page p_i and sets tag$[p_i] = t$. On the other hand, B incurs a hit and also sets the tag of p_i to t. At that point, the cache configurations of the two algorithms are identical (including the page tags), or more formally $C[A, \sigma[1, t]] = C[B, \pi(\sigma)[1, t]]$. In addition, A is tag-based on each request in $\sigma[t, n]$, and B is tag-based on each request in $\pi(\sigma)[t, n] = \sigma[t, n]$. We conclude that the actions of A on $\sigma[t+1, n]$ are identical to the actions of B on $\pi(\sigma)[t+1, n]$, request-by-request, and thus $B(\pi(\sigma)) < A(\sigma)$.
- Case 2: A incurs a fault in $\sigma[j+2, t-1]$. This case is similar to Case 1 (see Appendix B). $\square$

Proposition 1 and Proposition 2 conclude the proof of the Lemma. $\square$

For any given algorithm A, we will now repeatedly use Lemma 2 in order to construct a new algorithm, which makes LRU-like decisions on suffixes of any request sequence, and is no worse than A. Denote by $\mathcal{B}_i$ the class of algorithms that make LRU-like decisions on the last i requests of every request sequence $\sigma \in \mathcal{I}_n^f$.

Lemma 3. *For every algorithm A there exists an algorithm $B_i \in \mathcal{B}_i$ such that $B_i \preceq_{b,n}^f A$, and for every request sequence $\sigma \in \mathcal{I}_n^f$, B_i makes the same decisions as A in the first $n-i$ requests in σ.*

Proof. By induction on i. The lemma is vacuously true for $i = 0$. Let $B_i \in \mathcal{B}_i$ be such that $B_i \preceq_{b,n}^f A$, and for any request sequence $\sigma \in \mathcal{I}_n^f$, B_i makes the same decisions as A in the first $n-i$ requests in σ. We will show that the claim holds for $i+1$ as well. From Lemma 2 there exists an algorithm B such that $B \preceq_{b,n}^f B_i$, and for every $\sigma \in \mathcal{I}_n^f$, B makes an LRU-like decision on request σ_{n-i}, and the same decisions as B_i on the first $n-i-1$ requests in σ.

Note that B does not necessarily make LRU-like decisions for requests in $\sigma[n-i+1, n]$; however, we can apply the induction hypothesis, and again argue that there exists an algorithm $B'_i \in \mathcal{B}_i$ with the properties that $B'_i \preceq_{b,n}^f B$, and for every $\sigma \in \mathcal{I}_n^f$, B'_i makes the same decisions as B on the first $n-i$ requests of σ, and LRU-like decisions on the last i requests of σ. But then $B'_i \in \mathcal{B}_{i+1}$, and again from induction hypothesis, B'_i makes the same decisions as A in the first $n-i-1$ requests of $\sigma \in \mathcal{I}_n^f$, hence the lemma holds for $i+1$, which completes the proof. $\square$

Theorem 1. *For any online paging algorithm A, and every n, we have $LRU \preceq_{b,n}^f A$.*

Proof. For every online algorithm A, Lemma 3 guarantees the existence of an algorithm in $\mathcal{B}_n$ which makes LRU-like decisions on all n requests, given a request sequence $\sigma \in \mathcal{I}_n^f$, and is no worse than A. The only algorithm with this property is LRU itself. $\square$

4 Bijective analysis for list access

In this section we prove that MTF is optimal in the modified cost model. (Recall that in this cost formulation, accessing an item a which is the i-th item in the list requires time i, and the algorithm can rearrange the part of the list up to a without cost). In order to show this, we prove optimality of MTF within a model allowing a larger class of algorithms. More precisely, in the model in which accessing item a costs i, and in addition rearranging the entire list right after the access can be done at no cost. Interestingly, this generalization leads to a more accessible proof, and also implies optimality of MTF in the standard cost model.

Let $\{a_1, \dots, a_l\}$ denote the l items of a list, and σ denote a request sequence of items to be accessed. Also, let $L^j = \langle a_1^j, \dots, a_l^j \rangle$ denote the list produced by an algorithm A right after serving request σ_j. We say that L^j is *ordered according to* $\sigma[1, j]$ if $a_1^j, \dots, a_l^j$ are ordered by least recent access (i.e., using terminology of Section 3, we have[3] $\text{last}[a_1^j, \sigma[1, j]] \geq \dots \geq \text{last}[a_l^j, \sigma[1, j]]$).

A pair of elements $(a_{i_1}^j, a_{i_2}^j)$ in L^j, with $i_1 < i_2$, is said to be *ordered according to* $\sigma[1, j]$ if $a_{i_1}^j$ was requested more recently in $\sigma[1, j]$ than $a_{i_2}^j$, otherwise it is called an *inversion*.

Given the current request σ_j, we say that an algorithm performs an *ordering decision on* σ_j if after serving σ_j the list L^j produced by the algorithm is ordered according to $\sigma[1, j]$.

Lemma 4. *Consider the set $\mathcal{I}_n^f$ and a fixed integer $j \leq n$. Suppose that A is an online algorithm such that for every request sequence $\sigma \in \mathcal{I}_n^f$, A performs ordering decisions on all requests σ_k for $k \geq j + 1$. Suppose also that there exists a specific request sequence $r = r_1 \dots r_j \in \mathcal{I}_j^f$ such that upon serving r_j, A produces a list L^j in which the pair $(a_{k_1}^j, a_{k_2}^j)$ is not ordered according to r.*

Consider now algorithm B which serves any sequence $\sigma \in \mathcal{I}_n^f$ of the form $\sigma = r\sigma''$ by i) making decisions identical to A for the first $j - 1$ requests; ii) serving $\sigma_j = r_j$ as A does, with the exception that it exchanges items $a_{k_1}^j$ and $a_{k_2}^j$ in L^j; and iii) performs ordering decisions for all remaining requests $\sigma_{j+1} \dots \sigma_n$. For requests $\sigma \in \mathcal{I}_n^f$ for which $\sigma[1, j] \neq r$, B performs the same decisions as A on the whole sequence σ. Then B has the property that $B \preceq_{b,n}^f A$.

Proof. To show that $B \preceq_n^f A$, suffices to define an appropriate bijection π, satisfying Definition 1. For definitiveness, let $L_s^j = \langle a_1 \dots a_l \rangle$ be the list ordered according to $\sigma[1, j]$, which means that $a_{i_1} := a_{k_2}^j$ precedes $a_{i_2} := a_{k_1}^j$ in L_s^j. Let ϕ be the cyclic permutation of list items defined by:

$$\phi(a_m) = \begin{cases} a_{i_1} & \text{if } m = i_2, \\ a_{m+1} & \text{if } i_1 \leq m < i_2, \\ a_m & \text{otherwise.} \end{cases}$$

As a next step, for every k define $\pi' \colon \mathcal{I}_k \to \mathcal{I}_k$ to be the bijection

$$\pi'(\sigma_1 \dots \sigma_k) = \begin{cases} \phi(\sigma_1) \dots \phi(\sigma_k) & \text{if } \sigma_1 = a_{i_2}, \\ \phi^{-1}(\sigma_1) \dots \phi^{-1}(\sigma_k) & \text{if } \sigma_1 = a_{i_1}, \\ \sigma_1 \dots \sigma_k & \text{otherwise.} \end{cases}$$

We are now ready to define the required bijection $\pi \colon \mathcal{I}_n^f \to \mathcal{I}_n^f$. Given $\sigma \in \mathcal{I}_n^f$, we use the notation σ' (resp. σ'') to define the prefix of the j first requests (resp. suffix of the last $n - j$ requests) in σ, and thus can write $\sigma = \sigma'\sigma''$. Define π as

$$\pi(\sigma) = \begin{cases} \sigma' \cdot \pi'(\sigma'') & \text{if } \sigma' \cdot \pi'(\sigma'') \text{ is consistent with } f \text{ and } \sigma' = r, \\ \sigma & \text{otherwise.} \end{cases}$$

[3]Without loss of generality, items not accessed in $\sigma[1, j]$ have $\text{last}[\cdot, \sigma[1, j]]$ value equal to 0, breaking ties arbitrarily.

The intuition behind the definition of π is that $\pi'(\sigma'')$ is analogous to $\overline{\sigma''}$ which we used in the context of the paging problem. The purpose of $\overline{\sigma''}$ was to rectify the non-optimal eviction decision involving pages p_i and p_k. Similarly, the purpose of $\pi'(\sigma'')$ is to rectify the non-optimal inversion involving a_{i_1} and a_{i_2}. This is accomplished through ϕ which, not surprisingly, involves all items between a_{i_1} and a_{i_2} in the ordered list L_s^j.

Observe that $\pi' \circ \pi'$ is the identity and therefore $\pi \circ \pi$ is the identity as well. It follows that π is a bijection (Appendix B). It remains to show that for all sequences σ, $B(\pi(\sigma)) \leq A(\sigma)$. Since this is trivial when $\sigma' \neq r$, for the remainder of the proof we consider sequences σ such that $\sigma' = r$.

- *Case 1:* $\pi(\sigma) \neq \sigma$. In this case, from the definition of π we have that $\sigma_{j+1} \in \{a_{i_1}, a_{i_2}\}$. The following proposition establishes that in this case, $B(\pi(\sigma)) = A(\sigma)$.

Proposition 3 (Appendix B). *Let $\langle a_1^t \dots a_l^t \rangle$ denote the list produced by A after serving sequence $\sigma[1,t]$, for $j < t \leq n$. Then the list produced by B upon serving $\pi(\sigma)[1,t]$ is $\langle \phi(a_1^t) \dots \phi(a_l^t) \rangle$, if $\sigma_{j+1} = a_{i_2}$, and $\langle \phi^{-1}(a_1^t) \dots \phi^{-1}(a_l^t) \rangle$, if $\sigma_{j+1} = a_{i_1}$. Furthermore, $A(\sigma[1,t]) = B(\pi(\sigma)[1,t])$.*

- *Case 2:* $\pi(\sigma) = \sigma$. If $\sigma_{j+1} \notin \{a_{i_1}, a_{i_2}\}$, then both A and B incur the same cost in serving $\sigma_{j+1} = \pi(\sigma)_{j+1}$. From that point onwards, the list configuration right after A serves request σ_t ($t \geq j+1$) is identical to the list configuration after B serves $\pi(\sigma)_t = \sigma_t$, since both algorithms make ordering-decisions thereafter. Therefore, $B(\pi(\sigma)) = B(\sigma) = A(\sigma)$. If $\sigma_{j+1} = a_{i_1}$ then algorithm B pays less to serve σ_{j+1} compared to algorithm A. Afterwards, once again, A and B maintain the same lists throughout, threfore, $B(\pi(\sigma)) < A(\sigma)$. Thus, it only remains to consider the case $\sigma_{j+1} = a_{i_2}$. We show that this case cannot occur, in the sense that if $\sigma_{j+1} = a_{i_2}$ then $\sigma'\pi(\sigma'')$ must be consistent with f (hence Case 1 would apply instead). By way of contradiction, suppose $\sigma_{j+1} = a_{i_2}$: we will prove that in any window $[t_1, t_2]$ there are at least as many distinct items in $\sigma[t_1, t_2]$ as in $\pi(\sigma)[t_1, t_2]$, hence $\pi(\sigma)$ must be consistent with f. If $t_1 > j$ or $t_2 \leq j$ this is straightforward. So assume $t_1 \leq j < t_2$. Recall that $\langle a_1 \dots a_l \rangle$ denotes the list ordered according to $\sigma[1,j]$. This means that the set of distinct items in $\sigma[t_1, j]$ is of the form $D = \{a_1, \dots, a_k\}$ for some k. Since $\sigma_{j+1} = a_{i_2}$ we know that $\pi(\sigma) = \sigma'\pi'(\sigma'') = \sigma[1,j]\phi(\sigma_{j+1}) \dots \phi(\sigma_n)$.

We first claim that if a is such that $a \in D$ and $\phi(a) \notin D$ then it must be that $a = a_k$ with $k \neq i_2$ and $\phi(a) = a_{k+1}$. This follows from the following facts: i) for every item a_m, with $m \in [1, l]$, we have $\phi(a_m) \in \{a_m, a_{m+1}, a_{i_1}\}$; and ii) if $a = a_{i_2} \in D$ then $\phi(a) = a_{i_1} \in D$ since $i_1 < i_2$.

Let D_1 (resp. D_2) denote the set of distinct pages in $\sigma[j+1, t]$ (resp. $\pi(\sigma)[j+1, t]$) which are not in D. Suffices then to show that $|D_2| \leq |D_1|$.

Using the previous claim we can further deduce that if item a is such that $a \notin D_1$ and $\phi(a) \in D_2$ then once again we have $a = a_k$ with $k \neq i_2$ and $\phi(a) = a_{k+1}$. This is because $\phi(a) \in D_2$ implies $\phi(a) \notin D$ and that $\phi(a)$ occurs in $\pi(\sigma)[j+1, t]$, hence a must occur in $\sigma[j+1, t]$. Since $a \notin D_1$ we conclude that $a \in D$ and we can apply the previous claim.

Suppose then that a_k occurs in $\sigma[j+1, t]$, and that $\phi(a_k) = a_{k+1} \in D_2$ (since otherwise for every item $\phi(a) \in D_2$ we have $a \in D_1$ and therefore $|D_2| \leq |D_1|$.) This implies that $i_1 \leq k < i_2$, thus $a_{i_1} \in D$ and $a_{i_2} \notin D$. We then observe that item a_{i_2} has the property that $a_{i_2} \in D_1$, but $\phi(a_{i_2}) \notin D_2$. In plain words, when comparing the set of distinct items in $\sigma[t_1, t_2]$ and $\pi(\sigma)[t_1, t_2]$, if it so happens that $\pi(\sigma)[t_1, t_2]$ "gains" a new item (compared to $\sigma[t_1, t_2]$) due to the presence of a_{k+1}, then for sure it will "lose" one, compared to $\sigma[t_1, t_2]$, due to the substitution of a_{i_2} by a_{i_1}.

We conclude that σ_{j+1} cannot be a_{i_2}, thus $B(\pi(\sigma)) \leq A(\sigma)$ in Case 2, and the lemma follows. ☐

Lemma 5. *Suppose that A is an online algorithm with the property that for every sequence $\sigma_1, \dots \sigma_n \in \mathcal{I}_n^f$, A performs an ordering decision on request σ_k for all $k > j+1$. Then there exists an online algorithm B with the following properties: (i) For every sequence $\sigma \in \mathcal{I}_n^f$, B makes the same decisions as A on the first j requests of σ, and makes ordering decisions on all remaining requests σ_k with $k \geq j+1$; and (ii) $B \preceq_{b,n}^f A$.*

Proof. Apply Lemma 4 inductively over the total number of inversions incurred by A on all sequences of length $j + 1$ (see Appendix B for details). $\square$

Theorem 2. *For any online algorithm A, and every n, MTF* $\preceq^f_{b,n}$ *A.*

Proof. Using Lemma 5, we conclude by induction that for every $i \leq n$, there exists an algorithm B_i such that B_i performs ordering decisions on the last i requests of every sequence in $\mathcal{I}^f_n$ and $B_i \preceq^f_{b,n} A$. Since B_n is MTF itself, the theorem follows. $\square$

Recall that, from the earlier discussion, Theorem 2 applies in a general model and implies optimality of MTF in both the standard and modified cost models.

5 Extensions

The proofs of Theorem 1 and Theorem 2 are presented in terms of the the *Max-model* of locality of reference (see Section 2). However, the results extend straightforwardly to the *Average-model*, in which $f(w)$ reflects the average number of faults over all windows of size w. In fact, we observe that Lemma 1 as well as the arguments in Case 2 of Lemma 4 carry over to the Average-model, and this is all that is required for the lemmas to be applicable in this more general model.

Even more strongly, one observes that we can derive the optimality of LRU under any framework in which sequences with locality of reference belong in a set $L_n \subseteq \mathcal{I}_n$, provided that the statement of Lemma 1 is satisfied, i.e., by substituting "consistency with f" by "membership in the set L_n". Hence for every such set L_n of request sequences, LRU is still optimal according to bijective analysis over L_n. (See Appendix B for details).

6 Future work

In future work we intend to investigate further applications of bijective analysis. In very recent work Boyar, Irani and Larsen [11] studied the two-server problem on three collinear points under various measures, including bijective analysis. But can the latter be applied successfully into a broader class of problems? Since bijective analysis is significantly less amenable than, say, competitive analysis, a more relaxed measure may be desirable. For instance, can we formalize a statement of the form "algorithm A is at most two times worse than any other algorithm according to bijective analysis."? Can we extend this type of analysis to randomized algorithms?

In a more concrete setting, Albers and Mitzenmacher [2] study the list update problem under the assumption that request sequences are generated by a discrete memoryless source according to a probability distribution, and show that algorithm Timestamp(0) performs better than MTF in such a setting. Interestingly, on strings processed by the Burrows-Wheeler transform, i.e., strings with high locality of reference, MTF still appears to perform better in [2]. In a similar vein, Boyar *et al.* [9] argue that a variant of LRU, namely algorithm LRU-2, is better than LRU according to the relative worst-order ratio. Consider then the setting in which the set of allowable sequences is the set of all sequences consistent with a given frequency pattern. Can we show separation results based on bijective analysis? We observe that since the latter definition of allowable sequences is not precisely the set $\mathcal{I}^f_n$, it would be very interesting to show that variants of LRU and MTF which are biased towards frequent requests (such as LRU-2 and Timestamp(0)) are in fact better algorithms.

References

[1] Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *JCSS: Journal of Computer and System Sciences*, 70, 2005.

[2] Susanne Albers and Michael Mitzenmacher. Average case analyses of list update algorithms, with applications to data compression. *Algorithmica*, 21(3):312–329, 1998.

[3] Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. On the separation and equivalence of paging strategies. In *Proceedings of the he Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 229–237, 2007.

[4] Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. List update with locality of reference. In *Proceedings of the Eighth Latin American Theoretical Informatics Symposium*, pages 399–410, 2008.

[5] Luca Becchetti. Modeling locality: A probabilistic analysis of LRU and FWF. In *Proceedings of the 12th European Symposium on Algorithms (ESA)*, pages 98–109, 2004.

[6] Shai Ben-David and Allan Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994.

[7] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[8] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50:244–258, 1995.

[9] Joan Boyar, Martin R. Ehmsen, and Kim S. Larsen. Theoretical evidence for the superiority of lru-2 over lru for the paging problem. In *Proceedings of the 4th International Workshop on Approximation and Online Algorithms (WAOA)*, pages 95–107, 2006.

[10] Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. The relative worst-order ratio applied to paging. *J. Comput. Syst. Sci.*, 73(5):818–843, 2007.

[11] Joan Boyar, Sandy Irani, and Kim S. Larsen. A comparison of performance measures for online algorithms, 2008. Availabble from arXiv.org, number 0806.0983v1.

[12] Joan Boyar, Kim S. Larsen, and Morten N. Nielsen. The accommodating function: A generalization of the competitive ratio. *SIAM J. Comput.*, 31(1):233–258, 2001.

[13] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, DEC SRC, 1994.

[14] Marek Chrobak and John Noga. LRU is better than FIFO. *Algorithmica*, 23(2):180–185, 1999.

[15] Peter J. Denning. The working set model for program behaviour. *Communications of the ACM*, 11(5), May 1968.

[16] Reza Dorrigiv and Alejandro López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 36(3):67–81, September 2005.

[17] Amos Fiat and Gerhard J. Woeginger. *Online Algorithms—The State of the Art*, volume 1442 of *LNCS*. Springer-Verlag, 1998.

[18] James H. Hester and Daniel S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17(3):295–311, 1985.

[19] i. H. Witten and T. Bell. The Calgary/Canterbury text compression corpus. Anonymous ftp from `ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus`.

[20] Haim Kaplan, Shir Landau, and Elad Verbin. A simpler analysis of burrows-wheeler based compression. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching*, pages 282–293, 2006.

[21] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[22] Anna R. Karlin, Steven J. Phillips, and Prabhakar Raghavan. Markov paging. *SIAM J. Comput.*, 30(3):906–922, 2000.

[23] Claire Kenyon. Best-fit bin-packing with random order. In *In Proceedings of the Seventh ACM-SIAM Symposium on Discrete Algorithms*, pages 359–364, 1996.

[24] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. *SIAM J. Comput.*, 30:300–317, 2000.

[25] C. Martínez and R. Roura. The competitiveness of the move-to-front rule. *Theoretical Computer Science*, 1–2(242):313–325, 2000.

[26] James Ian Munro. On the competitiveness of linear search. In *Proceedings of the Eighth European Symposium on Algorithms*, pages 338–325, 2000.

[27] Konstantinos Panagiotou and Alexander Souza. On adequate performance measures for paging. In *Proceedings of the thirty-eighth Annual ACM Symposium on Theory of computing*, pages 487–496, 2006.

[28] Frank Schulz. Two new families of list update algorithms. In *Proceedings of the 9th International Symposium on Algorithms and Computation*, pages 99–108, 1998.

[29] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[30] Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28:202–208, 1985.

[31] Eric Torng. A unified analysis of paging and caching. *Algorithmica*, 20(2):175–200, 1998.

[32] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, June 1994.

[33] Neal E. Young. Bounding the diffuse adversary. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 420–425, 1998.

[34] Neal E. Young. On-line paging against adversarially biased random inputs. *J. Algorithms*, 37(1):218–235, 2000.

[35] Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

Appendix

A A comparison of measures for the analysis of online paging algorithms

Measure	**Positive results**	**Negative results**	**Comments**
Competitive analysis [30]	applicable to a variety of problems and settings	all marking algorithms are equivalent	standard model of analysis
Loose competitiveness [32] [35]	more realistic ratios for paging algorithms		
Adequate Performance Measure [27]	more realistic ratios for paging algorithms		
Diffuse adversary [24] [33] [34]	more realistic ratios for paging algorithms LRU<FWF, LRU<FIFO		probabilistic analysis
Access Graph [8]	LRU $\leq$ FIFO		
Max/Max ratio [6]	direct comparison, influence of lookahead	LRU=FIFO=FWF	
Relative worst order ratio [10] [9]	direct comparison, influence of lookahead, LRU $<$ FWF, LRU-2 $<$ LRU	LRU=FIFO	
Concave analysis [1]	LRU $<$ FIFO,FWF		fault-rate cost model
Torng's model [31]	more realistic ratios, influence of lookahead	LRU=FIFO=FWF	uses full access cost model
Bijective Analysis [3] [4]	direct comparison, influence of lookahead, FWF<LRU	all lazy algorithms are equivalent	No assumptions of locality of reference
Average analysis + Concave analysis [3] [4]	LRU is the sole optimal paging algorithm		Uses the locality of reference model of [1]

Table 1: Known measures for the analysis of online paging algorithms.

B Omitted Proofs

The mapping π is bijective:

• *For the paging problem (defined in the proof of Lemma 2).* If $\sigma' p \overline{\sigma''}$ is consistent with f and A does not make an LRU-like decision on $\sigma' p$, then A does not make an LRU-like decision on request p, when presented with $\pi(\sigma)$ as input either. Combined with the fact that the complement of $\overline{\sigma''}$ (wrt i,k) is σ'' itself, it then follows that $\pi(\pi(\sigma))$ is σ, in this case.

If either $\sigma' p \overline{\sigma''}$ is not consistent with f, or A does not make an LRU-like decision on $\sigma' p$, then from the definition of π, $\pi(\sigma) = \sigma$, hence $\pi(\pi(\sigma))$ is σ.

• *For the list access problem (defined in the proof of Lemma 4).* From the definitions of π' and ϕ, it is straightforward that $\pi'(\pi'(\sigma)) = \sigma$. Using this fact, we then deduce that $\pi(\pi(\sigma)) = \sigma$, for all σ by using an argument analogous to the one showing that the mapping we used for the paging problem is bijective (see above).

Proof of Proposition 1: Suppose, by induction on l, that the claim holds for $l < n$, we will show that it holds for $l+1$. Note that the claim is straightforward for $l = j+1$; in particular, observe that the actions of B on $\pi(\sigma)_{j+1}$ and the choice of initial tags guarantee that $C[A, \sigma[1, j+1]] = \overline{C}[B, \pi(\sigma)[1, j+1]]$. We consider cases concerning request σ_{l+1}.

• If $\sigma_{l+1} \notin \{p_i, p_k\}$, then $\overline{\sigma}_{l+1} = \sigma_{l+1}$. If σ_{l+1} is a hit for A, it is also a hit for B, and both A and B will only update the tag of page σ_{l+1} to $l+1$ in their corresponding caches. If σ_{l+1} results in a fault for A, then by the induction hypothesis concerning the cache configuration of B, $\overline{\sigma}_{l+1}$ will likewise result in a fault for B. In addition, A and B evict the same page from their cache, and set the tag of σ_{l+1} to $l+1$, thus the proposition holds for $l+1$.

• If $\sigma_{l+1} = p_i$, then $\overline{\sigma}_{l+1} = p_k$. Once again, we consider two cases concerning whether σ_{l+1} incurred a hit or fault for A. If the request resulted in a hit, then by induction hypothesis $p_k \in C[B, \pi(\sigma)[1, l]]$, thus $\overline{\sigma}_{l+1}$ is a hit for B. Furthermore, after serving request σ_{l+1}, A sets the tag of p_i equal to $l+1$, and likewise B sets the tag of p_k equal to $l+1$, thus $C[A, \sigma[1, l+1]] = \overline{C}[B, \pi(\sigma)[1, l+1]]$. If, on the other hand, σ_{l+1} incurred a fault for A, then from induction hypothesis $\overline{\sigma}_{l+1} = p_k \notin C[B, \pi(\sigma)[1, l]]$. At that point, A and B evict pages complementary wrt. to i and k, in order to bring pages p_i and p_k into their cache, respectively, and update the respective tags to $l+1$, which again guarantees that $C[A, \sigma[1, l+1]] = \overline{C}[B, \pi(\sigma)[1, l+1]]$.

• If $\sigma_{l+1} = p_k$, then we apply an argument symmetric to the case $\sigma_{l+1} = p_i$. □

Details for the proof of Proposition 2, Case 2. *A incurs a fault in $\sigma[j+2, t-1]$.* Let σ_l with $l \in [j+2, t-1]$ be the earliest request in $\sigma[j+2, t-1]$ on which A incurs a fault (recall that σ_l cannot be p_k). It is easy to see that every request in $\pi(\sigma)[j+2, l]$ is a hit for B, and that $C[A, \sigma[1, l-1]] = \overline{C}[B, \pi(\sigma)[1, l-1]]$. Upon request σ_l, A incurs a fault, evicts p_k (in an LRU-like decision), brings the page σ_l into the cache, and sets its tag to l. Since $\sigma_l \notin \{p_i, p_k\}$, it follows that B will likewise incur a fault in $\pi(\sigma)_l$, and replace p_i with σ_l in its cache in a tag-based eviction (also setting the tag of σ_l to l). These actions imply that $C[A, \sigma[1, l]] = C[B, \pi(\sigma)[1, l]]$. As argued in Case 1, it follows that the actions of A on $\sigma[l+1, n]$ are identical to the actions of B on $\pi(\sigma)[l+1, n]$, request-by-request, and thus $B(\pi(\sigma)) = A(\sigma)$.

Proof of Proposition 3. We show the statement by induction on t. We only treat the case in which $\sigma_{j+1} = a_{i_2}$; the case $\sigma_{j+1} = a_{i_1}$ is shown in a very similar manner. We start with the base case $t = j+1$. Since B behaves like A up to the j-the request and $\sigma[1, j] = \pi(\sigma)[1, j]$ we know that $A(\sigma[1, j]) = B(\pi(\sigma)[1, j])$. From the definition of B, the list produced by B after it has served $\sigma[1, j]$ is the list algorithm A produces after serving $\sigma[1, j]$, with a_{i_1} and a_{i_2} interchanged. Furthermore $\sigma[1, t] = \sigma[1, j]a_{i_2}$ and $\pi(\sigma)[1, t] = \sigma[1, j]\phi(a_{i_2}) = \sigma[1, j]a_{i_1}$. Algorithm A and algorithm B therefore pay the same cost to serve $\sigma_t = a_{i_2}$ and $\pi(\sigma)_t = a_{i_1}$ respectively. Recall that we denote by $L_s^j = \langle a_1 \dots a_l \rangle$ the list that is ordered according to $\sigma[1, j]$, then A produces the list $\langle a_{i_2} a_1 \dots a_{i_1} \dots \widehat{a_{i_2}} \dots a_l \rangle$ after serving σ_t and B produces the list

$\langle a_{i_1} a_1 \ldots \widehat{a_{i_1}} \ldots a_{i_2} \ldots a_l \rangle = \langle \phi(a_{i_2})\phi(a_1) \ldots \phi(a_{i_1}) \ldots \widehat{\phi(a_{i_2})} \ldots \phi(a_l) \rangle$ after serving $\pi(\sigma)_t$. Here, we denote by $\sigma_1 \ldots \widehat{\sigma_i} \ldots \sigma_n$ the sequence $\sigma_1 \ldots \sigma_{i-1}\sigma_{i+1} \ldots \sigma_n$ obtained by deleting σ_i from sequence $\sigma_1 \ldots \sigma_n$.

For the induction step assume the statement is true for $j < t < n$. Then $A(\sigma[1,t]) = B(\pi(\sigma[1,t]))$ and if σ_{t+1} is the k-th item in the list A produces after processing $\sigma[1,t]$ then $\pi(\sigma)_{t+1} = \phi(\sigma_{t+1})$ is the k-th element in the list B produced after processing $\pi(\sigma)[1,t]$, so $A(\sigma[1,t+1]) = B(\pi(\sigma)[1,t+1])$. Algorithm A then moves the k-th item in its list, namely σ_{t+1} to the front, and likewise algorithm B moves the k-th item in its list $\pi(\sigma)_{t+1} = \phi(\sigma_{t+1})$ to the front. Therefore, the list produced by B after serving σ_{t+1} is $\langle \phi(a_1^{t+1}) \ldots \phi(a_l^{t+1}) \rangle$, and the statement holds for $t+1$ as well. □

Details in the proof of Lemma 5. For an algorithm that performs ordering-decisions on requests σ_k for all $k > j+1$ define

$$inv(A) := \sum_{\sigma' \in \mathcal{I}^f_{j+1}} (\text{number of inversions in the list after serving } \sigma').$$

Let B be the algorithm with minimum $inv(B)$, and with the properties that i) $B \preceq^f_{b,n} A$; ii) B makes the same decisions on the first j requests as A; and iii) makes ordering decisions for $k > j+1$. Then B does not incur an inversion on any request σ_{j+1} (i.e. $inv(B) = 0$), since otherwise Lemma 4 would yield an algorithm B' with $inv(B') < inv(B)$ with the same properties. Therefore, since $inv(B) = 0$, algorithm B performs an ordering decision on the $(j+1)$-th request of every sequence $\sigma \in \mathcal{I}^f_n$.

Details on the Extensions (section 5).

• We argue that Lemma 1 is also applicable in the Average-model. If σ' is not consistent with f, then there is a window size w such that the average number of faults (over all subsequences of size w) in σ' exceeds $f(w)$. This implies that there exists a pair (j_1, j_2) such that the number of distinct pages in $\sigma[j_1, j_2]$ exceeds the number of distinct pages in $\sigma'[j_1, j_2]$. The proof then proceeds along the same lines as the proof of Lemma 1.

• Suppose that $L_n \subseteq \mathcal{I}_n$ represents the set of request sequences of size n with locality of reference. Suppose also that the following variant of Lemma 1 is shown to be true concerning L_n.

Lemma: Let $\sigma = \sigma_1 \ldots \sigma_n$ be a sequence of n requests with $\sigma \in L_n$. Let $t \leq n$ be an integer such that $\sigma[1,t]$ contains a request to p_i, and in addition, p_k does not appear in $\sigma[1,t]$ after the last request to p_i in $\sigma[1,t]$.

Let σ' denote the sequence $\sigma[1,t] \cdot \overline{\sigma}[t+1,n]$, and suppose that $\sigma' \notin L_n$. Then $\sigma[t+1,n]$ contains a request to p_i; furthermore, no request to p_k in $\sigma[t+1,n]$ occurs earlier than the first request to p_i in $\sigma[t+1,n]$.

We conclude then that in any model in which L_n represents the inputs of high locality of reference, Theorem 1 still applies. Informally, we observe that a set L_n complies with the statement of the above lemma, if it obeys the following property: if the sequence $\ldots p_k \ldots p_i \ldots p_k \ldots p_i \ldots$ is in L_n, then so is the sequence $\ldots p_k \ldots p_i \ldots p_i \ldots p_k \ldots$. Arguably, this is a property which one should expect from sequences with high locality of reference.

• For list access, we give a brief argument that shows that Theorem 2 applies to the Average model as well. The proof of Lemma 4, and in particular, Case 2, makes use of the following property:

Given $\sigma \in \mathcal{I}_n^f$, and a list configuration $\langle a_1, \ldots, a_l \rangle$ ordered according to $\sigma[1, t]$, if $\sigma_{t+1} = a_{i_2}$, then $\pi(\sigma)$ is also in $\mathcal{I}_n^f$. This holds for the average case as well, following the same reasoning as in the one presented in Case 2. In particular, any window in $\pi(\sigma)[t_1, t_2]$ contains no more distinct items than $\sigma[t_1, t_2]$.