

# Paging and List Update under Bijective Analysis\*

Spyros Angelopoulos<sup>†</sup>

Pascal Schweitzer<sup>‡</sup>

## 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, Favrholt 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 (deterministic) algorithms.

## 1 Introduction

In their seminal work, Sleator and Tarjan [35] 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 [25]) has led to the development of a mathematical theory of online algorithms (see [11] and [21] 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

---

\*A preliminary version of this paper will appear in the Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2009.

<sup>†</sup>Max-Planck-Institut für Informatik, Campus E1 4, 66123 Saarbrücken, Germany. [sangelop@mpi-inf.mpg.de](mailto:sangelop@mpi-inf.mpg.de)

<sup>‡</sup>Max-Planck-Institut für Informatik, Campus E1 4, 66123 Saarbrücken, Germany. [pascal@mpi-inf.mpg.de](mailto:pascal@mpi-inf.mpg.de)

(LRU). As shown in [35] and [25], all these strategies yield the same competitive ratio equal to the cache size (a result extended by [36] to the whole class of *marking* algorithms). However, in practice LRU (and some of its variants) are the preferred strategies for paging [34]. More specifically, experimental results suggest that the “empirical competitive ratio” of LRU is a small constant independent of the cache size [38].

A similar situation arises in the context of another fundamental online problem, namely *list update*. In one of their central results in [35], 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 by deterministic algorithms. Other algorithms, such as the Timestamp algorithm [3] (in particular, the Timestamp(0) algorithm) and the MRI family of algorithms [6] are also 2-competitive. 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 [35] (to which we refer as the *standard cost model*).

In subsequent work, Martínez and Roura [29] and Munro [30] 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 [5] 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”.

**Bijjective Analysis.** In this paper we study the performance of algorithms for paging and list update problems using *Bijjective Analysis*, a technique introduced by Angelopoulos, Dorrigiv and López-Ortiz [4] 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  $\sigma$ . For concreteness, consider cost-minimization problems.

**Definition 1** ([4]). The online algorithm  $A$  is *no worse* than the online algorithm  $B$  on inputs of size  $n$  according to bijjective analysis, if there exists a bijection  $\pi : \mathcal{I}_n \rightarrow \mathcal{I}_n$  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 bijjective 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 bijjective analysis if  $A \preceq_b B$  and  $B \not\preceq_b A$ .

We adopt bijjective analysis as our measure of choice since it has several advantages over the competitive ratio. More specifically, bijjective 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 [4] shows that LRU with lookahead is strictly better

(according to bijective analysis) than LRU without lookahead. In contrast, as argued in [9], 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<sup>1</sup>. 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 [4] 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 [4], 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, [4] 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, [5] 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 [4] and [5]. 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.

## 2 Preliminaries and related work

**Problem definitions and cost models.** We consider the *paging* problem in its standard version: Let  $P = \{p_1, \dots, p_N\}$  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.

---

<sup>1</sup>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.

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* [35], 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 [29] and Munro [30] observed that in a realistic setting, rearranging the first  $i$  items of the list according to a permutation of  $\{1, \dots, i\}$  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 [35] 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 [9]; the *diffuse adversary model* of Koutsoupias and Papadimitriou [28] and Young [39] [40]; *loose competitiveness* of Young [38] [41]; the *random order ratio* of Kenyon [27]; the *relative worst-order ratio* of Boyar, Favrholt and Larsen [14] and Boyar, Ehmsen and Larsen [13]; and the *accommodation function model* of Boyar, Larsen and Nielsen [16].

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 [12] (also in work by Chrobak and Noga [18]) and its generalization known as *the Markov paging model* due to Karlin, Phillips and Raghavan [26]; the *full access cost model* due to Torng [36]; the *random order ratio* of Kenyon [27]; the *adequate performance analysis* of Panagiotou and Souza [31]; and *probabilistic analysis* by Becchetti [8].

For a comprehensive summary of relevant results in this area, we refer the reader to the survey of Dorrigiv and López-Ortiz [20]. (See also Table 1 in the Appendix). Although previous work has shown separation results for certain specific algorithms, only average analysis has separated LRU and MTF from every other (deterministic) algorithm, to the best of our knowledge.

In our work we adopt an intuitive and well-established model of locality of reference due to Albers, Favrholt and Giel [1], which in turn is based on Denning’s concept of the *working set* [19]. 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<sup>2</sup>

---

<sup>2</sup>Albers *et al.* consider a slightly more restrictive class of functions called concave\* functions.

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 [22] [33], and efficient 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 [17] [24]. Experimental studies for the list update problem have been presented by Rivest [32], Bentley and McGeoch [10] and Bachrach, El-Yaniv and Reinstädler [7]. As observed in [2], the ranking of various algorithms appears to depend on the degree of the locality in the input.

Unlike the paging problem, there have been few formal studies of the phenomenon of locality in list update. Experiments over the Calgary Compression Corpus [37], which is a standard benchmark for file compression, show [5] 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. [1] can faithfully represent locality of reference in list accessing, at least in applications which arise from file compression. Similar to the paging problem, we will denote by  $\mathcal{I}_n^f$  the set of requests sequences (sequence of item accesses) which are consistent with a given concave function  $f$ . We are then interested in bijective and average analysis over request sequences from this set. In very recent work, Albers and Lauer [2] proposed a new model of locality, which is based in the concept of *runs*; here, a run is a subsequence of requests to the same list item. Within this model, they performed a comprehensive theoretical and experimental study of various list update algorithms. Their results confirm that Move-To-Front has excellent performance in inputs of high locality.

In terms of the above notation, [4] and [5] 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. For paging, in particular, we argue that the same result applies within broader and more permissive (but still natural) definitions of what constitutes locality of reference.

### 3 Bijective analysis for paging

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

**Definition 2.** Let  $p_i, p_k$  denote two distinct pages in  $P$ . Let  $\sigma_j$  denote the  $j$ -th request in some request sequence  $\sigma$ . We define the *complement of  $\sigma_j$  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  $\sigma = \sigma_1\sigma_2 \dots \sigma_n$  we also define  $\overline{\sigma}$  as the sequence  $\overline{\sigma_1} \dots \overline{\sigma_n}$ .

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

**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  $\sigma'$  denote the sequence  $\sigma[1, t] \cdot \overline{\sigma}[t+1, n]$ , and suppose that  $\sigma'$  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  $\sigma'$  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  $\omega$  of  $\sigma[t+1, n]$ ,  $\overline{\omega}$  has the same number of distinct pages as  $\omega$ . 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  $\sigma$  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  $p_k$ .  $\square$   $\square$

**Definition 3.** Let  $\sigma$  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  $\sigma_i$  when  $\sigma$  is clear from context) if after serving  $\sigma[1, i-1]$ , it serves request  $\sigma_i$  as LRU would: namely if  $\sigma_i$  incurs a fault, given  $A$ 's cache,  $A$  evicts the page requested the least recently in  $\sigma[1, i]$  (or does nothing if  $\sigma_i$  incurs a hit).

We aim to show that for every online algorithm  $A$ , we have  $LRU \preceq_{b,n}^f A$ . 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  $A \equiv B_1 \succeq_{b,n}^f \dots \succeq_{b,n}^f B_i \succeq_{b,n}^f \dots \succeq_{b,n}^f B_l \equiv LRU$ . The result then follows from transitivity of the " $\preceq_{b,n}^f$ " 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  $\mathcal{I}_n^f$  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  $\sigma = \sigma_1 \dots \sigma_n \in \mathcal{I}_n^f$ ,  $A$  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  $\sigma \in \mathcal{I}_n^f$ ,  $B$  makes the same decisions as  $A$  on the first  $j$  requests of  $\sigma$  (i.e.,  $A$  and  $B$  make the same eviction decisions for each fault due to requests in  $\sigma[1, j]$ ).
2. For every input  $\sigma \in \mathcal{I}_n^f$ ,  $B$  is LRU-like on  $\sigma[1, j] \cdot \sigma_{j+1}$ .
3.  $B \preceq_{b,n}^f A$ .

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  $\mathcal{I}_n^f$  (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  $\mathcal{I}_n^f$ , 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  $\sigma = \sigma_1 \dots \sigma_n \in \mathcal{I}_n^f$ . For simplicity of notation, define  $\sigma' = \sigma[1, j]$ ,  $p = \sigma_{j+1}$  and  $\sigma'' = \sigma[j+2, n]$  ( $\sigma'$  or  $\sigma''$  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  $\text{tag}_B[p', \sigma'p]$  denote the tag assigned to  $p'$ , right after  $B$  has served  $\sigma'p$ . We say that  $B$  is tag-based wrt  $T$  on  $\sigma''$  if it processes each request  $\sigma_l$  (with  $l \geq j + 2$ ) in  $\sigma''$  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  $\sigma_l$  to the index of its current request (namely  $l$ ). Otherwise (on the event of a hit),  $B$  only updates the tag of  $\sigma_l$  to  $l$ .

We now define the actions of  $B$ . First, we require that  $B$  makes the same decisions as  $A$  on all requests in  $\sigma'$ . 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  $\sigma''$  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  $\sigma'$ . 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:

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

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  $\text{last}[p_k, \sigma']$  to denote the index of the last access of page  $p_k$  in  $\sigma'$ . From that point onwards, we require that  $B$  is tag-based on  $\sigma''$  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  $\sigma_k$ , with  $k \geq j + 2$ , it is also tag-based on  $\sigma''$ , for the set of tags  $\text{tag}_A[p', \sigma'p] = \text{last}[p', \sigma p]$  (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  $\pi$  as follows:

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

Here,  $\overline{\sigma''}$  denotes the complement of  $\sigma''$  with respect to  $i$  and  $k$  (note that  $i$  and  $k$  depend on  $\sigma'p$ ). It is easy to show that the composition  $\pi \circ \pi$  is the identity map so  $\pi$  is a bijection: 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  $\sigma''$  itself, it then follows that  $\pi(\pi(\sigma))$  is  $\sigma$ , in this case. On the other hand, 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  $\pi$ ,  $\pi(\sigma) = \sigma$ , hence  $\pi(\pi(\sigma))$  is  $\sigma$ .

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  $\sigma$  and  $\pi(\sigma)$ , respectively. (The reader may want to skip the tedious details in the proofs of Proposition 1 and Proposition 2, on first read).

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  $\text{tag}_A[p', \sigma'p]$  equal to  $\text{last}[p', \sigma'p]$  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  $\sigma''$  (thus making it easy to compare  $A$  to  $B$ ). This is because  $A$  updates the tags of pages in  $\sigma''$  as a tag-based algorithm would do for pages in  $\sigma''$ . 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  $p_k \in P[A, \sigma[1, t]]$ ) 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.* 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  $\sigma_l$  if and only if  $B$  incurs a fault on request  $\pi(\sigma)_l$ . The proof proceeds by induction on  $l$ . Suppose then 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  $\sigma_{l+1}$ .

- If  $\sigma_{l+1} \notin \{p_i, p_k\}$ , then  $\overline{\sigma}_{l+1} = \sigma_{l+1}$ . If  $\sigma_{l+1}$  is a hit for  $A$ , it is also a hit for  $B$ , and both  $A$  and  $B$  will only update the tag of page  $\sigma_{l+1}$  to  $l+1$  in their corresponding caches. If  $\sigma_{l+1}$  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  $\sigma_{l+1}$  to  $l+1$ , thus the proposition holds for  $l+1$ .

- If  $\sigma_{l+1} = p_i$ , then  $\bar{\sigma}_{l+1} = p_k$ . Once again, we consider two cases concerning whether  $\sigma_{l+1}$  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  $\bar{\sigma}_{l+1}$  is a hit for  $B$ . Furthermore, after serving request  $\sigma_{l+1}$ ,  $A$  sets the tag of  $p_i$  equal to  $l + 1$ , and likewise  $B$  sets the tag of  $p_k$  equal to  $l + 1$ , thus  $C[A, \sigma[1, l + 1]] = \bar{C}[B, \pi(\sigma)[1, l + 1]]$ . If, on the other hand,  $\sigma_{l+1}$  incurred a fault for  $A$ , then from induction hypothesis  $\bar{\sigma}_{l+1} = p_k \notin C[B, \pi(\sigma)[1, l]]$ . 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  $C[A, \sigma[1, l + 1]] = \bar{C}[B, \pi(\sigma)[1, l + 1]]$ .
- If  $\sigma_{l+1} = p_k$ , then we apply an argument symmetric to the case  $\sigma_{l+1} = p_i$ . □

**Proposition 2.** *If  $\sigma'p\bar{\sigma}''$  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,  $C[A, \sigma[1, j + 1]] = \bar{C}[B, \pi(\sigma)[1, j + 1]]$ . 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  $\sigma'p\bar{\sigma}''$  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  $\text{tag}[p_i] = t$ . 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. More specifically,  $A$  incurs a fault in  $\sigma[j + 2, t - 1]$ . Let  $\sigma_l$  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  $\sigma_l$  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  $C[A, \sigma[1, l - 1]] = \bar{C}[B, \pi(\sigma)[1, l - 1]]$ . Upon request  $\sigma_l$ ,  $A$  incurs a fault, evicts  $p_k$  (in an LRU-like decision), brings the page  $\sigma_l$  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  $\sigma_l$  in its cache in a tag-based eviction (also setting the tag of  $\sigma_l$  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  $B(\pi(\sigma)) = A(\sigma)$ . □

Proposition 1 and Proposition 2 conclude the proof of the Lemma. □

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  $\sigma$ .*

*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  $\sigma$ . 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  $\sigma_{n-i}$ , and the same decisions as  $B_i$  on the first  $n - i - 1$  requests in  $\sigma$ .

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  $\sigma$ , and LRU-like decisions on the last  $i$  requests of  $\sigma$ . 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  $\sigma$  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  $\sigma_j$ . 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<sup>3</sup>  $\text{last}[a_1^j, \sigma[1, j]] \geq \dots \geq \text{last}[a_l^j, \sigma[1, j]]$ ).

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  $\sigma_j$ , we say that an algorithm performs an *ordering decision on*  $\sigma_j$  if after serving  $\sigma_j$  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  $\sigma_k$  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*

---

<sup>3</sup>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.

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  $\sigma$ . 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  $\pi$ , 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  $\phi$  be the cyclic permutation of list items defined by:

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

Figure 1 is a depiction of the permutation.

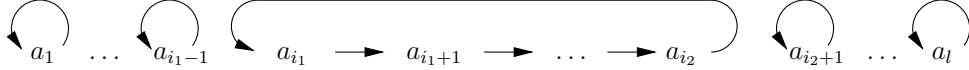


Figure 1: A depiction of the cyclic permutation  $\phi$ .

As a next step, for every  $k$  define  $\pi': \mathcal{I}_k \rightarrow \mathcal{I}_k$  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  $\pi: \mathcal{I}_n^f \rightarrow \mathcal{I}_n^f$ . Given  $\sigma \in \mathcal{I}_n^f$ , we use the notation  $\sigma'$  (resp.  $\sigma''$ ) to define the prefix of the  $j$  first requests (resp. suffix of the last  $n - j$  requests) in  $\sigma$ , and thus can write  $\sigma = \sigma' \sigma''$ . Define  $\pi$  as

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

The intuition behind the definition of  $\pi$  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  $\phi$  which, not surprisingly, involves all items between  $a_{i_1}$  and  $a_{i_2}$  in the ordered list  $L_s^j$ .

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

- *Case 1:*  $\pi(\sigma) \neq \sigma$ . In this case, from the definition of  $\pi$  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.** *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,  $A(\sigma[1, t]) = B(\pi(\sigma)[1, t])$ .*

*Proof.* 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  $j$ -th 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  $\sigma_t$  and  $B$  produces the list  $\langle a_{i_1}a_1 \dots \widehat{a_{i_1}} \dots a_{i_2} \dots a_l \rangle = \langle \phi(a_{i_2})\phi(a_1) \dots \phi(a_{i_1}) \dots \widehat{\phi(a_{i_2})} \dots \phi(a_l) \rangle$  after serving  $\pi(\sigma)_t$ . Here, we denote by  $\sigma_1 \dots \widehat{\sigma_i} \dots \sigma_n$  the sequence  $\sigma_1 \dots \sigma_{i-1}\sigma_{i+1} \dots \sigma_n$  obtained by deleting  $\sigma_i$  from sequence  $\sigma_1 \dots \sigma_n$ .

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  $\sigma_{t+1}$  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  $\sigma_{t+1}$  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  $\sigma_{t+1}$  is  $\langle \phi(a_1^{t+1}) \dots \phi(a_l^{t+1}) \rangle$ , and the statement holds for  $t+1$  as well.  $\square$

• *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  $\sigma_t$  ( $t \geq j+1$ ) 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  $\sigma_{j+1}$  compared to algorithm  $A$ . Afterwards, once again,  $A$  and  $B$  maintain the same lists throughout, therefore,  $B(\pi(\sigma)) < A(\sigma)$ . 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  $k$ . Also let  $D'$  be the set of distinct items in  $\sigma[j+1, t_2]$ . According to these definitions, the set of distinct items in the window  $\sigma[t_1, t_2]$  is the set  $D \cup D'$ .

Since  $\sigma_{j+1} = a_{i_2}$  we have  $\pi(\sigma) = \sigma' \pi'(\sigma'') = \sigma[1, j]\phi(\sigma_{j+1}) \dots \phi(\sigma_n)$ . Hence the set of distinct items in the window  $\pi(\sigma)[t_1, t_2]$  is the set  $D \cup \phi(D')$ , where  $\phi(D')$  is defined as the set  $\{\phi(a) \mid a \in D'\}$ . Figure 2 is an illustration of the definitions.

$$\begin{array}{ccccccc}
 & & \overbrace{D} & & \overbrace{D'} & & \\
 \sigma & = & \sigma_1 \dots \sigma_{t_1} \dots \sigma_j & a_{i_2} & \sigma_{j+2} \dots & a_{t_2} & \dots \sigma_l \\
 \updownarrow & & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\
 \pi(\sigma) & = & \sigma_1 \dots \sigma_{t_1} \dots \sigma_j & a_{i_1} \phi(\sigma_{j+2}) \dots \phi(a_{t_2}) & \dots & \phi(\sigma_l) & \\
 & & \underbrace{D} & & \underbrace{\phi(D')} & & 
 \end{array}$$

Figure 2: The figure depicts several definitions used in the proof.

Suffices now to show that  $|D \cup \phi(D')| \leq |D \cup D'|$ . To this end, we first show the following simple observation: For every  $i < k$ , if it is the case that  $a_i \in D \cap D'$ , then it is also the case that  $\phi(a_i) \in D \cap \phi(D')$ . We need to consider two cases: If  $i < i_1$  or  $i > i_2$ , then by definition  $\phi(a_i) = a_i$ , and the claim clearly follows. If, on the other hand,  $i_1 \leq i < i_2$ , then again by definition,  $\phi(a_i) = a_{i+1} \in D$ , since  $i < k$ . Thus it follows that  $\phi(a_i) \in D \cap \phi(D')$ .

Consider now item  $a_k$ . Using arguments as above, we can show that if  $k < i_1$  or  $k \geq i_2$ , then  $\phi(a_k) \in D \cap \phi(D')$ . In this case, the previous observation shows that  $|D \cap D'| \leq |D \cap \phi(D')|$ . Hence, it remains to consider the case  $i_1 \leq k < i_2$ . In this case, however,  $\phi(a_{i_2}) = a_{i_1} \in D$ , thus  $\phi(a_{i_2}) \in D \cap \phi(D')$ , whereas  $a_{i_2} \notin D$ , thus  $\phi(a_{i_2}) \notin D \cap D'$ . Combined with the previous observation, this proves that  $|D \cap D'| \leq |D \cap \phi(D')|$ .

In plain words, when comparing the set of distinct items in  $D \cap D'$  and  $D \cap \phi(D')$ , if it so happens that  $D \cap \phi(D')$  “looses” an item (compared to  $D \cap D'$ ) due to the presence of  $a_k$ , then for sure it will “gain” one, compared to compared to  $D \cap D'$ , because of the substitution of  $a_{i_2}$  by  $a_{i_1}$ .

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

**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  $\sigma_k$  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  $\sigma$ , and makes ordering decisions on all remaining requests  $\sigma_k$  with  $k \geq j + 1$ ; and (ii)  $B \preceq_{b,n}^f A$ .*

*Proof.* For an algorithm that performs ordering-decisions on requests  $\sigma_k$  for all  $k > j + 1$  define  $inv(A)$  as the sum, over all  $\sigma' \in \mathcal{I}_{j+1}^f$  of the number of inversions in the list after serving  $\sigma'$ .

Let  $B$  be the algorithm with minimum  $inv(B)$ , and with the properties that i)  $B \preceq_{b,n}^f A$ ; 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  $\sigma_{j+1}$  (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  $\sigma \in \mathcal{I}_n^f$ .  $\square$

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

*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  $\mathcal{I}_n^f$  and  $B_i \preceq_{b,n}^f A$ . 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  $L_n$ ”. More precisely, suppose that the set  $L_n$  is such that the following property holds.

**Property 1.** *Let  $\sigma = \sigma_1 \dots \sigma_n$  be a sequence of  $n$  requests which belongs 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  $\sigma'$  denote the sequence  $\sigma[1, t] \cdot \bar{\sigma}[t+1, n]$ , and suppose that  $\sigma'$  is not in  $L_n$ . 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  $\sigma[t+1, n]$ .*

It then follows (using the same arguments as in the proof of Lemma 2) that LRU is optimal among all paging algorithms. In other words, Property 1 identifies a structural property of typical sequences that LRU exploits, so as to achieve optimal performance. The intuitive interpretation of this property is similar to the interpretation of Lemma 1, in the context of sequences in  $\mathcal{I}_n^f$ .

## 6 Conclusions and future work

In this paper we provided strong theoretical justification of the superiority of MTF and LRU at the presence of locality of reference, by combining bijective analysis (as a measure of the comparison of online algorithms) with a formal model of locality of reference. Our work answers the open questions in [4] and [5]; we also believe that it manages to bridge (or at least significantly narrow) a long-known disconnect between theoretical analysis of paging and list update algorithms and empirical evaluation.

In future work we intend to investigate further applications of bijective analysis. In very recent work Boyar, Irani and Larsen [15] 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 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”?

A different direction is to extend bijective analysis to randomized algorithms. It is known that for both paging and list update, randomization helps in improving the performance of algorithms, and a variety of randomized algorithms have been proposed and analyzed. Very recently, Hiller and Vredeveld [23] use the notion of *stochastic dominance* in the context of probabilistic analysis of online bin coloring algorithms. We believe that both these two types of analysis could be useful in studying the effect of randomization in online computation, since they are closely related.

In a more concrete setting, Albers and Mitzenmacher [3] 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 [3]. For the paging problem, Boyar *et al.* [13] 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  $\mathcal{I}_n^f$ , 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.

## 7 Acknowledgments

We are thankful to Alejandro López-Ortiz and Reza Dorrigiv for earlier discussions on the problem. We also wish to thank Allan Borodin, Stefan Kratsch and Kurt Mehlhorn for their helpful comments.

## References

- [1] S. Albers, L. M. Favrholt, and O. Giel. On paging with locality of reference. *Journal of Computer and System Sciences*, 70(2), 2005.
- [2] S. Albers and S. Lauer. On list update with locality of reference. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 96–107, 2008.
- [3] S. Albers and M. Mitzenmacher. Average case analyses of list update algorithms, with applications to data compression. *Algorithmica*, 21(3):312–329, 1998.
- [4] S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. On the separation and equivalence of paging strategies. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 229–237, 2007.
- [5] S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. List update with locality of reference. In *Proceedings of the 8th Latin American Theoretical Informatics Symposium*, pages 399–410, 2008.
- [6] R. Bachrach and R. El-Yaniv. Online list accessing algorithms and their applications: Recent empirical evidence. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 43–62, 1997.
- [7] R. Bachrach, R. El-Yaniv and M. Reinstädler. On the competitive theory and practice of online list accessing algorithms. *Algorithmica*, 32, 201–245, 2002.
- [8] L. Becchetti. Modeling locality: A probabilistic analysis of LRU and FWF. In *Proceedings of the 12th European Symposium on Algorithms (ESA)*, pages 98–109, 2004.
- [9] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994.
- [10] J.L. Bentley and C.C. McGeoch. Amortized Analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28, 404–411, 1985.
- [11] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

- [12] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995.
- [13] J. Boyar, M. R. Ehmsen, and K. 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.
- [14] J. Boyar, L. M. Favrholdt, and K. S. Larsen. The relative worst-order ratio applied to paging. *Journal of Computer and System Sciences*, 73(5):818–843, 2007.
- [15] J. Boyar, S. Irani, and K. S. Larsen. A comparison of performance measures for online algorithms, 2008. Available from arXiv.org, number 0806.0983v1.
- [16] J. Boyar, K. S. Larsen, and M. N. Nielsen. The accommodating function: A generalization of the competitive ratio. *SIAM J. on Computing*, 31(1):233–258, 2001.
- [17] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, DEC SRC, 1994.
- [18] M. Chrobak and J. Noga. LRU is better than FIFO. *Algorithmica*, 23(2):180–185, 1999.
- [19] P. J. Denning. The working set model for program behaviour. *Communications of the ACM*, 11(5), 1968.
- [20] R. Dorrigiv and A. 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.
- [21] A. Fiat and G. J. Woeginger. *Online Algorithms—The State of the Art*, volume 1442 of *LNCS*. Springer-Verlag, 1998.
- [22] J. H. Hester and D. S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17(3):295–311, 1985.
- [23] B. Hiller and T. Vredeveld. Probabilistic analysis of online bin coloring algorithms via stochastic comparison. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA)*, pages 528–539, 2008.
- [24] H. Kaplan, S. Landau, and E. Verbin. A simpler analysis of burrows-wheeler based compression. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching*, pages 282–293, 2006.
- [25] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.
- [26] A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov paging. *SIAM J. on Computing*, 30(3):906–922, 2000.
- [27] C. Kenyon. Best-fit bin-packing with random order. In *In Proceedings of the Seventh ACM-SIAM Symposium on Discrete Algorithms*, pages 359–364, 1996.

- [28] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. *SIAM J. on Computing*, 30:300–317, 2000.
- [29] C. Martínez and R. Roura. The competitiveness of the move-to-front rule. *Theoretical Computer Science*, 1–2(242):313–325, 2000.
- [30] J. I. Munro. On the competitiveness of linear search. In *Proceedings of the 8th European Symposium on Algorithms*, pages 338–325, 2000.
- [31] K. Panagiotou and A. Souza. On adequate performance measures for paging. In *Proceedings of the 38th Annual ACM Symposium on Theory of computing*, pages 487–496, 2006.
- [32] R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, 19, 63–67, 1976.
- [33] F. Schulz. Two new families of list update algorithms. In *Proceedings of the 9th International Symposium on Algorithms and Computation*, pages 99–108, 1998.
- [34] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. John Wiley & Sons, Inc., New York, NY, USA, 2001.
- [35] D. D. Sleator and R. E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28:202–208, 1985.
- [36] E. Torng. A unified analysis of paging and caching. *Algorithmica*, 20(2):175–200, 1998.
- [37] I. H. Witten and T. Bell. The Calgary/Canterbury text compression corpus. Anonymous ftp from <ftp://ftp.cpsc.ucalgary.ca/pub/projects/>.
- [38] N. E. Young. The  $k$ -server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.
- [39] N. E. Young. Bounding the diffuse adversary. In *Proceedings of the 9th Annual ACM-SIAM symposium on Discrete algorithms*, pages 420–425, 1998.
- [40] N. E. Young. On-line paging against adversarially biased random inputs. *Journal of Algorithms*, 37(1):218–235, 2000.
- [41] N. 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 [35]                   | applicable to a variety of problems and settings                  | all marking algorithms are equivalent | standard model of analysis                  |
| Loose competitiveness [38] [41]             | more realistic ratios for paging algorithms                       |                                       |   |
| Adequate Performance Measure [31]           | more realistic ratios for paging algorithms                       |                                       |   |
| Diffuse adversary [28] [39] [40]            | more realistic ratios for paging algorithms LRU<FWF, LRU<FIFO     |                                       | probabilistic analysis                      |
| Access Graph [12]                           | LRU $\leq$ FIFO   |                                       |   |
| Max/Max ratio [9]                           | direct comparison, influence of lookahead                         | LRU=FIFO=FWF                          |   |
| Relative worst order ratio [14] [13]        | 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 [36]                          | more realistic ratios, influence of lookahead                     | LRU=FIFO=FWF                          | uses full access cost model                 |
| Bijjective Analysis [4] [5]                 | direct comparison, influence of lookahead, FWF<LRU                | all lazy algorithms are equivalent    | No assumptions of locality of reference     |
| Average analysis + Concave analysis [4] [5] | 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.