On the Separation and Equivalence of Paging Strategies and Other Online Algorithms

Spyros Angelopoulos, Reza Dorrigiv, Alejandro Lopez-Ortiz

On the separation and equivalence of paging strategies and other online algorithms [*]

Spyros Angelopoulos
CNRS and Sorbonne Université

Reza Dorrigiv
Samsung Research, Mountain View

Alejandro López-Ortiz
Univerity of Waterloo

Abstract

We introduce a new technique for the analysis of online algorithms, namely *bijective analysis*, that is based on pair-wise comparison of the costs incurred by the algorithms. Under this framework, an algorithm A is no worse than an algorithm B if there is a bijection π defined over all request sequences of a given size such that the cost of A on σ is no more than the cost of B on $B(\pi(\sigma))$. We also study a relaxation of bijective analysis, termed *average analysis*, in which we compare two algorithms based on their corresponding average costs over request sequences of a given size.

We apply these new techniques in the context of two fundamental online problems, namely *paging* and *list update*. For paging, we show that any two lazy online algorithms are equivalent under bijective analysis. This result demonstrates that, without further assumptions on characteristics of request sequences, it is unlikely, or even undesirable, to separate online paging algorithms based on their performance. However, once we restrict the set of request sequences to those exhibiting locality of reference, and in particular using a model of locality due to Albers, Favrholdt, and Giel [JCSS 2005], we demonstrate that Least-Recently-Used (LRU) is the unique optimal strategy according to average analysis. This is, to our knowledge, the first deterministic model to provide full theoretical backing to the empirical observation that LRU is preferable in practice. Concerning list update, we obtain similar conclusions, in terms of the bijective comparison of any two online algorithms, and in terms of the superiority (albeit not necessarily unique) of the Move-To-Front (MTF) heuristic in the presence of locality of reference.

1 Introduction

Paging is a fundamental problem in the context of analysis of online algorithms. A paging algorithm mediates between a slower memory and a faster memory, also called *cache*. Assuming a cache of size k, the paging strategy must decide which k memory pages to keep in the cache without the benefit of knowing in advance the *sequence* of upcoming page requests. Upon a request for a page, in the event the page is already in the cache the request is called a *hit* and is handled at no cost to the algorithm. On the other hand, in the event the request is not currently in the cache (namely, when

[*]This work is an extended, combined version of two papers: *On the separation and equivalence of paging strategies*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 229-237, and *List Update with Locality of Reference*, in Proceedings of the Eighth Latin American Theoretical Informatics Symposium, 2008, pp. 399-410.

a cache *miss* occurs) the online algorithm must decide irrevocably which page to evict, and then brings the requested page into the cache. The objective is to design efficient online algorithms, in the sense that on a given request sequence the total number of cache misses is as small as possible.

The best known theoretical framework for evaluating the performance of paging strategies (and online algorithms in general) is *competitive analysis*. The competitive ratio, first introduced formally by Sleator and Tarjan [51], has served as the standard measure for the study and classification of online algorithms. Assuming a cost-minimization problem (such as paging), an algorithm is said to be α-competitive if the cost of serving any specific request sequence does not exceed α times the optimal cost (up to some additive constant) of the best *offline* algorithm that knows the entire sequence. The competitive ratio has been applied to a variety of problems and settings, mainly due to its amenability to analysis: the measure is relatively simple to define, yet powerful enough to quantify the performance of online algorithms in a variety of settings. On the other hand, for the case of paging algorithms, competitive analysis has long been known to produce unrealistically pessimistic measures. More precisely, competitive analysis fails to distinguish between algorithms which differ vastly in performance in practical settings. Indeed, the first measure alternative to competitive analysis, namely resource augmentation, was proposed by Sleator and Tarjan in their original paper introducing competitive analysis [51].

As a concrete example, consider the well-studied paging strategies Flush-When-Full (FWF), Least-Recently-Used (LRU) and First-In-First-Out (FIFO). For the case of LRU and FIFO, Young established experimental values of the competitive ratio no larger than four [54]. In contrast, all three algorithms have a competitive ratio of (theoretical) value equal to the cache size, k. Furthermore, it has long been empirically established that LRU (and in particular, some of its more practical implementations) are among the most preferable paging strategies [50]. An additional drawback of competitive analysis is that finite lookahead yields no improvement in the performance of an online algorithm [16]. Once again, this is a rather counterintuitive conclusion: in practice, one expects that lookahead should improve performance, and that a "reasonable" theoretical measure should reflect this reality.

Such anomalies have been observed since the early days of competitive analysis, and there is a vast literature studying alternative proposals to the competitive analysis of online algorithms in general, and for the paging problem in particular, e.g., [3, 5, 14, 15, 19, 20, 22, 45, 42, 56, 52] (see the survey [32] for a discussion on performance measures). In general, known alternative approaches rely on one or more of the following: i) defining a new measure as substitute of the competitive ratio; ii) limiting the power of the adversary; iii) employing different definitions for the concept of the "cost" of an algorithm; iv) incorporating certain assumptions concerning the request sequences.

Note that competitive analysis uses the concept of an optimal offline algorithm as a baseline for comparing online algorithms. While this may be convenient, it is rather indirect: one could argue that in comparing two online algorithms, all we need to study is the relative cost of the algorithms on the request sequences. The approach we follow in this paper stems from this basic observation, and focuses not on a specific worst case request sequence, but rather on the performance of an algorithm on *all* possible sequences. We provide a formal definition of this intuitive observation, which is based on bijective mappings of the set of all possible request sequences of a given length onto itself. We term this technique *bijective analysis*. This form of analysis establishes a very strong relation between the compared algorithms which, however, may be difficult to prove or may even not exist. Thus, we also define a relaxation of bijective analysis based on the average cost incurred by an algorithm on all requests of the same length. We term this latter form of analysis *average* *analysis*. Formal definitions are provided in Section 3.

We also address the impact of a well-known phenomenon in the context of the paging problem, namely, *locality of reference*. It has been established that "real-life" sequences usually exhibit a high degree of locality of reference, and thus efficient algorithms tend to take advantage of this property. Informally, locality of reference suggests that the currently requested item is likely to be requested again in the near future. For the paging problem, several models for capturing locality of reference have been proposed [3, 14, 17, 40, 52] (more details are given in Section 2). In our work we adopt an intuitive model of locality of reference due to Albers, Favrholdt and Giel [3], termed *concave analysis*, which in turn is based on Denning's concept of the *working set* [29]. The reason is that the model is fairly inclusive, i.e., tends to incorporate any sequence that exhibits, to some degree, locality characteristics, but also is conceptually simple so as to facilitate the analysis based on the techniques of this work.

Our results We begin by showing that a very large class of natural paging strategies known as *lazy* algorithms are equivalent under bijective analysis. In contrast, we show that LRU is strictly better than FWF (note that the latter is not a lazy strategy). Both of these results describe natural, "to-be-expected" properties of the corresponding paging strategies which competitive analysis nevertheless fails to yield. The equivalence of lazy algorithms provides strong evidence of an inherent difficulty in separating algorithms in any general setting. In fact, it implies that in order to obtain a theoretical separation between paging algorithms we must either induce a partition of the space of request sequences (e.g. as in Albers et al. [3] and Borodin *et al.* [17]) or assume a distribution on the sequence space (e.g. as in Becchetti [14], Koutsoupias and Papadimitriou [42], Karlin *et al.* [40] and Young [55]). The latter group of approaches uses probabilistic assumptions on the sequence space. However, since we are interested in separating algorithms under a deterministic model, we adopt concave analysis as introduced by Albers *et al.*, which we then apply in the context of average analysis. Using this approach, we show formally our main result: namely that LRU is never outperformed in any possible subpartition on the request sequence space induced by concave analysis (c.f. Corollary 10), while it always outperforms *any other paging algorithm* in at least one subpartition of the request-sequence space (c.f. Theorem 11). This result proves separation between LRU and all other algorithms and provides theoretical backing to the observation that LRU is often preferable in practice.

We then apply the new technique to other problem domains. First, we provide concrete evidence that in the context of paging, lookahead is beneficial. Specifically, we show that under bijective analysis, LRU with lookahead as small as one (that is, the sequence is revealed to the algorithm as overlapping consecutive pairs of requests) is strictly better than LRU without any lookahead.

Second, we turn our attention to another fundamental problem in online computation, namely the *list update* problem. We first show that under the cost formulation of Martínez and Roura [43] and Munro [45] (to which we refer as the *modified cost model*) all online list update algorithms that do not use paid exchanges are equivalent according to bijective analysis. We then address the issue of locality of reference in the context of list update. In particular, we show how the model of Albers et al. [3] can be extended, so as to properly capture the effect of locality of reference in list-update applications related to compression algorithms. We provide experimental results obtained on the Calgary Corpus, which is frequently used as a standard benchmark for evaluating the performance of compression algorithms (and by extension for list update algorithms, e.g. [13]). We thus resolve the open problem posed by Hester and Hirschberg [35], in that we provide a theoretical model which captures the effect of locality in list update applications. Our main result proves that under both the standard cost model (i.e., the canonical cost formulation introduced by Sleator and Tarjan [51]), as well as under the modified cost formulation, the *Move-To-Front* algorithm (MTF) is an optimal algorithm according to average analysis (albeit not necessarily uniquely optimal).

Our results on list update also address a problem posed by Martínez and Roura [43], namely defining an alternative measure to the competitive ratio that demonstrates the superiority of MTF in the modified cost model. This is motivated by the observation that in the modified cost model, all list-update algorithms have asymptotically the same non-constant competitive ratio [43]. Our results provide evidence that bijective and average analysis are not tied to the paging problem, but rather can be applied, with success, to other online optimization problems.

Structure of the paper In Section 2 we give an overview of related work and introduce some standard definitions. In Section 3 we give formal definitions of the concepts of bijective and average analysis. In Subsection 4.1 we show strong equivalence between all lazy algorithms according to bijective analysis. These results formalize ideas that while perhaps familiar to many researchers of online problems had yet to be proved in a rigorous manner. We also show that LRU is strictly better than FWF under this measure. In Subsection 4.2 we present our main result, i.e., separation between LRU and all other paging strategies using average analysis coupled with concave analysis.

In Section 5 we demonstrate how to apply this framework to other problems, namely paging with lookahead and list update. More specifically, in Subsection 5.1 we show that bijective analysis captures the effect of lookahead. This is in contrast to the competitive ratio under which an algorithm with lookahead performs no better than a similar algorithm without lookahead. Finally, in Subsection 5.2 we apply our measures to list update algorithms and prove the optimality of MTF, under average analysis.

2 Preliminaries and Related Work

We begin with some well-known definitions concerning paging strategies. The algorithms *Least-Recently-Used* (LRU), *First-In-First-Out* (FIFO), and *Flush-When-Full* (FWF) are among the best known, and most studied eviction strategies. In the event of a cache miss, LRU evicts the page that is least recently used, whereas FIFO evicts the page that was brought into the cache earliest, and FWF simply empties the cache. All these paging algorithms have competitive ratio k, which is the best among all deterministic online paging algorithms [51, 52]. A paging algorithm is called *conservative* if it incurs at most k faults on any sequence that contains at most k distinct pages. On the other hand, a *marking* algorithm $\mathcal{A}$ works in phases, as follows. At the beginning of each phase, all pages in the cache are unmarked and a page becomes marked the first time it is requested. When an eviction is necessary, the marking algorithm $\mathcal{A}$ should evict an unmarked page, if possible; otherwise, it starts a new phase, removing all marks.

According to the above definitions, LRU and FIFO are conservative algorithms, whereas LRU and FWF are marking algorithms. A *lazy* paging algorithm (also called *demand* paging algorithm) does not evict any page upon a hit and evicts at most one page upon a fault. LRU and FIFO are lazy, while FWF is not.

Next we give an overview of some known alternative approaches to competitive analysis. We refer the reader to the survey [32] for a detailed discussion. *Loose competitiveness*, which was first proposed by Young in [54] and later refined in [57], considers an offline adversary that is oblivious to the cache size k. The adversary must then produce a sequence that is bad for most values of k rather than for just a specific value. It also ignores those sequences on which the online algorithm incurs a cost less than a certain threshold. This results in a weaker adversary and gives rise to paging algorithms of constant performance ratio. The *diffuse adversary* model by Koutsoupias and Papadimitriou [42] as well as Young [55, 56] refines the competitive ratio by restricting the set of legal request sequences to those derived from a class (family) of probability distributions. This restriction follows from the observation that although a good performance measure could in fact use the actual distribution over the request sequences, determining the exact distribution of real-life phenomena is a difficult task (e.g., depending on the intended application different distributions might arise). By restricting the input to a class Δ_ϵ of distributions, they are able to show more realistic ratios for the performance of well known paging algorithms. The *Max/Max ratio*, introduced by Borodin and Ben-David [15] compares online algorithms based on their amortized worst-case behaviour (here the amortization arises by dividing the cost of the algorithm over the length of the request sequence). This measure is based on directed comparison of online algorithms and reflects the influence of lookahead. However, it does not provide better separation results than competitive analysis of paging algorithms.

Continuing the discussion on measures alternative to the competitive ratio, the *relative worst order ratio* [19, 20, 25] combines some of the desirable properties of the Max/Max ratio and the *random order ratio* (this last introduced in [41] in the context of the online bin packing problem). As with the Max/Max ratio, it allows for direct comparison of two online algorithms. Informally, this measure compares the performance of two algorithms on a given request sequence by considering the worst-case ordering (permutation) of the sequence, for each algorithm. It then selects among all possible sequences the one that maximizes this worst-case performance. This measure reflects the influence of lookahead for paging and separates the performance of LRU from FWF [20]. Panagiotou and Souza proposed a model that explains the efficiency of LRU in practice [47]. In their work, they classified request sequences according to some parameters and proved an upper bound on the competitive ratio of LRU as a function of these parameters. They were able to argue that, in practice, typical request sequences have parameters that lead to a constant competitive ratio for LRU. Last, more recently, Dorrigiv *et al.* [30] introduced *relative order analysis* as a technique for directly comparing the performance of two online algorithms, based on the normalized difference of their cost, and they were able to obtain separation results for various algorithms.

Several theoretical models and techniques have been proposed in order to capture and exploit locality of reference. Borodin *et al.*[17] proposed the *access graph* model in which the space of request sequences can then be modeled by a graph in which paths between vertices correspond to request sequences. Chrobak and Noga showed that the competitive ratio of LRU is at least as good as FIFO on every access graph [28]. Irani et al. showed the existence of strongly competitive algorithms in the access graph model [37]. In a generalization of the access graph model, Karlin, Phillips, and Raghavan [40] proposed a model in which the space of request sequences has a distribution induced by a Markov chain process. In another generalization, Boyar *et al.* [22] applied relative interval analysis in the access graph model, as well as relative worst order analysis [21], again in the access graph model. In other work, Becchetti [14] refined the diffuse adversary model of Koutsoupias and Papadimitriou described earlier by considering only probabilistic distributions in which locality of reference is present. Using this model he proves that the performance of LRU improves as locality increases while the reverse is true for FWF. Torng [52] considered the decomposition of request sequences to appropriately defined phases, and modeled locality of reference by restricting the input to sequences with long average phase length. Using the *full access cost model*, [52] showed constant performance ratios for LRU on sequences with significant locality of reference and proved that the model reflects the influence of lookahead. However, all conservative and marking algorithms have the same performance in this model.

Most relevant to this paper is the work of Albers, Favrholdt, and Giel [3], which introduced a model in which request sequences are classified according to a measure of locality of reference. The measure itself is based on Denning's working set concept [29] which is supported by extensive experimental results. This technique, which we term *concave analysis*, reflects the fact that efficient algorithms must perform competitively in each class of inputs of comparable locality of reference, as opposed to the worst case alone. It should be noted that Albers et al. [3] focused on the *fault rate* as the measure of the cost of an algorithm, as opposed to the traditional definition of cost as the number of cache misses. Under this model, they showed that LRU has better performance than FWF and FIFO.

Subsequently to the conference versions of this paper, Angelopoulos and Schweitzer [12] were able to prove that LRU (as well as MTF in the context of list update) is optimal by combining bijective analysis and concave analysis. Although bijective analysis implies average analysis, we believe that the results of our work are still very much relevant, not only because of chronological precedence, but mostly because we provide different analysis techniques than [12]. Note that bijective analysis is not always applicable, and as a result two algorithms may be incomparable. In contrast, average analysis provides a less stringent way of comparing two algorithms, and although weaker than bijective analysis, is applicable to a much wider class of online problems[1]. We are hopeful that our analysis approach can be applied to a much wider range of online optimization problems.

3 Bijective analysis and average analysis

In this section we introduce a new technique for comparing the performance of online algorithms. At a high level, this is achieved by pairwise comparisons of the cost of algorithms over all possible request sequences of the same size.

Given an online algorithm $\mathcal{A}$ and a request sequence σ, let $\mathcal{A}(\sigma)$ be the cost incurred by $\mathcal{A}$ on σ, and let $\mathcal{I}_n$ be the set of all request sequences of length n. The first comparison model we introduce is bijective analysis. In this model, we aim to pair-up request sequences for two algorithms $\mathcal{A}$ and $\mathcal{B}$ using a bijective mapping in such a way that the cost of $\mathcal{A}$ on sequence σ is no more than the cost of $\mathcal{B}$ on the image of σ. More formally, we propose the following definition.

Definition 1. *We say that an online algorithm $\mathcal{A}$ is* no worse *than an online algorithm $\mathcal{B}$ according to* bijective analysis *if there exists an integer $n_0 \geq 1$ such that for each $n \geq n_0$, there is a bijection $b : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ satisfying $\mathcal{A}(\sigma) \leq \mathcal{B}(b(\sigma))$ for each $\sigma \in \mathcal{I}_n$. We denote this by $\mathcal{A} \preceq_b \mathcal{B}$. Otherwise we denote the situation by $\mathcal{A} \not\preceq_b \mathcal{B}$. Similarly, we say that $\mathcal{A}$ and $\mathcal{B}$ are* the same *according to bijective analysis if $\mathcal{A} \preceq_b \mathcal{B}$ and $B \preceq_b A$. This is denoted by $\mathcal{A} \equiv_b \mathcal{B}$. Finally we say $\mathcal{A}$ is* better *than $\mathcal{B}$ according to bijective analysis if $\mathcal{A} \preceq_b \mathcal{B}$ and $\mathcal{B} \not\preceq_b \mathcal{A}$. We denote this by $\mathcal{A} \prec_b \mathcal{B}$.*

Observe that, similar to Max/Max ratio, this measure considers sequences of the same length and allows for direct comparison of two online algorithms. However, it induces a comparison of

[1]For an extension of average analysis to problems in which the input contains elements with real-valued weights, see [24]

their performance on *all* sequences in $\mathcal{I}_n$, rather than only on the worst sequence. A related, and less stringent comparison model can be obtained by considering the average number of faults that a paging algorithm incurs on request sequences of a certain length.

Definition 2. *We say that an online algorithm $\mathcal{A}$ is* no worse *than an online algorithm $\mathcal{B}$ according to* average analysis *if there exists an integer $n_0 \geq 1$ such that for each $n \geq n_0$, $\sum_{I \in \mathcal{I}_n} \mathcal{A}(I) \leq \sum_{I \in \mathcal{I}_n} \mathcal{B}(I)$. We denote this by $\mathcal{A} \preceq_a \mathcal{B}$. Otherwise we denote the situation by $\mathcal{A} \not\preceq_a \mathcal{B}$. $\mathcal{A} \equiv_a \mathcal{B}$, and $\mathcal{A} \prec_a \mathcal{B}$ are defined as for bijective analysis.*

Observation 1. *If $\mathcal{A} \not\preceq_a \mathcal{B}$, then $\mathcal{A} \not\preceq_b \mathcal{B}$. As well, if $\mathcal{A} \preceq_b \mathcal{B}$, then $\mathcal{A} \preceq_a \mathcal{B}$ with similar statements holding for $\equiv_b$ and $\prec_b$.*

Example 3. *We use a simple example to illustrate the above definitions. Let $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ be three online algorithms and $\mathcal{I}_n = \{\sigma_1, \sigma_2, \cdots, \sigma_{10}\}$ be the set of all possible request sequences of length n. Suppose that the cost of $\mathcal{A}$, $\mathcal{B}$, C, and the optimal offline algorithm OPT on the request sequences are as follows:*

σ	$\mathcal{A}(\sigma)$	$\mathcal{B}(\sigma)$	$\mathcal{C}(\sigma)$	$OPT(\sigma)$
σ_1	5	6	6	3
σ_2	7	8	8	2
σ_3	3	4	3	3
σ_4	9	4	3	1
σ_5	7	10	8	3
σ_6	5	7	6	4
σ_7	3	6	4	2
σ_8	7	6	7	5
σ_9	5	8	5	2
σ_{10}	7	10	9	3

We have $\sum_\sigma \mathcal{A}(\sigma) = 58$, $\sum_\sigma \mathcal{B}(\sigma) = 69$, and $\sum_\sigma \mathcal{C}(\sigma) = 59$. Therefore $\mathcal{A} \prec_a \mathcal{B}$, $\mathcal{A} \prec_a \mathcal{C}$, and $\mathcal{C} \prec_a \mathcal{B}$. We have $\mathcal{B} \not\preceq_b \mathcal{A}$, because $\mathcal{B} \not\preceq_a \mathcal{A}$. We also have $\mathcal{A} \preceq_b \mathcal{B}$ by considering the bijection that maps $\sigma_1, \sigma_2, \ldots, \sigma_{10}$ to σ_1, σ_2, σ_3, σ_5, σ_6, σ_7, σ_4, σ_9, σ_8, and σ_{10}, respectively. Therefore $\mathcal{A}$ is better than $\mathcal{B}$ according to bijective analysis, i.e., $\mathcal{A} \prec_b \mathcal{B}$. Since $\mathcal{A} \prec_a \mathcal{C}$, we conclude that $\mathcal{C} \not\preceq_b \mathcal{A}$. We also have $\mathcal{A} \not\preceq_b \mathcal{C}$ because $\mathcal{C}$ incurs a cost less than 5 on 3 sequences while $\mathcal{A}$ incurs a cost less than 5 only on 2 sequences. Thus, although $\mathcal{A}$ is better than $\mathcal{C}$ according to average analysis, the two algorithms are not comparable according to bijective analysis. As a last example, consider the competitive ratio of these algorithms. The (strict) competitive ratios of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ are 9, 4, and 4 respectively. Although $\mathcal{A}$ seems to have better overall performance than $\mathcal{B}$ and $\mathcal{C}$, its bad performance on a single sequence, namely σ_4, results in a bad competitive ratio.

4 Separation between LRU and other paging algorithms

4.1 Paging without assumptions on request sequences

In this section we consider the model in which each request in $\mathcal{I}_n$ is possible, e.g., we consider all possible requests, whether or not they exhibit locality of reference. We begin with a simple, yet striking observation: as will be shown in Theorem 4, all *lazy* paging algorithms are equivalent according to bijective analysis. For the remainder of this section let N denote the number of pages in slow memory. In addition, given a sequence σ, we denote by $C(\sigma)$ the set of *continuations* of σ, namely the set of all sequences in $\mathcal{I}_{|\sigma|+1}$ of length $|\sigma|+1$ whose prefix is σ. For a continuation sequence $\sigma' \in C(\sigma)$, and a given paging algorithm A, we say that the last request of σ' is a *last-hit* (resp. last-fault) request with respect to A, if A incurs a page hit (resp. page fault) on the last request of σ'.

Theorem 4. *Let $\mathcal{A}$ and $\mathcal{B}$ be two lazy paging algorithms. Then we have $\mathcal{A} \equiv_b \mathcal{B}$.*

Proof. We will show that $A \preceq_b B$; using the same arguments, we can obtain that $B \preceq_b A$, hence $A \equiv_b B$. More precisely, we will show, by induction on n, that for all $n \geq 1$ there is a bijection $\pi_n : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ such that $A(\sigma) \leq B(\pi_n(\sigma))$, for every $\sigma \in I_n$. For $n \leq k$, this is true since on any sequence of length at most the cache size, A and B make the same decisions (i.e., no evictions). Suppose that the result holds for n, we will show the existence of an appropriate bijection $\pi_{n+1} : \mathcal{I}_{n+1} \leftrightarrow \mathcal{I}_{n+1}$. The crucial observation is that, on the one hand, for any $\sigma \in \mathcal{I}_n$ in which at least k pages have been requested, there are exactly k last-hit sequences in $C(\sigma)$ and exactly $N-k$ last-fault sequences in $C(\sigma)$, for both A and B (albeit not necessarily the same sequences). On the other hand, for any $\sigma \in \mathcal{I}_n$ in which fewer than k pages have been requested, A and B incur the same cost on σ. Combining these observations with the induction hypothesis, we obtain that for every $\sigma \in \mathcal{I}_n$, there exists a bijection $\phi_\sigma : \mathcal{I}_1 \leftrightarrow \mathcal{I}_1$ such that for any (single) request r we have that

$$A(\sigma r) \leq B(\pi_n(\sigma)\phi_\sigma(r)).$$

Therefore, the bijection $\pi_{n+1} : \mathcal{I}_{n+1} \leftrightarrow \mathcal{I}_{n+1}$ which maps σr to $\pi_n(\sigma)\phi_\sigma(r)$ has the desired property, and the theorem follows. □

Next, we will show how bijective analysis can provide a strict separation between LRU and FWF. Using the same arguments as in the proof of Theorem 4, one can argue that $LRU \preceq_b FWF$. The following lemma shows that LRU is strictly better than FWF, namely $LRU \prec_b FWF$. This implies that LRU is superior to FWF under bijective analysis.

Lemma 5. *$FWF \not\preceq_b LRU$.*

Proof. We prove this by contradiction. Suppose that we have $FWF \preceq_b LRU$ and thus there is an $n_0 \geq 1$ such that for each $n \geq n_0$ we have the bijection $b_n : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ with the corresponding properties. Recall that we can partition a sequence into a number of consecutive phases so that each phase contains exactly k distinct pages. LRU incurs at most k faults in each phase. On the other hand, FWF empties its cache at the beginning of each phase and incurs exactly k faults in each phase. Therefore we have $LRU(\sigma) \leq FWF(\sigma)$ for each sequence σ (hence also $LRU \preceq_b FWF$). Thus the desired bijection exists only if we have $FWF(\sigma) = LRU(\sigma)$ for every sequence of length $n \geq n_0$. Consider a sequence $\sigma = p_1 p_2 \ldots p_h$ ($h \geq \max(2, n_0)$) such that σ contains at least $k+1$ distinct pages and p_h is the first page of a phase. Therefore p_h causes FWF to flush its cache, which now contains only one page after serving σ. Now consider the set of continuations of σ. The number of last-fault sequences among these for LRU and FWF is $N-k$ and $N-1$, respectively. Therefore there are at least $k-1$ sequences for which the cost of LRU is strictly less than the cost of FWF and hence a bijection as required does not exist. □

It is worth noting that Theorem 4 explains why several performance measures fail to separate lazy algorithms such as LRU and FIFO, in that the multisets of the costs incurred by the algorithms on all requests of the same length are identical. In fact, Theorem 4 supports the observation that unless we constrain, or partition, the space of request sequences to the ones most often appearing in practice, it is unlikely (and arguably, even undesirable) to separate different paging algorithms. Since locality of reference is a definitive characteristic of typical sequences in paging, this naturally leads to the question of how it affects the comparison of algorithms, according to our measures.

4.2 Paging with locality of reference

In Subsection 4.1, we proved that all lazy algorithms are strongly equivalent according to bijective analysis. However, this analysis ignored that in practice request sequences exhibit *locality of reference*. We follow the model of Albers et al. [3], according to which a request sequence has high locality of reference if the number of distinct pages in a window of size n is small. More precisely, consider the function that represents the maximum number of distinct pages in a window of size w, in a request sequence of size $n \geq w$. Extensive experiments with real data have shown that this function can be bounded by a concave function for most practical request sequences [3]. Let then f denote an increasing concave function. We say that a request sequence is *consistent* with f if the number of distinct pages in any window of size w is at most $f(w)$, for any $w \in \mathcal{N}$. Now we can model locality by considering only those request sequences that are consistent with f. Albers et al. consider a slightly more restrictive class of functions called concave* functions.

Definition 6. *[3] A function $f : N \to R_+$ is* concave* *if*

1. $f(1) = 1$ *and*
2. $\forall w \in \mathcal{N} : f(w+1) - f(w) \geq f(w+2) - f(w+1)$.

We additionally require that f be surjective on the integers between 1 and its maximum value.

Note that with the exception of the requirement $f(1) = 1$, the definition describes a typical concave function. For instance, the function $f : \{1, \ldots, 8\} \to \{1, \ldots, 6\}$, with $f(x) = x$, if $1 \leq x \leq 4$, and $f(x) = x/2 + 2$, if $4 < x \leq 8$ is an example of a concave* function.

Let $\mathcal{I}^f$ denote the set of request sequences that are consistent with a given concave* function f. We can apply bijective and average analysis over this restricted set of sequences, by adapting Definition 1 and Definition 2 appropriately, i.e., by replacing the set $\mathcal{I}$ with the set $\mathcal{I}^f$. We denote the corresponding relations by $\mathcal{A} \preceq_b^f \mathcal{B}$, $\mathcal{A} \preceq_a^f \mathcal{B}$, etc. Note that given any sequence, we can obtain a new sequence that is consistent with f by repeating each request a sufficient number of times. Therefore, even if we restrict sequences to those with high locality of reference, there is still a worst-case sequence for LRU (compared to OPT) that is consistent with f. In other words, the competitive ratio of LRU is the same as in the standard model[2].

Figure 1 illustrates the partition of the request-sequence space induced by the choice of function f. Observe that the performance of a paging algorithm is now evaluated within the subset of request sequences of a given length whose locality of reference is consistent with f. For this purpose, we will denote by $\mathcal{I}_n^f$ the sequences of length n in $\mathcal{I}^f$.

Note that the inductive argument used to prove that all lazy algorithms (Theorem 4) are equivalent according to bijective analysis does not necessarily carry through under concave analysis.

[2]This is one of the reasons that Albers et al. [3] use the *fault rate*, instead of overall cost, as a performance measure.

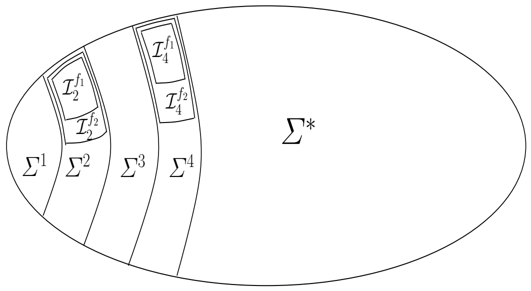

Figure 1: Partition of the input space induced by different choices of f.

The problem is that the bijective mapping may very well map a sequence consistent with f to a sequence which is not consistent with f.

Consider a fixed concave* function f, and let $\mathcal{A}$ be an arbitrary paging algorithm We call a sequence *bad* for $\mathcal{A}$ if $\mathcal{A}$ incurs a fault on its last request; otherwise we call it a *good* sequence for $\mathcal{A}$. Let $B_h(\mathcal{A})$ be the number of sequences in $\mathcal{I}_h^f$ that are bad for $\mathcal{A}$. For a sequence $\sigma \in \mathcal{I}_h^f$, let $B_{h+1}(\mathcal{A}\,|\,\sigma)$ denote the number of sequences in $\mathcal{I}_{h+1}^f$ that have σ as their prefix and are bad for $\mathcal{A}$. Define $G_h(\mathcal{A})$ and $G_{h+1}(\mathcal{A}\,|\,\sigma)$ in an analogous way for good sequences. Intuitively, an efficient algorithm maintains its good sequences in the set of sequences with high locality of reference. Observe that LRU naturally fits this criterion: the most recently accessed pages are exactly those that are in its cache, and therefore good (i.e. last-hit) sequences for LRU are more likely to be sequences with high locality of reference. We formalize this intuition in the remainder of this section.

Lemma 7. *For any integer $h > 0$ and any paging algorithm $\mathcal{A}$, $B_h(LRU) \leq B_h(\mathcal{A})$.*

Proof. We prove the lemma by induction on h. We can assume, without loss of generality, that $\mathcal{A}$ is a lazy paging algorithm (since any non-lazy algorithm can be transformed to a lazy one without increasing its cost). Let f be any concave* function. If $h = 1$, then every sequence of $\mathcal{I}_h$ is consistent with f and each algorithm incurs a fault on its last request (recall that algorithms start with an empty cache). Therefore we have $B_1(LRU) = |\mathcal{I}_1^f| = N = B_1(\mathcal{A})$. If $h > 1$, consider an arbitrary sequence $\sigma \in \mathcal{I}_{h-1}^f$. If σ has at most k distinct pages, then LRU and $\mathcal{A}$ have the same pages in their cache after serving σ and therefore $B_h(LRU\,|\,\sigma) = B_h(\mathcal{A}\,|\,\sigma)$. Otherwise, LRU has filled its cache with k pages after serving σ, while A's cache contains at most k pages. The next page requested can be an arbitrary page, provided that adding that page does not violate consistency with f.

From the definition of f, given $\sigma \in \mathcal{I}_{h-1}^f$, if we append the last request in σ to the end of σ, we obtain a sequence in $\mathcal{I}_h^f$ that is always consistent with f. If we append the second to last request, the resulting sequence may or may not be consistent with f, however if the second to last request is not consistent neither is any other request. This implies that for every good request for A that is consistent with f, there is a good request for LRU that is consistent with f. Hence $G_h(LRU\,|\,\sigma) \geq G_h(\mathcal{A}\,|\,\sigma)$. Since the good and bad sequences form a partition of the set of continuations of σ consistent with f [3], the inequality above implies $B_h(LRU\,|\,\sigma) \leq B_h(\mathcal{A}\,|\,\sigma)$. To

[3]Note that in the context of concave analysis, each continuation must naturally belong in the set $\mathcal{I}_h^f$.

conclude observe that $B_h(X) = \sum_{\sigma\in\mathcal{I}_{h-1}} B_h(X \mid \sigma)$ for any algorithm X. Hence

$$B_h(LRU) = \sum_{\sigma\in\mathcal{I}_{h-1}} B_h(LRU \mid \sigma) \le \sum_{\sigma\in\mathcal{I}_{h-1}} B_h(\mathcal{A} \mid \sigma) = B_h(\mathcal{A})$$

as claimed. □

Lastly, we will show that LRU strictly outperforms all other paging algorithms. To this end, we will use the following definition related to average analysis.

Definition 8. *Let m be an integer, $\mathcal{A}$ and $\mathcal{B}$ be online algorithms, and f be a* concave* *function. $\mathcal{A}$ is said to (m, f)*-dominate *$\mathcal{B}$ if we have*

$$\sum_{\sigma\in\mathcal{I}_m^f} \mathcal{A}(\sigma) \le \sum_{\sigma\in\mathcal{I}_m^f} \mathcal{B}(\sigma).$$

$\mathcal{A}$ is said to dominate *$\mathcal{B}$ if there exists an integer $m_0 \ge 1$ so that for each $m \ge m_0$ and every* concave* *function f, $\mathcal{A}$ (m, f)-dominates $\mathcal{B}$.*

Observation 2. *$\mathcal{A} \preceq_a^f \mathcal{B}$ if and only if there exists an integer $m_0 \ge 1$ so that $\mathcal{A}$ (m, f)-dominates $\mathcal{B}$ for each $m \ge m_0$.*

Lemma 9. *For every paging algorithm $\mathcal{A}$, LRU dominates $\mathcal{A}$.*

Proof. We can assume, without loss of generality, that $\mathcal{A}$ is a lazy paging algorithm. Let f be an arbitrary concave* function and m be a positive integer. For any $1 \le i \le m$, let $\mathcal{F}_{i,m}(\mathcal{A})$ be the number of sequences in $\mathcal{I}_m^f$ for which $\mathcal{A}$ incurs a fault on the i^{th} request. We will show that $\mathcal{F}_{i,m}(LRU) \le \mathcal{F}_{i,m}(\mathcal{A})$ for any $1 \le i \le m$ which will imply optimality of LRU. For $i = 1$, we have $\mathcal{F}_{1,m}(LRU) = \mathcal{F}_{1,m}(\mathcal{A}) = |\mathcal{I}_m^f|$. Assume that $i > 1$. Let σ be an arbitrary sequence in $\mathcal{I}_{i-1}^f$, and let T_σ denote the set of sequences in $\mathcal{I}_m^f$ that have σ as their prefix. Denote by $\mathcal{F}_{i,m}(\mathcal{A} \mid \sigma)$ the number of sequences in T_σ for which $\mathcal{A}$ incurs a fault on the i^{th} request.

If σ contains at most k distinct pages, then LRU and $\mathcal{A}$ behave the same on σ and we have $\mathcal{F}_{i,m}(LRU \mid \sigma) = \mathcal{F}_{i,m}(\mathcal{A} \mid \sigma)$. Suppose then that σ has more than k distinct pages. We can partition T_σ into four subsets: (1) T_σ^1: sequences in which neither LRU nor $\mathcal{A}$ incur a fault on the i^{th} page request, (2) T_σ^2: sequences in which both LRU and $\mathcal{A}$ incur a fault on the i^{th} page request, (3) T_σ^3: sequences in which $\mathcal{A}$ incurs a fault on the i^{th} page request, but LRU does not. (4) T_σ^4: sequences in which LRU incurs a fault on the i^{th} page request, but $\mathcal{A}$ does not.

We have $\mathcal{F}_{i,m}(LRU \mid \sigma) = |T_\sigma^2| + |T_\sigma^4|$, and $\mathcal{F}_{i,m}(\mathcal{A} \mid \sigma) = |T_\sigma^2| + |T_\sigma^3|$. We show that $|T_\sigma^4| \le |T_\sigma^3|$ by proving that there exists a one-to-one mapping d from T_σ^4 to T_σ^3. For $1 \le q \le 4$, let P_σ^q be the set of pages that are requested as the i^{th} page of a sequence in T_σ^q. Let $\rho \in \mathcal{I}_i^f$ be a sequence that contributes to $B_i(LRU \mid \sigma)$ and p be its i^{th} request. Denote by τ the sequence of length m obtained by appending $m - i$ requests to page p to ρ. Since ρ is consistent with f and p is the last request of ρ, τ is consistent with f and thus $\tau \in T_\sigma$. Note that LRU incurs a fault on the i^{th} request of τ. Therefore τ belongs to either T_σ^2 or T_σ^4 and p is in one of the sets P_σ^2 and P_σ^4. Also any page in $P_\sigma^2 \cup P_\sigma^4$ contributes (one unit of cost) to $B_i(LRU \mid \sigma)$. To see this, let q be a page in $P_\sigma^2 \cup P_\sigma^4$. Then q is the i^{th} request of a sequence $\rho' \in T_\sigma^2 \cup T_\sigma^4$. Let $\tau' \in \mathcal{I}_i^f$ be the prefix of ρ' that contains i requests. LRU incurs a fault on the last request of τ'. Thus τ' is a bad sequence for LRU and q contributes to $B_i(LRU \mid \sigma)$. Hence we have $B_i(LRU \mid \sigma) = |P_\sigma^2| + |P_\sigma^4|$. Using analogous arguments we get $B_i(\mathcal{A}\,|\,\sigma) = |P^2_\sigma| + |P^3_\sigma|$. We know that $B_i(LRU\,|\,\sigma) \leq B_i(\mathcal{A}\,|\,\sigma)$ from the proof of Lemma 7; therefore $|P^4_\sigma| \leq |P^3_\sigma|$ and there is a one-to-one mapping r from P^4_σ to P^3_σ.

We will now use the mapping r so as to define the desired mapping d. Consider an arbitrary sequence $S = p_1p_2 \ldots p_m \in T^4_\sigma$. Let $x = p_i$ and $y = r(x)$. According to definitions we know that on the i^{th} request of a sequence in T_σ, x is a fault for LRU and a hit for $\mathcal{A}$, whereas y is a hit for LRU and a fault for $\mathcal{A}$. Let $\sigma_x \in \mathcal{I}^f_i$ be the sequence obtained by appending the page x to σ, and $\sigma_y \in \mathcal{I}^f_i$ be the sequence obtained by appending the page y to σ. On serving σ_x, the last page (x) is a fault for LRU; therefore x is not among the last k distinct pages in σ. LRU does not incur a fault on the last page of σ_y; thus y is among the last k distinct pages of σ. In other words, if starting from the i-th request we convert each x in a sequence in T_{σ_x} to y, we obtain a sequence that is consistent with f, i.e., a sequence in T_{σ_y}. This gives us a one-to-one mapping from T_{σ_x} to T_{σ_y}. Using a similar process for all the pages in P^4_σ, we obtain a one-to-one mapping from T^4_σ to T^3_σ. Therefore

$$|T^4_\sigma| \leq |T^3_\sigma| \Rightarrow \mathcal{F}_{i,m}(LRU\,|\,\sigma) \leq \mathcal{F}_{i,m}(\mathcal{A}\,|\,\sigma). \tag{1}$$

Since

$$\mathcal{F}_{i,m}(LRU) = \sum_{\sigma \in \mathcal{I}_{i-1}} \mathcal{F}_{i,m}(LRU\,|\,\sigma) \tag{2}$$

and

$$\mathcal{F}_{i,m}(\mathcal{A}) = \sum_{\sigma \in \mathcal{I}_{i-1}} \mathcal{F}_{i,m}(\mathcal{A}\,|\,\sigma), \tag{3}$$

we obtain $F_{i,m}(LRU) \leq F_{i,m}(\mathcal{A})$. We also have

$$\sum_{\sigma \in \mathcal{I}^f_m} LRU(\sigma) = \sum_{1 \leq i \leq m} \mathcal{F}_{i,m}(LRU) \tag{4}$$

and

$$\sum_{\sigma \in \mathcal{I}^f_m} \mathcal{A}(\sigma) = \sum_{1 \leq i \leq m} \mathcal{F}_{i,m}(\mathcal{A}). \tag{5}$$

Therefore

$$\sum_{\sigma \in \mathcal{I}^f_m} LRU(\sigma) \leq \sum_{\sigma \in \mathcal{I}^f_m} \mathcal{A}(\sigma). \tag{6}$$

Thus LRU (m, f)-dominates $\mathcal{A}$ for every concave* function f, and every integer $m \geq 1$. Hence LRU dominates $\mathcal{A}$. □

Corollary 10. *For any* concave* *function f and any paging algorithm $\mathcal{A}$, $LRU \preceq^f_a \mathcal{A}$.*

Therefore LRU is an optimal algorithm when we restrict request sequences to those with high locality of reference (in the model of concave* functions). A natural question is whether LRU is the *unique* optimal algorithm. The following result answers this question in the affirmative.

Theorem 11. *No paging algorithm (other than LRU) dominates LRU, with respect to average analysis.*

Proof. Consider a paging algorithm $\mathcal{A}$ which is different from LRU. We will show that there exists a concave* function f such that $\mathcal{A} \not\preceq_a^f$ LRU, assuming that A is oblivious of the function f. We say that a concave* function f belongs to class F_z if $f(x) = x$ for $x < z + k$ and $f(x) = z + k$ for $x \geq z + k$. First, if the content of the cache of LRU and $\mathcal{A}$ is always the same there is nothing to show. Hence we focus on situations where their cache content differs.

Define ζ as the smallest z such that there is a concave* function $f_\zeta \in F_z$ such that $\mathcal{A}$ and LRU have different behavior on a sequence consistent with f_ζ. Let σ' denote a sequence of length $i-1$ consistent with f_ζ up to the point at which LRU's cache contents first differ from those of $\mathcal{A}$. That is, a request for a page p caused an eviction of a page p_a in LRU while it caused the eviction of a page p_b in $\mathcal{A}$. As in the proof of Lemma 9, we need only focus on the parameters $T^3_{\sigma'}$ and $T^4_{\sigma'}$. We will also use the same notation as in that proof. Consider a sequence $\sigma'' \in T^3_{\sigma'}$. $\mathcal{A}$ incurs a fault on the i^{th} request of σ'', while LRU does not. Just before the i^{th} request (i.e., just after serving σ'), $\mathcal{A}$'s cache contains all pages in LRU's cache except for p_b. Therefore, the only option for the i^{th} request of σ'' is p_b. Hence, $P^3_{\sigma'}$ can only contain p_b, i.e., $P^3_{\sigma'} \subseteq \{p_b\}$. We append some requests to page p to σ' so as to obtain a sequence σ for which $P^3_\sigma = \{p_b\}$. Slightly misusing our notation (and for the ease of the proof), let $|\sigma| = i - 1$. We can use the same one-to-one mapping d that was defined in the proof of Lemma 9 to show that $|T^4_\sigma| \leq |T^3_\sigma|$. Thus each sequence $\rho \in T^4_\sigma$ is mapped to a distinct sequence $d(\rho) \in T^3_\sigma$.

Our main argument will be to prove that $|T^4_\sigma| < |T^3_\sigma|$ by showing that there is a sequence ρ' in T^3_σ such that $d(\rho) \neq \rho'$ for any $\rho \in T^4_\sigma$. Let σ^0 be the suffix of σ starting just after the last request for p_a and $Q = \{q_1, q_2, \ldots, q_a\}$ be pages that do not appear in $\sigma^0 p_a$ (we will specify a lower bound on $|Q|$ later on). We clarify that Q has to include all pages in σ not requested in σ^0. Define α as the starting index of σ^0 in σ, i.e., the request at index $\alpha - 1$ of σ is to p_a and σ^0 is the subsequence of σ from index α to index $i-1$. LRU always keeps the k most recently accessed pages in its cache. Furthermore, on a fault it evicts the k-th most recently accessed page. Since p_a is the last page evicted by LRU in serving σ, it is the $(k+1)$-th most recently used page in σ. Similarly, since p_b is in LRU's cache after serving σ, p_b is among the k most recently accessed pages in σ. Therefore the number of distinct pages in $\sigma^0 p_a$ and $\sigma^0 p_b$ is $k+1$ and k, respectively.

Let ρ_1 and ρ_2 be continuations of σp_a and σp_b, respectively, defined as follows. We append new pages from Q to ρ_1 and ρ_2 until appending a new page q_t causes an inconsistency with f_ζ in the suffix starting with $\sigma^0 p_a$ in ρ_1. To see that this inconsistency with f_ζ will eventually occur, consider sequences ρ'_1 and ρ'_2 obtained after appending ζ pages from Q to σp_a and σp_b, respectively (note that we can pick Q such that $|Q| \geq \zeta$ since there is no restriction on the number of pages in the sequence). Let λ be the length of $\sigma^0 p_a$, then the suffix of ρ'_1 starting at index α has length $\lambda + \zeta$ and contains $k + 1 + \zeta$ distinct pages. Since $f_\zeta(\lambda + \zeta) = k + \zeta$, this subsequence is inconsistent with f_ζ. The suffix of ρ'_2 starting at index α contains $k + \zeta$ distinct pages and does not violate consistency with f. There is a small caveat in this construction. We might first violate consistency with f_ζ in a subsequence τ of ρ_1 and ρ_2 that does not start with σ^0. There are two cases that we should consider.

1. τ starts after the start of σ^0, i.e., at an index $\beta > \alpha$ of σ. This case cannot happen as τ will have at most $k + \zeta$ distinct pages and we have $f_\zeta(x) = x$ for $x \leq \zeta + k$. To see this, observe that we have not yet had an inconsistency with f_ζ in a suffix of ρ_1 that starts with

σ^0. Therefore τ contains fewer than ζ pages from Q. The prefix of τ not containing pages from Q (a suffix of $\sigma^0 p_a$ or $\sigma^0 p_b$) will have at most $k+1$ distinct pages. Therefore τ contains at most $k+\zeta$ distinct pages.

2. τ starts before the start of σ^0, say at an index $\gamma < \alpha$ of σ. We change our construction slightly to rule out this case. Recall that Q is defined as the set of pages that do not appear in $\sigma^0 p_a$. We partition Q into two disjoint subsets R and S as follows. Let σ^1 be the prefix of σ that ends just before the start of σ^0. Denote by $R = \{r_1, \ldots, r_b\}$ the pages in Q that are requested in σ^1, ordered by their last request in σ^1, e.g., if the last request to r_1 in σ^1 is at index u and the last request to r_2 in σ^1 is at index v, we have $u > v$. Let $S = \{s_1, \ldots, s_c\}$ be the pages in $Q \setminus R$. Now to construct ρ_1 and ρ_2, we append pages from $r_1, r_2, \ldots, r_b, s_1, \ldots, s_c$ in this order to σp_a and σp_b, respectively (note that we might not need all these pages to get our desired sequences). We claim that we cannot have an inconsistency with f_ζ in a suffix of ρ_1 (ρ_2) that starts at index γ before the desired inconsistency at the suffix of ρ_1 (ρ_2) that starts at index α. Assume for the sake of contradiction that the first inconsistency occurs after adding a page q on a suffix of ρ_1 (ρ_2) that starts at index γ while the suffix of ρ_1 (ρ_2) that starts at index α remains consistent with f_ζ.

   We will consider the two possible cases for q. Suppose first that $q = r_u \in R$. Let τ and τ^0 be the suffixes of ρ_1 (ρ_2) that start at indices γ and $\alpha > \gamma$, respectively. So, we have an inconsistency in τ while τ_0 remains consistent with f_ζ. Since this is the first inconsistency in τ, r_u is a new page in τ. Therefore τ does not contain r_v for $v > u$. τ^0 has all requests to r_v for $v \leq u$ and thus τ does not contain any additional pages, as compared to τ^0. Since τ is a longer sequence than τ, τ^0 contains all pages in τ and τ is not consistent with f_ζ, τ^0 cannot be consistent with f_ζ, which is a contradiction. Suppose, as the second case, that $q = s_u \in S$. In this case, the set of distinct pages in τ is the same as the set of distinct pages in τ^0 as both sequences have all pages in $R \cup \{s_1, \ldots, s_u\}$ as well as pages in $\sigma^0 p_a$. Therefore τ^0 is also not consistent with f_ζ and we have a contradiction.

Now consider the sequence $\overline{\sigma}$ that we have constructed as above. We have $\overline{\sigma} \in T^3_\sigma$ but it is easy to see that no sequence in T^4_σ is mapped by d to ρ_2 (in fact $d(\rho_1) = \rho_2$ but ρ_1 is not consistent with f_ζ). Therefore $|T^4_\sigma| < |T^3_\sigma|$ which leads to

$$\sum_{\sigma \in \mathcal{I}^f_m} \mathrm{LRU}(\sigma) < \sum_{\sigma \in \mathcal{I}^f_m} \mathcal{A}(\sigma),$$

for any $m > |\overline{\sigma}|$ by the same arithmetic manipulation as in equations (1)-(6) in the proof of Lemma 9. □

5 Further applications of bijective and average analysis

5.1 Influence of lookahead in paging

In this section we demonstrate that bijective analysis can properly capture the effects of lookahead in the context of the paging problem. We consider the setting in which the paging algorithm knows at any given point while serving a request sequence, which subsequent ℓ pages will be requested. Let LRU(ℓ) be a modification of LRU defined for a lookahead of size ℓ as follows [1]. On a fault, LRU(ℓ) evicts a page in the cache that is least recently used among the pages that are not in the current lookahead. We will show that our model reflects the influence of lookahead in that LRU(ℓ) $\prec_b$ LRU, i.e. LRU(ℓ) is better than LRU according to bijective analysis.

Lemma 12. *For any sequence σ, the cost of LRU(ℓ) on σ is no more than the cost of LRU on σ, that is, LRU(ℓ) $\preceq_b$ LRU.*

Proof. Assume for the sake of contradiction that there is a sequence $\sigma = p_1 \dots p_m$ on which LRU(ℓ) incurs strictly more faults than LRU. Let a be the smallest index so that p_a is a hit for LRU and a fault for LRU(ℓ). Suppose that the most recent eviction of p_a by LRU(ℓ) is at time r on the request p_r. Therefore we have $p_i \neq p_a$ for $r \leq i \leq a$ and furthermore LRU does not evict p_a at any time t, where $r \leq t \leq a$. Let $p_{r_1}, p_{r_2}, \dots, p_{r_k}$ be the pages in LRU's cache at time r so that p_{r_i} is less recently used than p_{r_j} at time r if and only if $i < j$. Note that since a is the smallest index so that p_a is a hit for LRU and a fault for LRU(ℓ), LRU incurs a fault on p_r and evicts p_{r_1}. Suppose that $p_{r_x} = p_a$ for some $1 < x \leq k$ and let $\mathcal{L}_r$ denote the set of pages in the lookahead of size ℓ at time r. We consider two cases:

Case 1: All pages $p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}$ are in LRU(ℓ)'s cache at time r. Since LRU(ℓ) evicts $p_{r_x} = p_a$ at time r, we should have $p_{r_i} \in \mathcal{L}_r$ for $1 \leq i \leq x-1$. Let y be the largest index such that $r \leq y \leq r+\ell$, $p_y \in \{p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}\}$, and LRU incurs a fault at time y. Note that since $p_{r_1} \in \mathcal{L}_r$ and LRU evicts p_{r_1} at time r, y exists. We claim that LRU should evict $p_{r_x} = p_a$ at time y. Since p_a has not been requested between times r and y, the only pages that can be less recently used than p_a are $p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}$. Assume, by way of contradiction, that at time y, LRU evicts page p_{r_z} for some $1 \leq z \leq x-1$. Note that p_{r_z} cannot be requested between times r and y; otherwise p_a would be less recently used than p_{r_z}. We know that $p_{r_z} \in \mathcal{L}_r$ and therefore p_{r_z} will be requested at least once between the times $y+1$ and $r+\ell$. The first such request is a fault on a page (p_{r_z}) that is in $\{p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}\}$; this contradicts the choice of y. Therefore p_a is the least recently used page for LRU at time y and LRU evicts it. This contradicts the fact that LRU does not evict p_a on a fault at any time $r \leq t \leq a$.

Case 2: There is a page $p \in \{p_{r_1}, p_{r_2}, \dots, p_{r_{x-1}}\}$ that is not in LRU(ℓ)'s cache at time r. Let $r' < r$ be the last time that LRU(ℓ) has evicted p. Since p is in LRU's cache but not in LRU(ℓ)'s cache at time r, we have $p_i \neq p$ for $r' \leq i \leq r-1$ and furthermore LRU does not evict p at any time t, where $r' \leq t \leq r-1$. This reduces to the situation discussed at the beginning of this proof, with $a = r$ and $r = r'$. Since a is a finite number and we strictly decrease our time of interest, after a finite number of applications of case 2 it will reduce to case 1.

We conclude that for every request on which LRU(ℓ) incurs a fault so does LRU, and hence LRU(ℓ) does not incur more faults than LRU on any sequence. □

Lemma 13. *There exists a sequence in which LRU(ℓ) outperforms LRU, therefore LRU $\not\preceq_b$ LRU(ℓ).*

Proof. From Lemma 12, we know that LRU has the same or higher number of page faults as LRU(ℓ) on each sequence of length at least n_0. So it suffices to show that on at least one sequence LRU(ℓ) strictly outperforms LRU. Consider a sequence σ of size $n_1 \geq n_0$, with n_0 as in Definition 1, which contains several consecutive copies of the subsequence $p_1 p_2 \dots p_k p_{k+1}$. LRU incurs a fault on all pages of σ and therefore the cost of LRU on σ is n_1. The cost of LRU(ℓ) on the other hand is $n_1/(\ell+1)$. □

Lemmas 12 and 13 imply the following theorem.

Theorem 14. *LRU(ℓ)* $\prec_b$ *LRU.*

5.2 The List Update Problem

As with the paging problem, *list update* is another fundamental problem in the context of on-line computation. In its standard form, the problem consists of maintaining an unsorted list of l items. More precisely, in the (related) *list access* problem the input is a sequence of n requests that must be served in an online manner. Let $\mathcal{A}$ be an arbitrary online list update algorithm. To serve a request to an item x, $\mathcal{A}$ linearly searches the list until it finds x. If x is the i^{th} item in the list, $\mathcal{A}$ incurs a cost i to access x. Immediately after this access, $\mathcal{A}$ can move x to any position closer to the front of the list at no extra cost. This is called a *free exchange.* In addition, $\mathcal{A}$ can exchange any two consecutive items at a cost of 1. These are called *paid exchanges.* An efficient algorithm can thus use free and paid exchanges so as to minimize the overall cost of serving a sequence. The above cost formulation describes the *standard cost model*, as proposed in the seminal work of Sleator and Tarjan [7]. 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.

The performance of list update algorithms has been an intensive topic of study under competitive analysis. Among the well-known deterministic on-line algorithms are *Move-To-Front* (MTF), *Transpose*, and *Timestamp* (TS). MTF always moves the requested item to the front of the list whereas Transpose exchanges the requested item with the item that immediately precedes it. Algorithm TS rearranges the list items in order of their second-to-last access. Sleator and Tarjan showed that MTF is 2-competitive, while Transpose does not have a constant competitive ratio [51]. Albers proved that TS is 2-competitive [2]. Since then, several other deterministic and randomized on-line algorithms have been studied using competitive analysis. (See [36, 2, 6, 34, 8] for some representative results, as well as the recent survey [38] on algorithms and models for list update).

Notwithstanding the extensive study of list update under the standard model, the validity of the latter has been debated. More precisely, Martínez and Roura [43] and Munro [45], independently drew attention to certain drawbacks of the standard cost model, which can better be demonstrated using a few examples. Let $(a_1, a_2, \ldots, a_l)$ be the list currently maintained by an algorithm $\mathcal{A}$. Martínez and Roura argued that in a realistic setting a complete rearrangement of all items in the list which precede item a_i would in practice require time proportional to i, while this operation has cost proportional to i^2 in the standard cost model. In a similar vein, Munro provided the example of accessing the last item of the list and then reversing the entire list. The real cost of this operation in an array or a linear link list should be $O(l)$; in contrast it costs about $l^2/2$ in the standard cost model.

As a consequence, one may informally observe that the standard model prevents online algorithms from using their true power. Martínez and Roura proposed a new model in which the cost of accessing the i-th item of the list plus the cost of reorganizing the first i items is linear in i. We will refer to this model as the *modified cost model.* Surprisingly, it turns out that the offline optimum benefits substantially more from this realistic adjustment than the online algorithms do. Indeed, under the modified model, every online algorithm has amortized cost of $\Theta(l)$ per access for some arbitrary long sequences, whereas an optimal offline algorithm incurs a cost of $\Theta(\log l)$ on every sequence and hence all online list update algorithm have a constant competitive ratio of $\Omega(l/\log l)$ [45]. One may be tempted to argue that this is proof that the modified model makes the offline optimum too powerful and hence this power should be removed, however this is not correct as in real life online algorithms can rearrange items at the cost prescribed by the model. Observe that the ineffectiveness of this power for improving the worst case competitive ratio does not preclude the possibility that under certain realistic input distributions (or other similar assumptions on the input) this power might be of use. Martínez and Roura observed this and posed the question of "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".

In this section we address this open problem by showing that under locality of reference assumptions, MTF is an optimal algorithm for list update. First, we show that under the modified cost model all list update algorithms are equivalent. This result parallels the equivalence of all lazy paging algorithms under bijective analysis as shown in Subsection 4.1, and in particular Theorem 4. We term a list update algorithm *economical* if it does not use paid exchanges.

Theorem 15. *Let $\mathcal{A}$ and $\mathcal{B}$ be two arbitrary economical online list update algorithms. Under the modified cost model, we have $\mathcal{A} \equiv_b \mathcal{B}$.*

Proof. We prove that for every $n \geq 1$ there is a bijection $b_n : \mathcal{I}_n \leftrightarrow \mathcal{I}_n$ so that $\mathcal{A}(\sigma) \leq \mathcal{B}(b_n(\sigma))$ for each $\sigma \in I_n$. We show this by induction on n, the length of the input sequence. Since $\mathcal{A}$ and $\mathcal{B}$ start with the same initial list, they incur the same cost on each sequence of length one. Therefore the statement trivially holds for $n = 1$. Assume that it is true for $n = k$. Thus there is a bijection $b_k : \mathcal{I}_k \leftrightarrow \mathcal{I}_k$ so that $\mathcal{A}(\sigma) \leq \mathcal{B}(b_k(\sigma))$ for each $\sigma \in I_k$. Let σ be an arbitrary sequence of length k and $\sigma' = b_k(\sigma)$. Denote by $\mathcal{I}_{k+1}(\sigma)$ the set of sequences in $\mathcal{I}_{k+1}$ which have σ as their prefix. We map $\mathcal{I}_{k+1}(\sigma)$ to $\mathcal{I}_{k+1}(\sigma')$ as follows. Let $\mathcal{L}(\mathcal{A}, \sigma) = (a_1, a_2, \ldots, a_l)$ be the list maintained by $\mathcal{A}$ after serving σ and $\mathcal{L}(\mathcal{B}, \sigma') = (b_1, b_2, \ldots, b_l)$ be the list maintained by $\mathcal{B}$ after serving σ'. Consider an arbitrary sequence $\sigma_1 \in \mathcal{I}_{k+1}(\sigma)$ and let its last element be a request to item a_i. We map σ_1 to the sequence $\sigma_2 \in \mathcal{I}_{k+1}(\sigma')$ that has b_i as its last request. Since $\mathcal{A}(\sigma) \leq \mathcal{B}(\sigma')$ and $\mathcal{A}$'s cost on the last request of σ_1 is the same as $\mathcal{B}$'s cost on the last request of σ_2, we have $\mathcal{A}(\sigma_1) \leq \mathcal{B}(\sigma_2)$. Therefore we get the desired mapping from $\mathcal{I}_{k+1}(\sigma)$ to $\mathcal{I}_{k+1}(\sigma')$. We obtain a bijection $b_{k+1} : \mathcal{I}_{k+1} \leftrightarrow \mathcal{I}_{k+1}$ by considering the above mapping for each sequence $\sigma \in \mathcal{I}_k$. Thus our induction statement is true and we have $\mathcal{A} \preceq_b \mathcal{B}$. Using a similar argument, we can show $\mathcal{B} \preceq_b \mathcal{A}$. Therefore we obtain that $\mathcal{A} \equiv_b \mathcal{B}$. □

Since an economical list update algorithm does not incur any cost for reorganizing the list we can prove the following statement using an argument analogous to the proof of Theorem 15.

Corollary 16. *All economical online list update algorithms are equivalent according to bijective analysis under the standard cost model.*

The results above suggest that so long as we consider the space of all possible request sequences, all on-line list update algorithms are equivalent in a strong sense. In the following section we address the issue of locality of reference in typical list-update sequences, and its effect in the relative performance of online algorithms.

5.2.1 List Update with Locality of Reference

Unlike the paging problem, the prevalence of locality of reference in list update is less self-evident. In practice, input sequences of list update algorithms indeed exhibit locality of reference [35, 49, 16] and efficient on-line list update algorithms try to take advantage of this property [35, 48]. In particular, locality is prevalent in applications of list update algorithms in data compression (see e.g. [13]).

This lack of formal models led Hester and Hirschberg [35] to pose the question of providing a satisfactory formal definition of locality of reference for list update as an open problem. In a preliminary version of this work [10] we addressed the problem of list update under locality of reference. To the best of our knowledge, this was the first formal study of locality of reference for list update. Subsequently to the conference versions of this paper, Albers and Lauer [4] proposed another model for list update with locality of reference that is based on the concept of runs; here a run is a subsequence of requests to the same list item. For this model [4] confirmed that MTF exhibits the best performance, and that its refined competitive ratio tends to 1, as locality increases. It should be noted that it has been commonly assumed, based on intuition and experimental evidence, that MTF is indeed an excellent algorithm on sequences with high locality of reference. For instance, Hester and Hirschberg [35] claim: "move-to-front performs best when the list has a high degree of locality" (see also [5], page 327). The results of Albers and Lauer [4] confirm, in a theoretical manner, that MTF has excellent performance at high locality. Later studies of the impact of locality of reference in list update can be found in [33, 31].

Following the model of concave analysis for the paging problem of Albers *et al.* [17], we say that a request sequence σ for list update is *consistent* with f if the maximum number of distinct items in a window of w consecutive items in σ is at most $f(w)$. In Section 5.2.2 we provide experimental results that demonstrate that for applications of list update related data compression, the function f has indeed an overall concave shape. Hence, the experimental evidence suggests that concave analysis can be applied not only to the paging problem but also in the context of list update.

Perhaps not surprisingly, this measure seems to parallel MTF's behaviour as the latter has been tailored to benefit from locality of reference. This should not be construed as a drawback of the measure, but rather as evidence of the fact that the design of the MTF algorithm incorporates the presence of this type of locality of reference into its choices. Our theoretical proof of the optimality of MTF in this context is then perhaps not surprising, yet this fact had eluded proof until now.

As with the paging problem, we can apply bijective and/or average analysis by restricting the set of request sequences to those consistent with f. For the remainder of the section we will thus make use of the notation and terminology introduced in Subsection 4.2.

Lemma 17. *For every online list update algorithm $\mathcal{A}$, MTF dominates $\mathcal{A}$.*

Proof. Let f be an arbitrary concave* function and m be a positive integer. For any $1 \leq i \leq m$, let $\mathcal{F}_{i,m}(\mathcal{A})$ be the total cost $\mathcal{A}$ incurs on the i^{th} request of all sequences in $\mathcal{I}_m^f$. We will first show that $\mathcal{F}_{i,m}(MTF) \leq \mathcal{F}_{i,m}(\mathcal{A})$ for any $1 \leq i \leq m$. For $i = 1$, we have $\mathcal{F}_{1,m}(MTF) \leq \mathcal{F}_{1,m}(\mathcal{A})$, as all algorithms start with the same list. Now suppose that $i > 1$. Let σ be an arbitrary sequence of length $i-1$, T_σ denote the set of all sequences in $\mathcal{I}_m^f$ that have σ as their prefix, and $\mathcal{F}_{i,m}(\mathcal{A}\,|\,\sigma)$ be the total cost $\mathcal{A}$ incurs on the i^{th} request of all sequences in T_σ. Denote by $\mathcal{L}(MTF,\sigma) = (a_1, a_2, \ldots, a_l)$ and $\mathcal{L}(\mathcal{A},\sigma) = (b_1, b_2, \ldots, b_l)$ the lists maintained by MTF and $\mathcal{A}$ after serving σ, respectively. Suppose that c_j (resp., d_j) sequences in T_σ have a_j (resp., b_j) as their i^{th} request, for $1 \leq j \leq l$. Note that $\sum_{1\leq j\leq l} c_j = \sum_{1\leq j\leq l} d_j = |T_\sigma|$ and $(d_1, d_2, \ldots, d_l)$ is a permutation of $(c_1, c_2, \ldots, c_l)$.

We first show that $c_{j+1} \leq c_j$ for $1 \leq j < l$. Let C_j and C_{j+1} denote the set of sequences in T_σ that have a_j and a_{j+1} as their i^{th} request. We provide a one-to-one mapping from C_{j+1} to

C_j which proves that $|C_{j+1}| \leq |C_j|$. We map every sequence τ in C_{j+1} to a sequence τ' in C_j by replacing every a_j with a_{j+1} and every a_{j+1} by a_j, starting from position i. Since a_j occurs before a_{j+1} in MTF's list after serving σ, we know that the last request to a_j occurs after the last request to a_{j+1} in σ. Therefore if τ is consistent with f, so is τ'. Thus every sequence in C_{j+1} is mapped to a unique sequence in C_j and we have $c_{j+1} = |C_{j+1}| \leq |C_j| = c_j$.

Therefore $(c_1, c_2, \ldots, c_l)$ is a permutation of $(d_1, d_2, \ldots, d_l)$ in non-increasing order, and thus $\mathcal{F}_{i,m}(MTF \,|\, \sigma) = \sum_{1 \leq j \leq l} j \times c_j \leq \sum_{1 \leq j \leq l} j \times d_j = \mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma)$. Now since

$$\mathcal{F}_{i,m}(MTF) = \sum_{\sigma \in \mathcal{I}_{i-1}} \mathcal{F}_{i,m}(MTF \,|\, \sigma) \text{ and } \mathcal{F}_{i,m}(\mathcal{A}) = \sum_{\sigma \in \mathcal{I}_{i-1}} \mathcal{F}_{i,m}(\mathcal{A} \,|\, \sigma),$$

we obtain $\mathcal{F}_{i,m}(MTF) \leq F_{i,m}(\mathcal{A})$. We have

$$\sum_{\sigma \in \mathcal{I}^f_m} MTF(\sigma) = \sum_{1 \leq i \leq m} \mathcal{F}_{i,m}(MTF) \leq \sum_{1 \leq i \leq m} \mathcal{F}_{i,m}(\mathcal{A}) = \sum_{\sigma \in \mathcal{I}^f_m} \mathcal{A}(\sigma).$$

Thus MTF (m, f)-dominates $\mathcal{A}$ for every concave* function f, and every integer $m \geq 1$. Hence MTF dominates $\mathcal{A}$. □

Corollary 18. *For any* concave* *function f and any online list update algorithm $\mathcal{A}$,*

$$MTF \preceq^f_a \mathcal{A}.$$

We can further prove separation with respect to bijective analysis between MTF and specific algorithms, e.g., Transpose, for a much larger family of concave* functions.

Theorem 19. *For all* concave* *functions f such that $f(l) < l$ (l is the size of list),*

$$\textit{Transpose} \npreceq^f_b MTF.$$

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_l)$ be the initial list. Assume by way of contradiction that $Transpose \preceq^f_b MTF$. Therefore there is an integer $n_0 \geq 1$ so that for each $n \geq n_0$, there is a bijection $b : \mathcal{I}^f_n \leftrightarrow \mathcal{I}^f_n$ satisfying $\textit{Transpose}(\sigma) \leq MTF(b(\sigma))$ for each $\sigma \in \mathcal{I}^f_n$. Let σ denote a sequence of of length $m \geq n_0$ obtained by considering the prefix of size m of the infinite sequence $a_l a_{l-1} a_l a_{l-1} \ldots$. Note that transpose incurs a cost of l on each request and we have $\textit{Transpose}(\sigma) = m \times l$. Note also that σ is consistent with f, since it has two distinct items.[4] Thus $\sigma \in \mathcal{I}^f_m$ and from the assumption there should exist some sequence $\sigma' \in \mathcal{I}^f_m$ such that $m \times l = \textit{Transpose}(\sigma) \leq MTF(\sigma')$. Therefore MTF incurs a cost of l on each request of σ'. Hence σ' is a prefix of the sequence $a_l a_{l-1} a_{l-2} \ldots a_1 a_l a_{l-1} a_{l-2} \ldots a_1 \ldots$. Note, however, that any window of size l in σ' has l distinct items. Since we started with $f(l) < l$, σ' is not consistent with f and this contradicts the assumption that $\sigma' \in \mathcal{I}^f_m$. □

5.2.2 Experimental Results and Analysis

In this section we test the validity of the concave-function model of locality of reference, as described in Section 5.2.1, against experimental data. In our experiments, we considered the fourteen files of

[4] We can assume that $f(2) = 2$ since otherwise we are restricted to sequences that contain only one item.

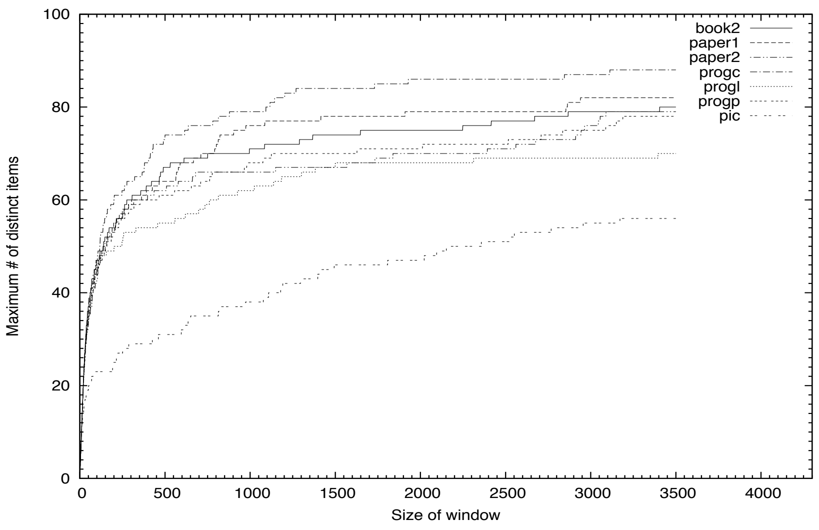

Figure 2: Maximum number of distinct characters in windows of size up to 3500 for the files in Calgary Compression Corpus (part 1).

the Calgary Compression Corpus [53] which are frequently used as the standard benchmark for file compression. Recall that list update algorithms can be used in a very direct way in file compression. For each file, we computed the maximum number of characters in windows of all possible sizes, up to the size of the whole file. Figures 2 and 3 show the resulting graphs. We note that in our experiments we observed that the maximum number of distinct items does not change significantly as the window size exceeds the value of 3500, and for this reason we only show the results for windows of size up to 3500.

As can be observed from these graphs, the curves have an overall concave shape. We should note that for some of the input files, the function we obtained is not concave for some intervals. However, this is not a major concern, since we can bound the said function by any concave function f which is such that $f(i)$ is an upper bound on the maximum number of distinct items in windows of size i. For instance, we can take the upper convex hull of the data points. It should be emphasized that Albers *et al.* [3] observed that similar non-concavity (mostly localized within small intervals) was present in their experimental results concerning locality of reference in typical request sequences for the paging problem. Albers *et al.* put forth this argument to justify the fact that local small deviations from concavity do not impose a serious problem.

Albers and Mitzenmacher [5] compared the efficiency of MTF and TS algorithms for compressing the files of the Calgary Compression Corpus. After accessing an item a, TS inserts a in front of the first item b that appears before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorganize the list. They compared MTF and TS in two settings: with or without Burrows-Wheeler transform (BWT). Informally, BWT transforms a string to one of its permutations that has more locality of reference, which is hence more readily compressible [27, 39]. Their results show that although TS outperforms MTF on compression without BWT, MTF usually has better performance when

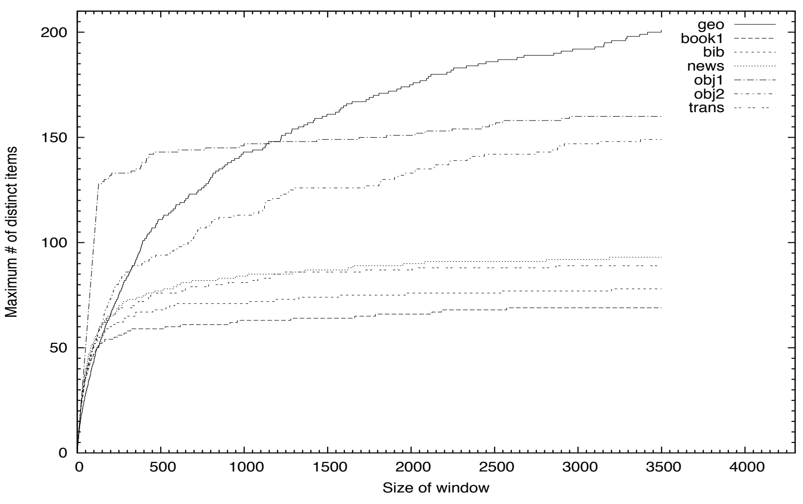

Figure 3: Maximum number of distinct characters in windows of size up to 3500 for the files in Calgary Compression Corpus (part 2).

we use BWT. This is consistent with our results as BWT is a transform designed with the goal of increasing the locality of reference in the representation of the string. We emphasize that our experimental results did not use BWT.

6 Conclusion

In this paper we introduced bijective analysis and average analysis as two new techniques for comparing the performance of online algorithms. These measures compare online algorithms over all sequences of the same length, rather than solely on the worst-case sequences. We demonstrated how the measure can be applied to two well-studied online problems, namely paging and list update.

For the paging problem, we showed that the new comparison techniques overcome some of the shortcomings of competitive analysis, namely they reflect the influence of lookahead and are able to separate the performance of LRU and FWF. We also proved that all lazy algorithms are equivalent according to bijective analysis. This result provides an intuitive explanation of why it is difficult, or even impossible, for several known measures to distinguish between the performance of known algorithms. Although a negative result at first sight, the result suggests that unless one considers typical request sequences for the problem, it is not likely that a meaningful measure can separate any two algorithms.

In view of this result, we turned our attention to the definitive property of typical request sequences for the paging problem, namely locality of reference. We relied on a natural model of locality of reference, due to Albers et al. [3], namely concave analysis. We then showed that when combining average and concave analysis, LRU emerges as the unique optimal on-line paging algorithm. Since, in practice, input sequences are known to exhibit locality of reference, this justifies theoretically why LRU is believed to have the best practical performance among on-line paging algorithms.

We then turned our attention to the list update problem, and in particular we focused in the cost formulation of Martínez and Roura, and Munro. We showed that under this cost model, all economical algorithms for list update are equivalent according to bijective analysis, a result that draws parallels with the equivalence of all lazy paging algorithms. Next, we investigated the issue of the locality of reference in list update, which, unlike the paging problem, has received considerably less attention in the literature. Here, we applied average analysis over request sequences exhibiting locality of reference, as defined by concave analysis, and proved that Move-to-Front is an optimal algorithm. We also provided experimental evidence that the model of Albers *et al.* can be adapted to applications of list update that emerge from text compression.

One may observe that in practice, and more specifically when considering disk accesses in databases and web servers, page replacement strategies that are modifications of LRU often outperform LRU. Examples of such strategies are the LRU-k algorithm [46], in which the k-th least recently accessed page is evicted upon a page fault, and the Adaptive Replacement Cache (ARC) algorithm [44], which, informally takes into account not only the "recency" of a request, but also its "frequency". This should not be seen as a deficiency of bijective and average analysis, but rather points to the direction of establishing much better theoretical models that capture the essential characteristics of typical request sequences. For instance, database applications often access data structures such as B-trees: in these cases, the locality model as provided by concave analysis is clearly inadequate, since it was not meant to model such accesses in the first place. An important open problem would be to confirm under the new measures the theoretical superiority of policies such as LRU-k and ARC, after providing, first, a model of typical request sequences. A first step towards this directions was done by Boyar *et al.* [18], which gave a theoretical separation of LRU-k from all other algorithms using relative worst-order analysis.

There are several other directions for future work involving the techniques we introduced in this paper. First and foremost, we would like to apply bijective and average analysis to other important online problems, e.g., the k-server and file caching problems. To this end, we believe that it is important to study relaxed versions of bijective analysis, namely by requiring that $A(\sigma) \leq cB(\pi(\sigma))$, for some (small) c, when comparing algorithms A and B. This relaxation, which was recently introduced in [11] guarantees that bijective analysis can be applied to any optimization problem, and can always reveal the pair-wise relative performance of two online algorithms. Since the conference versions of this paper appeared, further applications of these techniques emerged. More specifically, Angelopoulos and Schweitzer [12] extended the optimality of LRU and MTF to bijective analysis. In addition, Boyar, Irani and Larsen [23] showed that the greedy algorithm is optimal according to bijective analysis for the 2-server problem on three co-linear points. A natural question is whether the result extends in more complicated variants of the problem, e.g., for the 2-server problem on the line.

We believe that our techniques will prove their true potential if they further succeed to separate algorithms based on their intuitive, empirical, or even anticipated performance. For instance, can we separate the greedy and the naive algorithms in several natural variants of the Steiner tree problem (see [9] for the discussion of the deficiencies of competitive analysis when applied to these problems)? In a different vein, is it possible to extend bijective analysis to the context of randomized algorithms? Last, it would be interesting to explore the complexity of finding optimal algorithms (if possible) according to bijective analysis, along the lines of work by Burley and Irani [26] who studied this problem in the context of competitive analysis.

Acknowledgements. We are grateful to an anonymous referee for the extraordinary effort and care that gave to this paper; the in-depth comments of this referee helped improve substantially the paper's overall quality, and in particular, the proof of Theorem 11. We also thank Ian Munro for early discussions on alternative measures of performance for online algorithms, as well as Joan Boyar and Kim Larsen for many insightful comments on an earlier version of this paper.

This work was the result of an exciting collaboration with our mentor and friend, Alex López-Ortiz, who sadly passed away before this submission was accepted for publication. We would like to dedicate the final paper to his memory.

References

[1] S. Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, July 1997.

[2] S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, June 1998.

[3] S. Albers, L.M. Favrholdt, and O. Giel. On paging with locality of reference. *Journal of Computer and System Sciences*, 70(2):145–175, 2005.

[4] S. Albers and S. Lauer. On list update with locality of reference. *Journal of Computer and System Sciences*, 82(5):627–653, 2016.

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

[6] S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, 56:135–139, 1995.

[7] S. Albers and J. Westbrook. Self-organizing data structures. In *Online Algorithms: The State of the Art*, Lecture Notes in Computer Science 1442, pages 13–51. Springer, 1998.

[8] C. Ambühl, B. Gärtner, and B. von Stengel. Optimal lower bounds for projective list update algorithms. *ACM Transactions on Algorithms*, 9(4), 2013.

[9] S. Angelopoulos. Parameterized analysis of online Steiner tree problems. In *Adaptive, Output Sensitive, Online and Parameterized Algorithms*, number 09171 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[10] S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. List update with locality of reference. In *Proceedings of the 8th Latin American Symposium on Theoretical Informatics*, pages 399–410, 2008.

[11] S. Angelopoulos, M. Renault, and P. Schweitzer. Stochastic dominance and the bijective ratio of online algorithms, 2016. arXiv:1607.06132.

[12] S. Angelopoulos and P. Schweitzer. Paging and list update under bijective analysis. *Journal of the ACM*, 60(2), 2013.

[13] R. Bachrach, R. El-Yaniv, and M. Reinstädler. On the competitive theory and practice of online list accessing algorithms. *Algorithmica*, 32(2):201–245, 2002.

[14] L. Becchetti. Modeling locality: A probabilistic analysis of LRU and FWF. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA '04)*, volume 3221 of *Lecture Notes in Computer Science*, pages 98–109, 2004.

[15] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11:73–91, 1994.

[16] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[17] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50:244–258, 1995.

[18] J. Boyar, M. R. Ehmsen, J.S. Kohrt, and K. S. Larsen. A theoretical comparison of LRU and LRU-k. *Acta Inf.*, 47(7–8):359–374, 2010.

[19] J. Boyar and L. M. Favrholdt. The relative worst order ratio for on-line algorithms. *Transactions on Algorithms*, 3(2), 2007.

[20] 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.

[21] J. Boyar, S. Gupta, and K. S. Larsen. Access graph results for LRU versus FIFO under relative worst order analysis. In *Proceedings of the 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 328–339, 2012.

[22] J. Boyar, S. Gupta, and K. S. Larsen. Relative interval analysis of paging algorithms on access graphs. *Theoretical Computer Science*, 568:28–48, 2015.

[23] J. Boyar, S. Irani, and K. S. Larsen. A comparison of performance measures for online algorithms. *Algorithmica*, 72(4):969–994, 2015.

[24] J. Boyar, K. S. Larsen, and A. Maiti. A comparison of performance measures via online search. *Theor. Comput. Sci.*, 532:2–13, 2014.

[25] J. Boyar and P. Medvedev. The relative worst order ratio applied to seat reservation. *ACM Transactions on Algorithms*, 4(4), 2008.

[26] W. R. Burley and S. Irani. On algorithm design for metrical task systems. *Algorithmica*, 18(4):461–485, 1997.

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

[28] M. Chrobak and J. Noga. LRU is better than FIFO. *Algorithmica*, 23(2):180–185, 1999.

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

[30] R. Dorrigiv, Alejandro A. López-Ortiz, and J. I. Munro. On the relative dominance of paging algorithms. *Theoretical Computer Science*, 410(38-40):3694–3701, 2009.

[31] R. Dorrigiv, M. R. Ehmsen, and A. López-Ortiz. Parameterized analysis of paging and list update algorithms. *Algorithmica*, 71(2):330–353, 2015.

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

[33] R. Dorrigiv and A. López-Ortiz. List update with probabilistic locality of reference. *Information Processing Letters*, 112(13):540–543, 2012.

[34] R. El-Yaniv. There are infinitely many competitive-optimal online list accessing algorithms. Discussion paper, Hebrew University of Jerusalem, Center for Rationality and Interactive Decision Theory.

[35] J.H. Hester and D.S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17(3):295, September 1985.

[36] S. Irani. Two results on the list update problem. *Information Processing Letters*, 38:301–306, 1991.

[37] S. Irani, A. R. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing*, 25:477–497, 1996.

[38] S. Kamali and A. López-Ortiz. A survey of algorithms and models for list update. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 251–266, 2013.

[39] H. Kaplan, S. Landau, and E. Verbin. A simpler analysis of Burrows-Wheeler based compression. *Theoretical Computer Science*, 387(3):220–235, 2007.

[40] A.R. Karlin, S.J. Phillips, and P. Raghavan. Markov paging. *SIAM Journal on Computing*, 30(3):906–922, 2000.

[41] C. Kenyon. Best-fit bin-packing with random order. In *ACM-SIAM SODA '96*, pages 359–364, 1996.

[42] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. *SIAM J. Comput.*, 30:300–317, 2000.

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

[44] N. Megiddo and D.S. Modha. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the FAST '03 Conference on File and Storage Technologies.*

[45] J.I. Munro. On the competitiveness of linear search. In *Proceedings of the 8th Annual European Symposium on Algorithms (ESA '00)*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345, 2000.

[46] E. J. O'Neil, P. E. O'Neil, and G. Weikum. An optimality proof of the LRU-K page replacement algorithm. *Journal of the ACM*, 46(1):92–112, 1999.

[47] K. Panagiotou and A. Souza. On adequate performance measures for paging. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC '06)*, pages 487–496, 2006.

[48] N. Reingold, J. Westbrook, and D.D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11:15–32, 1994.

[49] F. Schulz. Two new families of list update algorithms. In *Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC '98)*, volume 1533 of *Lecture Notes in Computer Science*, pages 99–108. Springer, 1998.

[50] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts.* John Wiley & Sons, 2002.

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

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

[53] I.H. Witten and T. Bell. The Calgary/Canterbury text compression corpus. Anonymous ftp from ftp.cpsc.ucalgary.ca /pub/text.compression/corpus/text.compression.corpus.tar.Z.

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

[55] N.E. Young. Bounding the diffuse adversary. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA '98)*, pages 420–425, 1998.

[56] N.E. Young. On-line paging against adversarially biased random inputs. *Journal of Algorithms*, 37(1):218–235, 2000.

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