Online Algorithms for Weighted Paging with Predictions

Zhihao Jiang[1]
Tsinghua University, China

Debmalya Panigrahi
Duke University, USA

Kevin Sun
Duke University, USA

Abstract

In this paper, we initiate the study of the weighted paging problem with predictions. This continues the recent line of work in online algorithms with predictions, particularly that of Lykouris and Vassilvitski (ICML 2018) and Rohatgi (SODA 2020) on unweighted paging with predictions. We show that unlike unweighted paging, neither a fixed lookahead nor knowledge of the next request for every page is sufficient information for an algorithm to overcome existing lower bounds in weighted paging. However, a combination of the two, which we call the strong per request prediction (SPRP) model, suffices to give a 2-competitive algorithm. We also explore the question of gracefully degrading algorithms with increasing prediction error, and give both upper and lower bounds for a set of natural measures of prediction error.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, paging

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.69

Funding *Debmalya Panigrahi*: Supported in part by NSF grants CCF-1535972, CCF-1955703, an NSF CAREER Award CCF-1750140, and the Indo-US Virtual Networked Joint Center on Algorithms under Uncertainty.
Kevin Sun: Supported in part by NSF grants CCF-1535972, CCF-1527084, CCF-1955703, and an NSF CAREER Award CCF-1750140.

[1] Work done while visiting Duke University.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 69; pp. 69:1–69:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The paging problem is among the most well-studied problems in online algorithms. In this problem, there is a set of n pages and a cache of size $k < n$. The online input comprises a sequence of requests for these pages. If the requested page is already in the cache, then the algorithm does not need to do anything. But, if the requested page is not in the cache, then the algorithm suffers what is known as a *cache miss* and must bring the requested page into the cache. If the cache is full, then an existing page must be evicted from the cache to make room for the new page. The goal of the online algorithm is to minimize the total number of cache misses in the *unweighted paging* problem, and the total weight of the evicted pages in the *weighted paging* problem. It is well-known that for both problems, the best deterministic algorithms have a competitive ratio of $O(k)$ and the best randomized algorithms have a competitive ratio of $O(\log k)$ (see, e.g., [4, 2]).

Although the paging problem is essentially solved from the perspective of competitive analysis, it also highlights the limitations of this framework. For instance, it fails to distinguish between algorithms that perform nearly optimally in practice such as the *least recently used* (LRU) rule and very naïve strategies such as *flush when full* that evicts all pages whenever the cache is full. In practice, paging algorithms are augmented with predictions about the future (such as those generated by machine learning models) to improve their empirical performance. To model this, for unweighted paging, several lookahead models have been proposed where only a partial prediction of the future leads to algorithms that are significantly better than what can be obtained in traditional competitive analysis. But, to the best of our knowledge, no such results were previously known for the weighted paging problem. In this paper, we initiate the study of the *weighted paging problem with future predictions*.

For unweighted paging, it is well-known that evicting the page whose next request is *farthest in the future* (also called Belady's rule) is optimal. As a consequence, it suffices for an online algorithm to simply predict the next request of every page (we call this *per request prediction* or PRP in short) in order to match offline performance. In fact, Lykouris and Vassilvitskii [9] (see also Rohatgi [13]) showed recently that in this prediction model, one can simultaneously achieve a competitive ratio of $O(1)$ if the predictions are accurate, and $O(\log k)$ regardless of the quality of the predictions. Earlier, Albers [1] used a different prediction model called *ℓ-strong lookahead*, where we predict a sequence of future requests that includes ℓ distinct pages (excluding the current request). For $\ell = n - 1$, this prediction is stronger than the PRP model, since the algorithm can possibly see multiple requests for a page in the lookahead sequence. But, for $\ell < n - 1$, which is typically the setting that this model is studied in, the two models are incomparable. The main result in [1] is to show that one can obtain a constant approximation for unweighted paging for $\ell \geq k - 2$.

Somewhat surprisingly, we show that neither of these models are sufficient for weighted paging. In particular, we show a lower bound of $\Omega(k)$ for deterministic algorithms and $\Omega(\log k)$ for randomized algorithms in the PRP model. These lower bounds match, up to constants, standard lower bounds for the online paging problem (without prediction) (see, e.g., [11]), hence establishing that the PRP model does not give any advantage to the online algorithm beyond the strict online setting. Next, we show that for ℓ-strong lookahead, even with $\ell = k$, there are lower bounds of $\Omega(k)$ for deterministic algorithms and $\Omega(\log k)$ for randomized algorithms, again asymptotically matching the lower bounds from online paging without prediction. Interestingly, however, we show that a combination of these prediction models is sufficient: if $\ell = n - 1$ in the strong lookahead setting, then we get predictions that subsume both models; and, in this case, we give a simple deterministic algorithm with a competitive ratio of 2 for weighted paging, thereby overcoming the online lower bounds.

Obtaining online algorithms with predictions, however, is fraught with the risk that the predictions are inaccurate which renders the analysis of the algorithms useless. Ideally, one would therefore, want the algorithms to also be robust, in that their performance gracefully degrades with increasing prediction error. Recently, there has been significant interest in designing online algorithms with predictions that achieve both these goals, of matching nearly offline performance if the predictions are correct, and of gracefully degrading as the prediction error increases. Originally proposed for the (unweighted) paging problem [9], this model has gained significant traction in the last couple of years and has been applied to problems in data structures [10], online decision making [12, 6], scheduling theory [12, 8], frequency estimation [7], etc. Our final result contributes to this line of research.

First, if the online algorithm and offline optimal solution both use a cache of size k, then we show that no algorithm can asymptotically benefit from the predictions while achieving sublinear dependence on the prediction error. Moreover, if we make the relatively modest assumption that the algorithm is allowed a cache that contains just 1 extra slot than that of the optimal solution, then we can achieve constant competitive ratio when the prediction error is small.

1.1 Overview of models and our results

Our first result is a lower bound for weighted paging in the PRP model. Recall that in the PRP model, in addition to the current page request, the online algorithm is provided the time-step for the next request of the same page. For instance, if the request sequence is $(a, b, a, c, d, b, \ldots)$, then at time-step 1, the algorithm sees request a and is given position 3, and at time-step 2, the algorithm sees request b and is given position 6.

▶ **Theorem 1.** *For weighted paging with PRP, any deterministic algorithm is $\Omega(k)$-competitive, and any randomized algorithm is $\Omega(\log k)$-competitive.*

Note that these bounds are tight, because there exist online algorithms without prediction whose competitive ratios match these bounds (see Chrobak *et al.* [4] and Bansal *et al.* [2]).

Next, for the ℓ-strong lookahead model, we show lower bounds for weighted paging. Recall that in this model, the algorithm is provided a lookahead into future requests that includes ℓ distinct pages. For instance, if $\ell = 3$ and the request sequence is $(a, b, a, c, d, b, \ldots)$, then at time-step 1, the algorithm sees request a and is given the lookahead sequence (b, a, c) since it includes 3 distinct pages. At time step 2, the algorithm sees request b and is given (a, c, d). Note the difference with the PRP model, which would not be give the information that the request in time-step 5 is for page d, but does give the information that the request in time-step 6 is for page b.

▶ **Theorem 2.** *For weighted paging with ℓ-strong lookahead where $\ell \leq n-k$, any deterministic algorithm is $\Omega(k)$-competitive, and any randomized algorithm is $\Omega(\log k)$-competitive.*

For weighted paging with ℓ-strong lookahead where $n-k+1 \leq \ell \leq n-1$, any deterministic algorithm is $\Omega(n-\ell)$-competitive, and any randomized algorithm is $\Omega(\log(n-\ell))$-competitive.

In contrast to these lower bounds, we show that a prediction model that combines features of these individual models gives significant benefits to an online algorithm. In particular, combining PRP and ℓ-strong lookahead, we define the following prediction model:

SPRP (“strong per-request prediction”): On a request for page p, the predictor gives the next time-step when p will be requested *and all page requests till that request.*

This is similar to $(n-1)$-strong lookahead, but is slightly weaker in that it does not provide the first request of every page at the outset. After each of the n pages has been requested, SPRP and $(n-1)$-strong lookahead are equivalent.

▶ **Theorem 3.** *There is a deterministic 2-competitive for weighted paging with SPRP.*

So far, all of these results assume that the prediction model is completely correct. However, in general, predictions can have errors, and therefore, it is desirable that an algorithm gracefully degrades with increase in prediction error. To this end, we also give upper and lower bounds in terms of the prediction error.

For unweighted paging, Lykouris and Vassilvitski [9] basically considered two measures of prediction error. The first, called ℓ_{pd} in this paper, is defined as follows: For each input request p_t, we increase ℓ_{pd} by $w(p_t)$ times the absolute difference between the predicted next-arrival time and the actual next-arrival time. For unweighted paging, Lykouris and Vassilvitskii [9] gave an algorithm with cost $O(\mathsf{OPT} + \sqrt{\ell_{pd} \cdot \mathsf{OPT}})$. Unfortunately, we rule out an analogous result for weighted paging.

▶ **Theorem 4.** *For weighted paging with SPRP, there is no deterministic algorithm whose cost is $o(k)\cdot\mathsf{OPT}+o(\ell_{pd})$, and there is no randomized algorithm whose cost is $o(\log k)\cdot\mathsf{OPT}+o(\ell_{pd})$.*

It turns out that the ℓ_{pd} error measure is closely related to another natural error measure that we call the ℓ_1 measure. This is defined as follows: for each input request p_t, if the prediction q_t is not the same as p_t, then increase ℓ_1 by the sum of weights $w(p_t) + w(q_t)$. (This is the ℓ_1 distance between the predictions and actual requests in the standard weighted star metric space for the weighted paging problem.) The lower bound for ℓ_{pd} continues to hold for ℓ_1 as well, and is tight.

▶ **Theorem 5.** *For weighted paging with SPRP, there is no deterministic algorithm whose cost is $o(k)\cdot\mathsf{OPT}+o(\ell_1)$, and there is no randomized algorithm whose cost is $o(\log k)\cdot\mathsf{OPT}+o(\ell_1)$. Furthermore, there is a deterministic algorithm with SPRP with cost $O(\mathsf{OPT} + \ell_1)$.*

One criticism of both the ℓ_{pd} and ℓ_1 error measures is that they are not robust to insertions or deletions from the prediction stream. To counter this, Lykouris and Vassilvitski [9] used a variant of the classic edit distance measure, and showed a constant competitive ratio for this error measure. For weighted paging, we also consider a variant of edit distance, called ℓ_{ed} and formally defined in Section 5, which allows insertions and deletions between the predicted and actual request streams.[2] Unfortunately, as with ℓ_{pd} and ℓ_1, we rule out algorithms that asymptoticaly benefit from the predictions while achieving sublinear dependence on ℓ_{ed}. Furthermore, if the algorithm were to use a cache with even one extra slot than the optimal solution, then we show that even for weighted paging, we can achieve a constant competitive algorithm. We summarize these results in the next theorem.

▶ **Theorem 6.** *For weighted paging with SPRP, there is no deterministic algorithm whose cost is $o(k)\cdot\mathsf{OPT}+o(\ell_{ed})$, and there is no randomized algorithm whose cost is $o(\log k)\cdot\mathsf{OPT}+o(\ell_{ed})$. In the same setting, there exists a randomized algorithm that uses a cache of size $k+1$ whose cost is $O(\mathsf{OPT} + \ell_{ed})$, where OPT uses a cache of size k.*

[2] For technical reasons, neither ℓ_{ed} in this paper nor the edit distance variant in [9] exactly match the classical definition of edit distance.

1.2 Related work

We now give a brief overview of the online paging literature, highlighting the results that consider a prediction model for future requests. For unweighted paging, the optimal offline algorithm is Belady's algorithm, which always evicts the page that appears farthest in the future [3]. For online paging, Sleator and Tarjan [14] gave a deterministic k-competitive algorithm, and Fiat et al. [5] gave a randomized $O(\log k)$-competitive algorithm; both results were also shown to be optimal. For weighted online paging, Chrobak *et al.* [4] gave a deterministic k-competitive algorithm, and Bansal *et al.* [2] gave an $O(\log k)$-competitive randomized algorithm, which are also optimal by extension.

Recently, Lykouris and Vassilvitskii [9] introduced a prediction model that we call PRP in this paper: on each request p, the algorithm is given a prediction of the next time at which p will be requested. For unweighted paging, they gave a randomized algorithm, based on the "marker" algorithm of Fiat et al. [5], with competitive ratio $O(\min(\sqrt{\ell_{pd}/\mathsf{OPT}}, \log k))$. Here, ℓ_{pd} is the absolute difference between the predicted arrival and actual arrival times of requests, summed across all requests. They also perform a tighter analysis yielding a competitive ratio of $O(\min(\eta_{ed}/\mathsf{OPT}, \log k))$, where η_{ed} is the edit distance between the predicted sequence and the actual input. Subsequently, Rohatgi [13] improved the former bound to $O(1 + \min((\ell_{pd}/\mathsf{OPT})/k, 1) \log k)$ and also proved a lower bound of $\Omega(\log \min((\ell_{pd}/\mathsf{OPT})/(k \log k), k))$.

Albers [1] studied the ℓ-strong lookahead model: on each request p, the algorithm is shown the next ℓ distinct requests after p and all pages within this range. For unweighted paging, Albers [1] gave a deterministic $(k - \ell)$-competitive algorithm and a randomized $2H_{k-\ell}$-competitive algorithm. Albers also showed that these bounds are essentially tight: if $l \le k - 2$, then any deterministic algorithm has competitive ratio at least $k - \ell$, and any randomized algorithm has competitive ratio at least $\Omega(\log(k - \ell))$.

Finally, we review the paging model in which the offline adversary is restricted to a cache of size $h < k$, while the online algorithm uses a larger cache of size k. For this model, Young [16] gave a deterministic algorithm with competitive ratio $k/(k - h + 1)$ and showed that this is optimal. In another paper, Young [15] showed that the randomized "marker" algorithm is $O(\log(k/k - h))$-competitive and this bound is optimal up to constants.

Roadmap

In Section 2, we show the lower bounds stated in Theorem 1 for the PRP model. The lower bounds for the ℓ-strong lookahead model stated in Theorem 2 are proven in Section 3. In Section 4, we state and analyze the algorithm for the SPRP model with no error, thereby proving Theorem 3. Finally, in Section 5, we consider the SPRP model with errors, and focus on the upper and lower bounds in Theorems 4, 5, and 6. Detailed proofs of these bounds appear in the full version of this paper.

2 The Per-Request Prediction Model (PRP)

In this section, we give the lower bounds stated in Theorem 1 for the PRP model. Our strategy, at a high level, will be the same in both the deterministic and randomized cases: we consider the special case where the cache size is exactly one less than the number of distinct pages. We then provide an algorithm that generates a specific input. In the deterministic case, this input will be adversarial, based on the single page not being in the cache at any time. In the randomized case, the input will be oblivious to the choices made by the paging algorithm but will be drawn from a distribution. We will give a brief overview of the main ideas that are common to both lower bound constructions first, and then give the details of the randomized construction in this section. The details of the deterministic construction are deferred to the full paper.

Let us first recall the $\Omega(k)$ deterministic lower bound for unweighted caching without predictions. Suppose the cache has size k and the set of distinct pages is $\{a_0, a_1, \ldots, a_k\}$. At each step, the adversary requests the page a_ℓ not contained in the cache of the algorithm ALG. Then ALG incurs a miss at every step, while OPT, upon a miss, evicts the page whose next request is furthest in the future. Therefore, ALG misses at least k more times before OPT misses again.

Ideally, we would like to imitate this construction. But, the adversary cannot simply request the missing page a_ℓ because that could violate the predictions made on previous requests. Our first idea is to replace this single request for a_ℓ with a "block" of requests of pages *containing* a_ℓ in a manner that all the previous predictions are met, but ALG still incurs the cost of page a_ℓ in serving this block of requests.

But, how do we guarantee that OPT only misses requests once for every k blocks? Indeed, it is not possible to provide such a guarantee. Instead, as a surrogate for OPT, we use an array of k algorithms ALG_i for $1 \le i \le k$, where each ALG_i follows a fixed strategy: maintain all pages except a_0 and a_i permanently in the cache, and swap a_0 and a_i as required to serve their requests. Our goal is to show that the sum of costs of all these algorithms is a lower bound (up to constants) on the cost of ALG; this would clearly imply an $\Omega(k)$ lower bound.

This is where the weights of pages come handy. We set the weight $w(a_i)$ of page a_i in the following manner: $w(a_i) = c^i$ for some constant $c \ge 2$. Now, imagine that a block requested for a missing page a_ℓ only contains pages $a_0, a_1, \ldots, a_\ell$ (we call this an ℓ-block). The algorithms ALG_i for $i \le \ell$ suffer a cache miss on page a_i in this block, while the remaining algorithms ALG_i for $i > \ell$ do not suffer a cache miss in this block. Moreover, the sum of costs of all the algorithms ALG_i for $i \le \ell$ in this block is at most a constant times that of the cost of ALG alone, because of the geometric nature of the cost function.

The only difficulty is that by constructing blocks that do not contain pages a_i for $i > \ell$, we might be violating the previous predictions for these pages. To overcome this, we create an invariant where for every i, an $(i+1)$-block must be introduced after a fixed number of i-blocks. Because of this invariant, we are sometimes forced to introduce a larger block than that demanded by the missing page in ALG. To distinguish between these two types of blocks, we call the ones that exactly correspond to the missing page a *regular* block, and the ones that are larger *irregular* blocks. Irregular blocks help preserve the correctness of all previous predictions, but the sum of costs of ALG_i's on an irregular block can no longer be bounded against that of ALG. Nevertheless, we can show that the number of irregular blocks is small enough that this extra cost incurred by ALG_i's in irregular blocks can be charged off to the regular blocks, thereby proving the deterministic lower bound:

▶ **Theorem 7.** *For weighted paging with PRP, any deterministic algorithm is $\Omega(k)$-competitive.*

A formal proof of this theorem is deferred to the full paper. Instead, we focus on proving the lower bound for randomized algorithms.

2.1 Randomized Lower Bound

This subsection is devoted to proving the following theorem:

▶ **Theorem 8.** *For weighted paging with PRP, any randomized algorithm is $\Omega(\log k)$-competitive.*

Here, we still use the same idea of request blocks, but now the input is derived from a fixed distribution and is not aware of the state of ALG. The main idea is to design a distribution over block sizes in a manner that still causes any fixed deterministic algorithm ALG to suffer a large cost *in expectation*, and then invoke Yao's minimax principle to translate this to a randomized lower bound.

Let $H_k = 1 + 1/2 + \cdots + 1/k \approx \ln k$ denote the k-th harmonic number. The input is defined as follows:

1. For $0 \le i \le k$, set $u_i = (2ckH_k + 2)^i$ and let $y_i = 0$ for $i < k$.
2. Repeat the following:
   a. Select a value of ℓ according to the following probability distribution: $\Pr[\ell = j] = \frac{c-1}{c^{j+1}}$ for $j \in \{0, 1, \ldots, k-1\}$ and $\Pr[\ell = k] = \frac{1}{c^k}$.
   b. Increase ℓ until $\ell = k$ or $y_\ell < 2ckH_k$.
   c. For j from 0 to ℓ,
      i. Set all requests from time $t + 1$ through $u_j - 1$ as a_{j-1}. (Note: If $j = 0$, then $u_j = t + 1$, so this step is empty.)
      ii. Set the request at time u_j as a_j.
      iii. Let $t = u_j$.
   d. For $0 \le j \le \ell$, let $u_j = t + (2ckH_k + 2)^j$.
   e. For $0 \le j < \ell$, let $y_j = 0$. If $\ell < k$, increase y_ℓ by one.

Note that if ℓ is not increased in Step 2b, then this block is *regular*; otherwise, it is *irregular*. Let v_i denote the number of regular i-blocks, and let v'_i denote the number of irregular i-blocks. A j-block is an *i-plus* block if and only if $j \ge i$. We first lower bound the cost of ALG by the number of blocks.

▶ **Lemma 9.** *Every requested block increases $\mathbb{E}[\mathsf{cost}(\mathsf{ALG})]$ by at least a constant.*

Proof. At every time step, the cache of ALG is missing some page a_j. The probability that a_j is requested in the next block is at least $\Pr[\ell = j] \ge \frac{1}{2c^j}$, so the expected cost of serving this block is at least $c^j \cdot \Pr[\ell = j] = \Omega(1)$. ◀

For the rest of the proof, we upper bound the cost of OPT. We first upper bound the number of regular blocks, and then we use this to bound the number of irregular blocks.

▶ **Lemma 10.** *For every $i \in \{0, 1, \ldots, k\}$, we have $\mathbb{E}[v_i] \le 2c^{-i}m$.*

Proof. Consider the potential function $\phi(y) = \sum_{i=0}^{k-1} y_i \ge 0$. The initial value of $\phi(y)$ is 0. Notice that whenever a regular block is generated, $\phi(y)$ increases by at most 1, and whenever an irregular block is generated, $\phi(y)$ decreases by at least $2ckH_k$. Thus, the number of irregular blocks is at most the number of regular blocks, so the total number of blocks is at most $2m$. The lemma follows by noting that the probability that a block is a regular i-block is at most c^{-i}. ◀

▶ **Lemma 11.** *For every $i \in \{0, 1, \ldots, k\}$, we have $\mathbb{E}[v'_i] \le \frac{2m}{c^i kH_k}$.*

Proof. Observe that $v'_i \le \frac{1}{2ckH_k}(v'_{i-1} + v_{i-1})$ and $v'_1 \le \frac{1}{2ckH_k} v_0$. Repeatedly applying this inequality yields

$$\mathbb{E}[v'_i] \le \sum_{j=0}^{i-1} \frac{\mathbb{E}[v_j]}{(2ckH_k)^{i-j}} \le \sum_{j=0}^{i-1} \frac{2c^{-j}m}{(2ckH_k)^{i-j}} = \frac{2m}{c^i} \sum_{j=0}^{i-1} \frac{1}{(2kH_k)^{i-j}} \le \frac{2m}{c^i kH_k},$$

where the second inequality holds due to Lemma 10. ◀

Now let A denote the entire sequence of requests, B the subsequence of A comprising all regular blocks, and m the number of blocks in B. We bound $\mathsf{OPT} = \mathsf{OPT}(A)$ in terms of the optimal cost on B and the number of irregular blocks.

▶ **Lemma 12.** *Let $\mathsf{OPT}(A)$ and $\mathsf{OPT}(B)$ denote the optimal offline algorithm on request sequences A and B respectively. Then $\mathsf{cost}(\mathsf{OPT}(A)) \leq \mathsf{cost}(\mathsf{OPT}(B)) + 4c\sum_{i=0}^{k} v'_i c^i$.*

Proof. Consider the following algorithm ALG_A on request sequence A:

1. For requests in regular blocks, imitate $\mathsf{OPT}(B)$. That is, copy the cache contents when $\mathsf{OPT}(B)$ serves this block.
2. Upon the arrival of an irregular i-block, let a_ℓ denote the page not in the cache.
   a. If $\ell > i$, then the cost of serving this block is 0.
   b. If $1 \leq \ell \leq i$, evict a_0 when a_ℓ is requested. Then evict a_ℓ and fetch a_0 at the end of this block; the cost of this is $2(c^i + 1)$.
   c. If $\ell = 0$, we evict a_1 and fetch a_0 when a_0 is requested. Then we evict a_0 and fetch a_1 when a_1 is requested or at the end of this block (if a_1 is not requested in this block). The cost is $2(c + 1)$.

For each irregular block, notice that the cache of ALG_A is the same at the beginning and the end of the block. So Step 2 does not influence the imitation in Step 1. The cost of serving an irregular i-block is at most $4c^{i+1}$. Combining these facts proves the lemma. ◀

To bound $\mathsf{OPT}(B)$, we divide the sequence B into phases. Each phase is a contiguous sequence of blocks. Phases are defined recursively, starting with 0-phases all the way through to k-phases. A 0-phase is defined as a single request. For $i \geq 1$, let M_i denote the first time that an i-plus-block is requested and let Q_i denote the first time that c $(i-1)$-phases have appeared. An i-phase ends immediately after M_i and Q_i have both occurred. In other words, an i-phase is a minimal contiguous subsequence that contains c $(i-1)$-phases and an i-plus block. (Notice that for a fixed i, the set of i-phases partition the input sequence.)

For any k-phase, we upper bound OPT by considering an algorithm ALG_B^k that is optimal for B subject to the additional restriction that a_0 is not in the cache at the beginning or end of any k-phase. We bound the cost of ALG_B^k in any k-phase using a more general lemma.

▶ **Lemma 13.** *For any i, let ALG_B^i be an optimal algorithm on B subject to the following: a_0 is not in the cache at the beginning or the end of any i-phase. Then the cost of ALG_B^i within an i-phase is at most $4c^{i+1}$. In particular, in each k-phase, the algorithm ALG_B^k incurs cost at most $4c^{k+1}$.*

Proof. We shall prove this by induction on i. If $i = 0$, then the phase under consideration is one step. To serve one step, we can evict a_1 to serve a_0, and then evict a_0 if necessary for a total cost of $4c$. Now assume that the lemma holds for all values in $\{0, \ldots, i-1\}$. Let s_i denote the first i-plus block; there are two possible cases for the structure of an i-phase:

1. s_i appears after the c $(i-1)$-phases: In this case, the i-phase ends after this block. Thus, one strategy to serve the phase is to evict a_i at the beginning and evict a_0 when a_i is requested within s_i. These two evictions cost at most $4c^{i+1}$.
2. s_i appears within the first c $(i-1)$-phases: By the inductive hypothesis, the algorithm can serve these c $(i-1)$-phases with total cost at most $c \cdot 4c^i = 4c^{i+1}$. ◀

Finally, we lower bound the expected number of blocks in an i-phase. Since the total number of blocks is fixed, this allows us to upper bound the number of k-phases in the entire sequence. The next proposition forms the technical core of the lower bound:

▶ **Proposition 14.** *For $i \geq 1$, the expected number of blocks in an i-phase is at least $c^i H_i/4$.*

We defer the proof of Proposition 14 to the end of this section; first, we use it to prove Theorem 8.

Proof of Theorem 8. Let $\mathsf{OPT}(A)$ denote the cost of an optimal algorithm on the request sequence A, and let $\mathsf{OPT}(B)$ denote the cost of an optimal algorithm on the regular blocks B. Then we have the following:

$$\begin{aligned}\mathbb{E}\left[\mathsf{cost}(\mathsf{OPT}(A))\right] &\leq \mathbb{E}\left[\mathsf{cost}(\mathsf{OPT}(B))\right] + 4c\sum_{i=0}^{k} c^i \cdot \mathbb{E}\left[v_i'\right] && \text{(Lemma 12)}\\ &\leq \mathbb{E}\left[\mathsf{cost}(\mathsf{ALG}_B^k)\right] + 4c\sum_{i=0}^{k} c^i \cdot \frac{2m}{c^i k H_k} && \text{(Lemma 11)}\\ &\leq 4c^{k+1} \cdot \mathbb{E}\left[N_k(B)\right] + \frac{16cm}{H_k}, && \text{(Lemma 13)}\end{aligned}$$

where $N_k(B)$ denotes the number of k-phases in B. According to Proposition 14, the expected number of blocks in a k-phase is at least $c^k H_k/4$, which implies $\mathbb{E}\left[N_k(B)\right] \leq \frac{4m}{c^k H_k}$. Combining this with the above, we get

$$\mathbb{E}\left[\mathsf{cost}(\mathsf{OPT}(A))\right] \leq \frac{16cm}{H_k} + \frac{16cm}{H_k} = O\left(\frac{m}{H_k}\right).$$

Since any algorithm incurs at least some constant cost in every block by Lemma 9, its cost is $\Omega(m)$, which concludes the proof. ◀

Proof of Proposition 14

Let z_i be a random variable denoting the number of i-plus blocks in a fixed i-phase. We will first prove a sequence of three lemmas to yield a lower bound on $\mathbb{E}\left[z_i\right]$.

▶ **Lemma 15.** *For any $i \geq 1$, we have $\mathbb{E}\left[z_i\right] = \mathbb{E}\left[z_{i-1}\right] + \Pr\{M_i > Q_i\}$.*

Proof. Recall that an i-phase ends once it contains c $(i-1)$-phases and an i-plus block. In each of the $(i-1)$-phases, the expected number of $(i-1)$-plus blocks is $\mathbb{E}\left[z_{i-1}\right]$, so the total expected number of $(i-1)$-plus blocks in the first c $(i-1)$-phases of an i-phase is $c \cdot \mathbb{E}\left[z_{i-1}\right]$.

An elementary calculation shows that an $(i-1)$-plus block is an i-plus block with probability $1/c$. Thus, in expectation, the first c $(i-1)$-phases of this i-phase contain $\mathbb{E}\left[z_{i-1}\right]$ i-plus blocks.

If there are no i-plus blocks in the first c $(i-1)$-phases, then the i-phase ends as soon as an i-plus block appears. In this case, we have $z_i = 1$, and this happens with probability exactly $\Pr\{M_i > Q_i\}$. Otherwise, the i-phase ends immediately after the c $(i-1)$-phases, in which case no additional term is added. ◀

▶ **Lemma 16.** *For any $i \geq 1$, we have $\Pr\{M_i > Q_i\} \geq e^{-2\mathbb{E}[z_{i-1}]}$.*

Proof. We let $v_1, \ldots, v_c$ denote the number of i-plus blocks in the first c $(i-1)$-phases and let $V = \sum_{i=1}^{c} v_i$. As we saw in the proof of Lemma 15, an $(i-1)$-plus block is an i-plus block with probability $1/c$, so the probability that an $(i-1)$-plus block is an $(i-1)$-block is $1-1/c$. Thus, we have

$$\Pr\{M_i > Q_i\} = \mathbb{E}_{v_1,v_2,\ldots,v_c}\left[\left(1-\frac{1}{c}\right)^V\right] \geq \left(1-\frac{1}{c}\right)^{\mathbb{E}[V]} = \left(1-\frac{1}{c}\right)^{c\cdot\mathbb{E}[z_{i-1}]}$$

where the inequality follows from convexity and the second equality holds due to linearity of expectation. The lemma follows from this and the fact that $c \geq 2$. ◀

▶ **Lemma 17.** *For any $i \geq 0$, we have $\mathbb{E}[z_i] \geq \frac{1}{4} H_i$.*

Proof. When $i \leq 4$, we have $\mathbb{E}[z_i] \geq 1 \geq \frac{1}{4} H_i$. Now for induction, assume the statement holds for $j < i$, and consider the two possible cases:

1. If $\mathbb{E}[z_{i-1}] \geq \frac{1}{2} H_{i-1}$, then Lemma 15 implies $\mathbb{E}[z_i] \geq \mathbb{E}[z_{i-1}] \geq \frac{1}{4} H_i$.
2. If $\mathbb{E}[z_{i-1}] < \frac{1}{2} H_{i-1} < \frac{1}{2}(1 + \ln(i-1))$, then $\mathbb{E}[z_i] = \mathbb{E}[z_{i-1}] + \Pr\{M_i > Q_i\} \geq \frac{1}{4} H_{i-1} + e^{-2 \cdot \mathbb{E}[z_{i-1}]}$, where the equality follows from Lemma 15 and the inequality holds by the induction hypothesis and Lemma 16. Thus, $\mathbb{E}[z_i] \geq \frac{1}{4} H_{i-1} + \frac{1}{e} \cdot \frac{1}{i-1} \geq \frac{1}{4} H_i$. ◀

Now let L_i denote the number of blocks in an i-phase; recall that our goal is to lower bound its expectation by $c^i H_i / 4$. The following lemma relates L_i to z_i.

▶ **Lemma 18.** *For any $i \geq 0$, we have $\mathbb{E}[L_i] = c^i \cdot \mathbb{E}[z_i]$.*

Proof. When $i = 0$, the lemma holds because $E[L_0] = E[z_0] = 1$, so now we assume $i \geq 1$. Recall that an i-phase contains at least c $(i-1)$-phases, so the expected total number of blocks in the first c $(i-1)$-phases of this i-phase is $c \cdot \mathbb{E}[L_{i-1}]$.

If there are no i-plus-blocks in these c $(i-1)$-phases, we need to wait for an i-plus block to appear in order for the i-phase to end. This is a geometric random variable with expectation c^i. Thus, we have: $\mathbb{E}[L_i] = c \cdot \mathbb{E}[L_{i-1}] + c^i \cdot \Pr\{M_i > Q_i\}$. Applying this recursively,

$$\mathbb{E}[L_i] = c^i \left(\sum_{j=1}^{i} \Pr\{M_j > Q_j\} + \mathbb{E}[L_0] \right) = c^i \left(\sum_{j=1}^{i} \Pr\{M_j > Q_j\} + 1 \right)$$

Furthermore, from Lemma 15, we have

$$\mathbb{E}[z_i] = \mathbb{E}[z_{i-1}] + \Pr\{M_i > Q_i\} = \mathbb{E}[z_0] + \sum_{j=1}^{i} \Pr\{M_j > Q_j\} = 1 + \sum_{j=1}^{i} \Pr\{M_j > Q_j\}.$$

Combining the two equalities yields the lemma. ◀

We conclude by proving Proposition 14. Fix some $i \geq 1$. Using Lemma 18 and Lemma 17, we get $\mathbb{E}[L_i] = c^i \cdot \mathbb{E}[z_i] \geq \frac{c^i H_i}{4}$.

3 The ℓ-Strong Lookahead Model

Now we consider the following prediction model: at each time t, the algorithm can see request p_t as well as $L(t)$, which is the set of all requests through the ℓ-th distinct request. In other words, the algorithm can always see the next contiguous subsequence of ℓ distinct pages (excluding p_t) for a fixed value of ℓ. This model was introduced by Albers [1], who (among other things) proved the following lower bounds on algorithms with ℓ-strong lookahead.

▶ **Lemma 19** ([1])**.** *For unweighted paging with ℓ-strong lookahead where $\ell \leq k-2$, any deterministic algorithm is $\Omega(k-\ell)$-competitive. For randomized algorithms, the bound is $\Omega(\log(k-\ell))$.*

Notice that Lemma 19 implies that for small values of ℓ, ℓ-strong lookahead provides no asymptotic improvement to the competitive ratio of any algorithm. The proof proceeds by constructing a particular sequence of requests and analyzing the performance of any algorithm on this sequence. By slightly modifying the sequence, we can prove a similar result for the weighted paging problem.

▶ **Theorem 20.** *For weighted paging with ℓ-strong lookahead where $n - k + 1 \leq \ell \leq n - 1$, any deterministic algorithm is $\Omega(n - \ell)$-competitive, and any randomized algorithm is $\Omega(\log(n - \ell))$-competitive.*

Proof. We modify the adversarial input in Lemma 19 as follows: insert $n - k - 1$ distinct pages with very low weight between every two pages. This causes the lookahead to have effective size $\ell' = \ell - (n - k - 1)$, because at any point $L(t)$ contains at most ℓ' pages with normal weight. Note that if $\ell \leq n - k$, then $\ell' \leq 1$, and from Lemma 19, a lookahead of size 1 provides no asymptotic benefit to any algorithm.

If $\ell \leq n - 3$, then $\ell' \leq k - 2$. Thus, we can apply Lemma 19 to conclude that for any deterministic algorithm, the competitive ratio is $\Omega(k - \ell') = \Omega(n - \ell - 1)$, and for any randomized algorithm, the competitive ratio is $\Omega(\log(n - \ell - 1))$. Otherwise, if $\ell \geq n - 2$, then the lower bounds continue to hold because when $\ell = n - 3$, they are $\Omega(1)$. ◀

4 The Strong Per-Request Prediction Model (SPRP)

In this section, we define a simple algorithm called Static that is 2-competitive when the SPRP predictions are always correct. At any time step t, let $L(t)$ denote the set of pages in the current prediction. The Static algorithm runs on "batches" of requests. The first batch starts at $t = 1$ and comprises all requests in $L(1)$. The next batch starts once the first batch ends, i.e. at $|L(1)| + 1$, and comprises all predicted requests at that time, and so on. Within each batch, the Static algorithm runs the optimal offline strategy, computed at the beginning of the batch on the entire set of requests in the batch.

▶ **Theorem 21.** *The* Static *algorithm is 2-competitive when the predictions from SPRP are entirely correct.*

Proof. In this proof, we assume w.l.o.g. that evicting page p costs $w(p)$, and fetches can be performed for free. We partition the input into contiguous phases (which do not necessarily correspond to the batches of the algorithm) as follows: the first phase is simply the first request. Now for any $i \geq 2$, phase i is defined as the minimal subsequence of contiguous requests that contains all pages requested in phase $i - 1$, starting with the first request arriving after phase $i - 1$. In other words, if $P(i)$ denotes the set of pages that appear in phase i, then we require the $P(i)$ to be the minimal subsequence of contiguous requests that satisfy $\{p_1\} = P(1) \subseteq P(2) \subseteq \cdots \subseteq P(m - 1)$, where m denotes the total number of phases and p_1 is the first requested page. Note that we may not necessarily have $P(m - 1) \subseteq P(m)$ because of termination of the overall sequence.

Let OPT denote a fixed optimal offline algorithm for the entire sequence, and let OPT_i denote the cost of OPT incurred in phase i. Similarly, let S denote the total cost of Static, and let S_i denote the cost that Static incurs in phase i. So we have $\mathsf{OPT} = \sum_{i=1}^{m} \mathsf{OPT}_i$ and $S = \sum_{i=1}^{m} S_i$.

Now fix a phase index $j \in \{2, 3, \ldots, m\}$ and let $R(j)$ denote the sequence of requests in this phase. Furthermore, let $C(\mathsf{OPT}_{j-1})$ and $C(S_{j-1})$ denote the cache states of OPT and Static immediately before phase j. We know that Static runs an optimal offline algorithm on $R(j)$. One feasible solution is to immediately change the cache state to $C(\mathsf{OPT}_{j-1})$, and then imitate what OPT does in phase j. Since we charge for evictions, we have

$$S_j \leq \mathsf{OPT}_j + \sum_{p \in C(S_{j-1}) \setminus C(\mathsf{OPT}_{j-1})} w(p), \text{ for every } j \in \{2, 3, \ldots, m\}.$$

Consider some $p \in C(S_{j-1}) \setminus C(\mathsf{OPT}_{j-1})$: since $p \in C(S_{j-1})$, we know p must appear in $P(j-1)$ because Static does not fetch pages that have never been requested. Furthermore, since $p \notin C(\mathsf{OPT}_{j-1})$, then at some point in phase $j-1$, OPT must have evicted p because it appeared in $P(j-1)$ but is not in $C(\mathsf{OPT}_{j-1})$. Thus, $S_j \leq \mathsf{OPT}_j + \mathsf{OPT}_{j-1}$. Summing over all $j \geq 2$ and $S_1 \leq \mathsf{OPT}_1$ proves the theorem. ◀

5 The SPRP Model with Prediction Errors

In this section, we consider the SPRP prediction model with the possibility of prediction errors. We first define three measurements of error and then prove lower and upper bounds on algorithms with imperfect SPRP, in terms of these error measurements.

Let A denote a prediction sequence of length m, and let B denote an input sequence of length n. For any time t, let A_t and B_t denote the t-th element of A and B, respectively. We also define the following for any time step t:

- $\mathsf{prev}(t)$: The largest $i < t$ such that $B_i = B_t$ (or 0 if no such if no such i exists).
- $\mathsf{next}(t)$: The smallest $i > t$ such that $B_i = B_t$ (or $n+1$ if no such i exists).
- $\mathsf{pnext}(t)$: The smallest $i > t$ such that $A_i = B_t$ (or $m+1$ if no such i exists).
- We say two requests $A_i = B_j = p$ can be *matched* only if $\mathsf{pnext}(\mathsf{prev}(j)) = i$. In other words, A_i must be the earliest occurrence of p in A after the time of the last p in B before B_j. Furthermore, no edges in a matching are allowed to cross.

First, we define a variant of edit distance between the two sequences.

▶ **Definition 22.** *The* edit distance ℓ_{ed} *between A and B is the total minimum weight of unmatched elements of A and B.*

Next, we define an error measure based on the metric 1-norm distance between corresponding requests on the standard weighted star metric denoting the weighted paging problem.

▶ **Definition 23.** *The* 1-norm distance ℓ_1 *between A and B is defined as follows:*

$$\ell_1 = \sum_{\substack{i=1 \\ A_i \neq B_i}}^{n} \left(w(A_i) + w(B_i) \right). \tag{1-norm}$$

Third, we define an error measure inspired by the PRP model that was also used in [9].

▶ **Definition 24.** *The* prediction distance ℓ_{pd} *between A and B is defined as follows:*

$$\ell_{pd} = \sum_{i=1}^{n} w(B_i) \cdot |\mathsf{next}(i) - \mathsf{pnext}(i)|.$$

5.1 Lower Bounds

In this section, we give an overview of the lower bounds stated in Theorems 4, 5, and 6. We focus on the ℓ_{ed} (i.e., Theorem 6) error measurement; the proofs for ℓ_1 and ℓ_{pd} follow similarly. We defer some of the proofs to the full paper.

Our high-level argument proceeds as follows: recall that in Section 2, we showed a lower bound of $\Omega(k)$ on the competitive ratio of deterministic PRP-based algorithms. Given an SPRP algorithm ALG, we design a PRP algorithm ALG' specifically for the input generated by the procedure described in Section 2. (Recall that this input is a sequence of blocks, where a *block* is a string of a_0's, a_1's, and so on, ending with a single page a_ℓ for some ℓ.)

We show that if ALG has cost $o(k) \cdot \mathsf{OPT} + o(\ell_{ed})$ (where OPT is the optimal cost of the SPRP instance), then ALG' will have cost $o(k) \cdot \mathsf{OPT}'$ (where OPT' is the optimal cost of the PRP instance), which contradicts our PRP lower bound of $\Omega(k)$ on this input. For the randomized lower bound, we use the same line of reasoning, but replace $\Omega(k)$ with $\Omega(\log k)$.

Let k' denote the cache size of ALG'. Recall that the set of possible page requests received by ALG' is $A = \{a_0, a_1, \ldots, a_{k'}\}$ where $w(a_i) = c^i$ for some constant $c \geq 2$. The oracle ALG, maintained by ALG', has cache size $k = k' + 1$. The set of possible requests received by ALG is $A \cup \{b\}$ where $w(b) = 1/v$ for some sufficiently large value of v. (Thus, the instance for ALG has $k + 1$ distinct pages.) Our PRP algorithm ALG' must define a prediction and an input sequence for ALG.

The prediction sequence for ALG: For any strings X and Y, let $X + Y$ denote the concatenation of X and Y and let $\lambda \cdot X$ denote the concatenation of λ copies of X. Let $L = 2ck'H_{k'} + 1$, and consider the series of strings: $S_0 = 2 \cdot a_0$, and $S_i = L \cdot S_{i-1} + a_i$ for $i \in \{1, \ldots, k'\}$. We fix $S \coloneqq M \cdot S_{k'}$, for some sufficiently large M, as the prediction sequence for the SPRP algorithm. (Observe that S only contains k distinct pages, and the oracle ALG has cache size k.)

ALG' and the request sequence for ALG: Our PRP algorithm ALG' will simultaneously construct input for ALG while serving its own requests. Since randomized and fractional algorithms are equivalent up to constants (see Bansal et al. [2]), we view the SPRP algorithm ALG from a fractional perspective. Let $q_i \in [0, 1]$ denote the fraction of page a_i not in the cache of ALG. Notice that the vector $q = (q_0, q_1, \ldots, q_{k'})$ satisfies $\sum_{i=0}^{k'} q_i \geq 1$. (A deterministic algorithm is the special case where every $q_i \in \{0, 1\}$.) Similarly, let $q' = (q'_0, q'_1, \ldots, q'_{k'})$, where q'_i denotes the amount of request for a_i that is not in the cache in ALG'.

When a block ending with a_i is requested, ALG' scans S for the next appearance of a_i. It then feeds the scanned portion to ALG, followed by a single request for page b. In this case, the prediction error only occurs due to the requests for this page b. After serving this request b, the cache of ALG contains at most k' pages in A. This enables ALG' to mimic the behavior of ALG upon serving the current block. This process continues for every block: ALG' modifies the input by inserting an extra request b into the input for ALG, and mimics the resulting cache state of ALG. The details of our algorithm ALG' are given below:

1. Initially, let S be the input for ALG and $t = 0$. (We will modify S as time passes.)
2. For all $0 \leq i \leq k'$, let $q'_i = 1$. (Note that the initial value of every q_i is also 1.)
3. On PRP request block $s_i = (a_0, a_1, \ldots, a_i)$ (for some unknown i):
   - **a.** Let $q' = (q'_0, q'_1, \ldots, q'_{k'})$ denote the current cache state.
   - **b.** Set $q' = (0, \min\{1, q'_0 + q'_1\}, q'_2, q'_3, \ldots, q'_{k'})$ to serve a_0. Note that after we serve a_0, the PRP prediction tells us the value of i.
   - **c.** Find the first time t' after t when S requests a_i and set $t = t' + 2$.
   - **d.** Change the request at time t into b. (Note that the original request is a_0.)
   - **e.** Run ALG until this b is served to obtain a vector $q = (q_0, q_1, \ldots, q_{k'})$.
   - **f.** If $i \geq 1$, set $q' = (\min\{1, \sum_{j=0}^{i} q'_j\}, 0, 0, \ldots, 0, q'_{i+1}, q'_{i+2}, \ldots, q'_{k'})$; this serves the requests $(a_1, a_2, \ldots, a_i)$.
   - **g.** Set $q' = (q_0, q_1, \ldots, q_{k'})$.

Bounding the costs. The main idea in the analysis is the following: since the input sequences to ALG and ALG' are closely related, and they maintain similar cache states, we can show that they are coupled both in terms of the algorithm's cost and the optimal cost. Therefore, the ratio of $\Omega(k)$ for ALG' (from Theorem 7) translates to a ratio of $\Omega(k)$ for ALG. Furthermore, since the only prediction errors are due to the additional requests for page b, and this page has a very small weight, the cost of ALG is at least the value of ℓ_{ed}. (The same line of reasoning is used for randomized algorithms, but $\Omega(k)$ is replaced by $\Omega(\log k)$.)

We now formalize the above line of reasoning with the following lemmas.

▶ **Lemma 25.** *Using any SPRP algorithm* ALG *as a black box, the PRP algorithm* ALG' *satisfies the following:* $\mathsf{cost}(\mathsf{ALG}') \leq 2(c+1) \cdot \mathsf{cost}(\mathsf{ALG})$.

Proof. Note that $q = q'$ at the beginning and end of Step 3. For convenience, let q' denote the vector at the beginning of Step 3, and let q denote the vector at the end of Step 3. Let $\mathsf{cost}_{\mathsf{ALG}}$ and $\mathsf{cost}_{\mathsf{ALG}'}$ denote the cost of ALG and ALG' respectively incurred in a fixed Step 3.

Each time ALG' enters Step 3, the cost incurred is at most:

$$\text{Step 3b: } q'_0 \cdot (1+c),$$

$$\text{Step 3f: } (q'_0 + q'_1) \cdot (1+c) + \sum_{j=2}^{i} q'_j \cdot (1+c^j),$$

$$\text{Step 3g: } \left(\sum_{j=1}^{i} q_j \cdot (1+c^j)\right) + \left(\sum_{j=i+1}^{k} \left|q'_j - q_j\right| \cdot (1+c^j)\right).$$

Summing the above yields the following:

$$\mathsf{cost}_{\mathsf{ALG}'} \leq 2(c+1) \cdot \left(\left(\sum_{j=0}^{i} c^j \cdot \left(q_j + q'_j\right)\right) + \left(\sum_{j=i+1}^{k} c^j \cdot \left|q_j - q'_j\right|\right)\right).$$

Now we consider ALG. For each j, at the beginning of Step 3, there is q'_j amount of a_j not in the cache, and at the end of Step 3, there is q_j amount of a_j not in the cache.

If $j > i$, the cost incurred due to a_j is at least $c^j \cdot \left|q_j - q'_j\right|$. If $j \leq i$, ALG' must serve a_j at some point in Step 3e, so the incurred cost due to a_j is at least $c^j \cdot \left(q_j + q'_j\right)$. Summing the above yields the following:

$$\mathsf{cost}_{\mathsf{ALG}} \geq \left(\sum_{j=0}^{i} c^j \cdot \left(q_j + q'_j\right)\right) + \left(\sum_{j=i+1}^{k} c^j \cdot \left|q_j - q'_j\right|\right).$$

Combining the two inequalities above proves the lemma. ◀

Now let OPT denote the optimal SPRP algorithm for the input sequence served by ALG, and let OPT' denote the optimal PRP algorithm for the input sequence served by ALG'. We can similarly prove the following lemma (proof in full paper):

▶ **Lemma 26.** *The algorithms* OPT *and* OPT' *satisfy* $\mathsf{cost}(\mathsf{OPT}) \leq 2 \cdot \mathsf{cost}(\mathsf{OPT}')$.

We are now ready to bound the cost of any algorithm with SPRP (proof in full paper):

▶ **Theorem 27.** *For weighted paging with SPRP, there is no deterministic algorithm whose cost is* $o(k) \cdot \mathsf{OPT} + o(\ell_{ed})$*, and there is no randomized algorithm whose cost is* $o(\log k) \cdot \mathsf{OPT} + o(\ell_{ed})$.

Proof (Sketch). From Theorem 7, we know $\mathsf{ALG}' = \Omega(k) \cdot \mathsf{OPT}'$. Thus, applying Lemmas 25 and 26, we have $\mathsf{ALG} = \Omega(k) \cdot \mathsf{OPT}$. Furthermore (as we saw in Section 2), each PRP block increases ALG by at least a constant. At the same time, for each block, we can show that ℓ_{ed} increases by at most 2. As a result, we can conclude that $\mathsf{ALG} = \Omega(\ell_1)$. The theorem follows by combining these facts. For randomized algorithms, the same line of reasoning holds, but with $\Omega(\log k)$ instead of $\Omega(k)$. ◀

5.2 Upper Bounds

In this section, we give algorithms whose performance degrades with the value of the SPRP error. In particular, we first prove the upper bound in Theorem 6 for the ℓ_{ed} measurement, and then analyze the FOLLOW algorithm, which proves the upper bound in Theorem 5.

Now we present an algorithm that uses a cache of size $k+1$ whose cost scales linearly with $\mathsf{OPT} + \ell_{ed}$. Following our previous terminology, let A denote a prediction sequence of length m, and let B denote an input sequence of length n.

Our algorithm, which we call LEARN, relies on an algorithm that we call IDLE. At a high level, IDLE resembles STATIC (see Section 4): it partitions the prediction sequence A into batches and runs an optimal offline algorithm on each batch. The LEARN algorithm tracks the cost of imitating IDLE: if the cost is sufficiently low, then it will imitate IDLE on k of its cache slots; otherwise, it will simply evict the page in the extra cache slot.

Before formally defining IDLE, we consider a modified version of caching. Our cache has $k+1$ slots, where one slot is *memoryless*: it always immediately evicts the page it just fetched. In other words, this slot can serve any request, but it cannot store any pages. Let OPT^{+1} denote the optimal algorithm that uses a memoryless cache slot.

▶ **Lemma 28.** *For any sequences A and B, $\mathsf{cost}(\mathsf{OPT}^{+1}(A)) \leq \mathsf{cost}(\mathsf{OPT}(B)) + 2\ell_{ed}$, where ℓ_{ed} is the edit distance between A and B.*

Proof. Let M denote the optimal matching between A and B (for ℓ_{ed}). One algorithm for $\mathsf{OPT}^{+1}(A)$ is the following: imitate what $\mathsf{OPT}(B)$ does for requests matched by M, and use the memoryless slot for unmatched requests. The cost of this algorithm is $\mathsf{OPT}(B)+2\ell_{ed}$. ◀

Recall that the STATIC algorithm requires the use of an optimal offline algorithm. Similarly, for our new problem with a memoryless cache slot, we require a constant-approximation offline algorithm on A. This can be obtained from the following lemma (proof in full paper):

▶ **Lemma 29.** *Given a prediction sequence A, there is a randomized offline algorithm whose cost is at most a constant times the cost of $\mathsf{OPT}^{+1}(A)$.*

The Idle algorithm

Assume that our cache has size $k+1$ and the extra slot is memoryless (as defined above). For any time step t, let $L(t)$ denote the set of pages predicted to arrive starting at time $t+1$. At time step 1 (i.e., initially), IDLE runs the offline algorithm from Lemma 29 on $L(1)$, ignoring future requests. After the requests in $L(1)$ have been served, i.e., at time $|L(1)|+1$, IDLE then consults the predictor and runs the offline algorithm on the next "batch". The algorithm proceeds in this batch-by-batch manner until the end. We can show that the competitive ratio of this algorithm is at most a constant (see full paper).

▶ **Lemma 30.** *On the prediction sequence A, we have $\mathsf{cost}(\text{IDLE}) = O(1) \cdot \mathsf{cost}(\mathsf{OPT}^{+1}(A))$.*

The Learn algorithm

Before defining the algorithm, we introduce another measurement of error that closely approximates ℓ_{ed}. Recall that A denotes a prediction sequence of length m and B denotes an input sequence of length n. In defining ℓ_{ed}, two elements $A_i = B_j$ can be matched only if $\mathsf{pnext}(\mathsf{prev}(j)) = i$, and no matching edges are permitted to cross.

▶ **Definition 31.** *The* constrained edit distance ℓ'_{ed} *is the minimum weight of unmatched elements of A and B, with the following additional constraint: if $|P(A_i)| \geq 2$, then A_i can only be matched with the latest-arriving element in $P(A_i)$.*

We note that ℓ'_{ed} is a constant approximation of ℓ_{ed} (proof in full paper):

▶ **Lemma 32.** *For any sequences A, B, we have $\ell_{ed} \leq \ell'_{ed} \leq 3\ell_{ed}$.*

Now we are ready to define the Learn algorithm. For any $i \leq j$, we let $A(i,j)$ denote the subsequence $(A_i, A_{i+1}, \ldots, A_j)$. For any set (or multiset) of pages S, we let $w(S)$ denote the total cost of pages in S. The algorithm is the following:

1. Let $s = 0$; the variable s always denotes that we have imitated the Idle algorithm through the first s requests of the prediction.
2. Let $S = \emptyset$ be an empty queue.
3. On the arrival of request p, add p to S.
   a. If there is a t (in $[s+1, L]$ where L is the end of the current prediction) such that
   $$\ell'_{ed}(A(s+1,t), S) < \frac{1}{3}(w(A(s+1,t)) + w(S)), \tag{1}$$
   then imitate Idle through position t, empty S and let $s = t$. (If more than one t satisfies the above, select the minimum.)
   b. Otherwise, evict the page in the final slot.

We first observe that the algorithm is indeed feasible (proof in full paper).

▶ **Lemma 33.** *In the* Learn *algorithm, Step 3a is feasible, i.e., if t satisfies* (1)*, then $A_t = p$.*

Now we arrive at the heart of the analysis: we upper bound the cost of Learn against the cost of Idle (i.e., a surrogate for $\mathsf{OPT}(B)$) and the constrained edit distance ℓ'_{ed}. In particular, we sketch a proof of the following lemma and defer the full proof to the full paper.

▶ **Lemma 34.** *The algorithms* Learn *and* Idle *satisfy* $\mathsf{cost}(\text{Learn}) \leq \mathsf{cost}(\text{Idle}) + 12\ell'_{ed}$.

Proof (Sketch). Let cost_1 denote the total cost of Step 3a, and cost_2 denote the total cost of Step 3b so that $\mathsf{cost}(\text{Learn}) = \mathsf{cost}_1 + \mathsf{cost}_2$. From the algorithm, we see that $\mathsf{cost}_1 \leq \mathsf{cost}(\text{Idle})$, so now we must prove $\mathsf{cost}_2 \leq 12\ell'_{ed}$.

Now we establish some notation. Let $\ell'_{ed}((a,b)(c,d)) = \ell'_{ed}(A(a,b), B(c,d))$, and let $w_A(a,b) = w(A(a,b))$ and $w_B(a,b) = w(B(a,b))$.

We proceed by induction on the number of times we went Step 3a. Consider the first time we enter Step 3a; suppose we have read the input $B(1,b)$ and we now imitated Idle through $A(1,a)$ for some values a, b. Since the matched edges for ℓ'_{ed} do not cross, there exists some c such that $\ell'_{ed} = \ell'_{ed}(A,B)$ satisfies

$$\ell'_{ed} = \ell'_{ed}((1,a),(1,c)) + \ell'_{ed}((a+1,m),(c+1,n)).$$

We consider the case where $c < b$; the other cases follow similarly. Let $\mathsf{cost}(x,y)$ denote the cost incurred by the algorithm when serving $B(x,y)$ and notice that

$$\mathsf{cost}_2 \leq \mathsf{cost}(1,c) + \mathsf{cost}(c+1,b) + \mathsf{cost}(b+1,n).$$

The cost of serving $B(1, c)$ is at most the weight of the requested pages, so $\mathsf{cost}(1, c) \leq w_B(1, c)$. Furthermore, we can upper bound $\mathsf{cost}(c + 1, b)$ by a constant times $w_A(1, a)$ by analyzing a particular matching for $\ell'_{ed}((1, a)(1, c))$. Combining this together, we have

$$\mathsf{cost}(1, c) + \mathsf{cost}(c + 1, b) \leq 4(w_B(1, c) + w_A(1, a)) \leq 12 \cdot \ell'_{ed}((1, a), (1, c)),$$

where the second inequality follows from that we did not enter Step 3a when c arrived. Finally, applying the inductive hypothesis to $B(b + 1, n)$ and substituting the definition of c yields the lemma. ◀

The proof of Theorem 6 follows from Lemmas 28, 30, and 34.

The Follow algorithm

Now we show that the $\Omega(\ell_1)$ lower bound in Theorem 5 is tight, that is, we will give an SPRP algorithm Follow that has cost $O(1) \cdot (\mathsf{OPT} + \ell_1)$. Recall the Static algorithm from Theorem 21. The algorithm Follow ignores its input: it simply runs Static on the prediction sequence A and imitates its fetches/evictions on the input sequence B.

▶ **Theorem 35.** *The* Follow *algorithm has cost* $O(1) \cdot (\mathsf{OPT} + \ell_1)$.

Proof. Recall from Theorem 21 that $\mathsf{cost}(\text{Static}) \leq O(1) \cdot \mathsf{OPT}(A)$. Furthermore, we claim $\mathsf{OPT}(A) \leq \mathsf{OPT}(B) + 2\ell_1$. This is because on A, there exists an algorithm that imitates the movements of B: say at time t, $\mathsf{OPT}(B)$ evicts some element b that had appeared in B at time $v(t)$. Then $\mathsf{OPT}(A)$ can also evict whatever element appeared at time $v(t)$ in A, and if this is not b, then this cost can be charged to the $v(t)$ term of ℓ_1. Each term of ℓ_1 is charged at most twice because a specific request can be evicted and fetched at most once respectively.

By the same argument, we have $\mathsf{cost}(\text{Follow}) \leq \mathsf{cost}(\text{Static}) + 2\ell_1$. Combining these inequalities proves the theorem. ◀

6 Conclusion

In this paper, we initiated the study of weighted paging with predictions. This continues the recent line of work in online algorithms with predictions, particularly that of Lykouris and Vassilvitski [9] on unweighted paging with predictions. We showed that unlike in unweighted paging, neither a fixed lookahead not knowledge of the next request for every page is sufficient information for an algorithm to overcome existing lower bounds in weighted paging. However, a combination of the two, which we called the strong per request prediction (SPRP) model, suffices to give a constant approximation. We also explored the question of gracefully degrading algorithms with increasing prediction error, and gave both upper and lower bounds for a set of natural measures of prediction error. The reader may note that the SPRP model is rather optimistic and requires substantial information about the future. A natural question arises: can we obtain constant competitive algorithms for weighted paging with fewer predictions? While we refuted this for the PRP and fixed lookahead models, being natural choices because they suffice for unweighted paging, it is possible that an entirely different parameterization of predictions can also yield positive results for weighted paging. We leave this as an intriguing direction for future work.

References

1. Susanne Albers. The influence of lookahead in competitive paging algorithms. In *European Symposium on Algorithms*, pages 1–12. Springer, 1993.
2. Nikhil Bansal, Niv Buchbinder, and Joseph Seffi Naor. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM (JACM)*, 59(4):19, 2012.
3. Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.
4. Marek Chrobak, H Karloof, Tom Payne, and S Vishwnathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.
5. Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.
6. Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *International Conference on Machine Learning*, pages 2319–2327, 2019.
7. Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.
8. Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877, 2020.
9. Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3302–3311, 2018.
10. Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pages 464–473, 2018.
11. Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.
12. Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.
13. Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834–1845. SIAM, 2020.
14. Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
15. Neal Young. On-line caching as cache size varies. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, pages 241–250, 1991.
16. Neal E Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.