Online Algorithms for Weighted Paging with Predictions

ZHIHAO JIANG, Stanford University
DEBMALYA PANIGRAHI and KEVIN SUN, Duke University

39

In this article, 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 a knowledge of the next request for every page is sufficient information for an algorithm to overcome the existing lower bounds in weighted paging. However, a combination of the two, which we call strong per request prediction (SPRP), 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.

CCS Concepts: • **Theory of computation → Caching and paging algorithms;**

Additional Key Words and Phrases: Weighted paging, algorithms with predictions

ACM Reference format:
Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. 2022. Online Algorithms for Weighted Paging with Predictions. *ACM Trans. Algorithms* 18, 4, Article 39 (October 2022), 27 pages.
https://doi.org/10.1145/3548774

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 fetch 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. In the *unweighted paging* problem, the goal of the online algorithm is to minimize the total number of fetches. In the *weighted paging* problem, each page has an associated weight representing the cost to fetch it, and the goal is to minimize the total weight of fetched pages.

We measure the performance of any algorithm by its competitive ratio. In particular, an algorithm has competitive ratio c if, for every input sequence, the cost of the algorithm is at most c

This work was done while Zhihao Jiang was visiting Duke University, Durham, NC, USA. Debmalya Panigrahi was 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 was supported in part by NSF grants CCF-1535972, CCF-1955703, and an NSF CAREER Award CCF-1750140.
Authors. addresses: Z. Jiang, Stanford University, 475 Via Ortega Stanford, CA 94305 USA; D. Panigrahi and K. Sun, Duke University, 308 Research Drive Durham, NC 27708 USA.

1549-6325/2022/10-ART39 $15.00
https://doi.org/10.1145/3548774

times the cost of an optimal offline algorithm for that input, plus a fixed additive term. For randomized algorithms, we consider the usual model of an oblivious adversary, and we replace the cost of the algorithm by its expected cost. It is well known that for both the unweighted and weighted versions of the paging problem, the best deterministic algorithms have a competitive ratio of $O(k)$ and the best randomized algorithms have a competitive ratio of $O(\log k)$ against an oblivious adversary (see, e.g., [3, 5]).

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 article, 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* for short) in order to match offline performance. In fact, Lykouris and Vassilvitskii [10] (see also Rohatgi [14]) 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 currently requested page). 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., [12]), 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 [10], this model has gained significant traction

in the last couple of years and has been applied to problems in data structures [11], online decision making [7, 13], scheduling theory [9, 13], frequency estimation [8], and the like. 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 one extra slot than that of the optimal solution, then we can achieve constant competitive ratio when the prediction error is small.

1.1 Problems and Results

We now formally define the weighted paging problem. There is a cache of size k (initially empty), and a set of n pages. Each page p has some weight $w(p) \geq 0$, which denotes the cost of fetching the page. The input is a sequence $(p_1, p_2, \ldots, p_m)$, where p_t denotes the page requested at time t. When p_t arrives, if p_t is not in the cache, then the algorithm suffers a "cache miss" and must evict a page from its cache, and fetch p_t at a cost of $w(p_t)$. If p_t is already in the cache, then the algorithm incurs a "cache hit." The goal is to minimize the total cost of fetching pages into the cache. In the unweighted version of the problem, every page p has weight $w(p) = 1$, so the goal is to minimize the total number of cache misses.

For any input sequence σ, let $\mathsf{OPT}(\sigma)$ denote the minimum cost incurred by an optimal "offline" algorithm, that is, an algorithm that is given σ before it needs to make any decisions. We say an algorithm is α-competitive if, for every input σ, the cost incurred by the algorithm on σ satisfies

$$\mathsf{ALG}(\sigma) \leq \alpha \cdot \mathsf{OPT}(\sigma) + c,$$

where c is a constant not depending on σ. The competitive ratio of an algorithm is the infimum over all α such that the algorithm is α-competitive.

In the case of randomized algorithms, $\mathsf{ALG}(\sigma)$ is a random variable, so in the above inequality, we replace it with its expectation. Furthermore, we follow the standard notion that the input is generated by an oblivious adversary, that is, the adversary must specify σ before seeing the random choices made by the algorithm.

Our first result is a lower bound for weighted paging in the PRP model. In this 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 1.1. *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. [5] and Bansal et al. [3]).

Next, for the ℓ-strong lookahead model, we show lower bounds for weighted paging. In this model, the algorithm is provided a lookahead into future requests that includes (at most) ℓ distinct pages. For instance, suppose there are four pages a, b, c, d and $\ell = 2$. Let the request sequence be $(a, b, c, d, a, b, a, c, \ldots,)$. When page d is requested in time-step 4, the ℓ-strong lookahead model reveals the sequence (a, b, a), but it does not know the page requested at time-step 8 (whether it's c or d). In contrast, at time-step 4, the PRP model knows that the request in time-step 8 is for page c, but does not know the request at time-step 7 (whether it's a or b).

Theorem 1.2. *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. For instance, when the first request arrives, $(n - 1)$-strong lookahead has access to the next request time of every page,[1] while SPRP only reveals future requests till the next request for the first page. (If, say, the second request is for the first page again, then SPRP only reveals that second request and nothing else.) After each of the n pages has been requested at least once, SPRP and $(n - 1)$-strong lookahead are equivalent.

Recall that the algorithm incurs a cost whenever it fetches a page. Since the number of times any algorithm fetches and evicts a page differs by at most 1, asymptotically, it is equivalent to consider the problem assuming that costs are incurred upon evictions. This simplifies our first result, which assumes that the SPRP model produces predictions that are entirely correct.

Theorem 1.3. *There is a deterministic 2-competitive algorithm for weighted paging with SPRP.*

Note that all results so far 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 [10] 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 [10] 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 1.4. *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 1.5. *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 [10] used a variant of the classic edit distance measure, and showed a constant competitive ratio for this error measure.

[1]Note that this also includes the next request for the first page because it either gets requested again in the lookahead region containing the first request of every other page, or its next request is precisely the one after the lookahead region.

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

In the same setting, there exists a randomized algorithm that uses a cache of size $k+1$ *whose cost is* $O(\mathrm{OPT} + \ell_{ed})$*, where* OPT *uses a cache of size* k*.*

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 [4]. For online paging, Sleator and Tarjan [15] gave a deterministic k-competitive algorithm, and Fiat et al. [6] gave a randomized $O(\log k)$-competitive algorithm; both results were also shown to be optimal. For weighted online paging, Chrobak et al. [5] gave a deterministic k-competitive algorithm, and Bansal et al. [3] gave an $O(\log k)$-competitive randomized algorithm, which are also optimal by extension.

Recently, Lykouris and Vassilvitskii [10] introduced a prediction model that we call *PRP* in this article: 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. [6], with competitive ratio $O(\min(\sqrt{\ell_{pd}/\mathrm{OPT}}, \log k))$. 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 $O(\min(\eta_{ed}/\mathrm{OPT}, \log k))$, where η_{ed} is the edit distance between the predicted sequence and the actual input. Subsequently, Rohatgi [14] improved the former bound to $O(1 + \min((\ell_{pd}/\mathrm{OPT})/k, 1)\log k)$ and also proved a lower bound of $\Omega(\log \min((\ell_{pd}/\mathrm{OPT})/(k \log k), k))$. Most recently, Wei [16] gave an algorithm with competitive ratio $O(1 + \min((\ell_{pd}/\mathrm{OPT})/k, \log k))$.

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 $l \leq k-2$, 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 [18] gave a deterministic algorithm with competitive ratio $k/(k-h+1)$ and showed that this is optimal. In another paper, Young [17] showed that the randomized "marker" algorithm is $O(\log(k/k-h))$-competitive and this bound is optimal up to constants.

Remark. The independent, concurrent work of Antoniadas et al. [2] has slight overlap with ours. In particular, they also showed that the PRP prediction model does not provide asymptotic

[2]For technical reasons, neither ℓ_{ed} in this article nor the edit distance variant in [10] exactly match the classical definition of edit distance.

benefits for randomized algorithms. They also gave a prediction-based randomized algorithm for *unweighted* caching, and they note that their prediction error is not directly comparable to the error used by Lykouris and Vassilvitskii [10] and Rohatgi [14].

Roadmap. In Section 2, we show the lower bounds stated in Theorem 1.1 for the PRP model. The lower bounds for the ℓ-strong lookahead model stated in Theorem 1.2 are proven in Section 3. In Section 4, we state and analyze the algorithm for the SPRP model with no error, thereby proving Theorem 1.3. Finally, in Section 5, we consider the SPRP model with errors, and focus on the upper and lower bounds in Theorems 1.4, 1.5, and 1.6.

2 THE PER-REQUEST PREDICTION MODEL (PRP)

In this section, we give the lower bounds stated in Theorem 1.1 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 deterministic construction.

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. The randomized lower bound follows the same intuition.

2.1 Deterministic Lower Bound

Now we give a formal proof of the following theorem.

Theorem 2.1. *For weighted paging with PRP, any deterministic algorithm is $\Omega(k)$-competitive.*

The set of pages is $\{a_0, a_1, \ldots, a_k\}$, and the weight of a_i is $w(a_i) = c^i$. While generating the input sequence, we will maintain variables u_i and t that satisfy the following invariants:

- The value of u_i denotes the next time at which page a_i will arrive.
- The value of t is the number of requests that have been made, initialized to $t = 0$.

Fix some constant $c \geq 2$. The input is defined as follows:

(1) For $0 \leq i \leq k$, let $u_i = (2c+2)^i$, and for $0 \leq i < k$, let $y_i = 0$.
(2) Repeat the following:
  (a) Let ℓ denote the largest index such that a_ℓ is not in the cache.
  (b) Increase ℓ until $\ell = k$ or $y_\ell < 2c$.
  (c) For j from 0 to ℓ,
    (i) Set all the 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 to be a_j.
    (iii) Let $t = u_j$.
  (d) For $0 \leq j \leq \ell$, let $u_j = t + (2c+2)^j$.
  (e) For $0 \leq j < \ell$, let $y_j = 0$. If $\ell < k$, increase y_ℓ by one.

We call the requests generated each time we enter Step (2) a *block*; if the final value of ℓ is i then this is an *i-block*. If ℓ is not increased in Step (2b), then this block is *regular*; otherwise, it is *irregular*. Any j-block is an *i-plus* block if and only if $j \geq i$.

Let us give an overview of the lower bound argument. First, we show that every i-block is a contiguous sequence of a_0's, then a_1's, and so on, ending with a single a_i (Lemma 2.2). Thus, for each such block, ALG incurs a cost of at least c^i, because at the beginning of this block, the cache of ALG does not contain the page a_i.

On the other hand, for each $i \in \{1, 2, \ldots, k\}$, consider the algorithm ALG_i defined as follows: upon a cache miss, evict a_i if it is in the cache, and a_0 otherwise. Notice that ALG_i incurs a cost of roughly c^i in every j-block for any $j \geq i$. Thus, after we bound the total number of i-blocks (Lemma 2.3), we can conclude that cost(ALG) is $\Omega(k)$ times the average cost of the ALG_i (Lemma 2.4). Since the optimal algorithm is no worse than the average of these k algorithms, the theorem follows. We now begin with the formal analysis.

Lemma 2.2. *For every ℓ, an ℓ-block is a contiguous sequence of a_0's, then a_1's, and so on, ending with a single a_ℓ.*

Proof. It suffices to show $u_0 < u_1 < \ldots < u_k$ at Step (2); this clearly holds for the initial values of the u_i. Thus, it suffices to prove $u_\ell < u_{\ell+1}$ because $u_0 < u_1 < \ldots < u_\ell$ from Step (2d) and the value of u_j for $j \geq \ell + 1$ remains unchanged within each step.

Suppose $u_{\ell+1} = t_0 + (2c+2)^{\ell+1}$ for some t_0, and for contradiction, suppose the value of u_ℓ exceeds $u_{\ell+1}$ at some point $t > t_0$. Since the value of $u_{\ell+1}$ has not changed, the blocks between t and t_0 must

all be j-blocks for $j \le \ell$. Furthermore, the value of u_ℓ only changes after we create an ℓ-block, and each time, it increases by $(2c+2)^\ell$. However, the number of ℓ-blocks that have appeared is at most $2c$ because of the condition in Step (2b): if $y_\ell \ge 2c$, then we would have created an $(\ell+1)$-plus block. Thus, the value of u_ℓ is at most $t_0 + 2c \cdot (2c+2)^\ell < t_0 + (2c+2)^{\ell+1} = u_{\ell+1}$. □

Let v_i denote the number of regular i-blocks, and let v_i' denote the number of irregular i-blocks. For each $i \in \{0, 1, \ldots, k\}$, ALG will miss on page a_i in each of the v_i regular blocks. This implies

$$\text{cost}(\text{ALG}) \ge v_0 + v_1 c + v_2 c^2 + \cdots + v_k c^k.$$

Lemma 2.3. *For any $i \ge 1$, the total number of i-blocks is at most $\sum_{j=0}^{i} (\frac{v_j}{(2c)^{i-j}})$.*

Proof. An irregular i-block is created only after $2c$ $(i-1)$-blocks have been created since the last time an i-plus block was created. Since every i-block is also an i-plus block, the number of $(i-1)$ blocks since the last time an i-block was created must also be at least $2c$. So we have $v_i' \le \frac{1}{2c}(v_{i-1}' + v_{i-1})$. (Since every 0-block is regular, we have $v_1' \le \frac{1}{2c} v_0$.) Adding v_i to both sides and repeatedly applying this inequality proves the lemma. □

Now we analyze the cost of any algorithm ALG by bounding it against the performance of k algorithms, defined as follows: For any $i \in \{1, 2, \ldots, k\}$, the algorithm ALG_i evicts a_i if it is in the cache on a cache miss, and a_0 otherwise.

Lemma 2.4. *On the adversarial input generated by the procedure above, the total cost of the algorithms $\text{ALG}_1, \ldots, \text{ALG}_k$ is at most $32 \cdot \text{cost}(\text{ALG})$. That is,*

$$\sum_{i=1}^{k} \text{cost}(\text{ALG}_i) \le 32 \cdot \text{cost}(\text{ALG}).$$

Proof. Notice that ALG_i misses on request a_i at most once in any i-plus block, so ALG_i misses on page a_i at most $\sum_{j=i}^{k}(v_j + v_j')$ times. Furthermore, ALG_i alternates between evicting a_0 and a_i, so ALG_i misses on page a_0 at most $1 + \sum_{j=i}^{k}(v_j + v_j')$ times in total. We bound the cost of every ALG_i miss (either on a_0 or a_i) by $(c^i + 1)$.

Thus, by Lemma 2.3, we have the following:

$$\begin{aligned}
\text{cost}(\text{ALG}_i) &\le \left(1 + 2\sum_{j=i}^{k}(v_j + v_j')\right) \cdot (c^i + 1) \\
&\le 8c^i \sum_{j=i}^{k}(v_j + v_j') \\
&\le 8 \sum_{j=i}^{k} \sum_{j'=0}^{j} \left(\frac{v_{j'} \cdot c^i}{(2c)^{j-j'}}\right).
\end{aligned}$$

Summing across all values of $i \in \{1, 2, \ldots, k\}$, we have

$$\begin{aligned}
\sum_{i=1}^{k} \text{cost}(\text{ALG}_i) &\le 8 \sum_{i=1}^{k} \sum_{j=i}^{k} \sum_{j'=0}^{j} \left(\frac{v_{j'} \cdot c^i}{(2c)^{j-j'}}\right) \\
&\le 8 \sum_{j'=0}^{k} \sum_{j=j'}^{k} \sum_{i=1}^{j} \left(\frac{v_{j'} \cdot c^i}{(2c)^{j-j'}}\right)
\end{aligned}$$

$$\leq 8 \sum_{j'=0}^{k} v_{j'} \cdot \sum_{j=j'}^{k} \frac{1}{(2c)^{j-j'}} \cdot \sum_{i=1}^{j} c^i$$
$$\leq 16 \sum_{j'=0}^{k} v_{j'} \cdot \sum_{j=j'}^{k} \frac{1}{(2c)^{j-j'}} c^j$$
$$\leq 32 \sum_{j'=0}^{k} v_{j'} \cdot c^{j'} \leq 32 \cdot \text{cost}(\text{ALG}). \qquad \square$$

We now conclude the proof of Theorem 2.1. From Lemma 2.4, we have

$$\text{OPT} \leq \min\{\text{cost}(\text{ALG}_1), \text{cost}(\text{ALG}_2), \ldots, \text{cost}(\text{ALG}_k)\} \leq \frac{32}{k}\text{cost}(\text{ALG}),$$

so ALG is $\Omega(k)$-competitive, as desired.

2.2 Randomized Lower Bound

This subsection is devoted to proving the following theorem:

Theorem 2.5. *For weighted paging with PRP, any randomized algorithm is $\Omega(\log k)$-competitive.*

Again, the set of pages is $\{a_0, a_1, \ldots, a_k\}$, and the weight of a_i is $w(a_i) = c^i$. 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. Again, fix some constant $c \geq 2$. The input is defined as follows:

(1) Set $t = 0$. For $0 \leq i \leq k$, 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 $0 \leq j \leq \ell$, let $u_j = t + (2ckH_k + 2)^j$.
  (e) For $0 \leq j < \ell$, let $y_j = 0$. If $\ell < k$, increase y_ℓ by one.

Again, 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 $j \geq i$. Note that Lemma 2.2, which describes the structure of an ℓ-block, still applies in the randomized setting. We first lower bound the cost of ALG by the number of blocks.

Lemma 2.6. *Every requested block increases $\mathbb{E}[\text{cost}(\text{ALG})]$ 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 $\Pr[\ell = j] \geq \frac{1}{2c^j}$, so the expected cost of serving this block is at least $c^j \cdot \Pr[\ell = j] = \Omega(1)$. $\square$

Now 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. 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. As before, let v_i denote the number of regular i-blocks, and v_i' denote the number of irregular i-blocks.

Lemma 2.7. *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 $2ckH_k$. Since $\phi(y)$ is always non-negative, 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 because the probability that a block is a regular i-block is at most c^{-i}. □

Lemma 2.8. *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

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

where the second inequality holds due to Lemma 2.7. □

We can bound $\mathrm{OPT} = \mathrm{OPT}(A)$ in terms of the optimal cost on B and the number of irregular blocks.

Lemma 2.9. *Let $\mathrm{OPT}(A)$ and $\mathrm{OPT}(B)$ denote the optimal offline algorithm on request sequences A and B, respectively. Then, $\mathrm{cost}(\mathrm{OPT}(A)) \le \mathrm{cost}(\mathrm{OPT}(B)) + 4c\sum_{i=0}^{k} v_i' c^i$.*

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

(1) For requests in regular blocks, imitate $\mathrm{OPT}(B)$. That is, copy the cache contents when $\mathrm{OPT}(B)$ 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 $1 \le \ell \le i$, 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 $\mathrm{OPT}(B)$, we divide the sequence B into phases. Phases are defined recursively, starting with 0-phases all the way through to k-phases. A 0-phase is defined as a single block. For $i \ge 1$, 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 the events corresponding to 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 $\mathrm{OPT}(B)$ 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 2.10. *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 block. To serve one block, 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, which 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 2.11. *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 2.11 to the end of this section; first, we use it to prove Theorem 2.5.

Proof of Theorem 2.5. Let $\mathrm{OPT}(A)$ denote the cost of an optimal algorithm on the request sequence A, and let $\mathrm{OPT}(B)$ denote the cost of an optimal algorithm on the regular blocks B. Then we have the following:

$$\begin{aligned}
\mathbb{E}\left[\mathrm{cost}(\mathrm{OPT}(A))\right] &\leq \mathbb{E}\left[\mathrm{cost}(\mathrm{OPT}(B))\right] + 4c \sum_{i=0}^{k} c^i \cdot \mathbb{E}\left[v_i'\right] && \text{(Lemma 2.9)}\\
&\leq \mathbb{E}\left[\mathrm{cost}(\mathrm{ALG}_B^k)\right] + 4c \sum_{i=0}^{k} c^i \cdot \frac{2m}{c^i k H_k} && \text{(Lemma 2.8)}\\
&\leq 4c^{k+1} \cdot \mathbb{E}\left[N_k(B)\right] + \frac{16cm}{H_k}, && \text{(Lemma 2.10)}
\end{aligned}$$

where $N_k(B)$ denotes the number of k-phases in B. According to Proposition 2.11, 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

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

Since any algorithm incurs at least some constant cost in every block by Lemma 2.6, its cost is $\Omega(m)$, which concludes the proof. □

Proof of Proposition 2.11. 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 2.12. *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 $\mathbb{E}[z_{i-1}]$, so the total expected number of $(i-1)$-plus blocks in the first c $(i-1)$-phases of an i-phase is $c \cdot \mathbb{E}[z_{i-1}]$.

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 $\mathbb{E}[z_{i-1}]$ i-plus blocks.

With probability $\Pr\{M_i > Q_i\}$, there are no i-plus blocks in the first c $(i-1)$-phases. In this case, the i-phase ends as soon as an i-plus block appears, so $z_i = 1$ and $z_{i-1} = 0$. Otherwise, the i-phase ends immediately after the c $(i-1)$-phases, in which case $z_i = z_{i-1}$. Combining these cases proves the lemma. □

Lemma 2.13. *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 $(i-1)$-plus blocks in the first c $(i-1)$-phases and let $V = \sum_{i=1}^{c} v_i$. 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

$$\begin{aligned}
\Pr(M_i > Q_i) &= \sum_{n \geq 1} \Pr(M_i > Q_i \mid V = n) \cdot \Pr(V = n) \\
&= \sum_{n \geq 1} \left(1 - \frac{1}{c}\right)^n \cdot \Pr(V = n) \\
&\geq \left(1 - \frac{1}{c}\right)^{\mathbb{E}[V]} \\
&= \left(1 - \frac{1}{c}\right)^{c \cdot \mathbb{E}[z_{i-1}]}
\end{aligned}$$

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

Lemma 2.14. *For any $i \geq 1$, 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 2.12 implies $\mathbb{E}[z_i] \geq \mathbb{E}[z_{i-1}] \geq \frac{1}{4} H_i$.
(2) If $\mathbb{E}[z_{i-1}] < \frac{1}{2} H_{i-1} < \frac{1}{2}(1 + \ln(i))$, 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 2.12 and the inequality holds by the induction hypothesis and Lemma 2.13. Thus, $\mathbb{E}[z_i] \geq \frac{1}{4} H_{i-1} + \frac{1}{e} \cdot \frac{1}{i} \geq \frac{1}{4} H_i$. □

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

Proof. When $i = 0$, we have $L_0 = z_0 = 1$ because a 0-phase is defined as a single block, and every block is a 0-plus block. 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,

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

Furthermore, from Lemma 2.12, we have

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

Combining the two equalities yields the lemma. □

We conclude by proving Proposition 2.11. Fix some $i \geq 1$. Using Lemmas 2.14 and 2.15, 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 3.1 ([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 3.1 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 inserting pages with low weight between every two consecutive requests in the the sequence, we can similarly prove the following results for the weighted paging problem.

Theorem 3.2. *For weighted paging with ℓ-strong lookahead, any deterministic algorithm is $\Omega(k)$-competitive if $\ell \leq n-k$ and $\Omega(n-\ell)$-competitive if $n-k+1 \leq \ell \leq n-1$.*

Proof. Suppose the pages are labeled $\{x_1, \ldots, x_n\}$. Pages $x_1, \ldots, x_{k+1}$ are called "heavy" and have weight 1, and the remaining $n-k-1$ pages are "light" and have weight $\epsilon < 1$ (where ϵ will be specified later). Also, let $\ell' = \ell - (n-k-1)$, and assume $n-k+1 \leq \ell \leq n-4$ for now. We assume that the algorithm's cache and the optimal cache initially contain the same set of pages.

Our input sequence consists of a series of phases. A phase is constructed as follows: start with the request sequence $(x_1, \ldots, x_{\ell'}, y)$, where y is a heavy page not in the algorithm's cache. In addition, between every two consecutive requests in this phase so far, insert the $n-k-1$ light pages as additional requests. This completes the construction of a phase. Notice that to serve any phase, the algorithm incurs a cost of at least 1 due to page y.

Now consider a sequence of $n-\ell \leq k-1$ consecutive phases, which contain at most $k-1$ distinct heavy pages. The optimal algorithm can reserve $k-1$ of its cache slots to serve heavy pages, and the remaining cache slot serves requests to light pages. Whenever it incurs a cache miss on a heavy page, it can evict a heavy page that will not be requested in this sequence of $n-\ell$ consecutive phases. This prevents the optimal algorithm from missing on the subsequent requests for heavy pages. Furthermore, since the total number of requests is at most n^3, if $\epsilon < 1/n^3$, then the optimal algorithm pays at most 1 to serve all of the requests for light pages. On the other hand, the algorithm's cost increases by at least 1 per phase, so its total cost to serve the sequence is at least $n-\ell$. Thus, the ratio between the algorithm's cost and the optimal cost is $\Omega(n-\ell)$. If $\ell = n-4$,

then the algorithm is $\Omega(1)$-competitive, and this bound holds for any value of $\ell \geq n-4$. If $\ell \leq n-k$, then we consider an input sequence consisting of $k-1$ consecutive phases (instead of $n-\ell$); the same argument implies that the competitive ratio is $\Omega(k)$. □

Theorem 3.3. *For weighted paging with ℓ-strong lookahead, any randomized algorithm is $\Omega(\log k)$-competitive if $\ell \leq n-k$ and $\Omega(\log(n-\ell))$-competitive if $n-k+1 \leq \ell \leq n-1$.*

Proof. This proof is similar to that of Theorem 3.2. Suppose the pages are labeled $\{x_1, \ldots, x_n\}$. Pages $x_1, \ldots, x_{k+1}$ are called "heavy" and have weight 1, and the remaining $n-k-1$ pages are "light" and have weight $\epsilon < 1$ (where ϵ will be specified later). Also, let $\ell' = \ell - (n-k-1)$ and assume $n-k+1 \leq \ell \leq n-4$ for now, so $2 \leq \ell' \leq k-3$. We assume that the algorithm's cache and the optimal cache initially contain the same set of pages.

Our input sequence consists of a series of phases. The first phase is constructed as follows: start with the request sequence $(x_1, \ldots, x_{\ell'}, y)$, where y is a heavy page not in the algorithm's cache. In subsequent phases, y is chosen uniformly at random from the subset of heavy pages $S = \{x_{\ell'+1}, \ldots, x_{k+1}\}$. In addition, between every two consecutive requests in this phase so far, insert the $n-k-1$ light pages as additional requests. This completes the construction of a phase.

We assume the algorithm is deterministic, analyze its expected cost, and apply Yao's minimax principle to prove the theorem. Notice that to serve any phase, a deterministic algorithm incurs an expected cost of at least $1/(k-\ell'+1)$, because it either misses on a request for some heavy page x_i where $i \leq \ell'$, or it must be missing a page in S, and that page is requested with probability $1/(k-\ell'+1) = \Omega(1/(k-\ell'))$.

Now let m denote the number of consecutive phases constructed until $k-1$ distinct heavy pages have been requested. It is well known (by the coupon collector problem) that the expected value of m is $\Theta((k-\ell')\log(k-\ell'))$. To serve the heavy pages in these phases, the optimal algorithm can reserve $k-1$ of its cache slots, and it will use the remaining cache slot to serve all requests to light pages. Upon a cache miss for a heavy page, the optimal algorithm can evict the heavy page that appears farthest in the future, and it will not miss on subsequent requests for heavy pages in these m phases. Since the expected number of total requests is at most $n^2 k \log k$, if $\epsilon < 1/n^2 k \log k$, then the expected cost of the optimal algorithm to serve all m phases is at most 2.

Putting this together, the ratio between the expected cost of the algorithm and the expected cost of the optimal algorithm across these m phases is $\Omega(\log(k-\ell')) = \Omega(\log(n-\ell))$. If $\ell = n-4$, then this algorithm is $\Omega(1)$-competitive, and this bound holds for any value of ℓ. If $\ell \leq n-k$, then we set S as the entire set of heavy pages (i.e., $S = \{x_1, \ldots, x_{k+1}\}$). Now to serve any phase, a deterministic algorithm incurs an expected cost of at least $1/(k+1) = \Omega(1/k)$, while the expected number of phases until $k-1$ distinct heavy pages have been requested is $\Theta(k \log k)$. An optimal algorithm can serve these phases by paying a constant cost, so the expected competitive ratio is $\Omega(\log k)$. □

4 THE STRONG PER-REQUEST PREDICTION MODEL (SPRP)

In this section, we assume that we have SPRP predictions that are always correct, and we use them to design a simple 2-competitive algorithm called define a simple algorithm called Static. At any time step t, let $L(t)$ denote the sequence of requests in the current prediction. At a high level, the Static algorithm runs on "batches" of requests, where each batch is a contiguous subsequence of requests. For each batch, it processes the requests using an optimal offline algorithm.

More specifically, the first batch is $L(1)$ and contains all requests that are initially given by the SPRP predictions. The next batch starts once the first batch ends (i.e., at time $|L(1)|+1$), and comprises all future predictions at that time. 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, as described in Algorithm 1.

ALGORITHM 1: STATIC

while there exists a request q **do**
  Set t as the smallest time step such that p_t has not been served (initially $t = 1$).
  Let $L(t)$ denote the sequence of SPRP predictions given at time step t (starting with q).
  Serve the requests in $L(t)$ by executing an optimal offline strategy.

To illustrate the STATIC algorithm, we give a small example. Suppose the input begins with the following sequence: $(a, b, a, c, a, b, c, d, \ldots)$. Upon seeing the first request (i.e., for page a), the SPRP predictions contain (a, b, a). This is the first batch, and the STATIC algorithm runs an optimal offline algorithm to serve it. Next, after the algorithm serves the request at time step 3, the SPRP predictions contain (c, a, b, c). This is the second batch, and the STATIC algorithm runs an optimal offline algorithm to serve it. The third batch begins with d and ends with the farthest page seen by the SPRP predictions, and STATIC will serve it using an optimal offline algorithm. The algorithm continues to process the input in this manner until every request has been served.

THEOREM 4.1. *The STATIC algorithm is 2-competitive when the predictions from SPRP are entirely correct.*

PROOF. For this problem, we make a standard assumption that all algorithms start and end with an empty cache. This way, measuring the cost of algorithm by the total weight of fetches is equivalent to measuring its cost by the total weight of evictions. This is because every page that gets fetched also gets evicted (since the final cache is empty), and every page that gets evicted must have gotten fetched (since the initial cache is empty). So throughout this proof, we charge the cost of the algorithm to the cost of evictions, rather than fetches.

Suppose the algorithm runs a total of m batches $B_1, \ldots, B_m$. Consider a page p in some batch B_i where $i < m$. If p appears in some batch after B_i, then upon seeing the last request for p in B_i, SPRP will include p in the next batch B_{i+1}. (If p does not appear again, then the next batch must be the last batch, because in this case, SPRP can see the rest of the input.) Therefore, for any $i < m$, batch B_i contains a request for every page that was requested in batches $B_1, \ldots, B_{i-1}$.

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

Fix a batch index $j \in \{2, 3, \ldots, m\}$ and let $C(\mathsf{OPT}_{j-1})$ and $C(S_{j-1})$ denote the cache states of OPT and STATIC immediately before the arrival of the first request in batch B_j. We know that STATIC runs an optimal offline algorithm on B_j. One feasible solution is to immediately change the cache state to $C(\mathsf{OPT}_{j-1})$, and then imitate the choices that OPT makes to serve B_j. Since costs are charged to evictions, we have

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

Consider some page $p \in C(S_{j-1}) \setminus C(\mathsf{OPT}_{j-1})$. Since $p \in C(S_{j-1})$, we know p must have appeared before the start of B_j (because STATIC does not fetch pages that have never been requested). Since B_{j-1} contains all pages that were requested in batches $B_1, \ldots, B_{j-2}$, in particular, p must have been requested in B_{j-1}. Furthermore, since $p \notin C(\mathsf{OPT}_{j-1})$, OPT must have evicted p at some point while serving B_{j-1}. Thus, $S_j \le \mathsf{OPT}_j + \mathsf{OPT}_{j-1}$. Summing over all $j \ge 2$ and the inequality $S_1 \le \mathsf{OPT}_1$ 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 tth 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 $\mathsf{pnext}(\mathsf{prev}(j)) = i$. Furthermore, no edges in a matching are allowed to cross. 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.

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

Definition 5.1. 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 5.2. The *1-norm distance* ℓ_1 between A and B is defined as follows: for each input request B_i, consider the corresponding predicted request A_i. If $A_i \neq B_i$, then we add $w(A_i) + w(B_i)$ to ℓ_1 (initially 0). Equivalently,

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

Note that in order for ℓ_1 to be well defined, we can assume $|A| \geq |B|$.

Third, we define an error measure inspired by the PRP model that was introduced in [10].

Definition 5.3. The *prediction distance* ℓ_{pd} between A and B is defined as follows: for each input request B_i, consider the actual next-arrival time $\mathsf{next}(i)$ and predicted next-arrival time $\mathsf{pnext}(i)$. The contribution of B_i to ℓ_{pd} (initially 0) is $w(B_i)$ times the absolute difference between these two values. Equivalently,

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

Example. We give an example that illustrates how ℓ_1 and ℓ_{pd} are computed. Both error measurements involve taking a sum over the actual input sequence B, while the terms in the sum depend on both B and the prediction sequence A. Suppose A and B are the following:

$$A = (a, c, b, a, b), \quad B = (a, b, a, c, b).$$

To compute ℓ_1, we sum the total weight of predicted and actual page requests, over the i such that $A_i \neq B_i$. In this example, B_1 and B_5 are the only requests correctly predicted by A, so the sum is taken over $i \in (2, 3, 4)$. Thus, we have

$$\begin{aligned}\ell_1 &= (w(A_2) + w(B_2)) + (w(A_3) + w(B_3)) + (w(A_4) + w(B_4)) \\ &= (w(c) + w(b)) + (w(b) + w(a)) + (w(a) + w(c)) \\ &= 2 \cdot (w(a) + w(b) + w(c)).\end{aligned}$$

To compute ℓ_{pd}, we sum the weighted absolute differences between predicted and actual next-arrival times, over the input sequence B. We start at $B_1 = a$, requested at time 1. Its actual next-arrival time is $\text{next}(1) = 3$, and its predicted next-arrival time is $\text{pnext}(1) = 4$. Thus, $B_1 = a$ contributes $w(a) \cdot |3 - 4| = w(a)$ to ℓ_{pd}. At time 2, $B_2 = b$ is requested. Its actual next-arrival time is 5, and its predicted next-arrival time is 3. Thus, B_2 contributes $w(b) \cdot |5 - 3| = 2 \cdot w(b)$ to ℓ_{pd}. Continuing in this manner for B_3, B_4, and B_5, we get

$$\begin{aligned}\ell_{pd} &= w(a) + 2 \cdot w(b) + w(a) \cdot |6 - 4| + w(c) \cdot |6 - 6| + w(b) \cdot |6 - 6| \\ &= 3 \cdot w(a) + 2 \cdot w(b).\end{aligned}$$

Notice that at the end of both sequences, when no next-arrival times exist, they are assumed to be 1 greater than the length of the sequence. For example, B_3 is the final request for page a, so we set $\text{next}(3) = |B| + 1 = 6$.

5.1 Lower Bounds

In this section, we give an overview of the lower bounds stated in Theorems 1.4, 1.5, and 1.6. We focus on the ℓ_{ed} (i.e., Theorem 1.6) error measurement; the proofs for ℓ_1 and ℓ_{pd} follow similarly.

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 $o(k) \cdot \text{OPT} + o(\ell_{ed})$ (where OPT is the optimal cost of the SPRP instance), then ALG′ will have cost $o(k) \cdot \text{OPT}'$ (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 $w(b) = \epsilon$ for some sufficiently small value of $\epsilon > 0$. (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 $i \in \{1, \ldots, k'\}$. The final string S consists of copies of $S_{k'}$, and this is the prediction sequence for the SPRP algorithm.

Observe that S only contains k distinct pages, and the oracle ALG has cache size k. Also, the structure of S is closely related to the structure of the lower-bound inputs for PRP algorithms, as described in Section 2. After L requests for a_{i-1}, there must be a request for a_i. This is analogous to the notion of regular and irregular blocks in Section 2.

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. [3]), 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 request for page b (replacing the original request).

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 changing a request for a_0 into b, feeds this as 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 $0 \le i \le k'$, 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 the final a_0, we can deduce the value of i from the PRP predictions.
  (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 $i \ge 1$, 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 2.1) 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 5.4. *Using any SPRP algorithm* ALG *as a black box, the PRP algorithm* ALG$'$ *satisfies the following:* $\text{cost}(\text{ALG}') \le 2(c+1) \cdot \text{cost}(\text{ALG})$.

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 cost_{ALG} and $\text{cost}_{\text{ALG}'}$ 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:

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

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

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

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 that bounds the costs of OPT and OPT′ against each other.

Lemma 5.5. *The algorithms* OPT *and* OPT′ *satisfy* $\text{cost}(\text{OPT}) \le 2 \cdot \text{cost}(\text{OPT}')$.

Proof. Using OPT′ as an oracle, we can design a potential algorithm for OPT:

(1) Let S be the initial input sequence for ALG and let $t = 0$.
(2) For all $0 \le i \le k'$, let $q_i = 1$. Note that $q'_i = 1$ at the beginning.
(3) For each PRP block $s_i = (a_0, a_1, \ldots, a_i)$:
  (a) Find the first time t' after t when S requests a_i. Let $t = t' + 2$; note that $S_t = b$.
  (b) Run OPT′ to serve request $(a_0, a_1, \ldots, a_i)$ and obtain $q' = (q'_0, q'_1, \ldots, q'_{k'})$.
    (i) Let $q = (q_0, q_1, \ldots, q_{k'})$ denote the current cache state (i.e., immediately before we serve a_0).
    (ii) Set $q = (0, 0, \ldots, 0, q'_{i+1}, q'_{i+2}, \ldots, q'_{k'})$ to serve all requests until the requested b.
    (iii) Set $q = (q'_0, q'_1, \ldots, q'_{k'})$ to serve the b.

Note that we have $q = q'$ at the beginning and the end of Step (3) in ALG. For convenience, let q' denote the vector at the beginning of Step (3), and let q to denote the vector at the end of Step (3). Furthermore, let cost_{OPT} and $\text{cost}_{\text{OPT}'}$ denote the cost that OPT and OPT′ respectively incur in a fixed Step (3b).

Each time OPT enters Step (3), the incurred cost is at most:

$$\text{Step 3(b)ii: } \sum_{j=0}^{i} q'_j \cdot \left(\frac{1}{v} + c^j\right),$$

$$\text{Step 3(b)iii: } \sum_{j=0}^{i} q_j \cdot \left(\frac{1}{v} + c^j\right) + \sum_{j=i+1}^{k} \left(\frac{1}{v} + c^j\right) \cdot \left|q_j - q'_j\right|.$$

Summing the above yields the following:

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

Now we consider OPT′. At the beginning of Step (3b), there is q'_j amount of a_j is not in the cache, and at the end of Step (3b), there is q_j amount of a_j is 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 $j \le i$, OPT′ must serve a_j while it serving $(a_0, a_1, \ldots, a_i)$, so the cost due to a_j is at least $c^j \cdot (q_j + q'_j)$. Summing the above yields the following:

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

Combining the above inequalities proves the lemma. □

We are now ready to bound the cost of any algorithm with SPRP.

Theorem 5.6. *For any error measurement $\ell \in \{\ell_{ed}, \ell_{pd}, \ell_1\}$ for weighted paging with SPRP, there is no deterministic algorithm whose cost is $o(k) \cdot \mathrm{OPT} + o(\ell)$, and there is no randomized algorithm whose cost is $o(\log k) \cdot \mathrm{OPT} + o(\ell)$.*

Proof. From Theorem 2.1, we know $\mathrm{ALG}' = \Omega(k) \cdot \mathrm{OPT}'$, so we can apply Lemmas 5.4 and 5.5 to conclude $\mathrm{ALG} = \Omega(k) \cdot \mathrm{OPT}$. Furthermore (as we saw in Section 2), each PRP block increases ALG by at least a constant. At the same time, for each block, ℓ_1 increases by at most 2, because only one request is changed from a_0 to b. As a result, we can conclude $\mathrm{ALG} = \Omega(\ell_1)$. Similarly, for ℓ_{pd}, notice that the only mispredictions are due to a_0 and b. If $w(b) = \epsilon$ is sufficiently small, then $\ell_{pd} = O(\ell_1)$, so $\mathrm{ALG} = \Omega(\ell_{pd})$. Finally, we can also see that in this instance, we have $\ell_{ed} = \ell_1$, so $\mathrm{ALG} = \Omega(\ell_{ed})$. For randomized algorithms, the same line of reasoning holds 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 1.6 for the ℓ_{ed} measurement, and then analyze the Follow algorithm, which proves the upper bound in Theorem 1.5.

Now we present an algorithm that uses a cache of size $k + 1$ whose cost scales linearly with $\mathrm{OPT} + \ell_{ed}$. 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 5.7. *For any sequences A and B, $\mathrm{cost}(\mathrm{OPT}^{+1}(A)) \le \mathrm{cost}(\mathrm{OPT}(B)) + 2\ell_{ed}$, 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 $\mathrm{OPT}^{+1}(A)$ is the following: imitate what $\mathrm{OPT}(B)$ does for requests matched by M, and use the memoryless slot for unmatched requests. The cost of this algorithm is $\mathrm{OPT}(B) + 2\ell_{ed}$. □

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:

Lemma 5.8. *For our modified version of caching, there exists an optimal offline algorithm.*

Proof. We show that our problem, which we denote as P^m, can be solved by solving the following version of weighted paging, which we call P: there are $k + 1$ (normal) cache slots, and a dummy page of weight 0. The input sequence for P is the same as that of P^m, with the addition that the dummy page is requested between every two consecutive requests. Since P is an instance of weighted paging, it can be solved optimally offline; let OPT denote this solution.

Our algorithm for P^m maintains that the k pages in its cache are the k non-dummy pages in OPT's cache after is has served the most recent dummy page. To serve a page p, we first observe

what OPT does to serve p and the subsequent dummy page. If OPT already has p, then so does our cache, so we don't need to do anything. Now consider the remaining cases, when OPT must fetch p:

- If OPT evicts a non-dummy page q, then we can do the same thing. (In this case, the dummy page is in OPT's cache before and after it serves p.)
- If OPT evicts the dummy page to serve p, then it must evict a non-dummy page q to serve the dummy request after p. To serve p, we can evict q. If $p = q$, then we use our memoryless cache slot to fetch, serve, and evict p.

Notice that in every case, the cost of our algorithm is the same as that of OPT.

Now we show that OPT (for P) is at most OPT^{+1}. To do this, we show that one possibility for OPT (on P) is to essentially imitate OPT^{+1} (on P^m), as we did in the above analysis. In other words, the algorithm OPT can maintain that after serving a dummy page, the k non-dummy pages in its cache are the ones in the cache of OPT^{+1}. Again, suppose the current non-dummy request is p, and consider the following cases:

- If OPT^{+1} already has p, then so does OPT, so OPT does not need to do anything.
- If OPT^{+1} must fetch p and evict some page q (possibly equal to p), then OPT evicts the dummy page (which it must have, since it was requested right before p) and fetches p. Then, to serve the dummy request after p, it evicts q.

Again, in every case, OPT pays the same as OPT^{+1}. In summary, our algorithm for P^m has the same cost as OPT on P, whose cost is at most OPT^{+1} on P^m, so this proves the lemma. □

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 5.8 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; the proof is nearly identical to the proof of Theorem 4.1, so we omit it.

Lemma 5.9. *On the prediction sequence A, we have* $\text{cost}(\textsc{Idle}) = O(1) \cdot \text{cost}(\text{OPT}^{+1}(A))$.

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 $\text{pnext}(\text{prev}(j)) = i$, and no matching edges are permitted to cross.

Definition 5.10. For any request A_i, let $P(A_i)$ denote the set of requests B_j such that, for the purposes of ℓ_{ed}, we can match A_i and B_j. 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}, as shown in the following lemma.

Lemma 5.11. *For any sequences A, B, we have* $\ell_{ed} \leq \ell'_{ed} \leq 3\ell_{ed}$.

Proof. The first inequality follows directly from the definitions of ℓ_{ed} and ℓ'_{ed}.

Let $S = \{i : |P(A_i)| \geq 2\}$, and let $w(S) = \sum_{i \in S} w_{A_i}$. Let M be an optimal matching for ℓ_{ed}. For each $i \in S$, there is at least one unmatched $B_j \in P(A_i)$ because A_i can only get matched with one request in B. Each of these unmatched elements of B contributes to the value of ℓ_{ed}, so $\ell_{ed} \geq w(S)$.

Now we construct a feasible matching M' for ℓ'_{ed} by removing the edges incident to S from M, that is, $M' = \{(A_i, B_j) \in M | i \notin S\}$. Consider the requests unmatched by M': the weight is at most the amount originally unmatched by M together with the amount incurred from removing edges incident to S. The former contributes ℓ_{ed} weight while the latter contributes $2w(S)$ weight, so we have $\ell'_{ed} \le \ell_{ed} + 2w(S) \le 3\ell_{ed}$. □

Now we are ready to define the LEARN algorithm. For any $i \le j$, 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 if necessary.

We first prove that the algorithm is indeed feasible.

LEMMA 5.12. *In the LEARN algorithm, Step (3a) is feasible, i.e., if t satisfies (1), then $A_t = p$.*

PROOF. Consider the optimal matching M between $A(s + 1, t)$ and S; we will show that both A_t and p are matched in M, and this implies that (A_t, p) is an edge in M, so $A_t = p$.

For contradiction, first suppose that A_t is not matched in M, in which case
$$\begin{aligned}\ell'_{ed}(A(s + 1, t - 1), S) &= \ell'_{ed}(A(s + 1, t), S) - w(A_t)\\ &< \frac{1}{3}(w(A(s + 1, t)) + w(S)) - w(A_t)\\ &\le \frac{1}{3}(w(A(s + 1, t - 1)) + w(S)),\end{aligned}$$
which means $t - 1$ satisfies (1), contradicting our choice of the minimum t satisfying (1).

Now we will show that p is matched in M. For contradiction, suppose p is not matched in M, which means A_t is matched to some other request $B_i \in S'$. By the defined matching conditions, we have $\text{pnext}(\text{prev}(i)) = t$. This implies that when the algorithm was serving request B_i, it could see the prediction sequence $A(s, t)$.

Let S' denote the contents of the queue up through B_i, and let $w(S \setminus S')$ denote the weight of pages in $S \setminus S'$ (including p). Since A_t is matched to B_i, no pages in $S \setminus S'$ can be matched when considering request p. Thus, we have the following:
$$\begin{aligned}\ell'_{ed}(A(s + 1, t), S') &= \ell'_{ed}(A(s + 1, t), S) - w(S \setminus S')\\ &< \frac{1}{3}(w(A(s + 1, t)) + w(S)) - w(S \setminus S')\\ &\le \frac{1}{3}(w(A(s + 1, t)) + w(S')),\end{aligned}$$
which means S' satisfied (1) by matching B_i with A_t, contradicting the fact that the algorithm did not enter Step (3a) at the time the queue was S'. □

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 OPT(B)) and the constrained edit distance ℓ'_{ed}. In particular, we prove the following lemma.

Lemma 5.13. *The algorithms Learn and Idle satisfy* $\text{cost}(\textsc{Learn}) \le \text{cost}(\textsc{Idle}) + 12\ell'_{ed}$.

Note that the proof of Theorem 1.6 follows directly from Lemmas 5.7, 5.9, and 5.13.

Proof of Lemma 5.13. Let cost_1 denote the total cost of Step (3a) and let cost_2 denote the total cost of Step (3b), so $\text{cost}(\textsc{Learn}) = \text{cost}_1 + \text{cost}_2$. From the algorithm, we can see that $\text{cost}_1 \le \text{cost}(\textsc{Idle})$.

So now we will prove $\text{cost}_2 \le 12\ell'_{ed}$ by induction on the times we enter Step (3a). Let $w_A(a,b) = w(A(a,b))$ and $w_B(a,b) = w(B(a,b))$. Let $\text{cost}_2(a,b)$ denote the total cost of Step (3b) when it serves input requests from time a to time b. Finally, let $\ell'_{ed}((a,b),(c,d))$ be the distance between $A[a\ldots b]$ and $B[c\ldots d]$ according to the definition of ℓ'_{ed}.

If we never enter Step (3a), then the algorithm trivially evicts every page of the input B, so

$$\text{cost}_2 \le 2w_B(1,n) \le 2w_A(1,m) + 2w_B(1,n) \le 6\ell'_{ed}$$

where the final inequality follows from the fact that we never satisfied (1).

Now assume the $\text{cost}_2 \le 12\ell'_{ed}$ if we enter Step (3a) fewer than i times; we will show that $\text{cost}_2 \le 12\ell'_{ed}$ if we enter Step (3a) i times.

Consider the first time we enter Step (3a), at which point we have read input $B(1,b)$ and we imitate Idle on $A(1,a)$. From the definition of ℓ'_{ed}, there exists some integer c such that

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

Consider the following cases:

(1) $c = b$: In this case, we have

$$\begin{aligned}\text{cost}_2 &= \text{cost}_2(1,b-1) + \text{cost}_2(b+1,n)\\ &\le 6\cdot\ell'_{ed}((1,a-1),(1,b-1)) + 12\cdot\ell'_{ed}((a+1,m),(b+1,n)),\end{aligned}$$

where the equality holds due to Lemma 5.12, and the inequality follows from the fact that we did not enter Step (3a) on request B_{b-1} and the induction hypothesis. Again, Lemma 5.12 and our choice to enter Step (3a) imply that this quantity is equal to

$$6\cdot\ell'_{ed}((1,a),(1,b)) + 12\cdot\ell'_{ed}((a+1,m),(b+1,n)),$$

which is at most $12\cdot\ell'_{ed}$ by the definition of c and our case assumption.

(2) $c < b$: Since we did not enter Step (3a) earlier, we have

$$\ell'_{ed}((1,a_0),(1,c)) \ge \frac{1}{3}\left(w_A(1,a_0) + w_B(1,c)\right) \tag{2}$$

for every $a_0 < a$. Furthermore, since we are now entering Step (3a), we have

$$\ell'_{ed}((1,a),(1,b)) \le \frac{1}{3}\left(w_A(1,a) + w_B(1,b)\right). \tag{3}$$

Let M be the optimal matching for $\ell'_{ed}((1,a),(1,b))$, and consider the following matching M' for $\ell'_{ed}((1,a),(1,c))$:

$$M' = \{(A_i,B_j) \in M \mid j \le c\}.$$

Let $d_M = \sum_{(A_i,B_j)\in M} w_{A_i} - \sum_{(A_i,B_j)\in M'} w_{A_i}$, and let $a' = \arg\max_i(A_i,B_j) \in M'$. Now consider the constrained edit distance between $A(1,a')$ and $B(1,c)$: one option is to match $A(1,a)$ and $B(1,b)$ and remove the weight of unmatched requests. This implies the following:

$$\ell'_{ed}((1,a'),(1,c)) \le \ell'_{ed}((1,a),(1,b)) - w_A(a'+1,a) - w_B(c+1,b) + 2d_M$$

Rearranging the above yields

$$\begin{aligned} w_A(a'+1,a) + w_B(c+1,b) - 2d_M &\le \ell'_{ed}((1,a),(1,b)) - \ell'_{ed}((1,a'),(1,c)) \\ &\le \frac{1}{3}(w_A(1,a) + w_B(1,b) - w_A(1,a') - w_B(1,c)) \\ &= \frac{1}{3}\left(w_A(a'+1,a) + w_B(c+1,b)\right), \end{aligned}$$

where the second inequality follows from inequalities (2) and (3). Further rearranging and applying the inequality $d_M \le w_A(a'+1,a)$ yields

$$d_M \ge \frac{1}{2} w_B(c+1,b). \tag{4}$$

Now consider the optimal matching between $A(a+1,m)$ and $B(b+1,n)$. One way to form this matching is to match $A(a+1,m)$ and $B(c+1,n)$ (since $c<b$) and unmatch the requests matched to $B(c+1,b)$. The matching corresponding to $\ell'_{ed}((a+1,m),(c+1,n))$ is penalized by d_M when considered as a matching for $A(a+1,m)$ and $B(b+1,m)$. Furthermore, the amount of weight in $A(a+1,m)$ matched to $B(c+1,n)$ is at most $w_B(c+1,b) - d_M$. This gives us the following:

$$\begin{aligned} \ell'_{ed}((a+1,m),(b+1,n)) &\le \ell'_{ed}((a+1,m),(c+1,n)) - d_M + (w_B(c+1,b) - d_M) \\ &\le \ell'_{ed}((a+1,m),(c+1,n)), \end{aligned} \tag{5}$$

where the second inequality follows from (4). Letting $\mathrm{cost}(x,y)$ denote the cost incurred by the algorithm to serve $B(x,y)$, we have

$$\begin{aligned} \mathrm{cost}_2 &\le \mathrm{cost}(1,c) + \mathrm{cost}(c+1,b) + \mathrm{cost}(b+1,n) \\ &\le 2w_B(1,c) + 4d_M + 12\cdot\ell'_{ed}((a+1,m),(b+1,n)) && \text{(trivial upper bounds, (4), induction)} \\ &\le 4(w_B(1,c) + w_A(1,a)) + 12\cdot\ell'_{ed}((a+1,m),(c+1,n)) && \text{(trivial upper bounds and (5))} \\ &\le 12\cdot\ell'_{ed}((1,a),(1,c)) + 12\cdot\ell'_{ed}((a+1,m),(c+1,n)) && \text{(we did not enter Step (3a) at time } c) \\ &= 12\cdot\ell'_{ed}. \end{aligned}$$

(3) $c > b$: This case is very similar to the $c<b$ case. Define $d \le a$ such that

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

If $d = a$, then this case is analogous to the $c = b$ case, so from now on, we assume $d < a$. Then, for every $b' \le b$, we have

$$\ell'_{ed}((1,d),(1,b')) \ge \frac{1}{3}\left(w_A(1,d) + w_B(1,b')\right), \tag{6}$$

and since we are now entering Step (3a), we have

$$\ell'_{ed}((1,a),(1,b)) \le \frac{1}{3}\left(w_A(1,a) + w_B(1,b)\right). \tag{7}$$

Let M denote the optimal matching for $\ell'_{ed}((1,a),(1,b))$ and consider the following matching between $A(1,d)$ and $B(1,b)$:

$$M' = \{(A_i,B_j) \in M \mid i \le d\}.$$

Let $d_M = \sum_{(A_i,B_j)\in M} w_{A_i} - \sum_{(A_i,B_j)\in M'} w_{A_i}$, and let $b' = \arg\max_j (A_i,B_j) \in M'$. Since M' is a valid matching between $A(1,d)$ and $B(1,b')$, we have the following:

$$\ell'_{ed}((1,d),(1,b')) \le \ell'_{ed}((1,a),(1,b)) - w_A(d+1,a) - w_B(b'+1,b) + 2d_M. \tag{8}$$

Rearranging and applying inequalities (6) and (7) yields

$$w_A(d+1,a) + w_B(b'+1,b) - 2d_M \le \ell'_{ed}((1,a),(1,b)) - \ell'_{ed}((1,d),(1,b')) \le \frac{1}{3}\left(w_A(d+1,a) + w_B(b'+1,b)\right),$$

and further rearranging gives us

$$d_M \ge \frac{1}{3}\left(w_A(d+1,a) + w_B(b'+1,b)\right).$$

Since $d_M \le w_B(b'+1,b)$, we have

$$d_M \ge \frac{1}{2} w_A(d+1,a).$$

As in the previous case, we have

$$\ell'_{ed}((a+1,m),(b+1,n)) \le \ell'_{ed}((d+1,m),(b+1,n)) - d_M + (w_A(d+1,a) - d_M) \le \ell'_{ed}((d+1,m),(b+1,n)).$$

Letting $\text{cost}(x,y)$ denote the cost of serving $B(x,y)$, we have

$$\begin{aligned}\text{cost}_2 &\le \text{cost}(1,b) + \text{cost}(b+1,n)\\ &\le 2w_B(1,b) + 12\cdot\ell'_{ed}((a+1,m),(b+1,n))\\ &\le 2w_B(1,b) + w_A(1,d) + 12\cdot\ell'_{ed}((d+1,m),(b+1,n))\\ &\le 6\cdot\ell'_{ed}((1,d),(1,b)) + 12\cdot\ell'_{ed}((d+1,m),(b+1,n))\\ &= 12\cdot\ell'_{ed}.\end{aligned}$$

□

The Follow Algorithm. Now we show that the $\Omega(\ell_1)$ lower bound in Theorem 1.5 is tight, that is, we will give an SPRP algorithm Follow that has cost $O(1)\cdot(\text{OPT}+\ell_1)$. Recall the Static algorithm from Theorem 4.1. The algorithm Follow essentially ignores its input, following Static (on the prediction sequence A) as much as possible.

More specifically, we maintain that after each time step, Static and Follow have the same set of k pages in their cache, starting with the empty cache. At each time step t, let $p = A(t)$ and $q = B(t)$ denote the corresponding predicted and actual page requests. If $p = q$, then Follow does whatever Static does. Otherwise, consider the following cases:

- $p \in$ Static, $q \in$ Follow: Follow does whatever Static does (if anything).
- $p \in$ Static, $q \notin$ Follow: Follow evicts p, fetches q, evicts q, and fetches p.
- $p \notin$ Static, $q \in$ Follow: Suppose Static evicts p' to serve p. Then Follow serves q, evicts p', and fetches p.
- $p \notin$ Static, $q \notin$ Follow: Suppose Static evicts p' to serve p. Then Follow evicts p', fetches q, evicts q, and fetches p.

Theorem 5.14. *The Follow algorithm has cost* $O(1)\cdot(\text{OPT}+\ell_1)$.

Proof. First, we claim $\text{cost}(\text{Follow}) \le \text{cost}(\text{Static}) + \ell_1$. At time t, if $A(t) = B(t)$, then cost(Follow) and cost(Static) increase by the same amount. Otherwise, notice that in every case, cost(Follow) increases by at most $w(p) + w(q)$, which is the the term corresponding to t in the definition of ℓ_1 (see Definition 5.2).

By Theorem 4.1, we know $\text{cost}(\text{Static}) \le O(1)\cdot\text{OPT}(A)$. Using the same argument as above, we can prove $\text{OPT}(A) \le \text{OPT}(B) + \ell_1$ by replacing cost(Follow) and cost(Static) with OPT(A) and OPT(B), respectively. Combining these inequalities proves the theorem. □

6 CONCLUSION

In this article, 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 [10] 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. 1997. On the influence of lookahead in competitive paging algorithms. *Algorithmica* 18, 3 (1997), 283–305. https://doi.org/10.1007/PL00009158

[2] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. 2020. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th International Conference on Machine Learning, (ICML'20), 13–18 July 2020, Virtual Event (Proceedings of Machine Learning Research)*, Vol. 119. PMLR, 345–355. http://proceedings.mlr.press/v119/antoniadis20a.html.

[3] Nikhil Bansal, Niv Buchbinder, and Joseph Seffi Naor. 2012. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM (JACM)* 59, 4 (2012), 19.

[4] Laszlo A. Belady. 1966. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal* 5, 2 (1966), 78–101.

[5] Marek Chrobak, H. Karloof, Tom Payne, and S. Vishwnathan. 1991. New results on server problems. *SIAM Journal on Discrete Mathematics* 4, 2 (1991), 172–181.

[6] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. 1991. Competitive paging algorithms. *Journal of Algorithms* 12, 4 (1991), 685–699.

[7] Sreenivas Gollapudi and Debmalya Panigrahi. 2019. Online algorithms for rent-or-buy with expert advice. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, 9–15 June 2019, Long Beach, California, *(Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 2319–2327. http://proceedings.mlr.press/v97/gollapudi19a.html.

[8] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. 2019. Learning-Based frequency estimation algorithms. In *7th International Conference on Learning Representations (ICLR'19),* (New Orleans, LA, May 6–9, 2019*).* OpenReview.net. https://openreview.net/forum?id=r1lohoCqY7.

[9] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. 2020. Online scheduling via learned weights. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA 2020,)* (Salt Lake City, UT, January 5–8*), 2020,* Shuchi Chawla (Ed.). SIAM, 1859–1877. https://doi.org/10.1137/1.9781611975994.114

[10] Thodoris Lykouris and Sergei Vassilvitskii. 2021. Competitive caching with machine learned advice. *J. ACM* 68, 4 (2021), 24:1–24:25. https://doi.org/10.1145/3447579

[11] Michael Mitzenmacher. 2018. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018 (NeurIPS'18),* (December 3–8, 2018, Montréal, Canada), Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 462–471. https://proceedings.neurips.cc/paper/2018/hash/0f49c89d1e7298bb9930789c8ed59d48-Abstract.html.

[12] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms.* Cambridge University Press, New York, NY.

[13] Manish Purohit, Zoya Svitkina, and Ravi Kumar. 2018. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018 (NeurIPS 2018),* (December 3–8, 2018, Montréal, Canada), Samy Bengio, Hanna M. Wallach, Hugo Larochelle,

Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (Eds.). 9684–9693. https://proceedings.neurips.cc/paper/2018/hash/73a427badebe0e32caa2e1fc7530b7f3-Abstract.html.

[14] Dhruv Rohatgi. 2020. Near-Optimal bounds for online caching with machine learned advice. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, (SODA'20),* (Salt Lake City, UT, January 5–8, 2020*)*, Shuchi Chawla (Ed.). SIAM, 1834–1845.

[15] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 2 (1985), 202–208.

[16] Alexander Wei. 2020. Better and simpler learning-augmented online caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM'20)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[17] Neal Young. 1991. On-line caching as cache size varies. In *Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'91)*. 241–250.

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

Received July 2020; revised July 2022; accepted July 2022