Parsimonious Learning-Augmented Caching

Sungjin Im [1] **Ravi Kumar** [2] **Aditya Petety** [1] **Manish Purohit** [2]

Abstract

Learning-augmented algorithms—in which, traditional algorithms are augmented with machine-learned predictions—have emerged as a framework to go beyond worst-case analysis. The overarching goal is to design algorithms that perform near-optimally when the predictions are accurate yet retain certain worst-case guarantees irrespective of the accuracy of the predictions. This framework has been successfully applied to online problems such as caching where the predictions can be used to alleviate uncertainties. In this paper we introduce and study the setting in which the learning-augmented algorithm can utilize the predictions parsimoniously. We consider the caching problem—which has been extensively studied in the learning-augmented setting—and show that one can achieve quantitatively similar results but only using a *sublinear* number of predictions.

1. Introduction

Learning-augmented algorithms have recently emerged as a framework to strengthen traditional algorithms with machine-learned predictions. Traditional algorithm design focuses on formal guarantees for *all* inputs. Hence, it is often geared towards working well on worst-case inputs and not for typical, real-world instances. In contrast, machine learning (ML) performs extremely well on typical instances but can occasionally fail on rare instances. The learning-augmented framework aims to design algorithms that can benefit from the machine-learned predictions while retaining worst-case guarantees.

This framework was initiated by Kraska et al. (2018), who demonstrated that indexed data structures can be improved using learned predictions. Inspired by their work, Lykouris & Vassilvitskii (2018) studied the classic online caching problem and obtained an algorithm whose performance guarantee gracefully degrades as the prediction quality worsens but still remains robust regardless of the prediction quality. The learning-augmented framework has since found applications in streaming algorithms, data structures, and particularly for online algorithms where predictions can alleviate the uncertainties for unseen future inputs; see the survey by Mitzenmacher & Vassilvitskii (2020).

In this paper we focus on an important yet largely overlooked aspect in previous works—the *cost* of obtaining predictions. Predictions are typically obtained from a machine learned model, which can be computationally expensive; this makes it highly desirable to use predictions *parsimoniously*. We study online caching in the learning-augmented framework in which hints are used sparingly.

Online caching. In the online caching problem, a sequence of page requests arrives at a cache of size k. If the requested page is in the cache, then it can be served at no extra cost, but otherwise a cache miss occurs to fetch the missing page into the cache. The goal of an online algorithm is to minimize the number of cache misses. A number of randomized algorithms are known to be $\Theta(\log k)$-competitive (Achlioptas et al., 2000; Fiat et al., 1991)—meaning that they incur $O(\log k)$ times more cache misses than the offline optimal solution for all inputs—and this is the best possible. Belady's *furthest-in-future* algorithm (Belady, 1966) that always evicts the page whose next request is the furthest in the future is well-known to be the optimal offline algorithm

To exploit predictions, Lykouris & Vassilvitskii (2018) proposed an algorithm that assumes the knowledge of the next *predicted* request time of *all* pages in the cache. For the ℓ_1-norm prediction error with respect to the actual arrival times, they showed that the algorithm's competitive ratio improves to $O(1)$ as the error tends to 0 and remains $O(\log k)$ always. These bounds have further been quantitatively improved recently by Rohatgi (2020); Wei (2020).

Our contributions. We show that we can use significantly fewer predictions for online caching to obtain results similar to the aforementioned work. More precisely,

[1]UC Merced [2]Google Mountain View. Correspondence to: Sungjin Im <sjin.im@gmail.com>, Ravi Kumar <ravi.k53@gmail.com>, Manish Purohit <mpurohit@google.com>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

we allow our algorithm to *query* b pages in the cache to learn their predicted next arrival time for each cache miss. We show that we can obtain an $O(\log_{b+1} k)$-competitive ratio for good predictions while retaining the $O(\log k)$-competitive ratio always (Theorem 11). Thus, as long as the cache miss rate is $\frac{1}{k^\epsilon}$ for any constant $\epsilon > 0$, we can obtain a constant $O(\log \frac{1}{\epsilon})$-competitive ratio using a *sub-linear* number of queries in the number of page requests. We also show that our trade-off is near-optimal (Theorem 12). Our experiments show that even with very few queries, e.g., making two queries per cache miss, we can significantly improve upon traditional caching algorithms in practice. The experimental results also demonstrate that we can match (and even exceed) the performance of prior learning-augmented algorithms but querying only $\approx 11\%$ of the page requests.

As is typical in unweighted caching, our algorithm is based on the randomized marking algorithm. However, instead of evicting a randomly chosen unmarked page on a cache miss, our algorithm queries b unmarked pages in the cache and evicts the one with the furthest predicted request time. At a high-level, if there are k unmarked pages, then we can show that the evicted page is not requested before $k/(b+1)$ other unmarked pages in the cache in expectation, provided all the predictions are correct. Using this we can show how to reduce the number of cache misses. While this idea is easy to state, the analysis is delicate as the prediction error is defined only over the pages that were queried. To retain an $O(\log k)$-competitive ratio, we adapt the techniques of Lykouris & Vassilvitskii (2018) and switch to using the randomized marking strategy once we detect that the algorithm has made too many mistakes. The lower bound is shown by an explicit but intricate construction.

Related work. Online caching has been extensively studied in the literature. For generalizations of caching, including the k-server problem, see (Koutsoupias & Papadimitriou, 1995; Bansal et al., 2015; Bubeck et al., 2018; Lee, 2018; Bansal et al., 2012; Adamaszek et al., 2012). In order to circumvent the pessimistic lower bounds in the worst-case setting, a number of alternative models have been proposed to capture properties of real-world instances such as the access graph model (Borodin et al., 1991; Fiat & Mendel, 1997; Irani et al., 1996), Markov paging (Karlin et al., 2000), and interleaved paging (Barve et al., 2000; Kumar et al., 2020). The reader is referred to the book by Borodin & El-Yaniv (2005) for a general overview of online algorithms.

Learning-augmented algorithms largely fall in the rubric of "beyond worst-case algorithms"; see (Roughgarden, 2020) for an extensive survey of the field. They have recently been extensively explored particularly for online algorithms, including load balancing (Lattanzi et al., 2020; Li & Xian, 2021), rent-or-buy (Kumar et al., 2018), scheduling (Azar et al., 2021), online set cover (Bamas et al., 2020), metrical task systems (Antoniadis et al., 2020), and many others. For online caching, its weighted version has been studied in (Jiang et al., 2020; Bansal et al., 2022).

The problem of learning-augmented algorithms with sub-linear number of queries was recently studied by Bhaskara et al. (2021), but in the regret setting for online linear optimization. Our paper studies an analogous question for caching, but in the competitive ratio setting.

The online algorithms with advice model (Boyar et al., 2017) is loosely related to learning-augmented algorithms. Here the goal is to quantify the *amount* of advice necessary to obtain a (near-)optimal solution; the model assumes the advice is error-free. For online caching, Dobrev et al. (2009) show that one bit of advice per page request suffices to obtain an optimal solution, where the advice is whether a page should be kept in the cache until it is requested next. Unfortunately, such advice is derived from the optimal solution itself and is arguably hard to learn; in contrast, predicting the next arrival time of each page—which is our setting—is a much easier task to learn.

2. Model

Let $\mathcal{U}$ denote a universe of pages and k be the number of distinct pages that can be held in the cache at any time. In the classical unweighted *caching* problem, a sequence $\Gamma = \langle p_1, p_2, \ldots \rangle$, where each $p_i \in \mathcal{U}$, of page requests arrives online and the algorithm is required to maintain a set of at most k pages in the cache at any time. At any time t, if the currently requested page p_t is not in the cache, then the algorithm incurs a *cache miss* and must fetch the requested page in the cache (possibly by evicting some other page). The objective of the online algorithm is to minimize the total number of cache misses incurred.

Note that an online algorithm has to choose the page to be evicted without knowing Γ. We measure its performance by comparing against the *furthest-in-the-future (FiF)* algorithm of Belady (1966), which is the optimal offline algorithm that knows Γ. Let $\text{cost}_\Gamma(\cdot)$ be the total number of cache misses of an algorithm for the request sequence Γ and let $\text{OPT}_\Gamma = \text{cost}_\Gamma(\mathsf{FiF})$ be the optimal offline cost. An online (randomized) algorithm $\mathcal{A}$ is said to be *c-competitive* if for all request sequences Γ, it holds that

$$E[\text{cost}_\Gamma(\mathcal{A})] \le c \cdot \text{OPT}_\Gamma + b,$$

where $b \ge 0$ is a constant independent of the length of Γ, and the expectation is over the randomness (if any) of $\mathcal{A}$. For brevity, from now on we will work with a given Γ and omit it from all subsequent notation.

In the usual learning-augmented setting, at each time t, along with the requested page p_t, the algorithm is presented with a (possibly noisy) prediction $\tau_t \in \mathbb{N}$ for the next time after t that the page p_t will be requested again; hence the predicted arrival time of the next request is available for every page in the cache. In the learning-augmented setting *with queries*, at any time t and for any page p that is in the cache, the algorithm is allowed to *query* a possibly noisy (stochastic) oracle $\mathcal{Q}$ for the time, after t, of the next request for p. Let $\tau_{p,t} = \mathcal{Q}(p,t)$ denote such a predicted arrival time of the next request to page p after time t; let $a_{p,t} \geq t$ denote the actual arrival time of the next request to page p. Let Q be the set of queries made to $\mathcal{Q}$. We define the *error* of the oracle to be $\eta = \sum_{(p,t)\in Q} |\tau_{p,t} - a_{p,t}|$.

The learning-augmented setting with queries generalizes many well-studied caching problems. On one hand, if the algorithm makes no queries to the oracle, then it is the standard caching problem and we can get a $O(\log k)$-competitive solution, say, with a randomized marking algorithm (see Section 3.1). On the other hand, if the oracle is error-free and the algorithm queries it at every time step, Belady's algorithm yields the optimal solution. In a recent work, Lykouris & Vassilvitskii (2018); Wei (2020); Rohatgi (2020) designed a learning-augmented caching algorithm for noisy oracles, showing a tight trade-off between the error of the oracle and the competitive ratio of the algorithm; their algorithm, however, queries the oracle at every time step. The question we ask in this paper is: can we get similar trade-offs but using much fewer queries?

3. Preliminaries

A pair (p, t_1) and (q, t_2) of queries in Q is called an *inversion* if $\tau_{p,t_1} \geq \tau_{q,t_2}$ but $a_{p,t_1} < a_{q,t_2}$, i.e., the next request of page p is earlier than that of q although the predictions indicated otherwise. Let $I = |\{(p,t_1),(q,t_2) \mid \tau_{p,t_1} \geq \tau_{q,t_2} \text{ but } a_{p,t_1} < a_{q,t_2}\}|$ be the number of inversions. The following relates the number of inversions to the error.

Lemma 1 (Diaconis & Graham (1977); Rohatgi (2020))**.** *For any request sequence Γ and any set Q of queries,*

$$\eta \geq \frac{1}{2} I.$$

3.1. Marking algorithms

Marking algorithms are a class of caching algorithms that associate a "marking" bit with each page in the cache, and upon a cache miss only evict an *unmarked* page from the cache. Formally, the algorithm first divides the request sequence into phases where a *phase* is a maximal contiguous sequence of requests to only k distinct pages. At the beginning of each phase, all pages in the cache are unmarked. Pages that are requested during the phase get marked one by one and upon any cache miss, the algorithm only evicts some *unmarked* page. Once all the pages in the cache have been marked, a new phase begins and the process repeats. Algorithm 1 shows the pseudocode for a generic marking algorithm. It is well known that *any* marking algorithm is $O(k)$-competitive and the *randomized marking* algorithm (Fiat et al., 1991), which evicts an unmarked page chosen uniformly at random, is $O(H(k))$-competitive, where $H(k) := 1 + \frac{1}{2} + \cdots + \frac{1}{k} = \Theta(\log k)$.

Algorithm 1: A generic *marking* algorithm.

```
for each requested page $p$ do
    if $p$ in cache then
        "Mark" $p$
    else
        if all pages in cache are marked then
            Unmark all pages
        end
        Evict an unmarked page
        Fetch $p$ in cache and "mark" it
    end
end
```

Consider any phase h and an arbitrary page p that is requested in phase h. We say that page p is *clean* if p was not requested in the previous phase (i.e., phase $h-1$), and we say p is *stale* otherwise. Note that once k is known, the phases of the sequence—as well as clean and stale pages—are determined, *independent* of the algorithm. Let ℓ_h denote the total number of distinct clean pages requested in phase h. The following result bounds the number of cache misses incurred by the optimal offline algorithm in terms of the number of distinct clean pages.

Lemma 2 (Fiat et al. (1991))**.** $\frac{1}{2}\sum_h \ell_h \leq \text{OPT} \leq \sum_h \ell_h$.

4. Warm-up: Modified marking algorithm

We first show how the classic randomized marking algorithm (Fiat et al., 1991) can be modified to effectively use predictions but making fewer queries. For ease of exposition, we assume for now that the oracle is error-free; we extend the analysis to handle noisy predictions in Section 4.1.

We consider the following modification to the marking algorithm: whenever a page needs to be evicted, if there are at least ϵk *unmarked* pages in the cache, then evict an unmarked page chosen uniformly at random; otherwise, query $\mathcal{Q}(p,t)$ for all unmarked pages p and evict the page whose next request appears furthest in the future (i.e., apply Belady's method). We remark that once we query all the remaining unmarked pages in a phase, we can simply reuse these predictions for any further cache misses and hence make at most ϵk queries in any phase. Algorithm 2 describes this naive eviction policy formally.

Algorithm 2: Naive eviction.

```
Function: Evict():
    Data: $U \subseteq \mathcal{U}$: Set of unmarked pages in cache
    Result: $\alpha$: Page to be evicted
    if $|U| \geq \epsilon k$ then
        $\alpha \leftarrow$ Uniformly random page from $U$
    else
        if we have not already queried pages in $U$ in this phase then
            foreach page $p$ in $U$ do
                Let $\tau_p \leftarrow \mathcal{Q}(p, t)$
        $\alpha \leftarrow \text{argmax}_{p \in U}\, \tau_p$
    return $\alpha$;
```

Theorem 3. *For any $\epsilon > 0$, for any request sequence Γ, there is an $O(\log(1/\epsilon))$-competitive algorithm for caching that makes at most $\epsilon|\Gamma|$ queries.*

Proof. Consider any phase h of the marking algorithm and ℓ_h be the number of clean pages in that phase. Let $p_1, \ldots, p_k$ denote the k pages in cache at the beginning of the phase and further suppose that the pages are sorted in order of the arrival time of the first request to a page in this phase (breaking ties arbitrarily). In other words, pages p_1, $\ldots$, $p_{k-\ell_h}$ are the $k - \ell_h$ stale pages requested in this phase and further the first request to page p_i is earlier than that of page p_j for any $i < j$.

Consider any stale page p_i where $1 \leq i \leq k - \epsilon k$, and let $\ell^{(i)} \leq \ell_h$ be the number of clean pages that have been requested before the first request to page p_i. When the first request to page i arrives, there are exactly $k - i + 1$ unmarked stale pages of which $\ell^{(i)}$ pages have been evicted from the cache uniformly at random. Hence, the algorithm incurs a cache miss for page p_i with probability $\frac{\ell^{(i)}}{k-i+1} \leq \frac{\ell_h}{k-i+1}$.

Finally, consider the first request to page $p_{k-\epsilon k}$. If the algorithm incurs any cache miss after this time, then it queries all the ϵk remaining unmarked pages and evicts the page whose next request is furthest in the future, i.e., it evicts a page from the set $\{p_{k-\ell_h}, \ldots, p_k\}$ that is not requested in this phase. Thus, for any $i > k - \epsilon k$, page p_i incurs a cache miss only if it has already been evicted by the time page $p_{k-\epsilon k}$ is first requested. Thus, any such page p_i incurs a cache miss with probability at most $\min\{1, \frac{\ell_h}{\epsilon k}\}$.

By the linearity of expectation, summing over all pages p_i, the expected number of cache misses incurred by the algorithm for stale pages is at most $\sum_{i=1}^{k-\epsilon k} \frac{\ell_h}{k-i+1} + \sum_{i=k-\epsilon k+1}^{k-\ell_h} \frac{\ell_h}{\epsilon k} \leq \ell_h + \ell_h(H_k - H_{\epsilon k}) \leq O(\ell_h(\log \frac{1}{\epsilon}))$. In addition, the algorithm also incurs ℓ_h additional cache misses for the clean pages. Hence, the total expected number of cache misses incurred in phase h is $O(\ell_h \log(\frac{1}{\epsilon}))$. The desired competitive ratio now follows from Lemma 2.

To bound the total number of queries, we observe that the algorithm makes at most ϵk queries in each phase. Since each phase has at least k requests, any request sequence Γ has at most $|\Gamma|/k$ phases, and thus the total number of queries is at most $\epsilon|\Gamma|$. $\square$

4.1. Handling prediction errors

Let us now consider the case where the oracle can give erroneous predictions.

Since Algorithm 2 does not utilize predictions as long as there are at least ϵk unmarked pages left in the cache, we only need to reconsider the cache misses that occur after there are fewer than ϵk unmarked pages left. Consider any page p_i for $i > k - \epsilon k$ and let $\tilde{t}$ denote the time when the algorithm queries all the remaining unmarked pages. Suppose the algorithm incurs a cache miss on page p_i and evicts page $q = \text{argmax}_{p \in U}\, \tau_{p,\tilde{t}}$. Now, if the predictions are correct, then q belongs to the set $\{p_{k-\ell_h}, \ldots, p_k\}$ of pages that are not requested in this phase. However, suppose the predictions are incorrect and page q is requested in this phase, then the algorithm incurs an additional cache miss. However, in this case the pair $(q, \tilde{t})$ and $(p_k, \tilde{t})$ of queries is an inversion and we can charge the additional cache miss incurred to this inversion. Since we only incur at most one cache miss for a page, it can be easily verified that we charge at most one cache miss to a specific inversion. Let I_h be the total number of inversions for queries made in phase h, then from the above discussion we have that the expected number of cache misses incurred by the algorithm in phase h is at most $O(\ell_h(\log \frac{1}{\epsilon}) + \mathbb{E}[I_h])$. Hence, the total cost incurred over all phases is $O((\sum_h \ell_h)(\log \frac{1}{\epsilon}) + \mathbb{E}[I])$ where the expectation is over the randomness in the pages evicted by the algorithm. Using Lemma 1 and Lemma 2, we conclude that the total cost incurred by Algorithm 2 is at most $O(2\log(1/\epsilon)\text{OPT} + \mathbb{E}[\eta])$ and obtain the following:

Theorem 4. *For any $\epsilon > 0$, there is an $O(\log(1/\epsilon) + \mathbb{E}[\eta]/\text{OPT})$-competitive algorithm for caching that makes at most $\epsilon|\Gamma|$ queries.*

While this warm-up result is a proof of concept for parsimonious use of predictions, to achieve a constant competitive ratio, we still need to make a linear number of queries in the request sequence length. To overcome this weakness we propose a new algorithm in the following section that is more adaptive in deciding which pages to query.

5. Adaptive query algorithm

In this section we present a new algorithm that queries b unmarked pages uniformly at random on each cache miss and evicts the one that is predicted to be requested the furthest in the future. We call this the *adaptive query* algorithm (AdaptiveQuery-b); see Algorithm 3. Here, b is a parameter that governs a trade-off between the desired competitive ratio and the number of queries we are willing to make per cache miss.

Algorithm 3: Adaptive query eviction.

```
Function: Evict():
    Data: $U \subseteq \mathcal{U}$: Set of unmarked pages in cache
          and an integer $b > 0$
    Result: $\alpha$: Page to be evicted
    $S \leftarrow$ Sample $b$ pages from $U$ uniformly at
      random without replacement
    Let $\tau_p \leftarrow \mathcal{Q}(p, t)$ for all $p \in S$
    $\alpha \leftarrow \text{argmax}_{p\in S}\, \tau_p$
    return $\alpha$
```

As before we first analyze the algorithm assuming the predictions are all correct. We will show the following trade-off in Section 5.1.

Theorem 5. *Under the assumption that the oracle is error-free, for any integer $b > 0$, the adaptive query algorithm is $2(\log_{b+1} k + 3)$-competitive and makes at most $2b(\log_{b+1} k + 3) \cdot \text{OPT}$ queries in expectation.*

This bound is shown to be nearly tight in Section 6; see Theorem 12. We extend Theorem 5 in Section 5.2 to handle error-prone predictions.

5.1. Analysis

If we show that the adaptive query algorithm is c-competitive, then it immediately follows that the number of queries made is $cb \cdot \text{OPT}$. Thus, we only need to establish the desired competitive ratio.

Consider any fixed phase h of the marking algorithm and let $f_1, \dots, f_{\ell_h}$ be the clean pages requested in that phase. We consider the following notion of eviction chains (Lykouris & Vassilvitskii, 2018) for the sake of analysis. An *eviction chain* $C_i = \langle q_{i,0} := f_i, q_{i,1}, \dots, q_{i,M_i}\rangle$ is a sequence of pages constructed as follows: $q_{i,1}$ is the stale page that is evicted by the algorithm when it serves the clean page f_i; similarly for all $j \geq 1$, $q_{i,j+1}$ is the stale page that gets evicted when the algorithm serves the request to page $q_{i,j}$. Eventually, a stale page q_{i,M_i} gets evicted that is not requested in the phase and the sequence ends. We note that each eviction chain starts with a distinct clean page and ends with a stale page that is not requested in the phase. Further, the ℓ_h eviction chains are disjoint and each cache miss incurred by the algorithm is encoded in these chains. The ith eviction chain C_i leads to M_i cache misses where M_i is a random variable. Our goal is to bound the expected total number of cache misses, i.e., $\mathbb{E}[\sum_{i=1}^{\ell_h} M_i]$.

Page ranks. We first order all clean pages and stale pages in the cache by the arrival time of the first request to that page in this phase (the ℓ_h stale pages that are not requested in the phase appear last in the ordering, in an arbitrary order). For each stale page p evicted by the algorithm, we define its *rank* $r(p)$ as the number of stale pages after page p in the above ordering that have not yet been evicted (at the time p was evicted). By construction of the eviction chains, page $q_{i,j}$ is evicted when page $q_{i,j-1}$ is requested and hence $q_{i,j}$ is after $q_{i,j-1}$ in the ordering (since all pages before $q_{i,j-1}$ in the ordering have already been marked). Hence, we always have $r(q_{i,j}) \leq r(q_{i,j-1})$. Similarly, for each clean page f_i, we define its rank $r(f_i) = r(q_{i,0})$ to be the number of stale pages after f_i that have not yet been evicted when f_i was requested. Note that $r(f_i) \leq k$, for all $1 \leq i \leq \ell_h$.

We first show the following simple statement that uses the order statistics of the uniform distribution. We defer its proof to the Supplementary Material.

Lemma 6. *If $S = \{s_1, \dots, s_b\}$ is a set sampled uniformly at random without replacement from $\{0, 1, \dots, r\}$, then $\mathbb{E}[\min_{t\in[b]} s_t] \leq \frac{r}{b+1}$.*

The following two lemmas are used to bound the expected length of an eviction chain.

Lemma 7. *Consider any eviction chain C_i and suppose it evicts page $q_{i,j+1}$ to service a request to page $q_{i,j}$. Then we have $\mathbb{E}[r(q_{i,j+1}) \mid r(q_{i,j})] \leq \frac{r(q_{i,j})}{b+1}$.*

Proof. When a cache miss occurs for page $q_{i,j}$, note that all pages that appear before $q_{i,j}$ (when ordered by the arrival time of their first request in the phase) have already been marked. Thus, all the queried stale pages must appear after $q_{i,j}$. Suppose there are $r \leq r(q_{i,j})$ unmarked stale pages left. When the predictions are all correct, Algorithm 3 randomly samples b pages from all unmarked stale pages and evicts $q_{i,j+1}$ as the one that is latest in the ordering. In other words, $r(q_{i,j+1})$ is the minimum of b uniform samples from $\{0, 1, \dots, r-1\}$. Thus from Lemma 6, we have $\mathbb{E}[r(q_{i,j+1}) \mid r(q_{i,j})] \leq \frac{r-1}{b+1} \leq \frac{r(q_{i,j})}{b+1}$. □

Lemma 8. *For every $1 \leq i \leq \ell_h$, we have $\mathbb{E}[M_i] \leq \log_{b+1} k + 3$ where M_i is the length of the eviction chain beginning with the clean page f_i.*

Proof. An eviction chain ends when it evicts one of the ℓ_h stale pages that are not requested in the phase. Fix a particular chain C_i and for brevity, let $r_j := r(q_{i,j})$. Since we have $r_0 \le k$, using Lemma 7 and the law of iterated expectation, we have $\mathbb{E}[r_j] \le \frac{k}{(b+1)^j}$. By Markov's inequality, we have $\Pr[r_j \ge 1] \le \frac{k}{(b+1)^j}$. Note that if $M_i > j$, then it must be the case that $r_j \ge 1$. Indeed, if $r_j = 0$, then $q_{i,j}$ will not be evicted and the chain C_i must have length j.

Let $c := \lceil \log_{b+1}(k) \rceil$. We can now bound the expected length of the chain as follows.

$$\begin{aligned}\mathbb{E}[M_i] &= \sum_{j=0}^{c-1} \Pr[M_i \ge j] + \sum_{j \ge c} \Pr[M_i \ge j] \\ &= c + \sum_{j \ge 0} \Pr[M_i > c + j] = c + \sum_{j \ge 0} \Pr[r_{c+j} \ge 1] \\ &\le c + \sum_{j \ge 0} \frac{k}{(b+1)^{c+j}} = c + \sum_{j \ge 0} \frac{1}{(b+1)^j} \\ &\le \log_{b+1}(k) + 3.\end{aligned}$$ □

Proof of Theorem 5. Fix a phase h of the marking algorithm. Since every cache miss incurred by the algorithm is recorded in exactly one eviction chain, the expected total number of cache misses incurred by the algorithm in this phase is $\mathbb{E}[\sum_{i=1}^{\ell_h} M_i]$. Using Lemma 8, this is at most $\ell_h(\log_{b+1} k + 3)$. The desired competitive ratio now follows from Lemma 2. Further, the algorithm makes at most b queries for each cache miss it incurs and thus we have Theorem 5. □

5.2. Handling prediction errors

In this section we extend the analysis to allow for oracles that make erroneous predictions. In this scenario, since we evict a page that is only *predicted* to arrive furthest in the future (and not actually be the one to arrive the latest), Lemma 7 fails. However, as we show below, in this case the oracle has a large error and we can bound the expected cost of the algorithm in terms of the prediction error.

We first show the following technical statement that relates the rank of the page evicted by the algorithm and the rank of the page that actually arrives the furthest in the future from among the sampled pages (while processing any cache miss).

Lemma 9. *Let S be any set of b pages and let $a_1 < \cdots < a_b$ denote their actual next arrival times and let $\langle \tau_1, \ldots, \tau_b \rangle$ be the sequence of their predicted arrival times. Let $\eta_S = \sum_{i=1}^{b} |a_i - \tau_i|$ be ℓ_1-error of the predictions for the set S. If $\hat{b} = \operatorname{argmax}_\alpha \tau_\alpha$ is the page with the furthest predicted arrival time, then we have*

$$r(\hat{b}) \le r(b) + \eta_S.$$

Proof. We assume that $\hat{b} \ne b$ since otherwise the lemma is trivial. By definition of $\hat{b}$ we have $\tau_{\hat{b}} \ge \tau_b$ and $a_{\hat{b}} < a_b$. For convenience let $\hat{p}$ and p denote the corresponding pages. Since the number of unmarked pages between $\hat{p}$ and p, when ordered by the request time of their first request, is at most $a_b - a_{\hat{b}}$, by definition of rank we have

$$r(\hat{b}) \le r(b) + a_b - a_{\hat{b}}.$$

We now show $\eta_S \ge a_b - a_{\hat{b}}$, which will complete the proof. To show this, we observe that $\eta_S \ge |a_{\hat{b}} - \tau_{\hat{b}}| + |a_b - \tau_b|$. We consider three cases.

Case 1: $\tau_{\hat{b}} \le a_{\hat{b}}$. In this case we have $\eta_S \ge |a_b - \tau_b| = (a_b - a_{\hat{b}}) + (a_{\hat{b}} - \tau_b) \ge a_b - a_{\hat{b}}$.

Case 2: $a_b \ge \tau_{\hat{b}} > a_{\hat{b}}$. Since $\tau_b \le \tau_{\hat{b}}$, we have $\eta_S \ge |a_{\hat{b}} - \tau_{\hat{b}}| + |a_b - \tau_b| = \tau_{\hat{b}} - a_{\hat{b}} + a_b - \tau_b \ge a_b - a_{\hat{b}}$.

Case 3: $\tau_{\hat{b}} > a_b$. Here we have $|a_{\hat{b}} - \tau_{\hat{b}}| = \tau_{\hat{b}} - a_{\hat{b}} \ge a_b - a_{\hat{b}}$. □

Lemma 9 lets us prove the following analog of Lemma 8.

Lemma 10. *For every $1 \le i \le \ell_h$, we have $\mathbb{E}[M_i] \le \log_{b+1} k + 3 + 2\mathbb{E}[\eta_{S_i}]$ where M_i is the length of the eviction chain beginning with the clean page f_i and S_i is the set of pages queried when pages on path P_i are evicted.*

Proof. Fix a particular chain C_i. Let $r(q_{i,j})$ be the number of stale unmarked pages left in the cache when the algorithm incurs a cache miss for page $q_{i,j}$ at some time t. In this case, we sample a set S of b of those pages uniformly at random, and set $q_{i,j+1} = \operatorname{argmax}_{p \in S} \mathcal{Q}(p, t)$. Let $q^*_{i,j+1} = \operatorname{argmax}_{p \in S} a_{p,t}$ be the sampled page that actually arrives furthest in the future. Then by Lemma 7, we have $\mathbb{E}[r(q^*_{i,j+1}) \mid r(q_{i,j})] \le \frac{r(q_{i,j})}{b+1}$. Further, for any queried set S of pages, by Lemma 9 we have, $r(q_{i,j+1}) \le r(q^*_{i,j+1}) + \eta_S$ where η_S is the ℓ_1-error of the predictions for the set S. Thus we obtain the following where $\eta_{i,j+1}$ is defined to be the prediction error of the oracle for pages queried while evicting page $q_{i,j+1}$.

$$\mathbb{E}[r(q_{i,j+1}) \mid r(q_{i,j})] \le \frac{r(q_{i,j})}{b+1} + \mathbb{E}[\eta_{i,j+1}].$$

Now, since we have $r(q_{i,0}) \le k$, using the law of iterated expectation we have

$$\mathbb{E}[r(q_{i,j})] \le \frac{k}{(b+1)^j} + \sum_{j'=1}^{j} \frac{\mathbb{E}[\eta_{i,j'}]}{(b+1)^{j-j'}}.$$

Finally, using Markov's inequality, we have

$$\Pr[M_i > j] \le \Pr[r(q_{i,j}) \ge 1] \le \frac{k}{(b+1)^j} + \sum_{j'=1}^{j} \frac{\mathbb{E}[\eta_{i,j'}]}{(b+1)^{j-j'}}.$$

As earlier, let $c = \lceil \log_{b+1}(k) \rceil$. We can now bound the expected length of the chain.

$$\begin{aligned}
\mathbb{E}[M_i] &= c + \sum_{j \geq 0} \Pr[M_i > c + j] \\
&\leq c + \sum_{j \geq 0} \left(\frac{k}{(b+1)^{c+j}} + \sum_{j'=1}^{c+j} \frac{\mathbb{E}[\eta_{i,j'}]}{(b+1)^{c+j-j'}} \right) \\
&\leq c + \sum_{j \geq 0} \frac{1}{(b+1)^j} + \sum_{j' \geq 1} \mathbb{E}[\eta_{i,j'}] \sum_{j \geq j'} \frac{1}{(b+1)^{j-j'}} \\
&\leq \log_{b+1}(k) + 3 + 2 \sum_{j' \geq 1} \mathbb{E}[\eta_{i,j'}].
\end{aligned}$$

Finally, since the algorithm makes a distinct set of queries when evicting any page, we have $\sum_{j' \geq 1} \mathbb{E}[\eta_{i,j'}] = \mathbb{E}[\eta_{S_i}]$ and the lemma follows. □

5.3. Adding worst-case guarantees

In this section we show how a simple modification to the algorithm allows us to obtain an $O(\log k)$-competitive ratio even when the prediction error is arbitrarily large. In order to obtain this worst-case guarantee, we make the following modification: when processing a cache miss for the jth page ($q_{i,j}$) on chain C_i, if $j > \log k$, then the algorithm switches to evict an unmarked stale page uniformly at random (as opposed to querying b pages and evicting the one with the furthest predicted arrival).

Theorem 11. *For any integer $b > 0$, there is an $O(\min\{\log_{b+1} k + \mathbb{E}[\eta]/\text{OPT}, \log k\})$-competitive algorithm for caching that makes at most b queries per cache miss.*

Proof. When pages are evicted according to Algorithm 3, Lemma 10 shows that the expected length of any eviction chain is at most $O(\log_{b+1} k + \mathbb{E}[\eta_{S_i}])$. We consider the modified algorithm that switches to evicting a uniformly random unmarked page once the chain length exceeds $\log k$. Following the traditional analysis of the randomized marking algorithm (Fiat et al., 1991; Lykouris & Vassilvitskii, 2018), we observe that once the algorithm switches to random evictions, the length of the chain increases by at most $O(\log k)$ in expectation. Consequently, the modified algorithm incurs at most $O(1) \cdot \min\{\log_{b+1} k + 3 + 2\mathbb{E}[\eta_{S_i}], 2 \log k\}$ cache misses in expectation on each eviction chain.

Summing over all clean pages seen in the phase, the expected number of cache misses incurred in any phase h is at most $O(\min\{\log_{b+1} k + \mathbb{E}[\eta_h], \log k\})$, where η_h is defined to be the ℓ_1-error of all the queries made in this phase. The desired competitive ratio now follows from Lemma 2. □

5.4. Comparison to existing algorithms

There are three main learning-augmented algorithms for (unweighted) caching with access to full predictions: the works of (i) Lykouris & Vassilvitskii (2018), (ii) Rohatgi (2020), and (iii) Wei (2020). Both (i) and (ii) are very similar: they consider a marking algorithm that evicts a page with the furthest predicted next arrival time until the eviction chain becomes long enough. The only difference is that (ii) caps each eviction chain's length at $O(1)$ whereas (i) caps it at $O(\log k)$; the algorithmic difference and the resulting improvement in the competitive ratio is relatively minor, particularly for parsimonious predictions. On the other hand, (iii) shows that the competitive ratio of the algorithm that follows predictions blindly (called BlindOracle; see Section 7) can be bound in terms of the prediction error. In our work, we cap the chain length at $O(\log k)$ for simplicity. Unfortunately, we cannot directly use (iii) as we do not have predictions for all pages and hence have to use marking-based algorithms (i.e., akin to (i) and (ii)). Further, we observe experimentally that bounding the length of an eviction chain in a marking algorithm provides better practical performance while also providing worst-case guarantees as opposed to using the black-box combination with a robust algorithm as suggested by (iii).

6. Lower bound

The lower bound instance is fairly simple. Each phase starts with a request for a clean page that has never been requested before. Then, it is followed by requests for $k-1$ stale pages that are chosen uniformly at random among the k stale pages. We provide a formal description below.

Lower bound instance. The page requests proceed in phases. Let P_h denote the pages that are requested in phase h for $h \in [H]$, where H is a sufficiently large integer. We will have $|P_h| = k$ for all $h \in [H]$, and $|P_{h+1} \setminus P_h| = |P_h \setminus P_{h+1}| = 1$ for all $h \in [H-1]$. First, P_1 is an arbitrary set of k pages and there is one request for each page in P_1. We now iteratively construct P_{h+1} from P_h as follows: Let f_{h+1} be a clean page that has never been requested before. Let $p_1, \ldots, p_k$ be a uniformly random permutation of the set of pages in P_h. Then the request sequence for phase $h+1$ is f_{h+1}, $\langle f_{h+1} \rangle^k p_k$, $\langle f_{h+1} p_k \rangle^k p_{k-1}$, …, $\langle f_{h+1} p_k \ldots p_3 \rangle^k p_2$ in this order. Here, $\langle S \rangle^k$ implies k repetitions of the sequence S. Focusing on the page arriving after the repeated sequence, we will say that pages are requested in the order of $f_{h+1}, p_k, \ldots, p_2$.

The proof of Theorem 12 requires care to impose constraints on the structure of candidate algorithms, and formally demonstrate that a learning-augmented algorithm for caching can do no better than querying unmarked stale pages and always evict the one that arrives furthest in the future. Unlike in the analysis of the upper bound, the algorithm can make varying numbers of queries per cache miss, even stochastically, which renders the analysis considerably more challenging. We defer the full proof to the Supplementary Material.

Theorem 12. *For any integer $c \le \ln k$, any $(c+4)$-competitive algorithm must make at least $\frac{1}{12\ln(k+1)} ck^{1/c} \cdot$ OPT queries (with no error).*

7. Experiments

We experimentally evaluate our algorithm on a real-world dataset and demonstrate the empirical dependence of the competitive ratio on the number of queries made as well as on the prediction errors.

Input dataset. We use the CitiBike dataset as in Lykouris & Vassilvitskii (2018). The dataset comes from a publicly-available[1] bike sharing platform in New York City. For each month of 2018, we construct one instance where each page request corresponds to the starting point of a bike trip. We truncate each month's data to the first 25,000 events, and thus each input sequence length is 25,000. Finally we set the cache size $k = 500$ and obtain seven non-trivial instances[2]. We use a bigger cache than (Lykouris & Vassilvitskii, 2018) to illustrate our algorithm's trade-off between number of queries and the competitive ratio.

Predictions. To demonstrate the empirical dependence of different algorithms on the prediction error, we generate the following synthetic predictions. For each page p in the cache, its predicted next request time is set to its actual next request time plus a noise, which is drawn i.i.d. from a log-normal distribution whose underlying normal distribution has mean 0 and standard deviation σ. If the page is never requested in the future, we pretend its actual request time is the sequence length plus 1, i.e., 25,001.

We also use a very simple prediction model to demonstrate the efficacy of easy off-the-shelf predictors. For each page, we compute the average time μ_p elapsed between consecutive requests for that page. For any page p at time t, we set the predicted arrival time as $Q(p,t) = \tilde{t}_p + \mu_p$ where $\tilde{t}_p$ is the last time before t when page p was requested. We refer to these predictions as "Mean Predictions" in Table 1.

Algorithms. We implement the following algorithms.

- RandomMarker (Randomized Marking, Fiat et al. (1991)). Evicts a randomly chosen unmarked page; $\Theta(\log k)$-competitive.

[1] https://www.citibikenyc.com/system-data

[2] The other five sequences have less than 500 distinct pages and the caching problem becomes trivial.

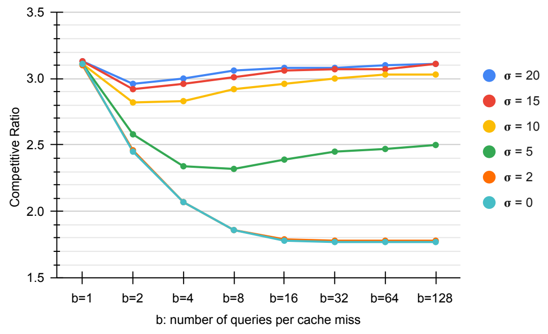

Figure 1. Average competitive ratio of AdaptiveQuery for different values of b and the error parameter σ.

Table 1. Average competitive ratio of algorithms for different error parameters. (Smaller values means better performance.)

Algorithms	Mean Predictions	Synthetic Predictions			
		$\sigma = 0$	$\sigma = 2$	$\sigma = 4$	$\sigma = 6$
RandomMarker	3.14	3.14	3.14	3.14	3.14
LRU	2.86	2.86	2.86	2.86	2.86
BlindOracle	1.92	1.00	1.02	3.92	4.15
LVMarker	2.49	1.77	1.81	2.94	3.11
RohatgiMarker	2.54	1.77	1.83	3.15	3.29
RobustOracle	4.29	1.80	1.83	4.48	4.51
AdaptiveQuery-2	2.91	2.46	2.46	2.52	2.65
AdaptiveQuery-4	2.71	2.07	2.07	2.20	2.49
AdaptiveQuery-8	2.59	1.86	1.86	2.07	2.54

- LRU (Least Recently Used). A widely used heuristic that evicts the least recently used page.
- BlindOracle. Evicts the page with the latest predicted next request time.
- LVMarker (Lykouris & Vassilvitskii, 2018). A learning-augmented marking algorithm that evicts the page with the furthest predicted arrival until the length of the eviction chain is $O(\log k)$ and then switches to evicting a randomly chosen unmarked page.
- RohatgiMarker (Rohatgi, 2020). Identical to LVMarker except the switch occurs after the chain length exceeds one.
- RobustOracle (Wei, 2020). Uses the *combiner* (Fiat et al., 1994) to combine BlindOracle and RandomMarker.
- AdaptiveQuery-b. Our algorithm (with worst-case guarantees) that is parameterized by b, the number of queries made per cache miss.

Results. Table 1 shows the competitive ratios of all the implemented algorithms averaged over the seven instances. We observe that our AdaptiveQuery algorithm performs significantly better that RandomMarker and LRU (that do not use any predictions), even while using very few predictions, e.g., making $b = 2$ queries on each cache miss. Since our algorithm is equivalent to RandomMarker when $b = 1$, this demonstrates than even minimal predictions can considerably help online algorithms.

At the same time, Table 1 also demonstrates that AdaptiveQuery performs as well as the three other learning-augmented algorithms even for relatively small values of b with both synthetic predictions as well as the simple mean predictions. In fact, for high prediction errors, the AdaptiveQuery algorithm is less affected by these errors and outperforms the other learning-augmented algorithms. For comparison, with $b = 8$, the AdaptiveQuery algorithm uses only 2839 queries for each instance on average, it utilizes predictions for about 11% of requests in the sequence.

We also compare the dependence of our algorithm on the number of queries and the prediction error. Figure 1 shows the competitive ratio of AdaptiveQuery-b for different values of b and different error parameters. Unsurprisingly, we observe that when predictions are (near-)perfect, the competitive ratio of the algorithm improves with the number of queries it is allowed to make. Surprisingly, however, when the prediction error is large, using more queries actually leads to a worse competitive ratio; indeed, in this case, more queries can lead to making poor eviction decisions.

8. Conclusions

In this paper we initiate the study of online algorithms augmented with parsimonious learned predictions. Both the theory and experimental results suggest that performance of online algorithms can be significantly improved by judiciously using just a few predictions. Such an approach can make learning-augmented algorithms more practically appealing since obtaining predictions is often computationally expensive. An interesting future direction is to further explore this parsimonious model for other online problems. For example, consider problems that involve predictions of locations, such as metric task system and online matching (Antoniadis et al., 2020). It is conceivable that one can use less predictions by spatial interpolation.

References

Achlioptas, D., Chrobak, M., and Noga, J. Competitive analysis of randomized paging algorithms. *TCS*, 234(1-2):203–218, 2000.

Adamaszek, A., Czumaj, A., Englert, M., and Räcke, H. An $O(\log k)$-competitive algorithm for generalized caching. In *SODA*, pp. 1681–1689, 2012.

Antoniadis, A., Coester, C., Elias, M., Polak, A., and Simon, B. Online metric algorithms with untrusted predictions. In *ICML*, pp. 345–355, 2020.

Azar, Y., Leonardi, S., and Touitou, N. Flow time scheduling with uncertain processing time. In *STOC*, pp. 1070–1080, 2021.

Bamas, É., Maggiori, A., and Svensson, O. The primal-dual method for learning augmented algorithms. In *NeurIPS*, 2020.

Bansal, N., Buchbinder, N., and Naor, J. S. A primal-dual randomized algorithm for weighted paging. *JACM*, 59 (4):19, 2012.

Bansal, N., Buchbinder, N., Madry, A., and Naor, J. A polylogarithmic-competitive algorithm for the k-server problem. *JACM*, 62(5):1–49, 2015.

Bansal, N., Coester, C., Kumar, R., Purohit, M., and Vee, E. Learning-augmented weighted paging. In *SODA*, 2022.

Barve, R. D., Grove, E. F., and Vitter, J. S. Application-controlled paging for a shared cache. *SICOMP*, 29(4): 1290–1303, 2000.

Belady, L. A study of replacement algorithms for a virtual-storage computer. *IBM Systems J.*, 5(2):78–101, 1966.

Bhaskara, A., Cutkosky, A., Kumar, R., and Purohit, M. Logarithmic regret from sublinear hints. In *NeurIPS*, 2021.

Borodin, A. and El-Yaniv, R. *Online Computation and Competitive Analysis*. Cambridge U. Press, 2005.

Borodin, A., Raghavan, P., Irani, S., and Schieber, B. Competitive paging with locality of reference. In *STOC*, pp. 249–259, 1991.

Boyar, J., Favrholdt, L. M., Kudahl, C., Larsen, K. S., and Mikkelsen, J. W. Online algorithms with advice: a survey. *ACM Computing Surveys (CSUR)*, 50(2):1–34, 2017.

Bubeck, S., Cohen, M. B., Lee, Y. T., Lee, J. R., and Madry, A. k-server via multiscale entropic regularization. In *STOC*, pp. 3–16, 2018.

Diaconis, P. and Graham, R. L. Spearman's footrule as a measure of disarray. *JRS Series B*, 39:262–268, 1977.

Dobrev, S., Královič, R., and Pardubská, D. Measuring the problem-relevant information in input. *RAIRO-Theoretical Informatics and Applications*, 43(3):585–613, 2009.

Fiat, A. and Mendel, M. Truly online paging with locality of reference. In *FOCS*, pp. 326–335, 1997.

Fiat, A., Karp, R. M., Luby, M., McGeoch, L. A., Sleator, D. D., and Young, N. E. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

Fiat, A., Rabani, Y., and Ravid, Y. Competitive k-server algorithms. *JCSS*, 48(3):410–428, 1994.

Irani, S., Karlin, A. R., and Phillips, S. Strongly competitive algorithms for paging with locality of reference. *SICOMP*, 25(3):477–497, 1996.

Jiang, Z., Panigrahi, D., and Sun, K. Online algorithms for weighted paging with predictions. In *ICALP*, pp. 69:1–69:18, 2020.

Karlin, A. R., Phillips, S. J., and Raghavan, P. Markov paging. *SICOMP*, 30(3):906–922, 2000.

Koutsoupias, E. and Papadimitriou, C. H. On the k-server conjecture. *JACM*, 42(5):971–983, 1995.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The case for learned index structures. In *SIGMOD*, pp. 489–504, 2018.

Kumar, R., Purohit, M., and Svitkina, Z. Improving online algorithms via ML predictions. In *NeurIPS*, pp. 9661–9670, 2018.

Kumar, R., Purohit, M., Svitkina, Z., and Vee, E. Interleaved caching with access graphs. In *SODA*, pp. 1846–1858, 2020.

Lattanzi, S., Lavastida, T., Moseley, B., and Vassilvitskii, S. Online scheduling via learned weights. In *SODA*, pp. 1859–1877, 2020.

Lee, J. R. Fusible HSTs and the randomized k-server conjecture. In *FOCS*, pp. 438–449, 2018.

Li, S. and Xian, J. Online unrelated machine load balancing with predictions revisited. In *ICML*, pp. 6523–6532, 2021.

Lykouris, T. and Vassilvitskii, S. Competitive caching with machine learned advice. In *ICML*, pp. 3296–3305, 2018.

Mitzenmacher, M. and Vassilvitskii, S. Algorithms with predictions. In Roughgarden, T. (ed.), *Beyond the Worst-Case Analysis of Algorithms*, pp. 646–662. Cambridge U. Press, 2020.

Rohatgi, D. Near-optimal bounds for online caching with machine learned advice. In *SODA*, pp. 1834–1845, 2020.

Roughgarden, T. (ed.). *Beyond the Worst-Case Analysis of Algorithms*. Cambridge U. Press, 2020.

Wei, A. Better and simpler learning-augmented online caching. In *APPROX/RANDOM*, 2020.

Supplementary Material

A. Proof of Lemma 6

Proof. We proceed assuming that replacement is allowed since it only increases $\mathbb{E}[\min_{t\in[b]} s_t]$. Then, the sampling process can be simulated by sampling $s'_1, \ldots, s'_b \sim \mathrm{Unif}(0, r)$ from the uniform distribution and taking their floor. Thus, we have $\mathbb{E}[\min_{t\in[b]} s_t] \le \mathbb{E}[\min_{t\in[b]} s'_t]$.

It well known that if $x_1, \ldots, x_b \sim \mathrm{Unif}(0, 1)$, then $\mathbb{E}[\min_{t\in[b]} x_t] = \frac{1}{b+1}$. (Indeed, this can be easily verified by observing that $\Pr[\min_{t\in[b]} x_t > x] = (1-x)^b$ and simple calculus.) Since we can set $s'_t = r \cdot x_t$, we have, $\mathbb{E}[\min_{t\in[b]} s_t] \le \mathbb{E}[\min_{t\in[b]} s'_t] = \frac{r}{b+1}$. □

B. Lower bound

We repeat the lower bound instance here for clarity.

Lower bound instance. The page requests proceed in phases. Let P_h denote the pages that are requested in phase h for $h \in [H]$, where H is a sufficiently large integer. We will have $|P_h| = k$ for all $h \in [H]$, and $|P_{h+1} \setminus P_h| = |P_h \setminus P_{h+1}| = 1$ for all $h \in [H-1]$. First, P_1 is an arbitrary set of k pages and there is one request for each page in P_1. We now iteratively construct P_{h+1} from P_h as follows: Let f_{h+1} be a clean page that has never been requested before, and let $\pi : [k] \to [k]$ denote a uniformly random permutation. Let $\langle S\rangle^k$ denote k repetitions of the sequence S. Then the request sequence for phase $h+1$ is f_{h+1}, $\langle f_{h+1}\rangle^k p_{\pi(k)}$, $\langle f_{h+1} p_{\pi(k)}\rangle^k p_{\pi(k-1)}$, $\ldots$, $\langle f_{h+1} p_{\pi(k)} \cdots p_{\pi(3)}\rangle^k p_{\pi(2)}$ in this order. Focusing on the page arriving after the repeated sequence, we will say that pages are requested in the order of $f_{h+1}, p_{\pi(k)}, \ldots, p_{\pi(2)}$. For ease of notation, we let $p_{\pi(k+1)} := f_{h+1}$.

We first observe that the optimum offline solution for such an instance always incurs a cache miss on the first, clean page of each phase (except the first phase) and evicts the unique page in $P_{h-1} \setminus P_h$. By construction, the optimum algorithm incurs no more cache misses in each phase. Finally, any algorithm must incur k cache misses for the first phase and we have the following claim.

Claim 13. *There is an offline solution that incurs exactly $k + H - 1$ cache misses in total.*

We would like to lower bound the number of cache misses incurred by any c-competitive algorithm. Consider a fixed optimum online algorithm $\mathcal{A}$.[3]

Claim 14. *At the beginning of each phase $h \ge 2$, we can assume without loss of generality that $\mathcal{A}$ has all pages in P_{h-1} in its cache.*

Proof. Assume that the universe of pages is infinite. Then, knowing that we cannot guess the clean page that will be requested in phase h and all the other requests are for the stale pages, the claim follows. □

Thanks to Claim 14 and the repeated identical structure of the lower bound instance in every phase, we can assume without loss of generality that we use the same optimum online algorithm that we call $\mathcal{A}$ in all phases except the first. If c' is the expected number of cache misses $\mathcal{A}$ incurs in each phase $h \ge 2$, for $\mathcal{A}$ to be c-competitive, it must be the case that $\frac{k+c'(H-1)}{k+H-1} \le c$ from Claim 13. Here, k in the numerator is the number of cache misses incurred by $\mathcal{A}$ in the first phase. Thus, we must have $c' = c$ as $H \to \infty$.

Therefore, we can focus on one phase and lower bound the expected number of queries made by $\mathcal{A}$ assuming that it incurs at most c cache misses in expectation. Henceforth we drop indices referring to phases from the notation. We will say that a page is marked in the phase if it has been requested in the phase. For simplicity, we will assume that $\mathcal{A}$ makes at least one query before each page eviction; this would have no effect on the asymptotic lower bound we aim to prove.

Lemma 15. *In a phase that is not the first, with a given limit $c > 0$ on the expected number of cache misses, there is an algorithm that makes the minimum number of queries in expectation and simultaneously satisfies the following:*

(i) evicts a page only when it is forced to do so;
(ii) never evicts marked pages and therefore it only needs to query unmarked pages;

[3] In this context, an optimal online algorithm is one that obtains a competitive ratio of c by making the fewest number of queries.

(iii) only queries pages just before a page eviction;
(iv) evicts the queried page with furthest arrive time, if it makes any queries.

We first prove (i)–(iii). After setting up additional notation that will be used throughout the analysis, we will prove (iv).

Proof of Lemma 15 (i)–(iii). As argued in Fiat et al. (1991), we can assume without loss of generality that the algorithm is *lazy*, i.e., it only evicts a page when it incurs a cache miss; this implies (i).

We now show (ii). Suppose algorithm $\mathcal{A}$ evicts a page $p_{\pi(i)}$ for some $i > j$ to service $p_{\pi(j)}$ in the request subsequence, $\langle p_{\pi(k+1)} p_{\pi(k)} \ldots p_{\pi(j+1)} \rangle^k p_{\pi(j)}$. Then, for every repetition of $\langle p_{\pi(k+1)} p_{\pi(k)} \ldots p_{\pi(j)} \rangle$ in the subsequence of $\langle p_{\pi(k+1)} p_{\pi(k)} \ldots p_{\pi(j)} \rangle^k p_{\pi(j-1)}$, until $\mathcal{A}$ fetches $p_{\pi(i)}$ and evicts an unmarked page, it incurs another cache miss. If it makes a cache miss for every repetition, we can make $\mathcal{A}$ better or no worse by instead evicting an arbitrary unmarked page (such an algorithm incurs at most k cache misses even without using randomization). Otherwise, if $\mathcal{A}$ ever replaces with an unmarked page before $p_{\pi(j-1)}$ is requested, $\mathcal{A}$ could have instead done so earlier to reduce cache misses. This will incur no cache misses for the repetition $\langle p_{k+1} p_k \ldots p_j \rangle^k$. We have thus shown that we can assume without loss of generality that $\mathcal{A}$ never evicts marked pages; thus, we have (ii).

Now (iii) follows as once a page gets marked it stays marked in the phase. Thus, by deferring the queries until being forced to evict a page, $\mathcal{A}$ can only potentially avoid querying about pages that will be marked soon. □

For the remaining analysis, we take the eviction chain view we used in the analysis of the adaptive query algorithm. As our analysis will require careful conditioning and deconditioning, we slightly override the notation. From the above reasoning, particularly from Lemma 15 (i)–(iii), we now have the following problem: We will see a request sequence for a clean page $p_{\pi(k+1)} = f_{h+1}$, and $k-1$ stale pages, $p_{\pi(k)}, p_{\pi(k-1)}, \ldots, p_{\pi(2)}$. Note that $p_{\pi(1)}$ is the stale page that is not requested. Then, we consider the eviction chain C that starts with $p_{\pi(k+1)}$ and ends with $p_{\pi(1)}$. Recall that an edge from q to q' means that we evict page q' to service page q. Our goal is to lower bound the number of queries made under the requirement that the expected length (number of cache misses) M of C is at most c. Page $p_{\pi(j)}$ is defined to have rank j; this definition is slightly simpler than the one in Section 5.1 as we have only one clean page, thus only one chain.

With this notation set up, we are now ready to prove Lemma 15(iv).

Proof of Lemma 15(iv). Our goal is to consider any algorithm $\mathcal{A}$ satisfying (i)–(iii), and to construct another algorithm $\mathcal{A}'$ satisfying (iv) as well, without increasing the number of cache misses but making no more queries.

In the execution of $\mathcal{A}$, suppose ith evicted page by $\mathcal{A}$ has rank R_i and $\mathcal{A}$ makes q_i queries just before the eviction. Now $\mathcal{A}'$ makes the same number q_i of queries just before evicting ith page, but it instead evicts the one with smallest rank among the queried pages, i.e., satisfies property (iv). By a simple induction on i, we can show that $\mathcal{A}'$ generates a sequence that stochastically dominates what $\mathcal{A}$ generates. More precisely, suppose $\mathcal{A}$ has made q queries, and the queried pages have ranks $S_1 < \cdots < S_q$. Then, by definition, $\mathcal{A}'$ has also made q queries (if there are not enough pages to query, then it is only better for $\mathcal{A}'$), and let $S'_1 < \cdots < S'_q$ be the rank of pages queried by $\mathcal{A}'$. Then, we say that the sequence S' stochastically dominates sequence S if $S'_j \le S_j$ for all $j \in [q]$. Because the chain C ends once the last page of rank 1 is evicted (we can assume we evict only a queried page under the assumption we query at least one page before each page eviction), $\mathcal{A}'$ make no more cache misses than $\mathcal{A}$ in expectation. Further, by construction, $\mathcal{A}'$ can only make less queries than $\mathcal{A}$ in expectation. □

Henceforth, we consider an algorithm $\mathcal{A}$ that satisfies Lemma 15(i)–(iv). Let $R_0 := k+1$, be the rank of the clean page. Let R_i denote the rank of the ith evicted page. As mentioned before, C eventually ends with $p_{\pi(1)}$. For notational convenience, once R_i becomes 1, we define all the subsequent $R_{i+1}, R_{i+2}, \ldots,$ to be 1. Let Q_i denote the number of queries $\mathcal{A}$ makes when we witness the ith cache miss. For analysis, we will assume that we do not make too many queries for each page eviction. This is because we can find the page $p_{\pi(1)}$ without making many more queries.

Lemma 16. *Suppose we show that any algorithm that incurs at most c cache misses in expectation makes at least d queries under the assumption that $R_i > Q_{i+1}^2$ for all i. Then, it implies the following lower bound: any algorithm that incurs at most $c + 4$ cache misses in expectation makes at least $d/5$ queries.*

Proof. If the fixed algorithm considered to show the lower bound makes Q_{i+1} queries such that $R_i \le Q_{i+1}^2$ for the first time, then instead we let it make $\sqrt{R_i}$ queries per cache miss until the phase ends. This is equivalent to a problem where the cache size is R_i and we make $\sqrt{R}_i$ queries per cache miss. Thus, by Lemma 8, we know that the number of cache misses is at most 5 in expectation, and we make at most $5\sqrt{R}_i \le 5Q_{i+1}$ queries in expectation. In summary, this change makes at most 4 extra cache misses in expectation and increases the number of queries by a factor of 5. $\square$

Let us fix i and let $R_i = r$ and $Q_i = q$. If we choose q points uniformly at random from $[0, r]$, the expected minimum of the samples is well known to be $\frac{r}{q+1}$. But, we want to know the expected value of $\frac{r}{R_{i+1}}$, where the only randomness comes from $\pi(1), \dots, \pi(r-1)$, which we denote as $\pi([r-1])$ for short. Bounding this quantity needs more care.

Now, our concern is to upper bound $\mathbb{E}_{\pi([r-1])} \frac{r}{S}$, where $S_1, \dots, S_q$ are sampled uniformly from $[r-1]$ without replacement and $S := \min_{j\in[q]} S_j$. This corresponds to making queries about q pages among $r-1$ unmarked ones and evicting the one with the minimum rank, i.e., the furthest request time in the future. For ease of analysis, we pretend that samples $S'_1, \dots, S'_q$ are made from $[1/r, 1]$ independently and uniformly at random and we want to upper bound $\mathbb{E}[\frac{1}{S'}]$, where $S' := \min_{j\in[q]} S'_j$. Here, we relax the random selection by making the sampling domain continuous.

We first show this relaxation does not change the expectation by much.

Lemma 17. *If* $1 \le q^2 < r$, *we have* $\mathbb{E}_{\pi(r-1)}\left[\frac{r}{S}\right] \le 6\mathbb{E}\left[\frac{1}{S'}\right]$.

Proof. We first scale down $S_1, \dots, S_q$ by a factor of r, so we can pretend that they are sampled from $D = \{\frac{1}{r}, \dots, \frac{r-1}{r}\}$ without replacement. Now, we want to upper bound $\mathbb{E}[\frac{1}{S}]$. Let $\mathcal{D} := \binom{D}{q}$.

We observe that

$$\mathbb{E}\left[\frac{1}{S}\right] = \mathbb{E}\left[\frac{1}{\bar{S}'} \;\mid\; \bar{\mathcal{S}}' \in \mathcal{D}\right],$$

where $\bar{\mathcal{S}}' := \{\lfloor rS'_i \rfloor / r \mid i \in [q]\}$ and $\bar{S}' = \min \bar{\mathcal{S}}'$. In other words, after "rounding" down each S'_i to the nearest multiple of $1/r$, if they are all distinct, we keep them. This is an equivalent way of getting samples $S_1, \dots, S_q$.

Further, the rounding changes the expectation by a factor of at most 2. Therefore, we have,

$$\mathbb{E}\left[\frac{1}{\bar{S}'} \;\mid\; \bar{\mathcal{S}}' \in \mathcal{D}\right] \le 2\mathbb{E}\left[\frac{1}{S'} \;\mid\; \bar{\mathcal{S}}' \in \mathcal{D}\right].$$

We would like to decondition on $\bar{S}' \in \mathcal{D}$.

$$\mathbb{E}\left[\frac{1}{S'} \;\mid\; \bar{\mathcal{S}}' \in \mathcal{D}\right] \cdot \Pr[\bar{\mathcal{S}}' \in \mathcal{D}] \le \mathbb{E}\left[\frac{1}{S'}\right].$$

As $\bar{\mathcal{S}}'$ is a uniform sample with replacement, we have, $\Pr[\bar{\mathcal{S}}' \in \mathcal{D}] = \frac{r(r-1)\cdots(r-q+1)}{r^q} \ge (1-q/r)^q \ge (1-q/(q^2+1))^q \ge 1/3$, where the last inequality follows from a simple calculation. Combining the above equations yields the lemma. $\square$

Lemma 18. *If* $1 \le q^2 < r$, *we have* $\mathbb{E}[\frac{1}{S'}] \le 2q\ln(k+1)$.

Proof. Observe that $\Pr[S' \ge x] = \left(\frac{1-x}{1-1/r}\right)^q$. Thus, the pdf of S' is $\frac{q(1-x)^{q-1}}{(1-1/r)^q}$ where $x \in [1/r, 1]$. For brevity, we omit the denominator in the following calculations and bring it back at the end.

$$\begin{aligned}
\mathbb{E}\left[\frac{1}{S'}\right] &= \int_{x=1/r}^{1} \frac{1}{x} q(1-x)^{q-1}\mathrm{d}x \\
&= \int_{x=1/r}^{1/q} \frac{1}{x} q(1-x)^{q-1}\mathrm{d}x + \int_{x=1/q}^{1} \frac{1}{x} q(1-x)^{q-1}\mathrm{d}x \\
&\le q\int_{x=1/r}^{1/q} \frac{1}{x}\mathrm{d}x + \int_{x=1/q}^{1} q \cdot q(1-x)^{q-1}\mathrm{d}x \\
&\le q\ln r/q + q \le q\ln r \\
&\le q\ln(k+1).
\end{aligned}$$

Thus, by factoring in the denominator $\frac{1}{(1-1/r)^q} \le \frac{1}{(1-1/(q^2+1))^q} \le 2$, we obtain the lemma. $\square$

Corollary 19. $\mathbb{E}[\frac{R_i}{R_{i+1}}] \leq 12q\ln(k+1)\mathbb{E}[Q_{i+1}]$.

Proof. From Lemmas 17 and 18, we know $\mathbb{E}_{\pi([r-1])}[\frac{r}{S(r,q,\pi)}] \leq 12q\ln(k+1)$. Here, the parameters r, q, π are used to make the dependency of S clear. As this holds for any $R_i = r$ and $Q_{i+1} = q$, by deconditioning, we have the desired result. □

Lemma 20. *If C has length M and for any integer $c > 0$, we have $\mathbb{E}[\sum_{i=1}^{c} Q_i \mid M = c] \geq \frac{1}{12\ln(k+1)} ck^{1/c}$.*

Proof. Using the linearity of expectation and Corollary 19,

$$\begin{aligned}
12\ln(k+1)\sum_{i=1}^{c}\mathbb{E}[Q_i \mid t = c] &\geq \mathbb{E}\left[\sum_{i=1}^{c}\frac{R_{i-1}}{R_i} \mid R_c = 1, R_{c-1} > 1\right] \\
&\geq \mathbb{E}\left[c(\prod_{i=1}^{c}\frac{R_{i-1}}{R_i})^{1/c} \mid R_c = 1, R_{c-1} > 1\right] \\
&= (k+1)^{1/c},
\end{aligned}$$

where the middle step follows from the AM–GM inequality and the last step follows from a telescoping product, $R_0 = k+1$, and $R_c = 1$. □

By Lemma 20, if algorithm $\mathcal{A}$ makes at most c cache misses in expectation, then the number of queries it makes is lower bounded by the optimum objective of the following LP:

$$\begin{aligned}
&\frac{1}{12\ln(k+1)} \cdot \min \sum_{i\geq 1} ik^{1/i}x_i \\
&\sum_{i\geq 1} ix_i \leq c \qquad (1) \\
&\sum_{i\geq 1} x_i = 1 \\
&x_i \geq 0 \quad \forall i \geq 1.
\end{aligned}$$

Here, $x_i := \Pr[M = i]$, i.e., the probability that the chain C has length i, or equivalently $\mathcal{A}$ makes i cache misses. The last two constraints define a probability distribution over the values M can have and constraint (1) means that we can afford to make at most c cache misses in expectation.

Lemma 21. *If $c \leq \ln k$, then the above LP's optimum objective is at least $\frac{1}{12\ln(k+1)} ck^{1/c}$.*

Proof. Let $f(y) := yk^{1/y}$. By simple calculus, we have $f'(y) = k^{1/y}\left(1 - \frac{\ln k}{y}\right)$ and $f''(y) = \frac{\ln^2 k}{y^2}k^{1/y}$. Thus, f decreases in y for $y \in [1, \ln k]$ and is convex. Then, the LP objective is $\frac{1}{12\ln(k+1)}\sum_{i\geq 1} f(i)x_i$. By convexity, we have $\sum_{i\geq 1} f(i)x_i \geq f\left(\sum_{i\geq 1} ix_i\right)$. Then, by constraint (1) and f being decreasing in y, we have $f\left(\sum_{i\geq 1} ix_i\right) \geq f(c)$. □

To summarize, we have shown that any c-competitive algorithm must make at least $\frac{1}{12\ln(k+1)}ck^{1/c}$ cache misses, but under the assumption stated in Lemma 16, i.e., $R_i > Q_{i+1}^2$ for all i. Thus, by the lemma, we have the following.

Theorem 22. *For any integer $c \leq \ln k$, any $c+4$-competitive algorithm must make at least $\frac{1}{12\ln(k+1)}ck^{1/c} \cdot \mathrm{OPT}$ queries.*