Better and Simpler Learning-Augmented Online Caching

Alexander Wei
Harvard University, Cambridge, MA, USA
https://www.alexwei.org
weia@college.harvard.edu

Abstract

Lykouris and Vassilvitskii (ICML 2018) introduce a model of online caching with machine-learned advice that marries the predictive power of machine learning with the robustness guarantees of competitive analysis. In this model, each page request is augmented with a prediction for when that page will next be requested. The goal is to design algorithms that (1) perform well when the predictions are accurate and (2) are robust in the sense of worst-case competitive analysis.

We continue the study of algorithms for online caching with machine-learned advice, following the work of Lykouris and Vassilvitskii as well as Rohatgi (SODA 2020). Our main contribution is a substantially simpler algorithm that outperforms all existing approaches. This algorithm is a black-box combination of an algorithm that just naïvely follows the predictions with an optimal competitive algorithm for online caching. We further show that combining the naïve algorithm with LRU in a black-box manner is optimal among deterministic algorithms for this problem.

2012 ACM Subject Classification Theory of computation → Caching and paging algorithms; Theory of computation → Machine learning theory; Computing methodologies → Machine learning

Keywords and phrases Online caching, learning-augmented algorithms, beyond worst-case analysis

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.60

Category APPROX

Related Version A full version of the paper is available at https://arxiv.org/abs/2005.13716.

Acknowledgements I would like to thank Jelani Nelson for advising this project and Bailey Flanigan for providing many helpful references.

1 Introduction

The study of online algorithms traditionally focuses on *worst-case* robustness, where algorithms provide the same competitive guarantee against the offline optimal over all possible inputs. In recent years, however, there has been a surge of interest in online algorithms for structured inputs [20, 18, 24, 11, 15, 16, 25, 17, 23]. A principal motivation for these works is the philosophy of "beyond worst-case analysis" [13, 26]: Many practical settings have inputs that follow restricted patterns, making worst-case competitive analysis too pessimistic to inform practice. Algorithms designed with the worst case in mind can be hamstrung by these considerations, sacrificing performance on "easy" inputs in order to eke out a better worst-case guarantee.

Learning-augmented online algorithms, introduced by Lykouris and Vassilvitskii [18] and Purohit et al. [24], is a "beyond worst-case" framework motivated by the powerful predictive abilities of modern machine learning. In this framework, the classical online algorithm model is augmented with a machine-learned oracle that predicts future inputs. A concern with simply relying on the oracle is that machine learning models typically have few worst-case guarantees. Thus, with learning-augmented online algorithms, we seek to obtain the best of both worlds. Given a predictor, our objective is to design algorithms that:

1. Perform well in the optimistic scenario, where the predictor has low error;
2. Remain robust in the classical worst-case sense, when the predictor can be arbitrarily bad.

In other words, we want our algorithm to be $c(\eta)$-competitive against the offline optimal on all inputs, for c a function of the predictor's total error η, such that $\sup_\eta c(\eta) \leq \gamma$ for some constant $\gamma \geq 1$. This γ is the classical worst-case competitive ratio and is a measure of the robustness of our algorithm.

This paper focuses on learning-augmented *online caching* [18, 25]. In the online caching (a.k.a. online paging) problem, one maintains a cache of size k online while serving requests for pages that may or may not be in the cache. For simplicity, assume that pages must always be served from the cache and that bringing a page into the cache has unit cost. (In particular, if the cache is full, bringing a page into the cache requires also *evicting* a page already in the cache.) Thus, one wishes to minimize the number of *cache misses*, i.e., requests for which the page is not already in the cache. This is a classical online problem that has been the subject of extensive study over the past several decades (see [6] for an overview). From the worst-case perspective, this problem is well-understood for not only the version stated above [27, 10, 1], but also for weighted generalizations [3, 4].

Online caching in the learning-augmented context was first considered by Lykouris and Vassilvitskii [18]. They introduce a model of prediction where the predictor, upon the arrival of each page, predicts the next time that this page will be requested. They show that the BlindOracle algorithm, which follows the predictor naïvely and evicts the page with the latest predicted arrival time, can have unbounded competitive ratio (i.e., is *non-robust*). They then give a different algorithm, PredictiveMarker, based on the Marker algorithm of Fiat et al. [10], that achieves a competitive ratio of

$$2 + O\left(\min\left(\sqrt{\frac{\eta}{\mathsf{OPT}}}, \log k\right)\right),$$

where η is the ℓ_1 error of the predictor and OPT is the cost of the offline optimal. (For precise definitions, refer to Section 2.1.) Notably, this competitive ratio approaches 2 as the error η goes to 0 and is bounded by $O(\log k)$ regardless of how large η gets.

In recent work, Rohatgi [25] introduces the LNonMarker algorithm, which is also based on randomized marking, and shows that LNonMarker achieves a competitive ratio of

$$O\left(1 + \min\left(\frac{\log k}{k}\frac{\eta}{\mathsf{OPT}}, \log k\right)\right).$$

This bound is obtained by constructing a non-robust algorithm and then using the black-box combination technique discussed in [18] to combine this non-robust algorithm with the Marker algorithm. Rohatgi also shows a lower bound of

$$1 + \Omega\left(\min\left(\log\left(\frac{1}{k \log k}\frac{\eta}{\mathsf{OPT}}\right), \log k\right)\right)$$

for the competitive ratio of any learning-augmented online algorithm for caching in terms of k, OPT, and η. The main difference between two bounds is that the former has a linear dependence on η/OPT, whereas the latter has only a logarithmic dependence on the same. We will make some progress in closing this gap and also show that deterministic algorithms fundamentally cannot approach Rohatgi's lower bound.

1.1 Our Contribution

We show that the strikingly simple approach of combining BlindOracle with an $O(\log k)$-competitive online caching algorithm (e.g., Marker) in a black-box fashion achieves state-of-the-art performance, improving over LNonMarker with a competitive ratio bound of

$$O\left(1+\min\left(\frac{1}{k}\frac{\eta}{\mathsf{OPT}}, \log k\right)\right).$$

Thus, although BlindOracle was previously shown to be non-robust [18], our result demonstrates that using BlindOracle appropriately can actually lead to very effective algorithms for learning-augmented online caching.

In addition to achieving better theoretical bounds, our approach to learning-augmented online caching is substantially simpler than previous work. The algorithms of Lykouris and Vassilvitskii [18] and Rohatgi [25] rely on intricate constructions based on randomized marking, whereas our main ingredient is a careful analysis of perhaps the simplest algorithm possible. We thus believe that our approach may yield better practical performance and may generalize more readily to other learning-augmented settings. We note that our optimal deterministic algorithm is especially simple: For each eviction, "follow" whichever of BlindOracle and LRU has performed better so far.

The crux of our result is a tight analysis of BlindOracle's performance as a function of η. We prove that BlindOracle has excellent performance when η/OPT is small, improving over $O(\log k)$-competitive online caching algorithms for η up to $O(k \log k) \cdot \mathsf{OPT}$. This is in contrast to the lower bound on BlindOracle given by Lykouris and Vassilvitskii [18], who rule out BlindOracle due to its poor performance when $k = 2$. In particular, our bound has a $1/k$ dependence on k, meaning BlindOracle obtains drastically better bounds for larger k. Stated formally, our main theorem is the following:

▶ **Theorem 1.1.** *For learning-augmented online caching, BlindOracle obtains a competitive ratio of*

$$\min\left(1+2\frac{\eta}{\mathsf{OPT}}, 4+\frac{4}{k-1}\frac{\eta}{\mathsf{OPT}}\right),$$

where η is the ℓ_1 loss incurred by the predictor and OPT is the optimal offline cost. (For precise definitions, see Section 2.1.)

We then obtain robust deterministic and randomized algorithms for learning-augmented online caching as corollaries, by combining BlindOracle with LRU and Equitable [1]. We apply the algorithms of Fiat et al. [10] and Blum and Burch [5] for combining online algorithms in a black-box manner online. These algorithms achieve better constants in the competitive ratio than the approach discussed by Lykouris and Vassilvitskii [18] and applied by Rohatgi [25]. By composing our analysis of BlindOracle with these "combiners," we obtain constants in the competitive ratio that are significantly lower than those of previous works. Indeed, we have the following corollaries for deterministic and randomized algorithms for learning-augmented online caching:

▶ **Corollary 1.2.** *There exists a deterministic algorithm for learning-augmented online caching that achieves a competitive ratio of*

$$2\min\left(\min\left(1+2\frac{\eta}{\mathsf{OPT}}, 4+\frac{4}{k-1}\frac{\eta}{\mathsf{OPT}}\right), k\right).$$

▶ **Corollary 1.3.** *There exists a randomized algorithm for learning-augmented online caching that achieves a competitive ratio of*

$$(1+\gamma)\min\left(\min\left(1+2\frac{\eta}{\mathsf{OPT}}, 4+\frac{4}{k-1}\frac{\eta}{\mathsf{OPT}}\right), H_k\right)$$

for any $\gamma \in (0, 1/4)$.[1] *(Here,* $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} = \ln(k) + O(1)$ *is the* k*-th harmonic number.)*

Finally, we show that combining BlindOracle with a k-competitive deterministic algorithm (e.g., LRU [27]) is the best one could hope to do among deterministic algorithms for learning-augmented online caching. In particular, we show that a linear dependence on $\eta/(k \cdot \mathsf{OPT})$ in the competitive ratio is necessary. Therefore, if a logarithmic dependence on $\eta/(k \cdot \mathsf{OPT})$ is to be achieved, as in Rohatgi's lower bound, then randomization is needed, (perhaps surprisingly) even in the regime where $\eta/(k \cdot \mathsf{OPT})$ is bounded.

▶ **Theorem 1.4.** *The competitive ratio bound for any deterministic learning-augmented online caching algorithm must be at least*

$$1 + \Omega\left(\min\left(\frac{1}{k}\frac{\eta}{\mathsf{OPT}}, k\right)\right)$$

in terms of k *and* η/OPT.

1.2 Related Work

In addition to the predecessor works by Lykouris and Vassilvitskii [18] and Rohatgi [25] on learning-augmented online caching, there have been several other recent papers in the space of learning-augmented online algorithms: Medina and Vassilvitskii [20] study repeated posted-price auctions, Purohit et al. [24] and Gollapudi and Panigrahi [11] study the ski rental problem, and Purohit et al. [24], Lattanzi et al. [17], and Mitzenmacher [23] study online scheduling. Of these, the scheduling algorithm of Purohit et al. [24] is the most similar in spirit to this present work: Both algorithms are based on combining a naïve and optimistic algorithm with a robust algorithm.

Other threads of research falling under beyond worst-case online algorithms include work on combining multiple algorithms with different performance characteristics [10, 5, 19, 11], designing online algorithms with distributional assumptions (e.g., stochasticity) on the input [13, 8, 21], and semi-online algorithms, where the input is assumed to have a predictable offline component and an adversarial online component [15, 16].

The idea of learning-augmentation has also been explored in many other algorithmic and data structural settings in recent years. These include learned indices [14], bloom filters [22], frequency estimation in streams [12], and nearest neighbor search [9], among others.

Finally, advice for online algorithms has also been considered with a more complexity theoretic spirit through the study of advice complexity of online algorithms; see the survey of Boyar et al. [7] for an overview.

1.3 Recent Developments

Recently, in work done independently of and concurrently with this paper, Antoniadis et al. [2] also study a BlindOracle-like algorithm, which they term FollowThePrediction, in the more general setting of learning-augmented metrical task systems; they also use the "combiner"

[1] The trade-off in γ and the cost is additive; thus, it does not factor into the competitive ratio.

of Blum and Burch [5] to make this algorithm robust. However, their prediction model, when specialized to online caching, is incomparable to that of Lykouris and Vassilvitskii [18] (which we follow).[2] Thus, the theoretical results proved in these two models do not imply each other.

1.4 Outline

The remainder of this paper is organized as follows: In Section 2, we formally describe our model of learning-augmented online caching and BlindOracle; we also introduce background results that will later be used in our proofs. In Section 3, we prove our upper bounds (i.e., Theorem 1.1 and Corollaries 1.2 and 1.3). In Section 4, we prove our lower bound (Theorem 1.4) for deterministic algorithms. Section 5 concludes.

2 Preliminaries

2.1 Setup and Notation

In the online caching problem, we receive a sequence $\sigma = (\sigma_1, \ldots, \sigma_n)$ of page requests online, and our goal is to serve these requests using a cache of size k while minimizing cost. In this problem, pages must be served from the cache and can be served at no cost; however, evicting a page from the cache has unit cost.[3]

We will establish competitive bounds comparing the performance of two caching algorithms $\mathcal{A}$ and $\mathcal{B}$. More precisely, we will show bounds of the form

$$\mathsf{ALG}_{\mathcal{B}}(\sigma) \leq \gamma \cdot \mathsf{ALG}_{\mathcal{A}}(\sigma) + O(1),$$

where $\mathsf{ALG}_{\mathcal{A}}(\sigma)$ and $\mathsf{ALG}_{\mathcal{B}}(\sigma)$ are the costs of $\mathcal{A}$ and $\mathcal{B}$, respectively, as measured in number of evictions made while serving a sequence σ of page requests. Ultimately, $\mathcal{A}$ will be the optimal offline algorithm and $\mathcal{B}$ will be BlindOracle. We will also use $\mathsf{OPT}(\sigma)$ to denote the optimal offline cost; that is, $\mathsf{OPT}(\sigma)$ is the minimum possible cost of serving the request sequence σ. We will omit the argument σ when the context is clear (i.e., just writing $\mathsf{ALG}_{\mathcal{A}}$ to represent $\mathsf{ALG}_{\mathcal{A}}(\sigma)$).

In our analysis, we use A_t and B_t to denote the cache states of $\mathcal{A}$ and $\mathcal{B}$, respectively, just before the t-th request. Formally, A_t and B_t are subsets of $\{1, \ldots, t-1\}$ of size at most k, containing for each cached page the index at which it was last served. That is, when serving the t-th request, we remove some old request index t' from the cache and insert t. Thus, if t' is such that $\sigma_t = \sigma_{t'}$, this operation is free; otherwise, it has unit cost. In the sequel, we will also refer to these indices t as *page requests*.

In the learning-augmented online caching problem, the t-th page request comes with a prediction h_t for the next time page σ_t is requested. That is, at the time of the t-th request, our algorithm receives the pair (σ_t, h_t). Let $h = (h_1, \ldots, h_n)$ be the tuple of all n predictions. To define a notion of loss, let y_t denote for each t the next time page t is actually requested, with $y_t = n + 1$ if page σ_t is never requested again. The ℓ_1 loss is then defined to be

$$\eta(\sigma, h) = \sum_t |h_t - y_t|.$$

[2] Namely, their algorithms expect predictions to be in a different form: They expect predictions to be cache states (i.e., the set of pages in the cache at time t) rather than next arrival times of pages. Moreover, there exist sequences of "corresponding" inputs for each of these two models such that the predictor error approaches infinity in one model while remaining constant in the other.

[3] Observe that this formulation is equivalent to the "standard" formulation of online caching, where each cache miss has unit cost, up to an additive constant of k.

We will omit arguments to η if the context is clear. Note that if $\eta(\sigma, h) = 0$, then the offline optimal can be obtained, as the optimal algorithm always evicts the page that is next requested furthest into the future.

In stating our bounds, the essential quantity is often η/OPT. To make this clear, we will take $\varepsilon = \eta/\mathsf{OPT}$ and state our bounds in terms of ε in the sequel.

2.1.1 Inversions

Call a pair (i, j) of page requests an *inversion* if $y_i < y_j$ but $h_i \geq h_j$. Let $M(\sigma, h)$ denote the total number of inversions between the pair of sequences σ and h. We will omit arguments to M when the context is clear.

2.1.2 BlindOracle

We formally define the BlindOracle algorithm as follows: For each page request, if the requested page is already in the cache, do nothing. Otherwise, evict the page request p whose predicted next arrival time h_p is furthest into the future, with ties broken consistently (e.g., by always evicting the least recently used page among those with maximal h_p).

2.2 Combining Online Algorithms Competitively

To define our algorithms, we will need two classical results on competitively combining online algorithms online, due to Fiat et al. [10] and Blum and Burch [5], respectively. This type of "black-box" combination was also considered by Lykouris and Vassilvitskii [18], but their approach has a worse constant than that of Fiat et al. [10]. We also note that results of a similar flavor are proven by Purohit et al. [24] and Mahdian et al. [19], but for other online problems.

The question of combining multiple online algorithms while remaining competitive against each was first considered in the seminal paper of Fiat et al. [10]. They consider combining n online algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_n$ for the online caching problem into a single algorithm $\mathcal{B}$ such that $\mathcal{B}$ is C_i-competitive against $\mathcal{B}_i$ for each i. They show such an $\mathcal{B}$ is achievable if and only if

$$\sum_{i=1}^{n} \frac{1}{C_i} \leq 1.$$

We will need only the special case of $n = 2$ and $C_1 = C_2 = 2$, which we state below:

▶ **Theorem 2.1** ([10], special case)**.** *Given two algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$ for the online caching problem, there exists an algorithm $\mathcal{B}$ such that*

$$\mathsf{ALG}_{\mathcal{B}}(\sigma) \leq 2\min(\mathsf{ALG}_{\mathcal{B}_1}(\sigma), \mathsf{ALG}_{\mathcal{B}_2}(\sigma)) + O(1).$$

Moreover, if $\mathcal{B}_1$ and $\mathcal{B}_2$ are deterministic, then so is $\mathcal{B}$.

We note that this can be done deterministically with a "follow-the-leader" approach, where we simulate both algorithms and at each step evict any page that is not in the cache of the better performing algorithm (as measured by total number of evictions after serving the current request).

Blum and Burch [5] show that one can obtain a better approximation factor using a *randomized* scheme, namely multiplicative weights.[4] That is, at each point in time, the probability that the combined algorithm is following one of the n algorithms is given by a probability distribution over the n algorithms governed by the multiplicative weights update rule. For $n = 2$, their result can be stated as follows:

▶ **Theorem 2.2** ([5], special case). *Given two algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$ for the online caching problem and any γ such that $0 < \gamma < 1/4$, there exists an algorithm $\mathcal{B}$ such that*

$$\mathsf{ALG}_{\mathcal{B}}(\sigma) \leq (1+\gamma)\min(\mathsf{ALG}_{\mathcal{B}_1}(\sigma), \mathsf{ALG}_{\mathcal{B}_2}(\sigma)) + O(\gamma^{-1}k).$$

▶ Remark. Although we do not state the versions of these results for combining several algorithms, one can imagine that they could be useful if one wishes to ensemble multiple machine-learned predictors.

2.3 From ℓ_1 Loss to Inversions

We now state a lemma of Rohatgi [25] that relates ℓ_1 loss to the number of inversions, letting us lower bound the ℓ_1 loss $\eta(\sigma, h)$ by lower bounding the number of inversions $M(\sigma, h)$. Thus, instead of reasoning in terms of ℓ_1 loss, we will reason in terms of inversions.

▶ **Lemma 2.3** ([25, Lemma 11]). *For any σ and h, $\eta(\sigma, h) \geq \frac{1}{2}M(\sigma, h)$.*

With this lemma, it suffices (up to a factor of 2) to give our competitive ratio upper bounds in terms of the number of inversions M.

3 Upper Bounds

3.1 A First Analysis of BlindOracle

In this section, we give a first analysis of BlindOracle, showing that it gets very good performance when the ratio $\varepsilon = \eta/\mathsf{OPT}$ is very small. In particular, our analysis shows that as $\varepsilon \to 0$, the competitive ratio achieved approaches 1.

Let $\mathcal{A}$ be the offline optimal algorithm (i.e., such that $\mathsf{ALG}_{\mathcal{A}} = \mathsf{OPT}$). Let $\mathcal{B}$ be BlindOracle. Note that we can think of each of $\mathsf{ALG}_{\mathcal{A}}$, $\mathsf{ALG}_{\mathcal{B}}$, and M as functions of the time t, i.e., they are the cost of $\mathcal{A}$, the cost of $\mathcal{B}$, and the number of inversions, respectively, on the prefix consisting of the first $t-1$ requests.[5] We use the Δ operator to denote the change (in a function of t) from time t to time $t+1$. For example, $\Delta\mathsf{ALG}_{\mathcal{A}} = 1$ if $\mathsf{ALG}_{\mathcal{A}}$ evicts an element upon the t-th request.

In our analysis, we maintain a matching X_t between A_t and B_t at all times t. Call a matching *valid* if it consists only of pairs $(a, b) \in A_t \times B_t$ such that the next arrival of b is no later than the next arrival of a. Our matching $X_t \subseteq A_t \times B_t$ will be valid throughout the execution of the algorithm.

We now proceed with a potential function analysis, taking our potential Φ (as a function of A_t, B_t, and X_t) to be the number of unmatched pages in B_t. For notational simplicity, we denote $\Phi(A_t, B_t, X_t)$ by $\Phi(t)$. Given this setup, we show:

[4] The result of Blum and Burch [5] in fact holds more generally for all metrical task systems.
[5] This indexing is to be consistent with the definitions of A_t and B_t.

▶ **Proposition 3.1.** *There exists a valid matching X_n such that*

$$\mathsf{ALG}_{\mathcal{B}} + \Phi(n) \leq \mathsf{OPT} + M.$$

Proof. We induct on the length n of the input and perform a case analysis to show that we can maintain a valid matching X_t such that at each time step, the right-hand side increases at least as much as the left-hand side, i.e., $\Delta\mathsf{ALG}_{\mathcal{B}} + \Delta\Phi \leq \Delta\mathsf{OPT} + \Delta M$.

For our base case, note that $A_1 = B_1$, so we may take X_1 to be the identity matching.

Now, upon a request at time t, we update X_t according to the following cases (and with the consequences listed for each case):

1. The requested page p is in both A_t and B_t.
   a. The cached pages are matched to each other.
      - Do nothing.
   b. Otherwise:
      i. Both cached pages are matched.
         - Remove the pairs (c, p) and (p, d) from X_t.
         - Add the pairs (p, p) and (c, d) to X_t.
         - As a result:
            - Nothing changes. (For conciseness, we omit obviously unchanged quantities here and for the remainder of this proof.)
      ii. Otherwise:
         - Remove any pairs involving p from X_t. (There is at most one such pair.)
         - Add the pair (p, p) to X_t.
         - As a result:
            - $\Delta\Phi \leq 0$.
2. The requested page p is in B_t only.
   - Remove any pairs involving the evicted page a from X_t. (There is at most one such pair.)
   - Remove any pairs involving the requested page p from X_t. (There is at most one such pair.)
   - Add the pair (p, p) to X_t.
   - As a result:
      - $\Delta\mathsf{OPT} = 1$.
      - $\Delta\Phi \leq 1$.
3. The requested page p is in A_t only.
   a. The evicted page $b \in B_t$ is unmatched.
      - Add the pair (p, p) to X_t. (The arriving page $p \in A_t$ cannot be in any valid matching.)
      - As a result:
         - $\Delta\mathsf{ALG}_{\mathcal{B}} = 1$.
         - $\Delta\Phi = -1$.
   b. The evicted page $b \in B_t$ is matched.
      i. b arrives later than all unmatched pages in B_t.
         - Remove the pair (c, b) involving the evicted page b from X_t.
         - Add the pair (c, b') to X_t, where $b' \in B_t$ is any unmatched page.
         - Add the pair (p, p) to X_t. (The arriving page $p \in A_t$ cannot be in any valid matching.)
         - As a result:
            - $\Delta\mathsf{ALG}_{\mathcal{B}} = 1$.
            - $\Delta\Phi = -1$.

   - **ii.** There is an unmatched page $b' \in B_t$ arriving later than b.
     - Remove the pair (c, b) involving the evicted page b from X_t.
     - Add the pair (p, p) to X_t. (The arriving page $p \in A_t$ cannot be in any valid matching.)
     - As a result:
       - $\Delta\mathsf{ALG}_{\mathcal{B}} = 1$.
       - $\Delta\Phi = 0$.
       - $\Delta M = 1$, as there is an inversion between b and b'. (Note that we do not count this inversion ever again, as b gets evicted.)

4. The requested page p is in neither A_t nor B_t.
   - **a.** $\mathcal{A}$ evicts an unmatched page $a \in A_t$.
     - **i.** $\mathcal{B}$ evicts an unmatched page $b \in B_t$.
       - Add the pair (p, p) to X_t.
       - As a result:
         - $\Delta\mathsf{OPT} = 1$.
         - $\Delta\mathsf{ALG}_{\mathcal{B}} = 1$.
         - $\Delta\Phi = 1$.
     - **ii.** $\mathcal{B}$ evicts a matched page $b \in B_t$.
       - Remove the pair (c, b) involving b from X_t.
       - Add the pair (p, p) to X_t.
       - As a result:
         - $\Delta\mathsf{OPT} = 1$.
         - $\Delta\mathsf{ALG}_{\mathcal{B}} = 1$.
   - **b.** $\mathcal{A}$ evicts a matched page $a \in A_t$.
     - **i.** $\mathcal{B}$ evicts an unmatched page $b \in B_t$.
       - Remove the pair (a, d) involving a from X_t.
       - Add the pair (p, p) to X_t.
       - As a result:
         - $\Delta\mathsf{OPT} = 1$.
         - $\Delta\mathsf{ALG}_{\mathcal{B}} = 1$.
     - **ii.** $\mathcal{B}$ evicts a matched page $b \in B_t$.
       - Remove the pair (a, d) involving a from X_t.
       - Remove the pair (c, b) involving b from X_t.
       - Add the pair (p, p) to X_t.
       - As a result:
         - $\Delta\mathsf{OPT} = 1$.
         - $\Delta\mathsf{ALG}_{\mathcal{B}} = 1$.
         - Note that either b arrives after d, in which case we can add (c, d) to X_t and $\Delta\Phi = 0$, or the pair (b, d) forms an inversion, in which case $\Delta\Phi = 1$ and $\Delta M = 1$. (As before, since b is getting evicted, we will not count this pair twice.)

It is not hard to verify that the change in the left-hand side of the bound is no more than the change in the right-hand side in each of the cases listed above, from which the proposition follows. ◀

▶ **Proposition 3.2.** *The competitive ratio of algorithm $\mathcal{B}$ is at most $1 + 2\varepsilon$.*

Proof. Note that 2η is bounded below by the number of inversions M of (σ, h) by Lemma 2.3. By Proposition 3.1, $\mathsf{ALG}_{\mathcal{A}} \leq \mathsf{OPT} + M$, so $\mathsf{ALG}_{\mathcal{A}}/\mathsf{OPT} \leq 1 + M/\mathsf{OPT} \leq 1 + 2\varepsilon$. ◀

3.2 A More Careful Analysis

We now give an asymptotically better (in k) bound for the performance of BlindOracle. A more careful analysis is needed to show an upper bound with a $1/k$ coefficient on the ratio $\varepsilon = \eta/\mathsf{OPT}$. We use the same high-level approach for the proof as before, but with a more complicated potential function. Again, $\mathcal{A}$ is the offline optimal algorithm and $\mathcal{B}$ is the BlindOracle algorithm, and also as before, we use Δ to denote change (in functions of t) from request t to request $t+1$.

We maintain in this proof a matching X_t over pairs of page requests $(a, b) \in A_t \times B_t$ such that $h_a \geq h_b$ for each time step t. (Recall that h_a is the prediction for the next time that page σ_a is requested.) We also require that X_t restricts to the identity matching on $A_t \cap B_t$. That is, if a page p is in both caches at time t, then it is matched to itself in X_t. If X_t satisfies both of these properties, we say that it is *allowable*. Our potential function Φ – which will really be a sum of three separate potential functions – will be a function of A_t, B_t, and an allowable matching X_t. For notational simplicity, we denote $\Phi(A_t, B_t, X_t)$ by $\Phi(t)$.

Given A_t, B_t, and X_t at time t, define $\Phi_0(t)$ to be the number of $b \in B_t$ that are unmatched. Define $\Phi_1(t)$ to be the number of $b \in B_t$ such that $(b, b) \notin X_t$. In other words, Φ_1 counts how many page requests in B_t are not matched to the *same* page request in A_t. For a page request p, let $z_p(t)$ be the number of pages in B_t predicted to appear before h_p (with pages predicted to appear at the same time tie-broken in a consistent manner, e.g., by the last time each page was requested[6]). Next, define

$$\Phi_2(t) = \sum_{(a,b)\in X_t} \big(z_b(t) - z_a(t)\big).$$

With these "sub"-potential functions defined, we take

$$\Phi(t) = (k-1)\Phi_0(t) + (k-1)\Phi_1(t) + \Phi_2(t)$$

as our overall potential function. This completes our setup for the analysis. But before we give the details of the proof, we make a few observations about these definitions and discuss some intuition.

First, observe that for any p, we have $0 \leq z_p(t) \leq k$. And if $p \in B_t$, then $z_p(t) \leq k-1$ because there are only $k-1$ other pages that could be predicted to appear before. Next, if $(a, b) \in X_t$, then $z_a(t) \geq z_b(t)$ as $h_a \geq h_b$. Hence $\Phi_2(t) \leq 0$ always. As a result, Φ may sometimes be negative, but it satisfies $|\Phi(t)| \leq 3k^2$ for all t. Finally, we note that Φ_2 is equivalently a sum over the $(a, b) \in X_t$ such that $a \neq b$.

In terms of intuition for these definitions, the main substance perhaps lies in Φ_2: Suppose $(a, b) \in X_t$ and $a \neq b$; in particular, $a \notin B_t$. The quantity $z_a(t) - z_b(t)$ is the future number of inversions that we are "guaranteed" from this pair (a, b). If we request σ_p for any p such that $h_b < h_p < h_a$, then we have an inversion (p, b). On the other hand, if σ_p is not requested before the next time σ_a is requested, then the request for σ_a forms an inversion (a, p). Thus, the quantity $M - \Phi_2$ accounts not only for the inversions already encountered, but also for future inversions. The matching of a and b also serves another purpose: We couple the decrease of z_a to the decrease of z_b whenever z_a decreases without generating an inversion. Therefore, if z_a is small when σ_a is next requested, then $z_b \leq z_a$ is also small. It follows that if we form a new pair (a', b), then $z_{a'} - z_b$ will be comparatively large, i.e., we get many future inversions.

[6] Alternatively, one can simply perturb all predictions by infinitesimal amounts so that there are no ties.

We execute on this intuition to prove the following proposition:

▶ **Proposition 3.3.** *For any input (σ, h), there exists a matching $X_n \subseteq A_n \times B_n$ consisting only of pairs (a, b) satisfying $h_a \geq h_b$ such that*

$$(k-1)\mathsf{ALG}_{\mathcal{B}} + \Phi(n) \leq 2k \cdot \mathsf{OPT} + 2M.$$

Proof. Like in the proof of Proposition 3.1, we induct on the length of the input and perform a case analysis to show that we can maintain a allowable matching X_t such that the right-hand side increases at least as much as the left-hand side for each page request.

We split the serving of each page request into two phases:

1. **Matching.** Update X_t so that the page requests in A_t and B_t that are to be removed are unmatched. (Note that page requests are removed either because the corresponding page was requested again or because the corresponding page was evicted.)
2. **Updating.** Replace a page request from each of A_t and B_t with the new request and add the new page request pair (t, t) into X_t.

In doing so, we ensure that the pages to be removed are unmatched by the time updating occurs, so they do not contribute to Φ_2.

We first analyze how updating affects the overall potential Φ. This operation decreases each of Φ_0 and Φ_1 by 1, since we remove an unmatched pair and replace it with the matched pair (t, t). For Φ_2, observe that for a matched pair (a, b), the difference $z_b - z_a$ increases on the t-th request only if a page predicted to arrive between b and p is removed. This occurs if and only if there is a $p \in B_t$ such that $\sigma_p = \sigma_t$ and $h_b < h_p < h_a$. In this case, the pair (p, b) forms an inversion, i.e., it increases M by 1. Any inversion (p, b) is counted at most once this way because p is getting removed from B_t. Thus $\Delta\Phi_2 \leq \Delta M$. In summary, updating leaves us with an extra $2(k-1)$ in potential due to the changes in Φ_0 and Φ_1; the change in potential of Φ_2 is offset by the change in one of the two M terms on the right-hand side.

We now specify the matching phase in greater detail and analyze it using a case analysis. We will show

$$(k-1)\Delta\mathsf{ALG}_{\mathcal{B}} + \Delta\Phi \leq 2k \cdot \Delta\mathsf{OPT} + \Delta M + 2(k-1),$$

with the extra $2(k-1)$ being covered by the updating step. The analysis proceeds as follows:

1. The requested page is in both A_t and B_t.
   - The previous page requests for σ_t in A_t and B_t are matched to each other by assumption, so we can just unmatch them.
   - As a result:
     - $(k-1)\Delta\Phi_0 = k-1$.
     - $(k-1)\Delta\Phi_1 = k-1$.
2. The requested page is in A_t only.
   a. The previous request $p \in A_t$ for the requested page is matched as (p, d) and the page request $b \in B_t$ evicted by $\mathcal{B}$ is matched as (c, b).
      - Unmatch (p, d) and (c, b) and match (c, d). The latter is allowable because $h_c \geq h_b \geq h_d$.
      - As a result:
        - $(k-1)\Delta\mathsf{ALG}_{\mathcal{B}} = k-1$.
        - $(k-1)\Delta\Phi_0 = k-1$.
        - $(k-1)\Delta\Phi_1 \leq k-1$, since $p \neq d$.
        - $\Delta\Phi_2 = z_p - (k-1)$, since $z_b = k-1$.
        - $\Delta M \geq z_p$, since the arrival of σ_p generates z_p inversions of the form (p, b') for all $b' \in B_t$ such that $h_p > h_{b'}$.

b. The previous request $p \in A_t$ for the requested page is matched as (p, d) and the page request $b \in B_t$ evicted by $\mathcal{B}$ is unmatched.
- Unmatch (p, d).
- As a result:
  - $(k-1)\Delta\mathsf{ALG}_{\mathcal{B}} = k-1$.
  - $(k-1)\Delta\Phi_0 = k-1$.
  - $(k-1)\Delta\Phi_1 = 0$, since $p \neq d$.
  - $\Delta\Phi_2 = z_p - z_b \leq z_p$.
  - $\Delta M \geq z_p$, since the arrival of σ_p generates z_p inversions of the form (p, b') for all $b' \in B_t$ such that $h_p \geq h_{b'}$.

c. The previous request $p \in A_t$ for the requested page is unmatched and the page request $b \in B_t$ evicted by $\mathcal{B}$ is matched as (c, b).
- Unmatch (c, b) and match (c, d) for an arbitrary unmatched $d \in B_t \setminus \{b\}$. This is allowable because $h_c \geq h_b \geq h_d$.
- As a result:
  - $(k-1)\Delta\mathsf{ALG}_{\mathcal{B}} = k-1$.
  - $(k-1)\Delta\Phi_0 = 0$.
  - $(k-1)\Delta\Phi_1 \leq k-1$.
  - $\Delta\Phi_2 = z_d - (k-1) \leq 0$, since $z_b = k-1$.

d. The previous request $p \in A_t$ for the requested page is unmatched and the page request $b \in B_t$ evicted by $\mathcal{B}$ is unmatched.
- Do nothing.
- As a result:
  - $(k-1)\Delta\mathsf{ALG}_{\mathcal{B}} = k-1$.

3. The requested page is in B_t only.

a. The previous request $p \in B_t$ for the requested page is matched as (c, p) and the page request $a \in A_t$ evicted by $\mathcal{A}$ is matched as (a, d).
- Unmatch (c, p) and (a, d).
- As a result:
  - $2k \cdot \Delta\mathsf{OPT} = 2k$.
  - $(k-1)\Delta\Phi_0 = 2(k-1)$.
  - If $a = d$:
    - $(k-1)\Delta\Phi_1 = k-1$, since $c \neq p$.
    - $\Delta\Phi_2 = z_c - z_p \leq k$, since $z_c \leq k$.
  - Otherwise, if $a \neq d$:
    - $(k-1)\Delta\Phi_1 = 0$, since $c \neq p$.
    - $\Delta\Phi_2 = (z_c - z_p) + (z_a - z_d) \leq 2k$, since $z_c, z_a \leq k$.

b. The previous request $p \in B_t$ for the requested page is matched as (c, p) and the page request $a \in A_t$ evicted by $\mathcal{A}$ is unmatched.
- Unmatch (c, p).
- As a result:
  - $2k \cdot \Delta\mathsf{OPT} = 2k$.
  - $(k-1)\Delta\Phi_0 = k-1$.
  - $(k-1)\Delta\Phi_1 = 0$, since $c \neq p$.
  - $\Delta\Phi_2 \leq z_c - z_p \leq k$, since $z_c \leq k$.

c. The previous request $p \in B_t$ for the requested page is unmatched and the page request $a \in A_t$ evicted by $\mathcal{A}$ is matched as (a, d).
- Unmatch (a, d).

- As a result:
  - $2k \cdot \Delta\mathsf{OPT} = 2k$.
  - $(k-1)\Delta\Phi_0 = k-1$.
  - $(k-1)\Delta\Phi_1 \leq k-1$.
  - $\Delta\Phi_2 \leq z_a - z_d \leq k$, since $z_a \leq k$.

d. The previous request $p \in B_t$ for the requested page is unmatched and the page request $a \in A_t$ evicted by $\mathcal{A}$ is unmatched.

- Do nothing.
- As a result:
  - $2k \cdot \Delta\mathsf{OPT} = 2k$.

4. The requested page is in neither A_t nor B_t.

a. The previous request $a \in A_t$ evicted by $\mathcal{A}$ is matched as (a, d) and the page request $b \in B_t$ evicted by $\mathcal{B}$ is matched as (c, b).

- If $(a, d) = (c, b)$, simply unmatch (a, d) (and thus (c, b)). Otherwise, unmatch (a, d) and (c, b) and match (c, d). The latter is allowable because $h_c \geq h_b \geq h_d$.
- As a result:
  - $2k \cdot \Delta\mathsf{OPT} = 2k$.
  - $(k-1)\Delta\mathsf{ALG}_{\mathcal{B}} = k-1$.
  - $(k-1)\Delta\Phi_0 = k-1$.
  - $(k-1)\Delta\Phi_1 \leq 2(k-1)$.
  - $\Delta\Phi_2 = z_a - (k-1) \leq 0$, since $z_b = k-1$.

b. The previous request $a \in A_t$ evicted by $\mathcal{A}$ is matched as (a, d) and the page request $b \in B_t$ evicted by $\mathcal{B}$ is unmatched.

- Unmatch (a, d).
- As a result:
  - $2k \cdot \Delta\mathsf{OPT} = 2k$.
  - $(k-1)\Delta\mathsf{ALG}_{\mathcal{B}} = k-1$.
  - $(k-1)\Delta\Phi_0 = k-1$.
  - $(k-1)\Delta\Phi_1 \leq k-1$.
  - $\Delta\Phi_2 = z_a - z_d \leq k$, since $z_a \leq k$.

c. The previous request $a \in A_t$ evicted by $\mathcal{A}$ is unmatched and the page request $b \in B_t$ evicted by $\mathcal{B}$ is matched as (c, b).

- Unmatch (c, b) and match (c, d) for an arbitrary unmatched $d \in B_t \setminus \{b\}$. This is allowable because $h_c \geq h_b \geq h_d$.
- As a result:
  - $2k \cdot \Delta\mathsf{OPT} = 2k$.
  - $(k-1)\Delta\mathsf{ALG}_{\mathcal{B}} = k-1$.
  - $(k-1)\Delta\Phi_0 = 0$.
  - $(k-1)\Delta\Phi_1 \leq k-1$.
  - $\Delta\Phi_2 = z_d - (k-1) \leq 0$, since $z_b = k-1$.

d. The previous request $a \in A_t$ evicted by $\mathcal{A}$ is unmatched and the page request $b \in B_t$ evicted by $\mathcal{B}$ is unmatched.

- Do nothing.
- As a result:
  - $2k \cdot \Delta\mathsf{OPT} = 2k$.
  - $(k-1)\Delta\mathsf{ALG}_{\mathcal{B}} = k-1$.

Recall that we have an extra $2(k-1)$ in potential from the updating phase that we can use to defray the costs of the matching phase. One can verify that this is sufficient for all of the cases described above – the "tight" cases are 1, 2(a), 2(b), 2(c), 3(a), and 4(a). The proposition now follows. ◀

▶ **Proposition 3.4.** *The competitive ratio of algorithm $\mathcal{B}$ is at most $2+2/(k-1)+4\varepsilon/(k-1)$.*

Proof. Compose Proposition 3.3 with Lemma 2.3. In particular, we have

$$\frac{\mathsf{ALG}_{\mathcal{B}}}{\mathsf{OPT}} + \frac{O(k)}{\mathsf{OPT}} \leq \frac{2k}{k-1} + \frac{2M}{(k-1)\mathsf{OPT}} \leq 2 + \frac{2}{k-1} + \frac{4}{k-1}\frac{\eta}{\mathsf{OPT}}.$$ ◀

3.3 Finishing Up

The results stated in Section 1 now follow easily from what we have already shown:

▶ **Theorem 1.1.** *For learning-augmented online caching, BlindOracle obtains a competitive ratio of*

$$\min\left(1 + 2\frac{\eta}{\mathsf{OPT}}, 4 + \frac{4}{k-1}\frac{\eta}{\mathsf{OPT}}\right),$$

where η is the ℓ_1 loss incurred by the predictor and OPT is the optimal offline cost. (For precise definitions, see Section 2.1.)

Proof. From the analysis of the previous two sections, the desired bound immediately follows from taking the minimum of the bounds in Propositions 3.2 and 3.4, noting that $2/(k-1) \leq 2$ when $k \geq 2$. ◀

▶ **Corollary 1.2.** *There exists a deterministic algorithm for learning-augmented online caching that achieves a competitive ratio of*

$$2\min\left(\min\left(1 + 2\frac{\eta}{\mathsf{OPT}}, 4 + \frac{4}{k-1}\frac{\eta}{\mathsf{OPT}}\right), k\right).$$

Proof. Combine BlindOracle with LRU using the "combiner" from Theorem 2.1, with the performance of BlindOracle being bounded by Theorem 1.1. ◀

▶ **Corollary 1.3.** *There exists a randomized algorithm for learning-augmented online caching that achieves a competitive ratio of*

$$(1+\gamma)\min\left(\min\left(1 + 2\frac{\eta}{\mathsf{OPT}}, 4 + \frac{4}{k-1}\frac{\eta}{\mathsf{OPT}}\right), H_k\right)$$

for any $\gamma \in (0, 1/4)$.[7] *(Here, $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} = \ln(k) + O(1)$ is the k-th harmonic number.)*

Proof. Like in the proof above, combine BlindOracle with algorithm Equitable of Achlioptas et al. [1][8], this time using the "combiner" from Theorem 2.2. ◀

[7] The trade-off in γ and the cost is additive; thus, it does not factor into the competitive ratio.

[8] We use Equitable because it achieves the optimal worst-case competitive ratio of H_k for online caching; Marker has a competitive ratio of $2H_k - 1$ [1].

4 Lower Bound for Deterministic Algorithms

We now show that combining BlindOracle with LRU achieves an optimal competitive ratio bound (in terms of k and $\varepsilon = \eta/\mathsf{OPT}$) among all deterministic algorithms for learning-augmented online caching by proving Theorem 1.4:

▶ **Theorem 1.4.** *The competitive ratio bound for any deterministic learning-augmented online caching algorithm must be at least*

$$1 + \Omega\left(\min\left(\frac{1}{k}\frac{\eta}{\mathsf{OPT}}, k\right)\right)$$

in terms of k and η/OPT.

Proof. Fix a $k > 2$, and let $\mathcal{A}$ be any deterministic algorithm for learning-augmented online caching. We show that for any even integer $2j$ such that $1 < 2j < k$, there exists a sequence of inputs with $\mathsf{OPT}(\sigma)$ growing arbitrarily large that satisfies $\varepsilon/k = 2j$ and

$$\frac{\mathsf{ALG}_{\mathcal{A}}(\sigma, h)}{\mathsf{OPT}(\sigma)} \geq 1 + \frac{\varepsilon}{4k}$$

for all (σ, h) in the sequence. As $2j$ nears k, this lower bound on competitive ratio approaches $1 + k/4$, so this lower bound binds up to a competitive ratio of $\Omega(k)$, thus yielding the stated result.

We spend the remainder of this proof constructing such inputs (σ, h). Let $P_1, \ldots, P_k, Q$ be $k + 1$ distinct pages. We make the following sequence of requests, which we call a *phase*:

1. For $i = 1, \ldots, k - 1$:
   a. Request pages $P_1, \ldots, P_k$ in order, predicting each page to next appear k requests from now.
2. Request pages $P_1, \ldots, P_k$ in order, predicting each page to next appear $k + j + 2$ requests from now.
3. Request page Q and predict that it will next appear $j + 1$ requests from now.
4. For $i = 1, \ldots, j$:
   a. Request the page evicted by $\mathcal{A}$ during the previous request if it exists. Otherwise, request an arbitrary page. For each page, provide the same prediction as the last time this page was requested.
5. Request page Q and predict that it will next appear $k^2 + 1 + z$ requests from now, for some $z \geq 0$ that we will specify below.

As an outer loop, we can repeat the above "phase" as many times as needed.

In a single phase, observe that OPT makes at most two evictions: One to evict Q if it is in the cache at the beginning of the phase and one upon the arrival of Q in step (3). For the latter, it suffices to evict a page that does not appear among the at most $j + 1 \leq k$ pages requested in steps (4) and (5). Note also that OPT makes at least one eviction in any phase, since each phase involves serving $k + 1$ distinct pages.

On the other hand, I claim $\mathcal{A}$ makes at least $j + 1$ evictions. First, if the cache of $\mathcal{A}$ at the end of step (2) does not consist of pages $P_1, \ldots, P_k$, then $\mathcal{A}$ must have incurred cost at least $k \geq j + 1$ during steps (1) and (2). Thus, we may assume that $\mathcal{A}$'s cache consists exactly of pages $P_1, \ldots, P_k$ at the end of step (2). If so, $\mathcal{A}$ has to evict a page for each of the $j + 1$ requests made in steps (3) and (4).

Finally, observe that all the predictions are accurate except those made for the pages that arrive in step (4) and the prediction in step (5). I claim that each prediction for a page arriving in step (4) is off by at most $k + j < 2k$. Indeed, a request to Q will be off by

at most j, as Q is requested again in step (5). And a request to P_ℓ will be off by at most $k + j$, as either page P_ℓ appears in step (1) of the next phase or the sequence of requests terminates. To bound the error for the former, note that P_ℓ will appear at most k requests into step (1) of the next phase, which starts j requests after the first request of step (4). For the prediction made in step (5), note that it is off from the ground truth by exactly z.

Putting everything together, we have that $1 \leq \mathsf{OPT} \leq 2$ and $\eta \leq j \cdot 2k + z$. In fact, we can set z so that $\eta = 2jk \cdot \mathsf{OPT}$ exactly. In this case,

$$\mathsf{ALG}_{\mathcal{A}} \geq j + 1 = 2 + \frac{1}{2}\left(1 - \frac{1}{j}\right) \cdot 2j \geq \mathsf{OPT} + \frac{1}{4}\frac{\eta}{k},$$

with $\varepsilon/k = 2j$. To finish, notice that we can make OPT arbitrarily large by repeating the "phase" defined above multiple times in sequence; the same analysis holds. ◀

5 Conclusion

In this paper, we show that the simple approach of combining BlindOracle with competitive algorithms for online caching surprisingly achieves state-of-the-art performance for learning-augmented online caching. We additionally show that combining BlindOracle with LRU is optimal among deterministic algorithms for learning-augmented online caching. An open question is whether the bounds we achieve can be improved using randomization to match the lower bound of Rohatgi [25].

References

1. Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.
2. Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *CoRR*, abs/2003.02144, 2020. To appear at ICML 2020. `arXiv:2003.02144`.
3. Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012.
4. Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. *SIAM J. Comput.*, 41(2):391–414, 2012.
5. Avrim Blum and Carl Burch. On-line learning and the metrical task system problem. *Mach. Learn.*, 39(1):35–58, 2000.
6. Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
7. Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2):19:1–19:34, 2017.
8. Sébastien Bubeck and Aleksandrs Slivkins. The best of both worlds: Stochastic and adversarial bandits. In *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, pages 42.1–42.23, 2012.
9. Yihe Dong, Piotr Indyk, Ilya P. Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.
10. Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.
11. Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 2319–2327, 2019.

12 Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

13 Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. *SIAM J. Comput.*, 30(1):300–317, 2000.

14 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 489–504, 2018.

15 Ravi Kumar, Manish Purohit, Aaron Schild, Zoya Svitkina, and Erik Vee. Semi-online bipartite matching. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 50:1–50:20, 2019.

16 Ravi Kumar, Manish Purohit, Zoya Svitkina, and Erik Vee. Interleaved caching with access graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1846–1858, 2020.

17 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877, 2020.

18 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3302–3311, 2018.

19 Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Online optimization with uncertain information. *ACM Trans. Algorithms*, 8(1):2:1–2:29, 2012.

20 Andres Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1858–1866, 2017.

21 Vahab S. Mirrokni, Shayan Oveis Gharan, and Morteza Zadimoghaddam. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1690–1701, 2012.

22 Michael Mitzenmacher. 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 2018, 3-8 December 2018, Montréal, Canada*, pages 462–471, 2018.

23 Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, pages 14:1–14:18, 2020.

24 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 9684–9693, 2018.

25 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834–1845, 2020.

26 Tim Roughgarden. Beyond worst-case analysis. *Commun. ACM*, 62(3):88–96, 2019.

27 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.