

Learning-Augmented Weighted Paging

Nikhil Bansal* Christian Coester† Ravi Kumar‡ Manish Purohit‡ Erik Vee‡

Abstract

We consider a natural semi-online model for weighted paging, where at any time the algorithm is given predictions, possibly with errors, about the next arrival of each page. The model is inspired by Belady’s classic optimal offline algorithm for unweighted paging, and extends the recently studied model for learning-augmented paging [45, 50, 52] to the weighted setting.

For the case of perfect predictions, we provide an ℓ -competitive deterministic and an $O(\log \ell)$ -competitive randomized algorithm, where ℓ is the number of distinct weight classes. Both these bounds are tight, and imply an $O(\log W)$ - and $O(\log \log W)$ -competitive ratio, respectively, when the page weights lie between 1 and W . Previously, it was not known how to use these predictions in the weighted setting and only bounds of k and $O(\log k)$ were known, where k is the cache size. Our results also generalize to the interleaved paging setting and to the case of imperfect predictions, with the competitive ratios degrading smoothly from $O(\ell)$ and $O(\log \ell)$ to $O(k)$ and $O(\log k)$, respectively, as the prediction error increases.

Our results are based on several insights on structural properties of Belady’s algorithm and the sequence of page arrival predictions, and novel potential functions that incorporate these predictions. For the case of unweighted paging, the results imply a very simple potential function based proof of the optimality of Belady’s algorithm, which may be of independent interest.

1 Introduction

Paging is among the most classical and well-studied problems in online computation. Here, we are given a universe of n pages and a cache that can hold up to k pages. At each time step, some page is requested, and if it is not in the cache (called a *cache miss* or *page fault*), it must be fetched into the cache (possibly evicting some other page), incurring a unit cost. The goal of the algorithm is to minimize the total cost incurred. The problem is well understood through the lens of competitive analysis [51], with several optimal k -competitive deterministic and $O(\log k)$ -competitive randomized algorithms known for it [1, 30, 47]. A remarkable property of paging is that the offline optimum can be computed with rather limited knowledge of the future: only the *relative order* of the *next* request times for pages. In particular, Belady’s classic Farthest in Future (FiF) algorithm [16], which at any time greedily evicts the page whose next request is farthest in the future, gives the optimal solution.

A natural and well-studied generalization of paging is *weighted* paging, where each page p has an arbitrary fetching cost $w_p > 0$, and the goal is to minimize the total cost. Besides the practical motivation, weighted paging is very interesting theoretically as the phase-based analyses for unweighted paging do not work anymore (even if there are only two different weights), and as it is a stepping stone in the study of more general problems such as metrical task systems (MTS) [20] and the k -server problem¹ [46]. In fact, $O(\log k)$ -competitive randomized algorithms for weighted paging were obtained relatively recently, and required new techniques such as the primal-dual method [11, 13] and entropic regularization [24]. These ideas have been useful for various other problems and also for MTS and the k -server problem [10, 24, 23, 28].

Learning-augmented setting. Motivated by advances in machine learning, Lykouris and Vassilvitskii [45] recently introduced a new semi-online model where at each step, the algorithm has access to some, possibly erroneous, machine-learned advice about future requests and studied the paging problem in this model. Here, at each time t , along with the current page request we are also given the predicted arrival time for the next request of the same page. This can be viewed as generalizing the setting for Belady’s FiF algorithm to allow

*University of Michigan, Ann Arbor. bansal@gmail.com. The research was done while the author was at CWI and it was supported by the NWO VICI grant 639.023.812.

†Tel Aviv University, Israel. christian.coester@gmail.com. Christian Coester is supported by the Israel Academy of Sciences and Humanities & Council for Higher Education Excellence Fellowship Program for International Postdoctoral Researchers. Research was carried out while he was at CWI in Amsterdam and supported by the NWO VICI grant 639.023.812.

‡Google Research, Mountain View. ravi.k53@gmail.com, mpurohit@google.com, erikvee@google.com

¹Weighted paging is equivalent to the k -server problem on weighted star metrics.

incorrect predictions. They design an algorithm with competitive ratio $O(1)$ when the predictions are accurate, and which degrades smoothly as the prediction error increases, but never exceeds $O(\log k)$. These results have been subsequently refined and improved in [50, 52].

In this work, we study whether Belady's algorithm and the results in the learning-augmented setting for unweighted paging can be extended to the weighted case. Suppose each page weight is one of distinct values w_1, \dots, w_ℓ ; the pages are thus divided into ℓ disjoint *weight classes*. Then recent work by Jiang et al. [37] and Antoniadis et al. [6] shows that even with perfect predictions, any deterministic (resp., randomized) online algorithm must have competitive ratio $\Omega(\ell)$ (resp., $\Omega(\log \ell)$), provided $\ell \leq k$.² In particular, for $\ell \geq k$, predictions do not give any advantage.

As Belady's algorithm is 1-competitive for $\ell = 1$, this raises the natural question whether there are algorithms with guarantees that are only a function of ℓ , and *independent* of the cache size k . In typical scenarios ℓ is likely to be small and much less than k . Also if the weights range from 1 to W , then one can assume $\ell = O(\log W)$ by rounding them to powers of 2.

1.1 Prediction model and error We consider the following model for learning-augmented weighted paging. At each time $t = 1, \dots, T$, the algorithm receives a request to some page σ_t as well as a prediction $\tau_t \in \mathbb{N}$ for the next time after t when σ_t will be requested again. Let $a_t \in \mathbb{N}$ be the actual time when σ_t is next requested (or $a_t = T + 1$ if it is not requested again). In the unweighted setting of [45, 50, 52], the prediction error was defined as the ℓ_1 -distance between a and τ , which in the weighted case generalizes naturally to

$$\eta := \sum_t w_{\sigma_t} \cdot |\tau_t - a_t|.$$

We remark that although the predictions τ_t are for the arrival times, we use them only to get a relative ordering of pages within the same weight class by their next predicted arrival times.

We define the following more nuanced error measure that allows us to obtain tighter bounds. For any weight class i , we call a pair (s, t) of time steps an *inversion* if both σ_s and σ_t belong to weight class i and $a_s < a_t$ but $\tau_s \geq \tau_t$. Let $\epsilon_i(\sigma, \tau) := |\{s \in \mathbb{N} \mid \exists t \in \mathbb{N}: (s, t) \text{ is an inversion for weight class } i\}|$. In other words, $\epsilon_i(\sigma, \tau)$ is the number of *surprises* within class i , i.e., the number of times some page σ_s arrives although some other page σ_t of the same class was expected earlier. Let

$$\epsilon(\sigma, \tau) := \sum_i w_i \cdot \epsilon_i(\sigma, \tau).$$

We drop σ, τ from the notation when it is clear from context and bound the competitive ratio of our algorithms in terms of ϵ . Since $\epsilon \leq 2\eta$ [50, Lemma 4.1], our bounds hold for the η error measure as well. In fact, the relationship holds even if ϵ_i is defined as the *total number of inversions* within weight class i and thus our notion of ϵ can be significantly smaller than η (see [29] for an example where $\eta = \Omega(T) \cdot \epsilon$).

1.2 Our results We obtain algorithmic results for learning-augmented weighted paging, both for the case of perfect predictions and for predictions with error. Even though the latter setting generalizes the former, we describe the results separately as most of the key new ideas are already needed for perfect predictions. To the best of our knowledge, no bounds better than $O(k)$ and $O(\log k)$ were previously known even for the case of $\ell = 2$ weight classes with perfect predictions.

We first consider the deterministic and the randomized settings when the predictions are perfect.

THEOREM 1.1. *There is an ℓ -competitive deterministic algorithm for learning-augmented weighted paging with ℓ weight classes and perfect predictions.*

The competitive ratio is the best possible by the lower bound of [37] and is $O(\log W)$ if page weights lie in the range $[1, W]$. Also, notice that the algorithm is exactly ℓ -competitive; in particular, for $\ell = 1$ we have an optimal algorithm. Since $\ell = 1$ corresponds to the unweighted case, Theorem 1.1 can be viewed as generalizing Belady's FIF algorithm to the weighted case.

²The lower bounds in [37, 6] are stated in terms of k rather than ℓ , using a construction with $\ell \approx k$. The effective cache size can be reduced to ℓ by forcing $k - \ell$ pages to be in cache at all times.

Our algorithm is quite natural, and is based on a water-filling (primal-dual) type approach similar to that for the deterministic k -competitive algorithm for weighted paging due to Young [54]. Roughly speaking, the algorithm evicts from each weight class at a rate inversely proportional to its weight, and the evicted page is the one whose next arrival is (predicted) farthest in the future for that weight class. While the algorithm is natural, the analysis is based on a novel potential function that is designed to capture the next predicted requests for pages. The algorithm and its analysis are described in Appendix B.

Furthermore, for $\ell = 1$, this gives a new potential-function proof for the optimality of FiF. This new proof seems simpler and less subtle than the standard exchange argument and might be of independent interest; see Appendix A.

THEOREM 1.2. *There is an $O(\log \ell)$ -competitive randomized algorithm for learning-augmented weighted paging with ℓ weight classes and perfect predictions.*

The competitive ratio is the best possible [6, 37], and is $O(\log \log W)$ for page weights in the range $[1, W]$. This result is technically and conceptually the most interesting part of the paper and requires several new ideas.

The algorithm splits the cache space into ℓ parts, one for each weight class. Within each class, the cached pages are selected according to a ranking of pages that is induced by running several copies of Belady's FiF algorithm simultaneously for different cache sizes. The key question is how to maintain this split of the cache space over the ℓ classes dynamically over time. To get an $O(\log \ell)$ guarantee, we need to do some kind of a *multiplicative update* on each weight class, however there is no natural quantity on which to do this update³. The main idea is to carefully look at the structure of the predicted requests and the recent requests and use this to determine the rate of the multiplicative update for each class. We give a more detailed overview in Section 1.3.

Both our algorithm and its analysis are rather complicated and we leave the question of designing a simpler randomized algorithm as an interesting open question.

Prediction errors and robustness. The algorithms above also work for erroneous predictions, and their performance degrades smoothly as the prediction error increases. In particular, our deterministic algorithm has cost at most $\ell \cdot \text{OPT} + 2\ell\epsilon$, and our randomized algorithm has expected cost $O(\log \ell \cdot \text{OPT} + \ell\epsilon)$. (Recall ℓ is the number of weight classes, and ϵ is the weighted number of surprises.) Using standard techniques to combine online algorithms [18, 32, 6], together with the worst case k - and $O(\log k)$ -competitive deterministic and randomized algorithms for weighted paging, this gives the following results.

THEOREM 1.3. *There is an $O(\min\{\ell + \ell\epsilon/\text{OPT}, k\})$ -competitive deterministic algorithm for learning-augmented weighted paging.*

THEOREM 1.4. *There is an $O(\min\{\log \ell + \ell\epsilon/\text{OPT}, \log k\})$ -competitive randomized algorithm for learning-augmented weighted paging.*

Implications for interleaved caching. Our algorithms actually only require the relative order of pages *within* each weight class, and not how the requests from different classes are interleaved. Unweighted paging has also been studied in the *interleaved* model, where ℓ request sequences $\sigma^{(1)}, \dots, \sigma^{(\ell)}$ are given in advance and the adversary interleaves them arbitrarily. Here tight $\Theta(\ell)$ deterministic and $\Theta(\log \ell)$ randomized competitive algorithms are known [15, 26, 41]. Our results thus extend these results to the weighted setting, where each sequence has pages of a different weight.

1.3 Overview of techniques We now give a more detailed overview of our algorithms. We mainly focus on the case of perfect predictions, and briefly remark how to handle errors, towards the end. As we aim to obtain guarantees as a function of ℓ instead of k , the algorithm must consider dynamics at the level of weight classes in addition to that for individual pages. Our algorithms have two components: a *global strategy* and a *local strategy*. The global strategy decides at each time t how many cache slots to dedicate to each different weight class i , denoted $x_i(t)$. Since we have k cache slots in total, we maintain $\sum_i x_i(t) = k$ with $x_i(t) \geq 0$ for all i .⁴ The local strategy decides, for each weight class i , which $x_i(t)$ pages to keep in the cache.

Suppose page σ_t requested at time t belongs to weight class r , and σ_t is fetched as it is not in the cache. This increases $x_r(t)$, the number of pages of class r in cache, and the global strategy must decide how to decrease $x_i(t)$

³The standard weighted paging algorithm does a multiplicative update for each page, but this necessarily loses a factor of $\Omega(\log k)$.

⁴ The $x_i(t)$ may be fractional, but let us assume that they are integral for now.

for each class $i \neq r$ to maintain $\sum_i x_i(t) = k$. In the deterministic case, roughly, the global strategy simply decreases the $x_i(t)$ uniformly at rate $1/w_i$ (some care is needed to ensure that $x_i(t)$ are integral, and the idea is implemented using a water-filling approach), and the local strategy is Belady's FIF algorithm.

This suffices for a competitive ratio of ℓ , but to get an $O(\log \ell)$ bound in the randomized case, one needs more careful *multiplicative* updates for $x_i(t)$. However, it is not immediately clear how to do this and naively updating $x_i(t)$ in proportion to, for example $x_i(t)/w_i$ or $(k - x_i(t))/w_i$ (analogous to algorithms for standard weighted paging), does not work.

Update rule. A key intuition behind our update rule is the following example. Suppose that for each class i , the adversary repeatedly requests pages from some fixed set P_i , say in a cyclic order. Assuming $|P_i| \geq x_i$, we claim that the right thing to do is to update each x_i multiplicatively in proportion to $|P_i| - x_i$ (and inversely proportional to w_i). Indeed, if $|P_i|$ is already much larger than $x_i(t)$, the algorithm anyway has to pay a lot when the pages in P_i are requested, so it might as well evict more aggressively from class i to serve requests to pages from other classes. On the other hand, if $x_i(t)$ is close to $|P_i|$, then the algorithm should reduce $x_i(t)$ at a much slower rate, since it is already nearly correct.

The difficulty in implementing this idea is that the request sequence can be completely arbitrary, and there may be no well-defined *working set* P_i of requests for class i . Moreover, even if the requests have such structure, the set P_i could vary arbitrarily over time.

Our key conceptual and technical novelty is defining a suitable notion of P_i . The definition itself is somewhat intricate, but allows us to maintain a “memory” of recent requests by utilizing a subset of the real line. (A formal description appears in Section 3.) Our definition relies on a crucial notion of page ranks that we describe next.

Page ranks. Let us fix a weight class i , and consider the request sequence σ restricted to this class. We say that page p has *rank* m at time t if Belady's algorithm running on σ with a cache of size m contains p at time t , but an alternate version of Belady's algorithm with a cache of size $m - 1$ does not. The rank of pages changes over time, e.g., a requested page always moves to rank 1 in its weight class. In Section 2, we describe various properties of this ranking.

Page ranks allow us to define a certain canonical local strategy. More importantly, they allow us to view the problem in a clean geometric way, where the requests for pages correspond to points on the line. In particular, if the requested page has rank m , we think of the request arriving at point m on the line. Even though the page request sequence can be arbitrary, the resulting *rank sequences* in the view above are not arbitrary but have a useful “repeat property”, which we crucially exploit in both designing the update rule for P_i and analyzing the algorithm. (Prediction errors are incorporated quite directly in the above approach, and require only an accounting of how these errors affect the ranks and the repeat property.)

The overall algorithm is described in Section 3 and the analysis is described in Section 4. The analysis uses several potential functions in a careful way. In particular, besides a relative-entropy type potential to handle the multiplicative update of x_i , we use additional new potentials to handle the dynamics and evolution of P_i .

1.4 Other related work Due to its relevance in computer systems and the elegance of the model, several variants of paging have been studied [11, 19, 35]. An important direction has been to consider finer-grained models and analyses techniques to circumvent the sometimes overly pessimistic nature of worst-case guarantees. In particular, several *semi-online* models, where the algorithm has some partial knowledge of the future input, have been studied, including paging with locality of reference [21, 36, 31], paging with lookahead [2, 22, 53], Markov paging [38], and interleaved paging [15, 26, 41]. Paging algorithms have also been explored using alternative notions of analysis such as loose-competitiveness [54], diffuse adversaries [39], bijective analysis [4], and parameterized analysis [3].

Learning-augmented algorithms have received a lot of attention recently. Besides paging [45], they have been considered for a wide range of problems such as ski-rental [33, 40], scheduling [8, 34, 40, 48], load balancing [42, 44], secretary [7], metrical task systems [6], set cover [9], flow and matching [43], and bin packing [5].

In [37], a different prediction model for weighted paging was considered where at each time, the algorithm has access to a prediction of the *entire* request sequence until the time when every page is requested at least once more. We note that this requires much more predicted information than our model and the analogous models for unweighted paging.

2 Page ranks, trustful algorithms and repeat violations

As discussed in Section 1.3, our algorithm will have two parts: a global strategy and a local strategy. At each time t , the global strategy specifies the cache space $x_i(t)$ for each weight class i and the local strategy decides which pages of class i to keep. In this section, we define a notion of *ranks* for pages within each class. This will allow us to not only define a local strategy for each class that is close to optimal given any global strategy, but also view the weighted paging problem with arbitrary request sequences in a very clean way in terms of what we call *rank sequences*.

2.1 Page ranks Fix a weight class i . We define a notion of time-varying *ranks* among pages of class i . Let $\sigma|_i$ be the actual request sequence and let $\tau|_i$ be the sequence of predictions, restricted to class i . We can view this as an input for unweighted paging. For brevity, we use $\sigma = \sigma|_i, \tau = \tau|_i$.

Let BELPRED(m) be the variant of Belady's algorithm for cache size m that, upon a cache miss, evicts the page with the farthest-in-future *predicted* next arrival time (breaking ties arbitrarily, but consistently for all m). Note that if all the predictions in τ are accurate, this is simply Belady's algorithm. For class i , let $C_{i,t}^m(\sigma, \tau)$ be the set of pages in the cache of BELPRED(m) at time t ; we call $C_{i,t}^m(\sigma, \tau)$ a *configuration* or *cache state*. We may drop i and σ and/or τ from the notation, and assume that $|C_t^m| = m$ for all t . A simple inductive argument, whose proof we defer to Appendix C, shows that the configurations C_t^m of BELPRED(m) differ in exactly one page for consecutive values of m :

LEMMA 2.1. (CONSISTENCY) *Let C_0^1, C_0^2, \dots be any initial configurations, satisfying $C_0^m \subset C_0^{m+1}$ for all m . Then for any sequences σ, τ , for all times t and all $m \geq 0$, we have $C_t^m(\sigma, \tau) \subset C_t^{m+1}(\sigma, \tau)$.*

Intuitively, Lemma 2.1 simply says that the set of items in the cache (of size m) when running BELPRED(m) will be a subset of the items when running BELPRED($m+1$) on a cache of size $m+1$. It leads to the following well-defined notion of rank⁵ on pages at any time t .

DEFINITION 2.1. (PAGE RANK) *A page p has rank m at time t if $C_t^m \setminus C_t^{m-1} = \{p\}$.*

We now describe how the ranks change when a page is requested. Suppose the requested page p had rank m_0 just before it was requested. Then p will have new rank 1 as it lies in the cache of BELPRED(m) for every $m \geq 1$. The pages with ranks $m > m_0$ do not change. Consider the set of pages with (old) ranks $1, \dots, m_0 - 1$. (Note that this is precisely $C_t^{m_0-1}$.) Among those pages, the one whose next predicted request is farthest in the future will be updated to have a new rank of m_0 ; denote its original rank by m_1 . All pages with rank between m_1 and m_0 will keep their ranks. Continuing this way, if we consider the pages of ranks $1, 2, \dots, m_1 - 1$ (corresponding to $C_t^{m_1-1}$), the page among those whose predicted request appears farthest in the future will have a new rank of m_1 ; denote its original rank by m_2 . We can recursively define m_3, m_4 and so on in a similar fashion. See Figure 1 for an illustration. More formally, we have the following lemma.

LEMMA 2.2. (RANK UPDATE) *For a given time t , let p_m denote the page with rank m , and let m_0 be the rank of the next requested page p_{m_0} . Starting from m_0 , define the sequence $m_0 > m_1 > \dots > m_b = 1$ inductively as follows: given m_a for $a \geq 0$, m_{a+1} is the rank of the page in $C_t^{m_a-1} = \{p_1, p_2, \dots, p_{m_a-1}\}$ with predicted next request farthest in the future. If $m_a = 1$, then $b := a$ and the sequence ends.*

Then at time $t+1$, page p_{m_0} will have rank 1, and for $a = 1, \dots, b$, page p_{m_a} will have rank m_{a-1} . All other ranks remain unchanged.

Proof. Clearly p_{m_0} will receive rank 1 as it must lie in the cache of BELPRED(1). Moreover, as BELPRED(m) for $m \geq m_0$ does not incur a cache miss, ranks greater than m_0 do not change.

Next, by definition of m_a , the page evicted by BELPRED(m) for $m \in \{m_a, m_a + 1, \dots, m_{a-1} - 1\}$ is p_{m_a} , so p_{m_a} will have new rank $\geq m_{a-1}$. However, as BELPRED(m) for $m \geq m_{a-1}$ keeps p_{m_a} in its cache (as it evicts another page), p_{m_a} will have new rank exactly m_{a-1} . Also as no page other than p_{m_0}, \dots, p_{m_b} will be loaded or evicted by any BELPRED(m), none of these other pages' ranks will change. \square

⁵This is very different from the ordering of pages according to their predicted next request time.

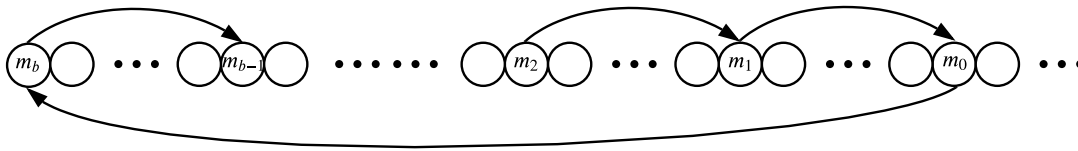


Figure 1: Change of ranks when rank m_0 is requested. m_{a+1} is the rank in $\{1, \dots, m_a - 1\}$ whose associated page has the latest predicted next request time, and $m_b = 1$.

2.2 Local strategy and trustful algorithms Recall that a global strategy specifies a vector $x(t) = (x_1(t), \dots, x_\ell(t))$ at each time t , describing the cache space for each weight class. For a weight class i , consider the local strategy that keeps the pages with ranks between 1 and $m = x_i(t)$ in cache at time t . Note that this is precisely the set $C_{i,t}^m$. If m is fractional, let us extend the definition of $C_{i,t}^m$ to be the cache state that fully contains the pages with ranks $1, \dots, \lfloor m \rfloor$, and an $m - \lfloor m \rfloor$ fraction of the page with rank $\lfloor m \rfloor + 1$. We call such algorithms trustful, defined next.

DEFINITION 2.2. (TRUSTFUL ALGORITHM) A weighted paging algorithm is trustful if at each time t its configuration (i.e. cache state) is of the form $\bigcup_{i=1}^\ell C_{i,t}^{x_i(t)}$ for some $(x_1(t), \dots, x_\ell(t))$.

Next, we show that we can restrict our attention to trustful algorithms⁶ without loss of generality.

LEMMA 2.3. Let A be an arbitrary (offline) weighted paging algorithm, and let $x_i(t)$ denote the cache space used by class i at time t under A . For any arbitrary σ, τ , let A^* be the trustful algorithm with configuration $\bigcup_{i=1}^\ell C_{i,t}^{x_i(t)}(\sigma, \tau)$ at any time t . Then, $\text{cost}_{A^*}(\sigma, \tau) \leq 3 \cdot \text{cost}_A(\sigma, \tau) + \epsilon(\sigma, \tau) + O(1)$.

The proof of Lemma 2.3 is in Appendix C and uses how ranks change over time. As we remark there, a minor modification also yields a much simpler proof of a bound of Wei [52] for learning-augmented unweighted paging, whose proof was based on an analysis of eleven cases.

Thanks to Lemma 2.3, we can assume that the *offline* algorithm is trustful at the expense of misjudging its cost by an $O(1 + \epsilon/\text{OPT})$ factor. Note that given a global strategy that computes $x(t)$ online, the corresponding trustful algorithm can be implemented in the learning-augmented setting; the algorithm we design will also be trustful.

2.3 Rank sequences and repeat violations Restricting ourselves to trustful algorithms and the local strategy above is useful as we can view the subsequence of page requests for a given weight class as a sequence of *rank requests*. Consider a single weight class and let r_1, r_2, \dots be the request sequence of pages within that class. Then together with sequence τ of predicted next arrivals, it induces the corresponding *rank sequence* h_1, h_2, \dots where h_t is the rank of r_t just before it is requested. A trustful algorithm has a page fault on the arrival of h_t if and only if it has less than h_t pages of that weight class in its cache.

Structure of rank sequences. It turns out that rank sequences have a remarkable structural property. In particular, for the case of perfect predictions ($\epsilon = 0$), the possible rank sequences are *exactly* characterized by the following “repeat property”, which we prove in Appendix C.

LEMMA 2.4. (REPEAT PROPERTY) Let $\epsilon = 0$ and let i be a weight class. A rank sequence corresponds to a request sequence of pages of class i if and only if it has the following repeat property: for any h , between any two requests to the same rank h , every rank $2, \dots, h - 1$ must be requested at least once.

With imperfect predictions (i.e., $\epsilon \neq 0$), while there is no clean characterization, the number of times the repeat property is violated can be bounded in terms of ϵ . We say that time t is a *repeat violation* if rank h_t was also requested at some earlier time (for the same weight class) and some rank from $\{2, \dots, h_t - 1\}$ has *not* been requested since then. The following lemma bounds the number of repeat violations.

⁶Note that a trustful algorithm is *not* the same as running BELPRED on each weight class (which is optimal for perfect predictions [49, 17]). E.g., if $x_i(t)$ reduces from m to $m - 1$, a trustful algorithm will evict the page with rank m , whereas BELPRED would evict the page with the farthest predicted next arrival.

LEMMA 2.5. (BOUNDED REPEAT VIOLATIONS) *The number of repeat violations during requests to pages of weight class i is at most ϵ_i .*

Proof. Let h_1, h_2, \dots be the rank sequence of weight class i . Consider some time t_2 that is a repeat violation of weight class i . Let t_1 be the last time before t_2 that rank h_{t_2} was requested, and let $h := h_{t_1} = h_{t_2}$. Since t_2 is a repeat violation, there exists a rank $h' \in \{2, \dots, h-1\}$ that is missing in the sequence $h_{t_1+1}, \dots, h_{t_2-1}$. For any m , denote by p_m the page with rank m at a given time. We will show that at time $t_2 - 1$, page $p_{h'}$ has an earlier predicted arrival time than page p_h . Thus, the request to p_h at time t_2 increases ϵ_i by 1.

Consider the last time in $\{t_1, \dots, t_2 - 1\}$ when the identity of p_h changes (this time exists as t_1 is one such time). By Lemma 2.2, the new page p_h will have the farthest predicted next arrival among the pages p_2, \dots, p_h . It suffices to show that it remains true until time $t_2 - 1$ that p_h has a farther predicted next arrival time than $p_{h'}$. This could only change if $p_{h'}$ changes. But as rank h' itself is not requested between times t_1 and t_2 , by Lemma 2.2, $p_{h'}$ can only change due to a request to a page with larger rank, and the predicted next arrival time of the new page $p_{h'}$ can only decrease due to this change. \square

3 Algorithm

We now describe and analyze an $O(\log \ell)$ -competitive algorithm for learning-augmented weighted paging without prediction errors (Theorem 1.2), and more generally show that it is $O(\log \ell + \ell \epsilon / \text{OPT})$ -competitive with imperfect predictions. Combining our algorithm with any $O(\log k)$ -competitive weighted paging algorithm via the combination method of Blum and Burch [18] yields Theorem 1.4.

By Lemma 2.3 we can assume that both the online and offline algorithms are trustful, and hence are fully specified by vectors $(x_1(t), \dots, x_\ell(t))$ and $(y_1(t), \dots, y_\ell(t))$ that describe the cache space used by each weight class under the online and offline algorithm, at each time t . For the online algorithm, we allow $x_i(t)$ to be fractional so that the cache contains pages with ranks $1, \dots, \lfloor x_i \rfloor$ fully and a $x_i - \lfloor x_i \rfloor$ fraction of the page with rank $\lfloor x_i \rfloor + 1$. By standard techniques [13], this can be converted into a randomized algorithm with integer $x_i(t)$, losing only a constant factor in the competitive ratio.

By the rank sequence view in Section 2.3, the problem can be restated as follows. At each time t , some rank p in some weight class r is requested. If $x_r(t) < p$, there is a cache miss and the algorithm incurs cost $w_r \cdot \min(1, p - x_r(t))$. The goal is to design the update of $x_i(t)$ so as to minimize the total cost. Before giving the details, we first present an overview of the algorithm.

3.1 Overview For convenience, we usually drop the dependence on time t from the notation. Consider a request to a page with rank p in weight class r and suppose our algorithm has a cache miss, i.e., $x_r < p$. Then x_r always increases and all other x_i with $i \neq r$ and $x_i > 0$ decrease. So the main question is at what rate to decrease the x_i for $i \neq r$ (since $\sum_i x_i = k$ at all times, the rate of increase $x'_r = -\sum_{i \neq r} x'_i$). A key idea is to have a variable μ_i for each weight class i that is roughly proportional to the eviction rate from weight class i . This variable also changes over time depending on the ranks of requested pages within each weight class. We now describe the main intuition behind how the μ_i are updated.

For each weight class i , we maintain a more refined memory of the past in the form of a set $S_i \subseteq [x_i, \infty)$, consisting of a union of intervals. Roughly speaking, the set S_i can be thought of as an approximate memory of ranks of weight class i that were requested relatively recently. So the P_i in Section 1.3 corresponds to $(0, x_i] \cup S_i$. We set $\mu_i = |S_i|$, the Lebesgue measure of S_i .

For weight classes $i \neq r$, the algorithm increases μ_i multiplicatively, so that eventually we will evict faster from weight classes that are only rarely requested. For the same reason, for the requested weight class r we would like to decrease μ_r (as we increased x_r , we would not want to decrease it rapidly right away when requests arrive next in other classes). However, decreasing μ_r could be highly problematic in the case that the offline algorithm has $y_r < x_r$, because this will slow down our algorithm in decreasing x_r in the future, making it very difficult to catch up with y_r later.

To handle this, crucially relying on the repeat property, we decrease μ_r if and only if $p \in S_r$. Informally, the reason is the following: if the last request of rank p was recent, then—assuming no repeat violation—all the ranks in $\{2, \dots, p\}$ were also requested recently. This suggests it may be valuable to hold p pages from weight class r in cache, but our algorithm only holds $x_r < p$ of them. To improve our chance of increasing x_r towards p in the future, we should reduce μ_r .

This simplified overview omits several technical details and in particular how to update the sets S_i . We now describe the algorithm in detail and then give the analysis in Section 4.

3.2 Detailed description As stated above, we denote by $x_i \in [0, k]$ the page mass from weight class i in the algorithm's cache, meaning that the pages with ranks $1, \dots, \lfloor x_i \rfloor$ of weight class i are fully in its cache and the next page is present to an $x_i - \lfloor x_i \rfloor$ extent. We maintain a variable μ_i and a set $S_i \subset [x_i, \infty)$ for weight class i . The x_i, μ_i, S_i are all functions of time t , but we suppress this dependence for notational convenience. Algorithm 1 contains a summary of our algorithm, that we now explain in detail.

Algorithm 1: Serving a request to rank q of weight class r .

```

Initialize  $p := q - 1$ 
while  $p < q$  do
    Increase  $p$  at rate  $p' = 8$ 
    if  $p > x_r$  then
        Update  $x$  and  $\mu$  according to (3.1), (3.2)
        for  $i \neq r$  do
            Add to  $S_i$  the points that  $x_i$  moves past
        end
        Remove points from the left and right of  $S_r$  at rates  $x'_r$  and 1
        if  $p \notin S_r$  then
            Add points from  $(q - 1, p] \setminus S_r$  to  $S_r$  at rate 2
        end
    end
end

```

Request arrival and continuous view. Consider a request to the page with rank q of weight class r . If $x_r \leq q - 1$ (resp. $x_r \geq q$), the page is fully missing (resp. fully present) in the algorithm's cache. But for $q - 1 < x_r < q$, the page is fractionally present. To obtain a view where each request is either fully present or fully missing, we break the request to rank q into infinitesimally small requests to each point $p \in (q - 1, q]$, by moving a variable p continuously along the interval $(q - 1, q]$. We call p the *pointer* (to the current request) and move it from $q - 1$ to q at speed 8, so the duration of this process is $1/8$ unit of time. During this process, we update x , μ , and the sets S_i continuously.

Update of x and μ . Let $\delta := \frac{1}{\ell}$ and define variables

$$M := \sum_{i=1}^{\ell} \mu_i, \quad \beta_i = \frac{\mu_i + \delta M}{w_i M}, \quad \text{and} \quad B := \sum_{i=1}^{\ell} \beta_i.$$

As μ changes over time, these quantities also change over time.

While $p \leq x_r$, the algorithm does nothing (the corresponding infinitesimal request is already present in its cache). While $p > x_r$, we update x_i and μ_i , for each class i , at rates

$$(3.1) \quad x'_i = \mathbb{1}_{\{i=r\}} - \frac{\beta_i}{B}$$

$$(3.2) \quad \mu'_i = \frac{\beta_i}{B} - 2 \cdot \mathbb{1}_{\{i=r \text{ and } p \in S_r\}}.$$

Let us make a few observations. Each β_i lies between 0 and B , and hence x_r increases while all x_i for $i \neq r$ decrease. Moreover, we have that $\sum_i x'_i = 0$, and so the total cache space $\sum_i x_i$ used by our algorithm remains constant at its initial value of k . Second, μ_i increases for all $i \neq r$. For $i = r$, we decrease μ_r if and only if $p \in S_r$.

Update of the sets S_i . Each set S_i maintained by our algorithm will be a (finite) union of intervals satisfying

- (i) $S_i \subset [x_i, \infty)$ and
- (ii) $|S_i| = \mu_i$,

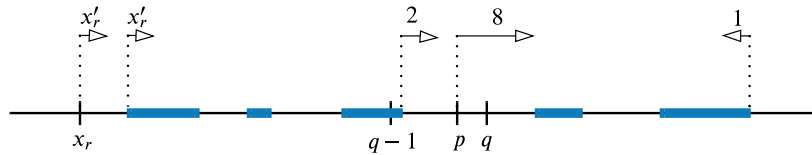


Figure 2: Update of S_r (blue) when $p \notin S_r$

where $|\cdot|$ denotes the Lebesgue measure. We initialize S_i arbitrarily to satisfy these properties (e.g., as $S_i := [x_i, x_i + \mu_i)$).

The precise update rule for the sets S_i is as follows. First consider $i \neq r$. Here x_i decreases, and we simply add to S_i all the points that x_i moves over. As $S_i \subset [x_i, \infty)$, these points are not in S_i yet, so S_i grows at rate $-x_i' = \beta_i/B$. As μ_i' is also β_i/B , both $|S_i| = \mu_i$ and $S_i \subset [x_i, \infty)$ remain satisfied.

For the requested weight class r , modifying S_r is somewhat more involved, consisting of three simultaneous parts (see Figure 2):

- We remove points from the left (i.e., minimal points) of S_r at the rate at which x_r is increasing (i.e., we increase the left boundary of the leftmost interval of S_r at rate x_r'). This ensures that the property $S_r \subset [x_r, \infty)$ is maintained.
- We also remove points from the right of S_r at rate 1. We can think of these points as “expiring” from our memory of points that were recently requested.
- While $p \in S_r$, we do nothing else. Observe that $|S_r| = \mu_r$ remains satisfied.
- While $p \notin S_r$, we also add points from $(q-1, p] \setminus S_r$ to S_r at rate 2, thereby ensuring again that $|S_r| = \mu_r$ remains satisfied. This can be achieved as follows, which will also ensure that S_r continues to be a finite union of intervals: Consider the leftmost interval of S_r that overlaps with $(q-1, p]$; if no such interval exists, consider the empty interval $(q-1, q-1]$ instead (or, if $x_r \in (q-1, q]$, consider the empty interval $(x_r, x_r]$). Since $p \notin S_r$, the right boundary of this interval lies in $[q-1, p]$. We can add points from $(q-1, p] \setminus S_r$ to S_r at rate 2 by shifting this boundary to the right at rate 2. (Since p moves at rate 8, the boundary will never “overtake” p .) If the considered interval happens to be the rightmost interval of S_r , then combined with the previous rule of removing points from the right at rate 1 this would mean that effectively the right boundary moves to the right only at rate 1 instead of 2.

Note that the removal of points from the left and right of S_r is always possible: If S_r were empty, then $p \notin S_r$ and the addition of points at rate 2 is faster than the removal.

Boundary conditions. The description of the algorithm is almost complete, except that it does not ensure yet that the requested page is fully loaded to its cache, which requires that $x_r \geq 1$ (recall that the requested page will receive new rank 1), and that no x_i decreases below 0. Both issues can be handled as follows. Add a dummy page d_i to each weight class i and increase the cache size by ℓ , where this extra cache space shall be used to hold the ℓ dummy pages. In the request sequence, replace any request to a page v by many repetitions of the request sequence (v, d_1, \dots, d_ℓ) . As the number of repetitions tends to infinity, our algorithm converges to a state with $x_i \geq 1$ for all i and $x_r \geq 2$ for the weight class r containing v . Thus, it holds v and all dummy pages in its cache. The corresponding algorithm obtained by removing the dummy pages (and reducing the cache size back to k) is a valid paging algorithm for the original request sequence.

4 Analysis

The goal of this section is to prove the following theorem:

THEOREM 4.1. *The algorithm is $O(\log \ell + \ell\epsilon/\text{OPT})$ -competitive for learning-augmented weighted paging.*

We employ a potential function based analysis. Our potential function Φ will consist of three potentials \mathcal{E}, \mathcal{R} ,

and \mathcal{S} and is defined as follows.

$$\begin{aligned}
(\text{Entropic potential}) \quad \mathcal{E} &:= \sum_i w_i \left[5[\mu_i + (x_i - y_i)_+] \log \frac{(1 + \delta)[\mu_i + (x_i - y_i)_+]}{\mu_i + \delta[\mu_i + (x_i - y_i)_+]} + [\mu_i + 2(x_i - y_i)_+] \right], \\
(\text{Repeat potential}) \quad \mathcal{R} &:= \sum_i w_i \int_{S_i} \frac{|(y_i, x_i] \cap R_{iu}|}{\mu_i + \delta(\mu_i + |(y_i, x_i] \cap R_{iu}|)} du, \\
(\text{Scatter Potential}) \quad \mathcal{S} &:= \sum_i w_i (|S_i \cap [y_i, \infty)| + (x_i - y_i)_+), \\
(\text{Overall potential}) \quad \Phi &:= 2\mathcal{E} + 5\mathcal{R} + 4\mathcal{S}.
\end{aligned}$$

Here y_i describes the state of a *trustful* offline algorithm (whose cost is within a factor $O(1 + \epsilon/\text{OPT})$ of the optimal offline algorithm by Lemma 2.3). We use $(x_i - y_i)_+ := \max\{0, x_i - y_i\}$. The set R_{iu} , for a weight class i and $u \geq 0$, consists of all points that were visited by the pointer since the last request at u (details in Section 4.4). In the definition of the repeat and scatter potentials, $|\cdot|$ denotes the Lebesgue measure.

The potential \mathcal{E} bears similarities with entropy/Bregman divergence type potentials used in other contexts [12, 23, 24, 25, 28]. The potential \mathcal{R} is carefully designed to exploit the repeat property. In particular, just before the pointer reaches u , the set R_{iu} contains $[1, u)$ provided there is no repeat violation, and it becomes empty immediately afterwards. The scatter potential is mostly for technical reasons to handle that the intervals in S_i may be non-contiguous.

To show that our algorithm has the desired competitive ratio, it suffices to show that

$$(4.3) \quad \text{On}' + \Phi' \leq O(\log \ell) \cdot \text{Off}' + O(\ell)\epsilon'$$

at all times, where On, Off, and ϵ are appropriate continuous-time notions of online cost, offline cost, and the prediction error and a' denotes the derivative of a with respect to time. We will actually take On to be a quantity we call *online pseudo-cost* that approximates the true online cost, as discussed in Section 4.2. For each request, we will consider the step where the offline algorithm changes its vector (y_1, \dots, y_ℓ) separately from the remaining events.

4.1 Offline cost We will charge the offline algorithm for both the weights of pages it fetches as well as pages it evicts.⁷ We may assume that for each request, the (trustful) offline algorithm proceeds in two steps: First it changes y and updates its cache content with respect to the *old* ranks. Then it updates its cache content to reflect the new ranks of weight class r . This means that it may fetch a page of weight class r in the first step that it evicts again in the second step in order to load the requested page (recall Lemma 2.2/Figure 1). Since both of these pages have the same weight, this overestimates the offline cost by only a constant factor.

Change of y . When the offline algorithm changes some y_i at rate y'_i , it incurs cost $w_i|y'_i|$.

We claim that the term corresponding to weight class i in each of the potentials changes at rate at most $O(\log \ell) w_i|y'_i|$. First, for the entropic potential, it is not hard to see that $\frac{d\mathcal{E}}{dy_i} = O(1 + \log \frac{1+\delta}{\delta}) w_i = O(\log \ell) w_i$. Therefore,

$$\mathcal{E}' \leq O(\log \ell) w_i|y'_i|.$$

Next, for the scatter potential, as $[y_i, \infty)$ and $(x_i - y_i)_+$ can change at rate at most $|y'_i|$, we also have that $\mathcal{S}' \leq 2w_i|y'_i|$. It remains to bound \mathcal{R}' .

Fix a $u \in S_i$ and consider the term in the integrand of \mathcal{R} corresponding to u . If y_i increases, then $(y_i, x_i] \cap R_{iu}$ can only decrease, in which case \mathcal{R} only decreases. If y_i decreases, then $(y_i, x_i] \cap R_{iu}$ increases at rate at most $|y'_i|$. Ignoring the increase in the denominator (which only decreases \mathcal{R}), the increase in the numerator leads to an increase of at most

$$\frac{|y'_i|}{\mu_i + \delta(\mu_i + |(y_i, x_i] \cap R_{iu}|)} \leq \frac{|y'_i|}{\mu_i}.$$

As $|S_i| = \mu_i$, integrating the above over points in S_i , the integral rises at rate at most $|S_i||y'_i|/\mu_i = |y'_i|$. So, $\mathcal{R}' \leq w_i|y'_i|$ as desired.

To summarize, when y changes, (4.3) holds as we can charge the change in the overall potential to $O(\log \ell)$ times the offline cost.

⁷Note that over all time steps, the total fetching and total eviction cost are within an additive constant of each other.

Change of ranks. We can assume that the offline vector y is integral. If $y_r < q$, then offline pays w_r to fetch the page. As the pointer p moves in $[q-1, q]$ at rate 8, equivalently, we can view this as charging the offline algorithm continuously at rate $\mathbb{1}_{\{y_r < p\}} 8w_r$ during the movement of p . This view will be useful for analyzing the online algorithm in a continuous way, as we do next.

4.2 Online pseudo-cost Instead of working with the actual cost incurred by the online algorithm, it will be convenient to work with a simpler *online pseudo-cost*. This is defined as the quantity that is initially 0 and grows at rate $1/B$ at all times during which the online algorithm changes x . Recall that B is itself changing over time. Formally, we define the online pseudo-cost as

$$\int_0^\infty \frac{1}{B(t)} \cdot \mathbb{1}_{\{x(t)' \neq 0\}} dt.$$

The following lemma shows that the online pseudo-cost is a good proxy for the true online cost.

LEMMA 4.1. *Up to a bounded additive error, the total cost of the online algorithm is at most 18 times the online pseudo-cost.*

Proof. The online cost is the weighted page mass loaded to its cache. The two events that lead to page mass being loaded to the cache are either an increase of x_r or a change of ranks.

By definition of ranks (or alternatively, Lemma 2.2/Figure 1), a request to rank q does not affect the (unordered) set of pages with ranks $1, 2, \dots, s$ for any $s \geq q$, and for $s < q$ it affects this set only by adding and removing one page. Thus, if rank q of class r is requested, the change of ranks incurs cost $w_r \min\{1, q - x_r\}_+$.

We can overestimate the cost due to increasing x_r by viewing the change of x_r as an increase at rate 1 separate from a decrease at rate β_r/B . In this view, the online cost for increasing x_r is w_r times the duration of the increase. Since the pointer p moves at speed 8 across $(q-1, q]$, and changes to x occur only while $p > x_r$, the duration of the update of x for this request is precisely $\frac{1}{8} \min\{1, q - x_r\}_+$ (where x_r denotes the value of this variable before the request arrives). Therefore, the online cost for increasing x_r is $\frac{1}{8} w_r \min\{1, q - x_r\}_+$, and hence the total online cost is 9 times the cost for increasing x_r (in our overestimating view).

Over the course of the algorithm, the overall increase of any x_i equals the decrease, up to an additive constant. Thus, instead of charging for increasing x_r and the change of ranks, we can charge only for decreasing each x_i (including $i = r$) at rate β_i/B (the associated cost being w_i times this quantity). This underestimates the true online cost by a factor of at most 9, up to an additive constant. The cost charged in this way increases at rate

$$\sum_i w_i \frac{\beta_i}{B} = \sum_i w_i \frac{\mu_i + \delta M}{B w_i M} = \frac{M + M}{BM} = \frac{2}{B},$$

which is twice the rate of increase of the pseudo-cost, and the result follows. \square

By Lemma 4.1 it now suffices (up to $O(1)$ factors) to assume that $\text{On}' = \frac{1}{B} \cdot \mathbb{1}_{\{x' \neq 0\}}$. Consequently, recalling that the offline algorithm suffers cost at rate $\mathbb{1}_{\{y_r < p\}} 8w_r$ due to the change of ranks, in order to prove the desired competitive ratio it suffices to show that

$$\frac{1}{B} \cdot \mathbb{1}_{\{x' \neq 0\}} + \Phi' \leq O(\log \ell) \cdot \mathbb{1}_{\{y_r < p\}} w_r + O(\ell) \epsilon'.$$

We now upper bound the rate of change of each of the potentials.

4.3 Rate of change of the entropic potential \mathcal{E}

LEMMA 4.2. *The rate of change of the entropic potential \mathcal{E} is at most*

$$\begin{aligned} \mathcal{E}' \leq & -\frac{1}{2B} + \mathbb{1}_{\{y_r < p\}} w_r O(\log \ell) - \frac{5}{2} \sum_{i \neq r} \frac{w_i (x_i - y_i)_+ \mu'_i}{\mu_i + \delta(\mu_i + x_i - y_i)} \\ & + \mathbb{1}_{\{p \in S_r\}} \frac{15w_r (x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)} + \mathbb{1}_{\{y_r < x_r + \mu_r \text{ and } p \notin S_r\}} 2w_r, \end{aligned}$$

Note that the first term $-\frac{1}{2B}$ can be charged against the online pseudo-cost and the second term can be charged to the offline cost resulting from the change of ranks. The third term is negative, which only helps. However, the last two terms are problematic. We will handle them using the repeat potential and the scatter potential.

Proof. [of Lemma 4.2] Define $\tilde{M} := \sum_i (\mu_i + x_i - y_i)_+$ and $L := \sum_i (x_i - y_i)_+ = \sum_i (y_i - x_i)_+$, where the equality is due to $\sum_{i=1}^\ell x_i = \sum_{i=1}^\ell y_i = k$. We note that

$$(4.4) \quad \max\{M, L\} \leq \tilde{M} \leq M + L,$$

where the first inequality follows as $M = \sum_i \mu_i = \sum_i (\mu_i + x_i - y_i) \leq \tilde{M}$ and $L \leq \tilde{M}$ as $\mu_i \geq 0$ for all i , and the second inequality follows from the triangle inequality.

Letting

$$\Psi := \sum_{i: x_i \geq y_i} w_i (\mu_i + x_i - y_i) \log \frac{(1 + \delta)(\mu_i + x_i - y_i)}{\mu_i + \delta(\mu_i + x_i - y_i)}$$

we can write

$$\mathcal{E} = 5\Psi + \sum_i w_i [\mu_i + 2(x_i - y_i)]_+.$$

We first bound the change of Ψ . We first consider the case $x_r < y_r$, so that the summand for $i = r$ is 0 and does not contribute to Ψ' . For $i \neq r$, the sum $\mu_i + x_i$ is unchanged by (3.1) and (3.2), and therefore

$$(4.5) \quad \begin{aligned} \Psi' &= - \sum_{i: x_i \geq y_i} w_i \frac{\mu_i + x_i - y_i}{\mu_i + \delta(\mu_i + x_i - y_i)} \mu'_i = - \sum_{i: x_i \geq y_i} \frac{\mu_i + x_i - y_i}{\mu_i + \delta(\mu_i + x_i - y_i)} \frac{\mu_i + \delta M}{BM} \\ &\leq - \sum_{i: x_i \geq y_i} \frac{\mu_i + x_i - y_i}{\mu_i + \delta(\mu_i + x_i - y_i)} \frac{\mu_i + \delta \tilde{M}}{B\tilde{M}} \leq - \sum_{i: x_i \geq y_i} \frac{x_i - y_i}{\mu_i + \delta \tilde{M}} \frac{\mu_i + \delta \tilde{M}}{B\tilde{M}} \\ (4.6) \quad &= - \frac{L}{B\tilde{M}} \leq - \frac{1}{B} \frac{L}{M + L} \quad (\text{if } x_r < y_r), \end{aligned}$$

where the first inequality uses $M \leq \tilde{M}$, and the second inequality uses $\mu_i \geq 0$ and $\mu_i + x_i - y_i \leq \tilde{M}$.

We will actually need the following slightly more complicated bound, which can be obtained by combining (4.5) and (4.6):

$$\Psi' \leq - \frac{1}{2} \sum_{i \neq r} \frac{w_i (x_i - y_i)_+ \mu'_i}{\mu_i + \delta(\mu_i + x_i - y_i)} - \frac{1}{2B} \frac{L}{M + L} \quad (\text{if } x_r < y_r).$$

Thus, the contributions considered so far only lead to a decrease of Ψ . However if $x_r \geq y_r$, then Ψ could also suffer an increase resulting from increasing x_r at rate 1 and, if $p \in S_r$, decreasing μ_r at rate 2. In this case, the change of Ψ can exceed the preceding bound by at most

$$(4.7) \quad \begin{aligned} &w_r \left[\log \frac{(1 + \delta)(\mu_r + x_r - y_r)}{\mu_r + \delta(\mu_r + x_r - y_r)} + 1 + \mathbb{1}_{\{p \in S_r\}} \frac{\mu_r + x_r - y_r}{\mu_r + \delta(\mu_r + x_r - y_r)} (1 + \delta) 2 \right] \\ &\leq w_r \cdot O(\log \ell) + \mathbb{1}_{\{p \in S_r\}} \frac{3w_r (x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)}. \end{aligned}$$

In summary, while x is changing, Ψ is changing at rate

$$(4.8) \quad \Psi' \leq - \frac{1}{2} \sum_{i \neq r} \frac{w_i (x_i - y_i)_+ \mu'_i}{\mu_i + \delta(\mu_i + x_i - y_i)} - \frac{1}{2B} \frac{L}{M + L} + \mathbb{1}_{\{y_r < p\}} w_r O(\log \ell) + \mathbb{1}_{\{p \in S_r\}} \frac{3w_r (x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)},$$

where the term $\mathbb{1}_{\{y_r < p\}}$ comes from the fact that the extra increase (4.7) is incurred only if $y_r \leq x_r$ and x is changing only if $x_r < p$.

We now bound the change of the part of \mathcal{E} not involving Ψ , i.e., of the quantity $\mathcal{E} - 5\Psi = \sum_i w_i [\mu_i + 2(x_i - y_i)]_+$. Using the update rules for μ and x , and cancelling some common terms, the rate of change can be written as

$$(4.9) \quad \begin{aligned} (\mathcal{E} - 5\Psi)' &= \mathbb{1}_{\{\mu_r > 2(y_r - x_r) \text{ and } p \notin S_r\}} 2w_r - \sum_{i: \mu_i > 2(y_i - x_i)} w_i \frac{\beta_i}{B} \\ &\leq \mathbb{1}_{\{y_r < x_r + \mu_r \text{ and } p \notin S_r\}} 2w_r - \frac{1}{BM} \sum_{i: \mu_i > 2(y_i - x_i)} \mu_i. \end{aligned}$$

As $\sum_i \mu_i = M$, the sum in (4.9) can be rewritten as

$$\sum_{i: \mu_i > 2(y_i - x_i)} \mu_i = \left[M - \sum_{i: \mu_i \leq 2(y_i - x_i)} \mu_i \right]_+ \geq \left[M - \sum_{i: 0 \leq 2(y_i - x_i)} 2(y_i - x_i) \right]_+ = [M - 2L]_+.$$

Thus,

$$(4.10) \quad (\mathcal{E} - 5\Psi)' \leq \mathbb{1}_{\{y_r < x_r + \mu_r \text{ and } p \notin S_r\}} 2w_r - \frac{1}{B} \left[1 - \frac{2L}{M} \right]_+.$$

The lemma follows by combining (4.8) and (4.10) and noting that

$$-\frac{5}{2B} \frac{L}{M+L} - \frac{1}{B} \left[1 - \frac{2L}{M} \right]_+ \leq -\max \left\{ \frac{5}{2B} \frac{L}{M+L}, \frac{1}{B} \left[1 - \frac{2L}{M} \right]_+ \right\} \leq -\frac{1}{2B},$$

which can be seen by considering separately the cases $M \leq 4L$ and $M > 4L$. \square

4.4 The repeat potential \mathcal{R} and its rate of change The purpose of the repeat potential is to cancel the term

$$\mathbb{1}_{\{p \in S_r\}} \frac{15w_r(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)},$$

from our bound on \mathcal{E}' in case the current request is not a repeat violation. We will crucially use that if the current request is not a repeat violation, then since the last request to rank q of weight class r , every rank less than q has been requested at least once.

For a weight class i and $u \in \mathbb{R}_+$, denote by R_{iu} the set of points across which the pointer p has moved during requests of weight class i after the time when the pointer was last located at u for a request to weight class i . In other words, after any request R_{iu} is the set $(u, [u]] \cup \bigcup_s [s-1, s]$, where s ranges over all ranks of weight class i that have been requested (so far) after the last request to rank $[u]$ of weight class i . If the pointer was never at u during a request to weight class i , define $R_{iu} := \mathbb{R}_+$ as the entire positive real line.

Recall that the *repeat potential* is defined as

$$\mathcal{R} := \sum_{i=1}^{\ell} w_i \mathcal{R}_i, \quad \text{where} \quad \mathcal{R}_i := \int_{S_i} \frac{|(y_i, x_i] \cap R_{iu}|}{\mu_i + \delta(\mu_i + |(y_i, x_i] \cap R_{iu}|)} du,$$

and $|\cdot|$ denotes the Lebesgue measure.

When online moves, \mathcal{R}_i will change due to the changes in the values of x_i , μ_i , the sets S_i and R_{iu} . We will consider the effect of each of these changes separately while analyzing \mathcal{R}' . We also consider $i \neq r$ and $i = r$ separately. The case $i \neq r$ is quite simple, and we describe it next.

Change of \mathcal{R}_i for $i \neq r$: Since the fraction in the definition of \mathcal{R}_i is non-decreasing in x_i , and x_i decreases for $i \neq r$, the change of x_i does not cause any increase of \mathcal{R}_i . Similarly, \mathcal{R}_i is non-increasing in μ_i and μ_i increases for $i \neq r$, so also the change of μ_i does not cause any increase of \mathcal{R}_i . Moreover, as the request is to weight class $r \neq i$, the sets R_{iu} do not change. However, \mathcal{R}_i could increase due to points being added to S_i (recall that as x_i decreases we add the points that x_i moves over to S_i).

The integrand corresponding to any $u \in S_i$ can be bounded by

$$\frac{|(y_i, x_i] \cap R_{iu}|}{\mu_i + \delta(\mu_i + |(y_i, x_i] \cap R_{iu}|)} \leq \frac{(x_i - y_i)_+}{\mu_i + \delta(\mu_i + x_i - y_i)}.$$

As new points u are added to S_i at rate μ'_i , the change of \mathcal{R}_i is bounded by

$$\mathcal{R}'_i \leq \frac{(x_i - y_i)_+ \mu'_i}{\mu_i + \delta(\mu_i + x_i - y_i)} \quad (\text{for } i \neq r).$$

Change of \mathcal{R}_r : For $i = r$, all the relevant quantities change, and we consider them separately.

Effect of changing x_r : Fix a point $u \in S_r$. The increase of x_r can increase the integrand corresponding to u at most at rate

$$\frac{x'_r}{(\mu_r + \delta(\mu_r + |(y_r, x_r] \cap R_{ru}|))} \leq \frac{1}{\mu_r},$$

as the numerator rises at rate at most $x'_r \leq 1$, and the denominator can only rise and reduce the integrand (which we ignore). Note that this increase occurs only if $y_r < x_r < p$, (as x_r does not increase if $p \leq x_r$, and the interval $(y_r, x_r]$ is empty if $y_r \geq x_r$). Thus, as $|S_r| = \mu_r$, the contribution of the change of x_r to \mathcal{R}'_r is at most

$$\mathbb{1}_{\{y_r < p\}} \int_{S_r} \frac{1}{\mu_r} du = \mathbb{1}_{\{y_r < p\}}.$$

Effect of changing μ_r : For a fixed u , the increase in the integrand due to the change in μ_r is

$$\begin{aligned} \frac{-(1+\delta)|(y_r, x_r] \cap R_{ru}| \cdot \mu'_r}{(\mu_r + \delta(\mu_r + |(y_r, x_r] \cap R_{ru}|))^2} &\leq \mathbb{1}_{\{p \in S_r\}} \frac{(1+\delta)2(x_r - y_r)_+}{(1+\delta)\mu_r(\mu_r + \delta(\mu_r + x_r - y_r))} \\ &= \mathbb{1}_{\{p \in S_r\}} \cdot \frac{1}{\mu_r} \cdot \frac{2(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)} \end{aligned}$$

where the first inequality uses that $\mu'_r \leq 0$ if and only if $p \in S_r$ and in that case $-\mu'_r \leq 2$.

As this expression does not depend on u and as $|S_r| = \mu_r$, the change of μ_r contributes to \mathcal{R}'_r

$$\int_{S_r} \mathbb{1}_{\{p \in S_r\}} \frac{1}{\mu_r} \frac{2(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)} du \leq \mathbb{1}_{\{p \in S_r\}} \frac{2(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)}.$$

Effect of changing S_r : Removing points from S_r can only decrease \mathcal{R}_r . Any point u added to S_r comes from $(q-1, p]$, and since these points have only just been passed by the pointer, $R_{ru} = (u, p]$ for such points; as any such point u added to S_r is also in $[x_r, \infty)$, we have $|(y_r, x_r] \cap R_{ru}| = 0$ for any u added to S_r . Thus, the change of the set S_r does not increase \mathcal{R}_r any further.

Effect of changing R_{ru} : Finally and most crucially, we consider the change of \mathcal{R}_r resulting from the change of the sets R_{ru} . If the current request is not a repeat violation, then just before the pointer reaches position p during the current request, R_{rp} must contain the interval $(1, p)$. Once the pointer reaches p , the set R_{rp} becomes empty. In particular, using that $y_r \geq 1$ and given that x is changing only while $x_r < p$, the quantity $|(y_r, x_r] \cap R_{rp}|$ then changes from $(x_r - y_r)_+$ to 0. If $p \in S_r$, this contributes to a decrease of \mathcal{R}_r . Since the pointer p moves at speed 8, the contribution of the change of the set R_{rp} to \mathcal{R}'_r is then

$$-\mathbb{1}_{\{p \in S_r\}} \frac{8(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)}.$$

However, if there is a repeat violation, then in the worst case R_{rp} may already be empty so this would not yield any contribution to \mathcal{R}_r . In this case, by Lemma 2.5, ϵ_r increases by 1 due to this request. In continuous time, as the pointer moves for $1/8$ unit of time, this corresponds to increasing ϵ_r at rate $\epsilon'_r = 8$. We claim that regardless of a repeat violation or not, the contribution of the change of R_{rp} to \mathcal{R}'_r is at most

$$-\mathbb{1}_{\{p \in S_r\}} \frac{8(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)} + \ell \epsilon'_r.$$

In case of no repeat violation this is just our statement above. If there is a repeat violation then since $\mu_r \geq 0$, $\delta = 1/\ell$ and $\epsilon'_r = 8$ this quantity is ≥ 0 , and since R_{rp} becoming empty cannot increase \mathcal{R}_r , this is a valid upper bound.

Note that \mathcal{R}_r can also increase as points are added to some R_{ru} . The only point added to any R_{ru} is the current pointer position p . The intersection $(y_r, x_r] \cap R_{ru}$ can be increased by this only if $p \leq x_r$ (when x_r is *not* changing). During those times, the integrand increases at most at rate $8/\mu_r$ and only if $p \in (y_r, x_r]$. This can cause \mathcal{R}_r to increase at rate at most $\mathbb{1}_{\{y_r < p\}} 8$.

Overall, while x_r is changing, \mathcal{R}_r changes at rate

$$\begin{aligned}\mathcal{R}'_r &\leq \mathbb{1}_{\{y_r < p\}} + \mathbb{1}_{\{p \in S_r\}} \frac{2(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)} - \mathbb{1}_{\{p \in S_r\}} \frac{8(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)} + \ell\epsilon'_r \\ &\leq \mathbb{1}_{\{y_r < p\}} + \ell\epsilon'_r - \mathbb{1}_{\{p \in S_r\}} \frac{6(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)},\end{aligned}$$

and while p is moving but x_r is not changing, \mathcal{R}_r changes at rate

$$\mathcal{R}'_r \leq \mathbb{1}_{\{y_r < p\}} 8.$$

Combining this with our bound on the change of \mathcal{R}_i for $i \neq r$, we obtain the bounds on the change of \mathcal{R} that are summarized in the following lemma.

LEMMA 4.3. *The overall rate of change of \mathcal{R} can be upper bounded as*

$$\begin{aligned}\mathcal{R}' &\leq \mathbb{1}_{\{y_r < p\}} w_r + \ell\epsilon' - \mathbb{1}_{\{p \in S_r\}} \frac{6w_r(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)} + \sum_{i \neq r} \frac{w_i(x_i - y_i)_+ \mu'_i}{\mu_i + \delta(\mu_i + x_i - y_i)} \quad (\text{if } x \text{ is changing}), \\ \mathcal{R}' &\leq \mathbb{1}_{\{y_r < p\}} 8w_r \quad (\text{while } p \text{ is moving but } x \text{ is unchanged}).\end{aligned}$$

The $\mathbb{1}_{\{y_r < p\}} w_r$ term can be charged to the offline cost, and then note crucially that the third and fourth terms in \mathcal{R}' (when x is changing) are the same but with opposite signs (and up to a factor $5/2$) as the third and fourth term in our bound on \mathcal{E}' .

4.5 The scatter potential \mathcal{S} and its rate of change It remains to cancel the term $\mathbb{1}_{\{y_r < x_r + \mu_r \wedge p \notin S_r\}} 2w_r$ in our bound on \mathcal{E}' . If S_r were the single interval $[x_r, x_r + \mu_r]$, then this term would be $\mathbb{1}_{\{y_r < p\}} 2w_r$ and we could charge it to the offline cost. However, in general S_r can be a union of several intervals of total size μ_r , scattered somewhere in $[x_r, \infty)$ and containing gaps within $[x_r, x_r + \mu_r]$. In this case, such a bound need not hold, and we use the following scatter potential.

$$(\text{Scatter potential}) \quad \mathcal{S} := \sum_{i=1}^{\ell} w_i (|S_i \cap [y_i, \infty)| + (x_i - y_i)_+).$$

LEMMA 4.4. *The rate of decrease of the scatter potential \mathcal{S} satisfies*

$$\mathcal{S}' \leq -\mathbb{1}_{\{y_r < x_r + \mu_r\}} w_r + \mathbb{1}_{\{y_r < p\}} 2w_r.$$

Proof. For $i \neq r$, the term $(x_i - y_i)_+$ is non-increasing. The term $|S_i \cap [y_i, \infty)|$ can increase for $i \neq r$ only if $x_r > y_r$ (as points are added to S_i at x_i). Thus, any possible increase of $|S_i \cap [y_i, \infty)|$ is cancelled by a decrease of $(x_i - y_i)_+$. So \mathcal{S}' is bounded by the rate of change for $i = r$.

A possible increase of the term $(x_r - y_r)_+$ would be cancelled by a decrease of $|S_r \cap [y_r, \infty)|$ caused by removing points from the left of S_r (as $S_r \subset [x_r, \infty)$). The removal of points from the right of S_r at rate 1 contributes a decrease at rate w_r if $y_r < \sup S_r$. Since $x_r + \mu_r \leq \sup S_r$, this contributes at most $-\mathbb{1}_{\{y_r < x_r + \mu_r\}} w_r$ to the change of \mathcal{S} . Any other change of \mathcal{S} could only be due to adding points from $(q-1, p]$ to S_r at rate 2. This can increase \mathcal{S} only if $y_r < p$ and hence contributes $\mathbb{1}_{\{y_r < p\}} 2w_r$. Together, this gives the claimed bound. \square

4.6 Putting it all together Consider our overall potential $\Phi = 2\mathcal{E} + 5\mathcal{R} + 4\mathcal{S}$. Using the bounds from Lemmas 4.2–4.4, we see that while x is changing, this potential is changing at rate

$$\begin{aligned} 2\mathcal{E}' + 5\mathcal{R}' + 4\mathcal{S}' &\leq -\frac{1}{B} + \mathbb{1}_{\{y_r < p\}} w_r O(\log \ell) - 5 \sum_{i \neq r} \frac{w_i(x_i - y_i)_+ \mu'_i}{\mu_i + \delta(\mu_i + x_i - y_i)} \\ &\quad + \mathbb{1}_{\{p \in S_r\}} \frac{30w_r(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)} + \mathbb{1}_{\{y_r < x_r + \mu_r \text{ and } p \notin S_r\}} 4w_r \\ &\quad + \ell\epsilon' - \mathbb{1}_{\{p \in S_r\}} \frac{30w_r(x_r - y_r)_+}{\mu_r + \delta(\mu_r + x_r - y_r)} + \sum_{i \neq r} \frac{5w_i(x_i - y_i)_+ \mu'_i}{\mu_i + \delta(\mu_i + x_i - y_i)} \\ &\quad - \mathbb{1}_{\{y_r < x_r + \mu_r\}} 4w_r \\ &\leq -\frac{1}{B} + \mathbb{1}_{\{y_r < p\}} w_r O(\log \ell) + \ell\epsilon', \end{aligned}$$

where in the first step the term $\mathbb{1}_{\{y_r < p\}} w_r O(\log \ell)$ absorbs any other terms of the form $\mathbb{1}_{\{y_r < p\}} O(1)w_r$.

The competitive ratio. By the bounds in Section 4.1, when y is changing, the increase in potential is $O(\log \ell)$ times the offline cost, and we can charge additional cost at rate $\mathbb{1}_{y_r < p} 8w_r$ to the offline algorithm while the pointer p is moving. By Lemma 4.1, the online algorithm suffers pseudo-cost at rate $1/B$ while it is moving. Thus, inequality (4.3) follows from the bound on the increase in potential above. Integrating over time, we get

$$\begin{aligned} \text{On} &\leq O(\log \ell) \text{Off} + O(\ell)\epsilon + O_{\ell, w, n}(1) \\ &\leq O(\log \ell)(3\text{OPT} + \epsilon) + O(\ell)\epsilon + O_{\ell, w, n}(1) \\ &\leq O(\log \ell + \ell\epsilon/\text{OPT}) \cdot \text{OPT} + O_{\ell, w, n}(1), \end{aligned}$$

where the second inequality uses Lemma 2.3 and $O_{\ell, w, n}(1)$ denotes any constants that may depend on ℓ, n , the weights, but are independent of the input sequence. We conclude that our algorithm is $O(\log \ell)$ -competitive in case of perfect predictions and $O(\log \ell + \ell\epsilon/\text{OPT})$ -competitive in general.

References

- [1] D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. *TCS*, 234(1-2):203–218, 2000.
- [2] S. Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, 1997.
- [3] S. Albers, L. M. Favrholt, and O. Giel. On paging with locality of reference. *JCSS*, 70(2):145–175, 2005.
- [4] S. Angelopoulos, R. Dorriv, and A. López-Ortiz. On the separation and equivalence of paging strategies. In *SODA*, pages 229–237, 2007.
- [5] S. Angelopoulos, S. Kamali, and K. Shadmehi. Online bin packing with predictions, 2021. arXiv:2102.03311.
- [6] A. Antoniadis, C. Coester, M. Eliás, A. Polak, and B. Simon. Online metric algorithms with untrusted predictions. In *ICML*, 2020.
- [7] A. Antoniadis, T. Gouleakis, P. Kleer, and P. Koley. Secretary and online matching problems with machine learned advice. In *NeurIPS*, 2020.
- [8] É. Bamas, A. Maggiori, L. Rohwedder, and O. Svensson. Learning augmented energy minimization via speed scaling. In *NeurIPS*, 2020.
- [9] É. Bamas, A. Maggiori, and O. Svensson. The primal-dual method for learning augmented algorithms. In *NeurIPS*, 2020.
- [10] N. Bansal, N. Buchbinder, A. Madry, and J. Naor. A polylogarithmic-competitive algorithm for the k -server problem. *JACM*, 62(5):1–49, 2015.
- [11] N. Bansal, N. Buchbinder, and J. Naor. Randomized competitive algorithms for generalized caching. *SICOMP*, 41(2):391–414, 2012.
- [12] N. Bansal, N. Buchbinder, and J. S. Naor. A simple analysis for randomized online weighted paging. *Manuscript*, 2010.
- [13] N. Bansal, N. Buchbinder, and J. S. Naor. A primal-dual randomized algorithm for weighted paging. *JACM*, 59(4):19, 2012.
- [14] Y. Bartal and E. Koutsoupias. On the competitive ratio of the work function algorithm for the k -server problem. *TCS*, 324(2-3):337–345, 2004.

- [15] R. D. Barve, E. F. Grove, and J. S. Vitter. Application-controlled paging for a shared cache. *SICOMP*, 29(4):1290–1303, 2000.
- [16] L. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.
- [17] M. A. Bender, R. Ebrahimi, J. T. Fineman, G. Ghasemiesfeh, R. Johnson, and S. McCauley. Cache-adaptive algorithms. In *SODA*, pages 958–971, 2014.
- [18] A. Blum and C. Burch. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, 2000.
- [19] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- [20] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *JACM*, 39(4):745–763, 1992.
- [21] A. Borodin, P. Raghavan, S. Irani, and B. Schieber. Competitive paging with locality of reference. In *STOC*, pages 249–259, 1991.
- [22] D. Breslauer. On competitive on-line paging with lookahead. *TCS*, 209(1-2):365–375, 1998.
- [23] S. Bubeck, M. B. Cohen, J. R. Lee, and Y. T. Lee. Metrical task systems on trees via mirror descent and unfair gluing. In *SODA*, pages 89–97, 2019.
- [24] S. Bubeck, M. B. Cohen, Y. T. Lee, J. R. Lee, and A. Madry. k -server via multiscale entropic regularization. In *STOC*, pages 3–16, 2018.
- [25] N. Buchbinder, A. Gupta, M. Molinaro, and J. S. Naor. k -servers with a smile: Online algorithms via projections. In *SODA*, pages 98–116, 2019.
- [26] P. Cao, E. W. Felten, and K. Li. Application-controlled file caching policies. In *USENIX Summer*, pages 171–182, 1994.
- [27] M. Chrobak and L. L. Larmore. An optimal on-line algorithm for k -servers on trees. *SICOMP*, 20(1):144–148, 1991.
- [28] C. Coester and J. R. Lee. Pure entropic regularization for metrical task systems. In *COLT*, pages 835–848, 2019.
- [29] Y. Emek, S. Kutten, and Y. Shi. Online paging with a vanishing regret. In *ITCS*, pages 67:1–67:20, 2021.
- [30] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.
- [31] A. Fiat and M. Mendel. Truly online paging with locality of reference. In *FOCS*, pages 326–335, 1997.
- [32] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k -server algorithms. *JCSS*, 48(3):410–428, 1994.
- [33] S. Gollapudi and D. Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, pages 2319–2327, 2019.
- [34] S. Im, R. Kumar, M. Montazer Qaem, and M. Purohit. Non-clairvoyant scheduling with predictions. In *SPAA*, pages 285–294, 2021.
- [35] S. Irani. Competitive analysis of paging: A survey. In *In Proceedings of the Dagstuhl Seminar on Online Algorithms, Dagstuhl*, 1996.
- [36] S. Irani, A. R. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. *SICOMP*, 25(3):477–497, 1996.
- [37] Z. Jiang, D. Panigrahi, and K. Sun. Online algorithms for weighted paging with predictions. In *ICALP*, pages 69:1–69:18, 2020.
- [38] A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov paging. *SICOMP*, 30(3):906–922, 2000.
- [39] E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. *SICOMP*, 30(1):300–317, 2000.
- [40] R. Kumar, M. Purohit, and Z. Svitkina. Improving online algorithms using ML predictions. In *NeurIPS*, pages 9661–9670, 2018.
- [41] R. Kumar, M. Purohit, Z. Svitkina, and E. Vee. Interleaved caching with access graphs. In *SODA*, pages 1846–1858, 2019.
- [42] S. Lattanzi, T. Lavastida, B. Moseley, and S. Vassilvitskii. Online scheduling via learned weights. In *SODA*, pages 1859–1877, 2020.
- [43] T. Lavastida, B. Moseley, R. Ravi, and C. Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In *ESA*, volume 204, pages 59:1–59:17, 2021.
- [44] S. Li and J. Xian. Online unrelated machine load balancing with predictions revisited. In *ICML*, pages 6523–6532. PMLR, 2021.
- [45] T. Lykouris and S. Vassilvitskii. Competitive caching with machine learned advice. In *ICML*, pages 3302–3311, 2018.
- [46] M. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.
- [47] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(1-6):816–825, 1991.
- [48] M. Mitzenmacher. Scheduling with predictions and the price of misprediction. In *ITCS*, pages 14:1–14:18, 2020.
- [49] E. Peserico. Paging with dynamic memory capacity. In *STACS*, pages 56:1–56:18, 2019.
- [50] D. Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *SODA*, pages 1834–1845, 2020.
- [51] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *CACM*, 28(2):202–208, 1985.
- [52] A. Wei. Better and simpler learning-augmented online caching. In *APPROX/RANDOM*, pages 60:1–60:17, 2020.

- [53] N. Young. *Competitive paging and dual-guided on-line weighted caching and matching algorithms*. Princeton University, 1991.
- [54] N. Young. The k -server dual and loose competitiveness for paging. *Algorithmica*, 11:525–541, 1994.

A A simple analysis of Belady's FiF algorithm

We present a simple potential function argument for Belady's FiF algorithm for unweighted paging, that evicts the page whose next request is the farthest in the future.

We first set up some notation. At any time t , let $C(t)$ be the set of pages in the cache of FiF and $C^*(t)$ be those in the cache of some fixed offline optimum solution. At any time t , let us order the pages according to their next request (this order only depends on the request sequence and not on $C(t)$ or $C^*(t)$). This order evolves as follows. At time t , the page at position 1 is requested. It is then reinserted in some position v and the pages in positions $2, \dots, v$ previously move one position forward to $1, \dots, v-1$, while the pages in positions $v+1, \dots, n$ stay unchanged.

Let $\text{pos}(p, t) \in [n]$ denote the position of page p at time t in the order above. If a page needs to be evicted at time t , FiF evicts the page in $C(t)$ with the largest position. Let $n(s, t) = |\{p \mid \text{pos}(p, t) \geq s, p \in C(t)\}|$ denote the number of pages in the cache of FiF with position is at least s . Similarly, let $n^*(s, t) = |\{p \mid \text{pos}(p, t) \geq s, p \in C^*(t)\}|$. Let $e(s, t) = n(s, t) - n^*(s, t)$. We define the following potential function at time t

$$\Phi(t) = \max_s \{n(s, t) - n^*(s, t)\} = \max_s e(s, t).$$

In other words, this is maximum over all suffixes of the ordering, of the difference between pages from $C(t)$ and $C^*(t)$ in that suffix. Note that $n(1, t) = n^*(1, t) = k$, for all t and hence $\Phi(t) \geq 0$. For any function $f : [T] \rightarrow \mathbb{R}$, let $f'(t) := f(t) - f(t-1)$.

LEMMA A.1. *For any time t , let $\text{FiF}(t)$ and $\text{OPT}(t)$ denote the total cost incurred up to time t by FiF and the offline optimum respectively. Then,*

$$(A.1) \quad \text{FiF}'(t) + \Phi'(t) \leq \text{OPT}'(t).$$

Proof. We show that (A.1) holds in three steps: when offline serves the request at t , then when online serves the request, and finally when the ordering of pages changes. Let p be the request at time t .

OPT moves. When OPT serves p , if it evicts some page q , then $\Phi(t)$ increases by at most 1, as $n^*(s, t)$ changes by at most 1 for any s . So, $\Phi'(t) \leq 1 = \text{OPT}'(t)$. On the other hand, if OPT(t) does not evict any page, then $\Phi'(t) = \text{OPT}'(t) = 0$.

FiF moves. If p is already in the cache, then $\text{FiF}'(t) = \Phi'(t) = 0$. So, we assume that p (of position 1) is not in FiF's cache. But as this page is in OPT's cache (as it has just served the request), we have $e(2, t) = k - (k-1) = 1$, and hence $\Phi(t) \geq 1$. Let $r = \max\{\text{pos}(p, t) \mid p \in C(t)\}$ be the position of the farthest in future page in FiF's cache. By definition, we have $e(s, t) \leq 0$, $\forall s \geq r+1$. As FiF evicts page q with $\text{pos}(q, t) = r$, $e(s, t)$ decreases by 1 for all $s \in [2, r]$, and thus $\Phi'(t) = -1$. Thus, $\text{FiF}'(t) + \Phi'(t) = 0$ as desired.

Finally, consider when t gets incremented. As p is inserted in position v : position of p becomes v , the positions of pages in positions $2, \dots, v$ decrease by 1, and pages with positions $\geq v+1$ stay unchanged. As p lies both in the offline and online cache (as it was just served), for every s both $n(s, t)$ and $n^*(s, t)$ change by the same amount and hence $\Phi'(t) = 0$. \square

THEOREM A.1. *FiF is 1-competitive for unweighted paging.*

Proof. The optimality of FiF now follows directly from Lemma A.1 as $\Phi(t) \geq 0$ for all t , and

$$\text{FiF}(t) = \sum_{\tau=1}^t \text{FiF}'(\tau) \leq \sum_{\tau=1}^t (\text{OPT}'(\tau) - \Phi'(\tau)) = \text{OPT}(t) - \Phi(t) \leq \text{OPT}(t).$$

\square

B Deterministic algorithm

We now give a natural extension of the FIF algorithm to the weighted case that yields an ℓ -competitive deterministic algorithm for learning-augmented weighted paging with perfect predictions (Theorem 1.1), and an $\ell + 2\ell\epsilon/\text{OPT}$ -competitive deterministic algorithm in the case of imperfect predictions. Combined with any k -competitive online algorithm for weighted paging [27, 54, 14] using the method of [32, 6] to deterministically combine several online algorithms, this yields Theorem 1.3.

Let $\text{pos}_i(p, t)$ denote the position of page p among all pages of weight w_i when the pages are sorted by the time of their predicted next request, just before the t -th request.

Algorithm. For each weight class i , we maintain a water-level $\alpha_i(t)$ that is initialized to w_i . At any time t when a page eviction is necessary, the page to evict from cache is decided as follows.

Let $J(t) \subseteq [\ell]$ be the set of weight classes from which the algorithm holds at least one page in its cache. Let $i' = \arg \min_{i \in J(t)} \alpha_i(t)$ be the weight class among them with the least level (ties broken arbitrarily). Then we evict the page from weight class i' with the highest position and set the levels as

$$\alpha_i(t+1) = \begin{cases} w_{i'} & i = i' \\ \alpha_i(t) - \alpha_{i'}(t) & i \in J(t) \setminus \{i'\} \\ \alpha_i(t) & i \notin J(t). \end{cases}$$

In other words, the level of the class i' from which the page is evicted is reset to $w_{i'}$, and the levels for all other classes with at least one page in the cache are decreased by $\alpha_{i'}(t)$.

Potential function analysis. For any weight class i , let $C_i(t)$ and $C_i^*(t)$ be the set of pages of weight w_i maintained in cache by the online algorithm and some optimal offline algorithm, respectively, just before serving the request for time t . Let $n_i(s, t) = |\{p \in C_i(t) \mid \text{pos}_i(p, t) \geq s\}|$ denote the number of pages of weight w_i in the cache of the online algorithm whose position is at least s . Similarly, let $n_i^*(s, t) = |\{p \in C_i^*(t) \mid \text{pos}_i(p, t) \geq s\}|$.

Let $e_i(t) = \max_s (n_i(s, t) - n_i^*(s, t))$ denote the “excess” for weight class i . We note that $e_i(t) \geq 0$, since s can be chosen greater than the maximal position so that $n_i(s, t) = n_i^*(s, t) = 0$. For each weight class i , we also define a term $\beta_i(t)$ as follows.

$$(B.2) \quad \beta_i(t) = \begin{cases} w_i(e_i(t) - 1) + \alpha_i(t) & \text{if } e_i(t) \geq 1 \\ 0 & \text{otherwise } (e_i(t) = 0). \end{cases}$$

We define the following potential function at time t :

$$\Phi(t) = \sum_{i \in [\ell]} \ell \beta_i(t) - \alpha_i(t).$$

THEOREM B.1. *The algorithm is $\ell + 2\ell\epsilon/\text{OPT}$ -competitive for learning-augmented weighted paging.*

Proof. It suffices to show for each time step that

$$(B.3) \quad \Delta \text{On} + \Delta \Phi \leq \ell \Delta \text{OPT} + 2\ell \Delta \epsilon,$$

where ΔOn and ΔOPT denote the cost incurred in this time step by the online algorithm and the optimum offline algorithm respectively, $\Delta \Phi$ is the associated change in potential, and $\Delta \epsilon$ is the increase of the prediction error ϵ .

Consider any fixed time step t where page σ_t is requested, and let r denote the weight class of σ_t . Note that either the prediction is correct and $\text{pos}_r(\sigma_t, t) = 1$ or otherwise $\Delta \epsilon = w_r$. For ease of analysis, we consider the events at time t in three stages and will show that (B.3) holds for each of them: (1) First the offline algorithm serves the page request, then (2) the online algorithm serves the request and a possible increase of ϵ is charged, and finally (3) the positions of pages in weight class r are updated.

1. If σ_t is already in the offline cache, then (B.3) holds trivially. Otherwise, let j be the weight class from which the offline algorithm evicts a page. Then e_j can increase by at most 1 and no other e_i can increase, so $\Delta \Phi \leq \ell w_j$. As $\Delta \text{OPT} = w_j$, (B.3) holds.

2. Now consider the actions of the online algorithm. If σ_t is already in the online cache, then (B.3) holds trivially. So suppose the requested page is not in the cache, and let $i' = \arg \min_{i \in J(t)} \alpha_i(t)$ be the class from which the online algorithm evicts a page. We consider the following three substeps: (a) level α_i of each class $i \in J(t)$ is decreased by $\alpha_{i'}(t)$, (b) a page is evicted from class i' and $\alpha_{i'}$ is reset from 0 to $w_{i'}$, and finally (c) the requested page σ_t is fetched into cache. Since $\Delta \text{On} = w_{i'}$, to get (B.3) it suffices to show that $\Delta \Phi \leq \ell \Delta \epsilon$ in steps (a) and (c), and $w_{i'} + \Delta \Phi \leq 0$ in step (b).

(a) If there exists a class $i \in J(t)$ with $e_i(t) \geq 1$: The decrease of α_i by $\alpha_{i'}(t)$ contributes $-\ell \alpha_{i'}(t)$ to $\Delta \Phi$ due to the first term in Φ . The second term can increase by at most $\alpha_{i'}(t)$ for each class, so overall $\Delta \Phi \leq -\ell \alpha_{i'}(t) + \ell \alpha_{i'}(t) = 0$.

Otherwise, we have $e_i(t) = 0$ for each $i \in J(t)$, and clearly $e_i(t) = 0$ holds also for $i \notin J(t)$. In particular, the online and offline algorithm have the same number of pages in cache from each class. Since the offline cache contains page σ_t , this means that also the online cache must contain some page from class r , so $r \in J(t)$. Then $\Delta \Phi \leq \ell \alpha_{i'}(t) \leq \ell \alpha_r(t) \leq \ell w_r$. To conclude this step, it suffices to show that $\Delta \epsilon = w_r$. Suppose not, then $\text{pos}_r(\sigma_t, t) = 1$. But then the fact that σ_t is in the offline but not the online cache and that they have the same number of pages from class r in their cache would imply that $e_r(t) \geq n_r(2, t) - n_r^*(2, t) = 1$, a contradiction.

(b) We claim that $\Delta \Phi = -w_{i'}$. As $\alpha_{i'}$ is reset from 0 to $w_{i'}$, the second term in Φ contributes $-w_{i'}$ to $\Delta \Phi$. If $e_{i'}(t) \geq 1$, then $e_{i'}$ decreases by 1 upon the eviction because the evicted page has maximum position, and as $\alpha_{i'}$ changes from 0 to $w_{i'}$, $\beta_{i'}$ stays unchanged. If $e_{i'}(t) = 0$, $\beta_{i'}$ does not change anyways.

(c) If $\text{pos}_r(\sigma_t, t) = 1$, then fetching σ_t to the online cache does not change e_r (as σ_t was already in the offline cache) and therefore $\Delta \Phi = 0$. Otherwise, we have $\Delta \epsilon = w_r$ and e_r increases by at most 1, so $\Delta \Phi \leq \ell w_r = \ell \Delta \epsilon$.

3. Finally, when page σ_t is re-inserted in some position of the predicted order, this does not affect e_r because page σ_t is in both the online and offline cache. Therefore, $\Delta \Phi = 0$ and (B.3) holds.

The theorem now follows by summing up (B.3) over all time steps. \square

C Missing proofs from Section 2

LEMMA C.1. (CONSISTENCY, REPEATED LEMMA 2.1) *Let C_0^1, C_0^2, \dots be any initial configurations, satisfying $C_0^m \subset C_0^{m+1}$ for all m . Then for any sequences σ, τ , for all times t and all $m \geq 0$, we have $C_t^m(\sigma, \tau) \subset C_t^{m+1}(\sigma, \tau)$.*

Proof. We use induction on time t . The base case for $t = 0$ holds by assumption. For the induction step, let p be the page requested at time $t + 1$, and let a denote the smallest index such that $p \in C_t^a$.⁸ By the inductive hypothesis, as $C_t^m \subset C_t^{m+1}$ for all m , we have that p lies in C_t^m for all $m \geq a$, and none of these caches incur a page fault. So $C_{t+1}^m = C_t^m$ and the property $C_{t+1}^m \subset C_{t+1}^{m+1}$ for all $m \geq a$ is maintained.

For $m < a$, each cache C_t^m evicts its page with the farthest predicted re-arrival time and fetches p . Let us consider this in two steps. First, adding p to each C_t^m for $m < a$ maintains the property that $C_{t+1}^m \subset C_{t+1}^{m+1}$ for all m , since $p \in C_{t+1}^a$. Let us now consider the eviction step. Fix some $m < a$, and suppose C_t^m evicts q . If $q \in C_t^m$, then as $C_t^m \subset C_t^{m+1}$ by the inductive hypothesis, q is also the page with the farthest predicted re-arrival time in C_t^m and hence evicted from C_t^m . Otherwise $q \notin C_t^m$ and some other page q' is evicted from C_t^m . In either case, $C_{t+1}^m \subset C_{t+1}^{m+1}$ is maintained for all $m < a$. \square

LEMMA C.2. (REPEATED LEMMA 2.3) *Let A be an arbitrary (offline) weighted paging algorithm, and let $x_i(t)$ denote the cache space used by class i at time t under A . For any arbitrary σ, τ , let A^* be the trustful algorithm with configuration $\bigcup_{i=1}^{\ell} C_{i,t}^{x_i(t)}(\sigma, \tau)$ at any time t . Then, $\text{cost}_{A^*}(\sigma, \tau) \leq 3 \cdot \text{cost}_A(\sigma) + \epsilon(\sigma, \tau) + O(1)$.*

Proof. We will use a potential function for analysis. For any weight class i at any time t , we order all pages of class i in increasing order of the predicted arrival time of their next requests (breaking ties in the same way as BELPRED). We call the position of pages in this ordering the *predicted position*. Note that this predicted position of pages within a weight class is very different from the *rank* of pages defined in Section 2.1. We observe the following property: at each time step t , when a page σ_t from some weight class i is requested, either σ_t has predicted position 1 in its class or the request contributes to the prediction error ϵ_i .

For any integer s , let $n_i(s)$ denote the total number of pages in the cache of algorithm A with predicted position at least s . Let $n_i^*(s)$ denote the respective quantity for algorithm A^* . We note that these quantities vary with time t , but we suppress the dependence in the notation for brevity. Let $\Phi_i := \max_s n_i^*(s) - n_i(s)$ and consider the potential function

$$\Phi := 2 \sum_{i=1}^{\ell} w_i \Phi_i.$$

We consider the setting where algorithms pay cost w_i whenever they evict or fetch a page of weight class i . As this doubles the cost of any algorithm compared to the original setting where algorithms only pay for evictions, up to an additive constant, it suffices to show that for each request,

$$(C.4) \quad \Delta \text{cost}_{A^*} + \Delta \Phi \leq 3 \Delta \text{cost}_A + 2 \Delta \epsilon,$$

where Δcost_{A^*} and Δcost_A are the costs incurred for this request, $\Delta \Phi$ is the change in potential and $\Delta \epsilon$ is the increase of ϵ .

For any request to some page p from weight class i , we break the analysis into three steps: (1) First A serves the request and A^* updates its cache accordingly with respect to the *old* ranks of each weight class. (2) Then A^* updates its cache content to reflect the new ranks of weight class i . (3) Finally page p might move to a later position in the predicted order for class i . In each step, we will show that inequality (C.4) is satisfied.

In step (1), suppose A evicts page q from some weight class j . In this case, $\Delta \text{cost}_A = w_i + w_j$ and $\Delta \text{cost}_{A^*} \leq w_i + w_j$. Moreover, both Φ_i and Φ_j increase by at most 1 and hence $\Delta \Phi \leq 2(w_i + w_j)$ and thus the inequality is maintained.

In step (2), after page p is requested the ranks of pages of weight class i change according to Lemma 2.2 (see Figure 1). In particular, p moves to rank 1 and, if p is not in cache yet, p is fetched and the page in cache from class i with the highest predicted position gets evicted. Let q be the evicted page. When A^* fetches page p and evicts q , it incurs a cost of $\Delta \text{cost}_{A^*} = 2w_i$. We analyze the change in potential $\Delta \Phi$ and error $\Delta \epsilon$ due to fetching of page p and evicting q separately.

Let s be such that $\Phi_i = n_i^*(s) - n_i(s)$. We observe that the predicted position of page q must be at least s (since otherwise, we would have $n_i^*(s) = 0$ and hence $\Phi_i \leq 0$, but $\Phi_i \geq n_i^*(1) - n_i(1) = 0$). Thus, evicting q decreases Φ_i

⁸Note that such an index a must exist since we are guaranteed that $r \in C_t^n$ where n is the total number of pages.

by 1 and we have $\Delta\Phi = -2w_i$ and inequality (C.4) is maintained. To account for the change in potential due to fetching page p , we consider separately the cases that p has (old) predicted position 1 in class i or not. In the former case, we have $s \geq 2$ and hence the potential does not change. Otherwise, Φ_i increases by at most 1, so $\Delta\Phi \leq 2w_i$. However, in this case, the prediction error ϵ_i also increments and we have $\Delta\epsilon = w_i$ and inequality (C.4) is maintained.

Finally in step (3), when page p is re-inserted in some position of the predicted order, the potential is not affected since now p is present in the cache of both algorithms. \square

REMARK C.1. *A slight modification of the proof of this lemma yields a much simpler proof of the result from [52] that in unweighted paging, BELPRED(k) has competitive ratio at most $1 + \epsilon/\text{OPT}$: In this case, we have a single weight class and $x_1(t) = k$ remains fixed. Therefore, $A^* = \text{BELPRED}(k)$ does nothing in step (1), and we can avoid losing a factor 3 by considering the setting where algorithms are charged only for evictions (not for fetching) and omitting the factor 2 in the definition of Φ . The proof of this result in [52] uses a case analysis involving eleven cases.*

LEMMA C.3. (REPEAT PROPERTY, REPEATED LEMMA 2.4) *Let $\epsilon = 0$ and let i be a weight class. A rank sequence corresponds to a request sequence of pages of class i if and only if it has the following repeat property: for any h , between any two requests to the same rank h , every rank $2, \dots, h-1$ must be requested at least once.*

Proof. Lemma 2.5 shows that if $\epsilon = 0$, any rank sequence must satisfy the repeat property.

Conversely, we show that for any sequence h_1, h_2, \dots satisfying the repeat property, there exists a corresponding paging request sequence r_1, r_2, \dots . Let $n := \max_t h_t$ be the number of distinct pages. We construct the request sequence online by specifying, whenever a page is requested, the time when the same page will be requested next.

We will maintain the invariant that for each t and integer $m = 1, \dots, n$, after the request at time t , the page with rank m has next-request time given as follows.

$$\begin{aligned} \inf\{t' > t : h_{t'} = m\} & \quad \text{if } m \geq 2 \text{ or } m = h_{t+1}, \\ \inf\{t' > t+1 : h_{t'} = h_{t+1}\} & \quad \text{if } m = 1 \text{ and } m \neq h_{t+1}. \end{aligned}$$

The invariant ensures that the next request will be to the page in position h_{t+1} , as required. It also implies that different pages have different next-request times. It remains to show that we can maintain this invariant over time. We can satisfy the invariant initially by defining the first-request times of the pages according to the condition of the invariant for $t = 0$.

Suppose the invariant holds for some t . By the invariant, the page r_{t+1} requested at time $t+1$ is the one with rank h_{t+1} . We define the next request time to page r_{t+1} to be $t+2$ if $h_{t+2} = 1$ and $\inf\{t' > t+2 : h_{t'} = h_{t+2}\}$ if $h_{t+2} \geq 2$. Since r_{t+1} will receive new rank 1, we see that the condition for $m = 1$ of the invariant is satisfied for the next time step. If $h_{t+1} = 1$, then the ranks remain unchanged and the invariant continues to be satisfied. So suppose $h_{t+1} \geq 2$. By assumption on the sequence h_1, h_2, \dots , for each $m = 2, \dots, h_{t+1} - 1$ we have

$$\inf\{t' > t+1 : h_{t'} = h_{t+1}\} \geq \inf\{t' > t+1 : h_{t'} = m\} = \inf\{t' > t : h_{t'} = m\},$$

with the inequality being strict unless both sides are ∞ . Thus, by the invariant, the page previously in position 1 is the one with the farthest next-request time among the pages in positions $1, 2, \dots, h_{t+1} - 1$. so the new ranks are the same as the old ones except that the pages with ranks 1 and h_{t+1} swap, and it is directly verified that the invariant is again satisfied at the next time step. \square