Tight Results for Online Convex Paging

Anupam Gupta
New York University
New York, USA
anupam.g@nyu.edu

Amit Kumar
IIT Delhi
New Delhi, India
amitk@cse.iitd.ac.in

Debmalya Panigrahi
Duke University
Durham, USA
debmalya@cs.duke.edu

Abstract

Online convex paging (Menache and Singh, 2015; Chiplunkar, Henzinger, Kale, and Vötsch, 2023) models a broad class of cost functions for the classical paging problem. In particular, it naturally captures fairness constraints: e.g., that no specific page (or groups of pages) suffers an "unfairly" high number of evictions by considering ℓ_p norms of eviction vectors for $p > 1$. The case of the ℓ_∞ norm has also been of special interest, and is called *min-max paging*.

We give tight upper and lower bounds for the convex paging problem for a broad class of convex functions. Prior to our work, only fractional algorithms were known for this general setting. Moreover, our general result also improves on prior works for special cases of the problem. For example, it implies that the randomized competitive ratio of the min-max paging problem is $\Theta(\log k \log n)$; this improves both the upper bound and the lower bound given in prior work. It also shows that the randomized and deterministic competitive ratios for ℓ_p-norm paging are $\Theta(p \log k)$ and $\Theta(pk)$ respectively; the randomized results are completely new, as is the deterministic lower bound.

All previous algorithms we know for paging with non-linear costs used fractional relaxations. We show a fundamental limitation of this approach — we give integrality gap instances for the natural relaxation used in these works. This shows that a generic relax-and-round framework—solving the relaxation and then rounding it—is insufficient for obtaining tight bounds for this problem.

To bypass this bottleneck, we work with the integer versions of the problems directly. Somewhat surprisingly, we show how to take an *arbitrary* online algorithm for the weighted paging problem (with linear costs), and convert it in a black-box way to an online algorithm for convex paging, losing just an optimal factor in this reduction. This reduction proves especially challenging in the randomized case, where the underlying weighted paging algorithm is randomized, and the analysis needs to proceed via a delicate martingale argument. We believe this approach of lifting arbitrary (weighted linear) online algorithms to convex objectives may be of broader interest.

CCS Concepts

• **Theory of computation → Caching and paging algorithms**; **Online algorithms**.

STOC '25, Prague, Czechia

ACM ISBN 979-8-4007-1510-5/25/06
https://doi.org/10.1145/3717823.3718217

Keywords

Online Paging, Convex Objectives

ACM Reference Format:
Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. 2025. Tight Results for Online Convex Paging. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25), June 23–27, 2025, Prague, Czechia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3717823.3718217

1 Introduction

Paging/caching is perhaps the most well-studied problem in online algorithms. Given a universe of n pages and a cache that can only hold any k pages at any time, a paging algorithm must serve an online sequence of page requests by ensuring that the currently requested page is in the cache. Traditionally, the goal has been to minimize the total number of page evictions, i.e., the ℓ_1-norm of the *page eviction vector*. For this problem, we have long known tight k-competitive deterministic and H_k-competitive randomized algorithms.

In recent years, there has been increased focus on paging applications in decentralized settings, such as in cloud computing and web caching. One key difference in these applications is that the different pages (e.g., web pages) are owned/requested by different agents. It is natural to ask that any paging algorithm be *fair* to these agents, and not evict pages for any one agent too often (see e.g. [24, 26]). Motivated by these considerations, Chiplunkar, Henzinger, Kale, and Vötsch [11] defined the *min-max paging* problem. Here the goal is to minimize the *maximum* number of evictions suffered by any page, i.e., the ℓ_∞-norm of the page eviction vector. They gave an upper bound of $O(\log^2 n \log k)$ and a lower bound of $\Omega(\log n)$ on the competitive ratio of min-max paging; this left an $O(\log n \log k)$ gap between these upper and lower bounds.

Chiplunkar *et al.* [11] also considered the more general setting of minimizing an arbitrary convex function $g(\mathbf{x})$ of the n-dimensional page eviction vector $\mathbf{x}$ (where the i^{th} coordinate represents the eviction count for page i). They focused on the widely-studied class of p-bounded of convex functions g, which contains the p^{th} moments—i.e., the functions $\sum_i x_i^p$—as a special case. (See Section 2.1 for a formal definition.) For such functions, they gave $(O(p \log k))^p$-competitive *fractional* paging algorithms. As a corollary, this implied an $O(p \log k)$-competitive algorithm for the *fractional* ℓ_p-norm paging problem, where the goal is to minimize $\|\mathbf{x}\|_p$. [1] Prior to our work, no such results for the *integral* convex paging problem were known.

It turns out that the extension of paging to ℓ_p-norms had previously been studied from both theoretical and empirical standpoints

[1] Since the ℓ_p norms for n-dimensional space with values of $p \geq \ln n$ are within constant factors of each other, we can restrict our attention to $p \leq \ln n$.

by Menache and Singh [23]. They gave an $O(pk)$-competitive *deterministic* algorithm for this problem, leaving open the question of getting a matching lower bound for deterministic algorithms, as well as better results for randomized algorithms.

Given these past results, the following question remains open:

> *Can we get tight results for the minmax paging and ℓ_p-norm paging problems, or the more general p-bounded convex paging problem, both in the randomized and deterministic cases?*

1.1 Our Results

In this work, we resolve this question, giving matching upper bound and lower bounds for all these problems using a broad new framework.

1.1.1 Randomized Algorithms. Our main result shows a black-box transformation from a randomized online algorithm for classical weighted ℓ_1 paging to a corresponding randomized online algorithm for p-bounded convex paging.

Theorem 1.1 (Randomized Upper Bound). *Suppose there is a randomized online algorithm for weighted paging with competitive ratio β. Then, there is a randomized $(O(p\beta))^p$-competitive online algorithm for the p-bounded convex paging problem. Using known randomized $O(\log k)$-competitive weight paging algorithms, this implies the following randomized online algorithms:*

1. *an $O(\log n \log k)$-competitive algorithm for the min-max paging problem, and*
2. *an $O(p \log k)$-competitive algorithm for the ℓ_p-norm paging problem.*

Our min-max paging result improves on the $O(\log^2 n \log k)$-competitive algorithm given by [11]. No previous randomized algorithm was known for the generalization to ℓ_p-norms (and hence, for p-bounded convex paging).

Next, we show our matching lower bounds. In fact, our lower bounds hold for the most restrictive settings of min-max paging and ℓ_p-norm paging—and hence for p-bounded convex functions as well.

Theorem 1.2 (Randomized Lower Bound). *There exists a constant $C > 0$ such that for any $p \le C \ln n$, any randomized algorithm (that is even allowed to produce a fractional solution) for ℓ_p-norm paging has an $\Omega(p \log k)$ competitive ratio against an optimal integer solution. This result implies an $\Omega(\log n \log k)$ lower bound for randomized min-max paging.*

This lower bound improves and generalizes a lower bound of $\Omega(\log n)$ given by [11] for the integral min-max paging problem. Moreover, it shows that for min-max paging, both our randomized (integral) algorithm from Theorem 1.1, as well as the $O(\log k \log n)$-competitive (fractional) algorithm of [11] are asymptotically tight. For ℓ_p-norm paging, to our knowledge, this is the first lower bound result.

1.2 Integrality Gap

Before we present the analogous reductions for deterministic algorithms, let us address a question that immediately arises when faced with these problems:

> *Could we get the randomized algorithm of Theorem 1.1, by applying the relax-and-round framework, e.g., for the special case of min-max paging? In particular, could we take the $O(\log k \log n)$-competitive fractional solutions from [11] (or Theorem 1.4), and round these online with only a constant factor loss?*

Indeed, relax-and-round has been the dominant paradigm for paging problems, as well as for generalizations such as weighted paging and k-server. However, we show a significant barrier: starting with a generic fractional solution to the natural convex relaxation and rounding it must lose a nearly-logarithmic factor to get from fractional to integer solutions. Indeed, the integrality gap of the natural linear programming formulation is nearly logarithmic.

Theorem 1.3 (Integrality Gap). *There exist instances of min-max paging where the natural linear program has an integrality gap of $\Omega(\log n / \log\log n)$. Further, for any p, $2 \le p \le \ln n$, there exist instances of ℓ_p-norm paging where the natural convex program has an integrality gap of $\Omega(p/\log p)$.*

Observe that Theorem 1.2 and Theorem 1.3 together imply lower bounds on the relax-and-round framework— any (randomized) online algorithm for min-max paging using the relax-and-round framework must have $\Omega(\log k \log^2 n / \log\log n)$-competitive ratio (and similarly, $\Omega(\log k p^2/\log p)$ competitive ratio for ℓ_p-norm paging). We manage to bypass the challenge of overcoming this integrality gap in Theorem 1.1 by avoiding working with fractional solutions entirely, and using a direct reduction between integral paging algorithms.

1.3 Deterministic Algorithms

Finally, we show that our techniques also extend to deterministic algorithms: we give a black-box transformation from a deterministic (fractional or integral) online algorithm for the ℓ_1 weighted paging problem to a corresponding online algorithm for the p-bounded convex paging objective.

Theorem 1.4 (Deterministic Upper Bound). *Suppose there is a deterministic α-competitive online algorithm $\mathbb{A}$ for the weighted paging problem. Then there is an $(O(p\alpha))^p$-competitive deterministic online algorithm for the p-bounded convex paging problem. This algorithm yields a fractional or an integral solution depending on whether $\mathbb{A}$ is fractional or integral.*

Given deterministic fractional $O(\log k)$-competitive [6] and integral $O(k)$-competitive [12] algorithms for weighted paging, we recover two known results for ℓ_p-norm paging:

(1) an $O(p \log k)$-competitive fractional algorithm [11], and
(2) an $O(pk)$-competitive integral algorithm [23].

The first result is tight by Theorem 1.2. We also give a new lower bound matching the second result.

Theorem 1.5 (Deterministic Lower Bound). *There exists a constant $C > 0$, such that for any $p \le C \ln n$, any deterministic integral online algorithm for ℓ_p-norm paging must have $\Omega(pk)$ competitive ratio. This implies an $\Omega(k \ln n)$ lower bound for deterministic min-max paging.*

For general p, we are not aware of any prior lower bound. For (integral) min-max paging, the best previous deterministic lower bound was $\Omega(k \log n/\log k)$ [11], which we improve by $\log k$.

1.4 Our Techniques

Algorithms. First, we describe the *deterministic* reduction of Theorem 1.4, given in Section 3. Recall that we want to reduce paging with a convex objective g to a series of weighted ℓ_1 paging instances. A natural idea is to define the weight of page i as the marginal cost of evicting a single page given the current situation, i.e., the gradient $\nabla g(\mathbf{x})$ w.r.t. the page eviction vector $\mathbf{x}$. This allows the weighted paging objective to locally track the growth of the convex objective. However, if we strictly implemented this, we would have to update page weights after every eviction, and run a new weighted paging instance which terminates upon a single eviction. Clearly, this is infeasible because we would not be able to account for the (additive) cost incurred by the black-box weighted paging algorithm, e.g., for resetting its initial state. To avoid this, we need to let the weighted paging instance run until its additive cost can be charged to the cost of an optimal solution with the same set of page weights. But, this means that the page weights determined at the beginning of an instance do not accurately reflect the growth of $g(\cdot)$ later on in the same instance.

To reconcile these differences, we need two ideas. For any page, there is an initial period of time where we do not maintain explicit control over how the gradient changes within a weighted paging instance. Instead, we ensure that the total weighted cost of evictions of the page in this initial time period is bounded. Once a page crosses this initial threshold, we switch to maintaining explicit control over the change in gradient within a weighted paging instance; in particular, the gradient at the end is at most twice the gradient at the beginning of the instance. These phases for different pages are asynchronous, which makes the analysis more subtle. Moreover we need to introduce extra lags in this process to control the number of phases; we give these details in Section 3.

The randomized reduction (Theorem 1.1) is more involved. Ideally, we would like to use the same strategy as above, while replacing the deterministic weighted paging algorithm with a randomized one. However, each weighted paging instance is a function of the algorithm's choices in the previous instances, i.e., is itself random. Moreover, the bound on the cost of a weighted paging instance only holds in expectation, and with some probability an instance can generate an expensive solution. Crucially, this can have a cascading effect, raising the (optimal) cost of later instances. We counter this by terminating an instance if the weighted paging cost exceeds a set threshold. The key idea is to show that this extra termination condition does not increase the total number of instances significantly, since the latter would make the additive cost of the instances unacceptable. We use a careful martingale argument to bound the probability of this bad event, which allows us to restart the algorithm every time this bad event happens, i.e., if the algorithm is creating too many weighted paging instances. The details of the randomized reduction appear in Section 4.

Lower Bounds. We outline the main ideas for the randomized lower bound for fractional min-max paging (Theorem 1.2); the ideas for the deterministic lower bound are similar. We start off with $n \gg k$ pages, which are grouped arbitrarily into disjoint groups of $k+1$ pages each. In the first epoch, we iterate over all the $\frac{n}{k+1}$ groups. For each group we give some M requests for the $k+1$ pages in the group, each request for a uniformly random page in the group. Each such request causes an eviction with probability $1/k+1$, and hence any algorithm pays $\approx M/k^2$ for each page during all these requests. Now a random half of the pages in each group are deactivated, and the remaining half of the pages survive to the next epoch. There are $\log_2(n/(k+1)) \approx \log n$ such epochs.

Since the algorithm does not know the identity of the pages to be deactivated, a random page suffers approximately M/k^2 evictions, and hence pages that survive all the epochs suffer approximately $(M/k^2)\log n$ evictions. However, an optimal solution can save on two fronts. First, it can ensure that the $\frac{k+1}{2}$ pages that remain active in the following epoch are always kept in cache and hence suffer a constant number of evictions in this epoch. In other words, a page is subjected to a substantial number of evictions only in its final epoch; in this case, the $\frac{k+1}{2}$ such pages in its group vie for $\frac{k-1}{2}$ slots in the cache. The instance now behaves like a paging lower bound with $\approx M/2$ requests and $\frac{k+1}{2}$ pages: using a coupon-collector argument we know that the optimal solution suffers $\approx \frac{M/2}{(k/2)\log(k/2)}$ evictions overall, and hence a random page suffers $\approx M/(k^2 \log k)$ evictions in the optimal solution. Since a random page suffers $\approx (M/k^2)\log n$ evictions in the online algorithm, this gives the gap of $\Omega(\log k \log n)$. The formal argument for this lower bound, along with its extension to ℓ_p-norm paging, appears in Section 5; the modification to the deterministic case (Theorem 1.5) is deferred to the full version.

1.5 Other Related Works

In classical paging, Bélády's offline algorithm (Farthest in Future) is known to be optimal for minimising the number of evictions [9]. We know deterministic k-competitive and randomized $O(\log k)$-competitive algorithms for the caching problem; both are optimal [18, 25]. Weighted paging is equivalent to the k-server problem on a weighted star, so deterministic k-competitiveness follows from a k-server algorithm on trees [12]. Bansal et al. [6] gave a randomized $O(\log k)$-competitive algorithm for weighted paging, illustrating the power of the relax-and-round framework for these problems. They used an interval covering IP given by [8, 13], which also extends to the setting of ℓ_p norms.

Azar *et al.* [3] give an algorithm for solving online covering LPs with supermodular convex objective functions f; their approximation factor depends upon bounds on the derivatives of f. These ideas were extended by Chiplunkar et al. [11] to solve fractional ℓ_p-norm paging, where the natural LP also has box constraints. (Alternatively, we can write the stronger covering-only LP given in Section 2 and directly use the algorithm from [3] to get an $O(p \log k)$-competitive algorithm for the fractional ℓ_p-norm paging problem.) Menache and Singh [23] used KKT conditions on the natural convex program to obtain deterministic algorithms for paging with convex objectives.

The use of ℓ_p norms and other convex objectives to capture the notions of fairness and balance is widespread in both offline and online algorithms. E.g., see [1, 2, 4, 5, 7, 10, 14, 15, 17, 19–22].

Roadmap. We now give an overview of the rest of the paper. In Section 2, we give some preliminaries and notations, including formally defining the class of p-bounded convex functions. We prove Theorem 1.4 giving our deterministic algorithms for p-bounded convex functions in Section 3; this allows us to develop many of our ideas. In Section 4, we prove Theorem 1.1 by extending our ideas to randomized algorithms. We prove the lower bound result Theorem 1.2 for randomized online algorithms in Section 5; Theorem 1.5 for deterministic online algorithms apears in the full version of the paper. The integrality gap result Theorem 1.3 is also deferred to the full version. All missing proofs appear in the full version.

2 Preliminaries

We start with a formal definition of the paging problem. We have a universe U of n pages, and a cache of size k. At any given time, at most k of the n pages can be in the cache. Let the cache contents at time t be C_t. Requests for pages arrive online: the page requested at timestep t is denoted σ_t, and it must belong to the cache C_t at the end of time t. W.l.o.g., we assume that the algorithm does nothing if $\sigma_t \in C_{t-1}$, and hence $C_t \leftarrow C_{t-1}$. Otherwise, if $|C_{t-1}| = k$ and $\sigma_t \notin C_{t-1}$, the algorithm must *evict* some page $\sigma'_t \in C_{t-1}$ and replace it with page σ_t; i.e., $C_t \leftarrow (C_{t-1} \cup \{\sigma_t\}) \setminus \{\sigma'_t\}$. Let x_i^t denote the number of times that the page $i \in U$ is evicted from the cache by the algorithm until the end of time t, and let the eviction vector be $\mathbf{x}^t := (x_1^t, \ldots, x_n^t)$.

We consider the *convex paging* problem, that of minimizing $g(\mathbf{x}^t)$ for a convex function $g : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$. This captures the unweighted (ℓ_1*-norm paging*) problems [18, 25], where $g(\mathbf{x}^t) := \sum_i x_i^t$, and the *min-max paging* problem [11] where $g(\mathbf{x}^t) := \|\mathbf{x}^t\|_\infty = \max_i x_i^t$. Moreover, we may have the ℓ_p*-norm paging* problem, where $g(\mathbf{x}^t) := \|\mathbf{x}^t\|_p = \left(\sum_i (x_i^t)^p\right)^{1/p}$ for any $p \in [1, \infty]$. As mentioned in footnote 1, we focus our attention on values of $p \leq O(\log n)$, and the result for $p = O(\log n)$ gives us min-max paging.

The ℓ_p-norm paging problem can be further generalized to the weighted versions with $g(\mathbf{x}^t) := \left(\sum_i w_i (x_i^t)^p\right)^{1/p}$. In particular, the *weighted* ℓ_1*-norm paging* problem is $g(\mathbf{x}^t) := \sum_i w_i x_i^t$, and is well-studied in the literature [6, 12]. Secondly, the pages can be partitioned into c groups and the weighted ℓ_p-norm is now defined on the groups: $g(\mathbf{x}^t) := \left(\sum_c w_c \left(\sum_{i \in c} x_i^t\right)^p\right)^{1/p}$. The intuition is that each group represents a separate entity such a client in a cloud service, and the goal is to achieve fairness across these groups [23].

2.1 p-Bounded Convex Functions

All the cases above are modeled by the class of p-bounded convex functions. These were previously studied by [3, 11] (and called *nice* functions in [3]):

(i) g is convex, differentiable, monotone, with $g(\mathbf{0}) = 0$;
(ii) g is supermodular, which implies that ∇g are monotone, and
(iii) the linear approximation $\langle \nabla g(\mathbf{x}), \mathbf{x} \rangle$ is within a factor of p of the function value $g(\mathbf{x})$: formally,

$$\langle \nabla g(\mathbf{x}), \mathbf{x} \rangle \leq p\, g(\mathbf{x}) \text{ for all } \mathbf{x}. \tag{1}$$

Note that all the paging objectives considered above satisfy the properties of a p-bounded convex function, where the value of p coincides with the index of the ℓ_p-norm. The following properties hold for p-bounded convex functions (see [3, Lemma 4(a,d)]):

Lemma 2.1. *For a p-bounded convex function g:*

$$g(\delta \mathbf{x}) \leq \delta^p \cdot g(\mathbf{x}) \qquad \text{for any } \delta \geq 1 \tag{2}$$

$$g(\mathbf{x} + \mathbf{y}) \leq 2^p \cdot g\left(\tfrac{\mathbf{x}+\mathbf{y}}{2}\right) \leq 2^{p-1}(g(\mathbf{x}) + g(\mathbf{y})). \tag{3}$$

2.2 Weighted ℓ_1-Norm Paging

Our main contribution is to reduce convex paging to weighted ℓ_1-norm problem. In an instance $\mathbb{I}$ of the weighted ℓ_1-norm problem, each page i has a weight w_i, and the goal is to minimize the total weighted eviction cost of the pages. We will use algorithms for the weighted ℓ_1-norm problem in a black-box fashion, and assume one of the two below:

(i) An α-competitive deterministic algorithm $\mathbb{A}$: given an instance $\mathbb{I}$ of weighted ℓ_1-norm paging with page weight vector $\mathbf{w}$ and optimal solution $\mathbf{x}^*$, and an *arbitrary* starting configuration, the algorithm $\mathbb{A}$ produces an integral solution with objective value at most $\alpha \langle \mathbf{w}, \mathbf{x}^* \rangle + w_{\max} \cdot \gamma_n$, where $w_{\max}$ is the maximum weight, and the term γ_n depends only on the number of pages n and not on the input sequence.
(ii) A randomized algorithm $\mathbb{B}$ with expected competitive ratio β, i.e., in the same setting as above, it produces a solution with expected objective value at most $\beta \langle \mathbf{w}, \mathbf{x}^* \rangle + w_{\max} \cdot \gamma_n$.

2.3 Assumptions on opt

Given an instance $\mathbb{I}$ of the convex paging problem, we denote the optimal solution by opt($\mathbb{I}$), or opt when the instance is unambiguous. We assume that the algorithm knows the optimal objective function value $g(\text{opt})$, and that $g(\text{opt}) \geq B(n)$ for some large enough function of n that we will define later. Both assumptions are without loss of generality (see full version for details).

3 Deterministic Online Algorithm for Convex Paging

We now present our reduction for deterministic algorithms. This allows us to develop some of our techniques, which will subsequently be useful for the reduction for randomized algorithms.

3.1 The Intuition for Our Algorithm

Consider an instance $\mathbb{I}$ of p-bounded convex paging. Our algorithm divides time into stages. It maintains a solution $\mathbf{x}$, where x_i denotes the number of evictions of page i. Observe that if $\mathbf{x}$ is the final solution produced by the algorithm, then

$$g(\mathbf{x}) = \int_t \nabla g(\mathbf{x}^t) \cdot d\mathbf{x} \approx \sum_s \nabla g(\mathbf{x}_{\text{init}}^{(s)}) \cdot \Delta \mathbf{x}^{(s)},$$

where $\mathbf{x}^t$ denotes the solution $\mathbf{x}$ at time t, $\mathbf{x}_{\text{init}}^{(s)}$ the solution $\mathbf{x}$ at the beginning of stage s, and $\Delta \mathbf{x}^{(s)}$ the change in $\mathbf{x}$ during this stage. In other words, the cost incurred during a stage can be approximated by the weighted ℓ_1 paging cost with weight of page i given by $(\nabla g(\mathbf{x}_{\text{init}}^{(s)}))_i$. Hence it is natural for the algorithm to run the following greedy procedure: in each stage s, run the weighted

paging algorithm $\mathcal{A}$ with weights given by the gradient of function g at the *beginning* of this stage.

However, there is an inherent trade-off in this approach: On one hand, if a stage s runs for too long, $(\nabla g(\mathbf{x}))$ may differ significantly from the gradient $\nabla g(\mathbf{x}^{(s)}_{\text{init}})$ at the beginning of the stage, and hence the weighted paging cost with fixed weights may not reflect the true costs incurred during this stage. On the other hand, there is an initial set-up cost (and an additive term in the competitive ratio of $\mathcal{A}$) each time we start a new stage. If there are many stages, these set-up costs can be large fraction of the overall cost incurred by the algorithm. The challenge lies in finding the right balance between these two factors.

We adopt the following approach: each page i goes through two conceptual "phases". In the initial phase (for page i), we do not control how fast $(\nabla g(\mathbf{x}))_i$ grows within a stage, but ensure that the total weighted paging cost (over all stages so far) incurred for page i forms a small fraction of $g(\text{opt})$ (at most $p/n \cdot g(\text{opt})$ to be precise). Once this initial phase for page i is over, we ensure that $(\nabla g(\mathbf{x}))_i$ at most doubles during a stage. Formally, a stage ends due to one of two reasons: (i) the total weighted paging cost of a page i reaches the threshold $p/n \cdot g(\text{opt})$, or (ii) the gradient of g along page i, i.e., $(\nabla g(\mathbf{x}))_i$, doubles during this stage. There are at most n stages of the first type, and we show that the doubling process cannot happen too many times, thereby bounding the number of stages.

The algorithm has to overcome several additional technical issues to get our claimed guarantee: (i) Due to the factor p in (3) and the competitive ratio α of $\mathcal{A}$, we maintain a scaled down version $\mathbf{z} := \varepsilon \cdot \mathbf{x}$, where $1/\varepsilon \approx p\alpha$), and the weights of pages must be defined with respect to $\nabla g(\mathbf{z})$. (ii) It is possible that x_i does not change a lot during a stage (and hence the change is small compared to the stage set-up cost), but $(\nabla g(x))_i$ changes by almost a factor of two. In such cases, we do not want to change the weight of page i. To ensure this, we change z_i, which dictates the weight of page i, only when x_i changes by a quantity greater than some threshold. We now present the formal description, which pins down these and other subtleties.

3.2 Algorithm Description

The *page eviction* vector is denoted $\mathbf{x} \in \mathbb{R}^n_+$, i.e., x_i is the total number of evictions of page i. We define an auxiliary vector $\mathbf{z}$, which is essentially an ε-scaled version of $\mathbf{x}$, but which omits some of the page evictions. The pages in $[n]$ are partitioned into two subsets J_L and J_S, loosely corresponding to pages whose x_i values are "large" and "small", respectively. Let

$$\varepsilon := 1/(cp\alpha)$$

for some suitably large constant $c \geq 8$.

(1) The input is divided into *stages*, where stage 1 begins at the first timestep. Let $\mathbf{z}^{(s)}_{\text{init}}$ denote the vector $\mathbf{z}$ at the beginning of stage s. Define the weight $w_i^{(s)} := (\nabla g(\mathbf{z}^{(s)}_{\text{init}}))_i$ of page i for stage s. Initialize the vectors $\mathbf{x} = \mathbf{x}^{(1)}_{\text{init}} = \eta \cdot \mathbf{1}$ for some parameter $\eta = 8p^2 n\gamma_n$; set $\mathbf{z} = \mathbf{z}^{(1)}_{\text{init}} = \varepsilon\, \mathbf{x}^{(1)}_{\text{init}}$. (Recall that the parameter γ_n is in the additive term of the competitive ratio guarantee of algorithm $\mathbb{A}$.) All the pages initially belong to J_S.

(2) Within each stage s, we run the black-box algorithm $\mathbb{A}$ with page weights $w_i^{(s)}$. Without loss of generality, this algorithm is *lazy* and evicts at most one page at each time.

(3) Whenever a page i is evicted in some stage s by algorithm $\mathbb{A}$, we add 1 to x_i. Since z_i is a scaled down version of x_i, one expects z_i to increase by ε. This is almost the case, with two differences. First, we increase z_i *after an initial lag*. In other words, at any time during a phase s, we ideally want z_i to equal

$$(\mathbf{z}^{(s)}_{\text{init}})_i + \varepsilon \cdot \max(0, x_i - (\mathbf{x}^{(s)}_{\text{init}})_i - \lambda), \tag{4}$$

for some lag parameter $\lambda = 4pn\,\gamma_n$. Second, after each eviction we increase z_i continuously, until it either reaches the value in (4), or else the stage ends—the condition for which we describe in the next bullet point. Hence the only case when z_i does not satisfy (4) is at the end of the stage.

(4) The stage s ends when one of the following conditions is true:

$$(\nabla g(\mathbf{z}))_i \geq 2 \cdot (\nabla g(\mathbf{z}^{(s)}_{\text{init}}))_i \text{ for some page } i \in J_L \tag{5}$$

$$\text{or, } (\nabla g(\mathbf{z}))_i \cdot (\mathbf{z})_i \geq p/n \cdot g(\text{opt}) \quad \text{for some page } i \in J_S. \tag{6}$$

We stop the stage as soon as one of these conditions happens. Since we raise z_i continuously, it follows that the condition above that triggers the end of the stage holds with equality. Moreover, if page i satisfies condition (6), it is moved from J_S to J_L. (We show in Claim 3.2 that none of the pages satisfies condition (6) initially, i.e., all pages are initially in J_S).

(5) When the stage ends, we evict all pages from the cache and start a new stage. To account for this last eviction for every page, we add 1 to each x_i, but keep z_i unchanged. This incurs a further discrepancy between $\mathbf{x}$ and $\mathbf{z}$.

3.3 Analysis

We now give an overview of the analysis.

Recall the definition of page weights $\mathbf{w}^{(s)} := \nabla g(\mathbf{z}^{(s)}_{\text{init}})$. Also, $\mathbf{z} \leq \varepsilon \mathbf{x}$, where $\mathbf{x}$ is our actual (integer) solution.

(1) We first show that due to the properties of the algorithm $\mathbb{A}$, the weighted eviction cost incurred by our algorithm during a stage is within a factor α of that incurred by the optimal solution opt with the same weights, plus additive terms.

(2) Recall that the page weights are defined by the gradients of our $\mathbf{z}$ solution at the beginning of each phase. Hence the cost incurred by the *optimal* solution using *our* weights may have nothing to do with our algorithm and these weights. Nonetheless, we show that the total weighted paging cost of this optimal solution (using our weights) can be upper bounded by $g(\text{opt}) + p \cdot g(\mathbf{z})$ (using Claim 3.1). Hence, this implies the same bound for our algorithm with an additional α factor (details in Claim 3.4).

(3) Next, we bound the final value of $g(\mathbf{z})$ by *half* the total weighted paging cost of our algorithm, which uses the fact that the gradient does not change significantly in a stage and the convexity of the function $g(\mathbf{z})$ (Claim 3.4 and Corollary 3.5).

One difficulty in carrying out this step is that the additive term involved in the guarantee of $\mathbb{A}$ (or the set-up cost for each stage) can accumulate over all the stages to give a large error. Therefore, we show that the number of stages T is $O(pn)$ (see Lemma 3.6). Using this fact, we show that the total weighted set-up cost over all the stages is also bounded (see Claim 3.3).

Let us now proceed with the details of the sketch above. The first simple but crucial claim uses the p-boundedness of the gradients to show an approximate sub-additivity-type property for g.

Claim 3.1. *For any non-negative convex function g that satisfies (1)*

$$\langle \nabla g(\mathbf{z}), \mathbf{v} \rangle \le g(\mathbf{v}) + p \cdot g(\mathbf{z}).$$

Proof. For a convex function

$$g(\mathbf{v}) \ge g(\mathbf{z}) + \langle \nabla g(\mathbf{z}), \mathbf{v} - \mathbf{z} \rangle.$$

Now rearrange, use the non-negativity of g to drop $g(\mathbf{z})$, and use the upper bound from (1) on $\langle \nabla g(\mathbf{z}), \mathbf{z} \rangle$ to complete the proof. □

In the following, we will instantiate $\mathbf{v}$ with the optimal solution, and $\mathbf{z}$ with the algorithm's solution, to infer that the weighted ℓ_1 cost of *the optimal solution with respect to the algorithm's weights* is at most $g(\text{opt}) + p \cdot g(\mathbf{z})$. The following claim shows some monotonicity conditions, and that if the initial solution has small cost relative to $g(\text{opt})$, all pages initially satisfy the condition of belonging to the set J_S.

(In this claim and in the subsequent analysis, we assume that

$$g(\mathbf{z}_{\text{init}}^{(1)}) < 1/n \cdot g(\text{opt}). \tag{7}$$

This is due to $g(\text{opt})$ being at least some large enough function of n—see Section 2.3. We will make this function precise later in (14).)

Claim 3.2 (Initialization and Monotonicity)**.** *Suppose $g(\mathbf{z}_{\text{init}}^{(1)}) < 1/n \cdot g(\text{opt})$, then none of the pages satisfy (6) initially. If a page satisfies (6) at some time, it continues to do so from then onward.*

Let $\mathbf{x}_{\text{final}}^{(s)}, \mathbf{z}_{\text{final}}^{(s)}$ denote the vectors $\mathbf{x}, \mathbf{z}$ at the end of any stage s; define $\Delta\mathbf{x}^{(s)} := \mathbf{x}_{\text{final}}^{(s)} - \mathbf{x}_{\text{init}}^{(s)}$ and $\Delta\mathbf{z}^{(s)}$ analogously. Let $J_L^{(s)}$ and $J_S^{(s)}$ denote the sets J_L and J_S respectively during stage s. The next claim shows that as long as the number of stages is not too large, the algorithm can account for the additive terms (i.e., about γ_n evictions for each page) in each of the stages.

Claim 3.3 (Additive Term)**.** *Suppose we run the algorithm for $T \le 4pn$ stages. Then for any page i,*

$$2\gamma_n \cdot \sum_{s \le T} w_i^{(s)} \le \sum_{s \le T} w_i^{(s)} (\Delta\mathbf{x}^{(s)})_i + g(\text{opt})/n.$$

Proof Sketch. As long as a page i does not satisfy property (6), the quantity $\gamma_n\, w_i^{(s)}$ summed over all stages can be upper bounded by $p/n \cdot g(\text{opt})$. Once condition (6) is satisfied, the page weight w_i changes after a stage s only when x_i changes by a significant amount during this stage: this is due to the lag term in (4) built into each stage. In this case, the quantity $\gamma_n\, w_i^{(s)}$ can be charged to the weighted change in x_i. □

We can now relate the cost incurred by the algorithm to the cost of an optimal solution. This uses the competitive ratio of $\mathbb{A}$ and the fact that the additive terms corresponding to $\gamma_n w_i$ for each page i during each of the stages can be bounded using Claim 3.3.

Claim 3.4 (Linearized Scaled Cost)**.** *Suppose we run the algorithm for T stages and $T \le 4pn$. Then, if $c \ge 8$, we have*

$$\sum_{s \le T} \sum_i (\nabla g(\mathbf{z}_{\text{final}}^{(s)}))_i \cdot (\Delta\mathbf{z}^{(s)})_i \le (1/2) \cdot g(\mathbf{z}_{\text{final}}^{(T)}) + (p+1)\, g(\text{opt}).$$

Proof Sketch. Note that the LHS in the claim is closely related to the total weighted eviction cost over all stages—it would precisely be the ε-scaled weighted cost if $\nabla g(\mathbf{z}_{\text{final}}^{(T)})$ were replaced by $\nabla g(\mathbf{z}_{\text{init}}^{(T)})$. The weighted paging algorithm $\mathbb{A}$ gives a bound on the weighted eviction cost of a single stage; adding/telescoping this bound over all the stages gives

$$\begin{aligned} &\sum_{s \le T} \sum_i \nabla g(\mathbf{z}_{\text{init}}^{(s)}) \cdot (\Delta\mathbf{z}^{(s)})_i \\ &= \sum_{s \le T} \sum_i w_i^{(s)} \cdot (\Delta\mathbf{z}^{(s)})_i \le 2\varepsilon\alpha \cdot \langle \mathbf{w}^{(T)}, \bar{\mathbf{x}}^{(T+1)} \rangle + \varepsilon/2 \cdot g(\text{opt}), \end{aligned} \tag{8}$$

where $\bar{\mathbf{x}}^{(T+1)}$ is any feasible solution and $\mathbf{w}^{(T)}$ is any upper bound on the weights in the individual stages. The first term on the right comes from the multiplicative loss of α in $\mathbb{A}$ and the second term $1/2 \cdot g(\text{opt})$ bounds the cumulative effect of the additive losses across all stages.

We now relate the two sides of (8) to the two sides of the inequality in the claim. On the right side, we set $\bar{\mathbf{x}}^{(T+1)}$ to be (offline) opt and $\mathbf{w}^{(T)}$ to be the page weights in the last stage of the algorithm. Then, using Claim 3.1, we have

$$\begin{aligned} \langle \mathbf{w}^{(T)}, \bar{\mathbf{x}}^{(T+1)} \rangle &= \langle \nabla g(\mathbf{z}_{\text{init}}^{(T)}), \bar{\mathbf{x}}^{(T+1)} \rangle \\ &\le g(\bar{\mathbf{x}}^{(T+1)}) + p \cdot g(\mathbf{z}_{\text{init}}^{(T)}) \le g(\text{opt}) + p \cdot g(\mathbf{z}_{\text{final}}^{(T)}). \end{aligned} \tag{9}$$

To relate the left side of (8) to the claim, we observe that

$$\nabla g(\mathbf{z}_{\text{final}}^{(T)})_i \le 2 \nabla g(\mathbf{z}_{\text{init}}^{(T)})_i \text{for each page } i \in J_L,$$

and $\nabla g(\mathbf{z}_{\text{init}}^{(T)})_i \cdot \mathbf{z}_i \le p/n \cdot g(\text{opt})$ for each page in J_S. This lets us infer

$$\begin{aligned} &\sum_{s \le T} \sum_i (\nabla g(\mathbf{z}_{\text{final}}^{(s)}))_i \cdot (\Delta\mathbf{z}^{(s)})_i \\ \le &2 \sum_{s \le T} \sum_i \nabla g(\mathbf{z}_{\text{init}}^{(s)}) \cdot (\Delta\mathbf{z}^{(s)})_i + p \cdot g(\text{opt}). \end{aligned} \tag{10}$$

Substituting (9) and (10) into (8) and setting $c \ge 8$ gives the claim. □

Corollary 3.5 (Scaled Cost Bound)**.** *Suppose we run the algorithm for T stages and $T \le 4pn$. Moreover, suppose $g(\mathbf{z}_{\text{init}}^{(1)}) \le 1/n \cdot g(\text{opt})$. Then for $c \ge 8$, we have $g(\mathbf{z}_{\text{final}}^{(T)}) \le 2(p+2)\, g(\text{opt})$.*

Proof. Since $\mathbf{z}_{\text{final}}^{(s)} = \mathbf{z}_{\text{init}}^{(s+1)}$, we can write $g(\mathbf{z}_{\text{final}}^{(T)})$ as a telescoping sum to get

$$g(\mathbf{z}_{\text{final}}^{(T)}) - g(\mathbf{z}_{\text{init}}^{(1)}) = \sum_{s \le T} \left(g(\mathbf{z}_{\text{final}}^{(s)}) - g(\mathbf{z}_{\text{init}}^{(s)}) \right)$$

$$\leq \sum_{s \leq T} \langle \nabla g(\mathbf{z}_{\text{final}}^{(s)}), \Delta \mathbf{z}^{(s)} \rangle \leq 1/2\, g(\mathbf{z}_{\text{final}}^{(T)}) + (p+1)\, g(\text{opt}). \quad (11)$$

The first inequality uses convexity, and the next one uses Claim 3.4. Using $n \geq 1$ and our assumption on the initial function value, and simplifying, we get the desired result. □

3.3.1 Bounding the Number of Stages. The previous results considered the situation at any stage $T \leq 4pn$. We now show that the algorithm must stop after at most $2pn$ stages, assuming we have a correct assumption on the cost of the optimal solution. The main idea is to show that once a page i satisfies (6) and is added to J_L, its weight cannot double too many times. (The assumption is from (7), which will follow from (14).)

Lemma 3.6. *Suppose $g(\mathbf{z}_{\text{init}}^{(1)}) \leq 1/n \cdot g(\text{opt})$. Then the total number of stages is at most $2pn$.*

Proof. Suppose for the sake of contradiction that the algorithm runs for more than $2pn$ stages. We consider the first $T := 2pn$ stages. The number of stages that end because of condition (6) is at most n, since each time a page is moved from J_S to J_L, and no page can move in the other direction. For a stage s that ends because of condition (5), there must be some page $i_s \in J_L^{(s)}$ such that $(\nabla g(\mathbf{z}_{\text{final}}^{(s)}))_{i_s} = 2 \cdot (\nabla g(\mathbf{z}_{\text{init}}^{(s)}))_{i_s}$; call this page *responsible* for such a stage s.

For sake of brevity, let J_L denote the set $J_L^{(T+1)}$. For a page $i \in J_L$, let n_i be the number of times it has been responsible for some stage s ending due to condition (5). Then $\sum_{i \in J_L} n_i \geq T - n$. For each such page i that is responsible at least once, let f_i be the final stage s in which it belonged to $J_S^{(s)}$. (Such a stage must exist, since a page is responsible only if it belongs to J_L.) Now,

$$p \cdot g(\mathbf{z}_{\text{final}}) \overset{(1)}{\geq} \langle \nabla g(\mathbf{z}_{\text{final}}), \mathbf{z}_{\text{final}} \rangle \geq \sum_{i \in J_L} (\nabla g(\mathbf{z}_{\text{final}}))_i \cdot (\mathbf{z}_{\text{final}}^{(f_i)})_i$$

$$\overset{(\dagger)}{\geq} \sum_{i \in J_L} 2^{n_i} (\nabla g(\mathbf{z}_{\text{final}}^{(f_i)})_i \cdot (\mathbf{z}_{\text{final}}^{(f_i)})_i \overset{(\ddagger)}{\geq} p/n \cdot g(\text{opt}) \cdot \sum_{i \in J_L} 2^{n_i}, \quad (12)$$

where inequality (†) uses that the gradient doubles each time the page is responsible for the stage, and inequality (‡) uses the definition of f_i, and the criterion (6) for a page to move from J_S to J_L. Now, since the sum $\sum_{i \in J_L} n_i \geq T - n$, the quantity $\sum_{i \in J_L} 2^{n_i}$ is minimized when all the n_i's are equal and $|J_L| = n$, in which case it is at least $n \cdot 2^{T/n-1}$. Substituting this into (12), we get

$$g(\mathbf{z}_{\text{final}}) \geq 2^{T/n-1} \cdot g(\text{opt}). \quad (13)$$

The desired contradiction, i.e., $T < 2pn$, now follows from Corollary 3.5. □

3.3.2 Bounding the Competitive Ratio. Finally, suppose

$$g(\text{opt}) \geq \max\{\zeta_{p,n}\, g(\mathbf{1}), n\, g(\mathbf{z}_{\text{init}}^{(1)})\}; \quad (14)$$

this is wlog (proof deferred to full version). Given that the total number of stages $T \leq 2pn$, we can bound the competitive ratio of the algorithm. (Note that assumption follows from (14).)

Theorem 3.7 (Cost). *Given $g(\mathbf{z}_{\text{init}}^{(1)}) \leq 1/n \cdot g(\text{opt})$, the total cost of the algorithm is at most*

$$(c_1 p)(2cp\alpha)^p \cdot g(\text{opt}) + \zeta_{p,n} \cdot g(\mathbf{1}),$$

for some $\zeta_{p,n}$ that depends only on p and n, and not on the sequence.

4 Randomized Algorithm for Convex Paging

In this section, we give a randomized online algorithm for the convex paging problem. Ideally, we would like to adapt the algorithm presented in Section 3 with the deterministic weighted paging algorithm $\mathbb{A}$ replaced by the randomized one $\mathbb{B}$, the latter having (expected) competitive ratio β. However, there are several issues in this direct approach:

- Each stage now uses a randomized algorithm, and hence, the start time of a stage depends on the coin tosses of the prior stages. Hence, we cannot argue that the expected weighted paging cost in this stage is at most β times the optimal weighted paging cost of this stage (because we can make such statements only for *fixed* times).
- Since the number of stages can be $\Omega(n)$, it is possible that in some stage the algorithm $\mathbb{B}$ incurs a large weighted paging cost, say $\Omega(\beta n)$ times the optimal cost in this stage. It is difficult to directly bound the cost incurred in such stages.
- Unlike the deterministic setting, the number of stages need not be $O(pn)$.

We address the above issues as follows: (i) We explicitly ensure that the weighted ℓ_1 cost incurred in every stage is not too large by adding a third condition (besides (5) and (6)) for ending a stage, (ii) we use martingale based tail bounds to show that with high probability, the number of stages ending because of this third condition cannot be too large, and finally, (iii) if the number of stages becomes too large, we *restart* the algorithm on the remaining input.

We now describe the randomized algorithm. Its execution is divided into phases. Each new phase runs the algorithm in Section 3 from scratch, i.e., it initializes the $\mathbf{x}$ and $\mathbf{z}$ variables, and runs on the remaining input. The phase ends if the number of stages exceeds a threshold T^* which is $O(\beta p^2 n)$.

We now give the details of any phase. For ease of notation, we describe the first phase – every other phase is identical, the only difference being the start times. Since the intuition for these steps remains same as that in Section 3.2, we only give the technical details for every step.

4.1 Algorithm

We maintain the page eviction vector $\mathbf{x}$ and an auxiliary vector $\mathbf{z}$ which is roughly $\varepsilon \cdot \mathbf{x}$. Here $\varepsilon = 1/cp\beta$, where c is a large enough constant. These variables are re-initialized at the beginning of every phase; the actual page eviction vector across all phases is only computed at the end as the sum of page eviction vectors in each phase. Our algorithm uses an additional constant c_1 satisfying

$$c_1 \gg c \gg 1. \quad (15)$$

We now set a limit $T^* := c_1 p^2 \beta n$ on the number of stages in a phase.

(1) As in the deterministic setting, the pages are partitioned into two sets J_L and J_S; all pages are in J_S initially. The vector $\mathbf{x}$ is initialized as $\mathbf{x}_{\text{init}}^{(1)} := \eta \cdot \mathbf{1}$ with the parameter $\eta = 2c_1 p \gamma_n T^*$, and the vector $\mathbf{z}$ is initialized as $\varepsilon \cdot \mathbf{x}$.
(2) The input is divided into *stages*, where stage 1 begins at the first timestep of the phase. Let $\mathbf{z}_{\text{init}}^{(s)}$ denote the vector $\mathbf{z}$ at

the beginning of the stage s. Define the weight $w_i^{(s)}$ of page i for stage s to be:

$$w_i^{(s)} := (\nabla g(\mathbf{z}_{\text{init}}^{(s)}))_i. \tag{16}$$

Within the stage, we run the randomized algorithm $\mathbb{B}$ to handle the input, with these page weights $w_i^{(s)}$ remaining unchanged. Whenever a page i is evicted by $\mathbb{B}$, we add 1 to x_i. Again, z_i is a scaled-down version of x_i *after an initial lag*. In other words, at any time during a phase s, the ideal value of z_i is

$$(\mathbf{z}_{\text{init}}^{(s)})_i + \varepsilon \cdot \max(0, x_i - (\mathbf{x}_{\text{init}}^{(s)})_i - \lambda), \tag{17}$$

where $\lambda = 2c_1 T^* \gamma_n$ is the lag parameter. We increase z_i continuously until it reaches this value, or the stage ends (as explained in the next bullet point).

(3) Let $\mathbf{z}_{\text{final}}^{(s)}$ denote the vector $\mathbf{z}$ at the end of stage s, and let $\Delta\mathbf{z}^{(s)}$ denote $\mathbf{z}_{\text{final}}^{(s)} - \mathbf{z}_{\text{init}}^{(s)}$. The stage s ends when one of the conditions (5) or (6) used for the deterministic algorithm hold, or if the following new condition holds:

$$\sum_i w_i^{(s)} ((\mathbf{x}_{\text{final}}^{(s)})_i - (\mathbf{x}_{\text{init}}^{(s)})_i) \geq \frac{g(\text{opt})}{n}. \tag{18}$$

We stop as soon as one of these conditions happen. As in the deterministic algorithm (Section 3.2) we can assume that if a stage ends due to either condition (5) or (6), then that condition holds with equality. We cannot state the same for the new condition (18), because x_i variables change in discrete unit-sized steps.[2]

(4) When the stage ends, we evict all pages from the cache and start a new stage. To account for this last eviction for every page, we add 1 to each $\mathbf{x}_i$, but keep $\mathbf{z}_i$ unchanged. This incurs a further discrepancy between $\mathbf{x}$ and $\mathbf{z}$.

(5) Finally, the *phase* ends when the number of *stages* within the phase exceeds the threshold T^* (or the input ends).

4.2 Analysis

We analyze the first phase; the same arguments would hold for subsequence phases. However, for any particular phase, we would condition on the coin tosses used by $\mathbb{B}$ in the previous phases. This would ensure that the start time of the current phase is deterministic.

We begin by stating the analog of Claim 3.2 justifying adding all the pages to J_S at the beginning of the phase. The proof follows exactly in the same manner as that of Claim 3.2.

Claim 4.1. *Suppose $g(\mathbf{z}_{\text{init}}^{(1)}) < 1/n \cdot g(\text{opt})$, then none of the pages satisfy (6) initially. If a page satisfies (6) at some time, it continues to do so from then onward.*

Let T_1, T_2, T_3 be the number of stages which terminate because of conditions (5), (6) and (18) respectively.

[2] One might wonder if this condition could also have been written in terms of z_i variables, but the presence of the additive γ_n term relating z_i and x_i makes it challenging if we state this condition in terms of z_i's.

4.2.1 Bounding $g(\mathbf{z}_{\text{final}}^{(T)})$ in terms of T_3. Next, we show that an upper bound on T_3 would lead to the desired upper bound on $g(\mathbf{z}_{\text{final}}^{(T)})$. The intuition is that condition (18) ensures that $\mathbf{z}$ does not increase a lot in a single stage, and hence the overall increase can be bounded in terms of the number of such stages. We first upper bound $\Delta\mathbf{z}^{(s)}$ for a stage s.

Claim 4.2. *For a stage s, $\sum_i w_i^{(s)} (\Delta\mathbf{z}^{(s)})_i \leq \frac{\varepsilon \cdot g(\text{opt})}{n}$.*

Proof. Since x_i changes in steps of +1, we know from (18) that for any stage s,

$$\sum_i w_i^{(s)} ((\mathbf{x}_{\text{final}}^{(s)})_i - (\mathbf{x}_{\text{init}}^{(s)})_i) \leq \frac{g(\text{opt})}{n} + \sum_i w_i^{(s)}. \tag{19}$$

It follows from inequality (18) that if $\Delta\mathbf{z}^{(s)}$ is positive, then (note that $\lambda \geq 1$)

$$\Delta\mathbf{z}^{(s)} \leq \varepsilon(\Delta\mathbf{x}^{(s)} - 1).$$

The desired result now follows from (19). □

The following claim bounds $g(\mathbf{z}_{\text{final}}^{(T)})$ in terms of $\Delta\mathbf{z}^{(s)}$ over all stages s.

Claim 4.3. *Assume $g(\mathbf{z}_{\text{init}}^{(1)}) \leq 1/n \cdot g(\text{opt})$. Then*

$$\begin{aligned} g(\mathbf{z}_{\text{final}}^{(T)}) &\leq 2 \sum_{s \leq T} \langle w^{(s)}, \Delta\mathbf{z}^{(s)} \rangle + (p + 1/2) \cdot g(\text{opt}) \\ &\leq \left(\frac{2\varepsilon\ (T_1 + T_3)}{n} + 2p \right) \cdot g(\text{opt}). \end{aligned}$$

Proof. By telescoping sum and convexity,

$$\begin{aligned} g(\mathbf{z}_{\text{final}}^{(T)}) - g(\mathbf{z}_{\text{init}}^{(1)}) &= \sum_{s \leq T} g(\mathbf{z}_{\text{final}}^{(s)}) - g(\mathbf{z}_{\text{init}}^{(s)}) \\ &\leq \sum_{s \leq T} \langle \nabla g(\mathbf{z}_{\text{final}}^{(s)}), \Delta\mathbf{z}^{(s)} \rangle. \end{aligned}$$

In a stage s, if a page $i \in J_L^{(s)}$, then (5) ensures that $(\nabla g(\mathbf{z}_{\text{final}}^{(s)})_i \leq 2(\nabla g(\mathbf{z}_{\text{init}}^{(s)})_i = 2w_i^{(s)}$. For a page i, let f_i be the last stage such that $i \in J_S^{(f_i)}$. Then the r.h.s. above is at most

$$\begin{aligned} &2 \sum_{s \leq T} \langle w^{(s)}, \Delta\mathbf{z}^{(s)} \rangle + \sum_i \sum_{s : i \in J_S^s} \langle \nabla g(\mathbf{z}_{\text{final}}^{(s)}), \Delta\mathbf{z}^{(s)} \rangle \\ \leq &2 \sum_{s \leq T} \langle w^{(s)}, \Delta\mathbf{z}^{(s)} \rangle + \sum_i (\nabla g(\mathbf{z}_{\text{final}}^{(f_i)}))_i (\mathbf{z}_{\text{final}}^{(f_i)})_i, \end{aligned}$$

The first inequality in the claim now follows from condition (6) and Claim 4.1.

Now, Claim 4.2 shows that

$$\sum_{s \leq T} \langle w^{(s)}, \Delta\mathbf{z}^{(s)} \rangle \leq \frac{\varepsilon T \cdot g(\text{opt})}{n}.$$

The second inequality in the claim now follows from the fact that $T = T_1 + T_2 + T_3$ and $T_2 \leq n$. □

Now we upper bound T_1.

Lemma 4.4. *Assume that $g(\mathbf{z}_{\text{init}}^{(1)}) \leq 1/n \cdot g(\text{opt})$. Then $T_1 \leq T_3 + 2np$.*

Proof. Consider a stage s that ends with (5). We know that there is a page $i_s \in J_L^{(s)}$ such that $(\nabla g(\mathbf{z}_{\text{final}}^{(s)}))_{i_s} = 2 \cdot (\nabla g(\mathbf{z}_{\text{init}}^{(s)}))_{i_s}$. For a page i, let n_i be the number of times it appears as i_s for some stage s. We know that $T_1 = \sum_i n_i$. For sake of brevity, let $\mathbf{z}_{\text{final}}$ denote $\mathbf{z}_{\text{final}}^{(T)}$.

For each page i, let f_i be the final stage in which it appears as a small page. Now,

$$p \cdot g(\mathbf{z}_{\text{final}}) \overset{(1)}{\geq} \langle \nabla g(\mathbf{z}_{\text{final}}), \mathbf{z}_{\text{final}} \rangle \geq \sum_i (\nabla g(\mathbf{z}_{\text{final}}))_i \cdot (\mathbf{z}_{\text{final}}^{(f_i)})_i$$

$$\geq \sum_i 2^{n_i} (\nabla g(\mathbf{z}_{\text{final}}^{(f_i)})_i \cdot (\mathbf{z}_{\text{final}}^{(f_i)})_i \geq p/n \cdot g(\text{opt}) \cdot \sum_i 2^{n_i}.$$

Now, conditioned on $\sum_i n_i = T_1$, the sum $\sum_i 2^{n_i}$ is minimized when each of the n_i's is equal to T_1/n. Substituting this above, we get

$$g(\mathbf{z}_{\text{final}}) \geq 2^{T_1/n} \cdot g(\text{opt}).$$

Using Claim 4.3 we get

$$2^{T_1/n} \leq \frac{2\varepsilon(T_1 + T_3) + 2np}{n} \leq \frac{T_1 + T_3 + 2np}{n} \tag{20}$$

If $T_1 \leq 2np$, there is nothing to prove. So assume $T_1 \geq 2np$. Then, $2^{T_1/n} \geq 2T_1/n$. Substituting in the above, we get the desired result. □

Combining the above two results, we get:

Corollary 4.5. *Assume that $g(\mathbf{z}_{\text{init}}^{(1)}) \leq 1/n \cdot g(\text{opt})$. Then $g(\mathbf{z}_{\text{final}}^{(T)}) \leq \left(\frac{4\varepsilon \cdot T_3}{n} + 4p\right) \cdot g(\text{opt})$*

4.2.2 Defining good and bad stages. Corollary 4.5 shows that an upper bound on T_3 would lead to an upper bound on $g(\mathbf{z}_{\text{final}}^{(T)})$. If the optimal solution were also incurring high cost in stages ending with condition (18), then we would be done. However, the algorithm $\mathbb{B}$ is randomized and hence, it is possible that in some stages, it incurs high cost whereas the optimal weighted paging cost in this stage is small. The crux of the analysis in this section lies in showing that such events are not frequent. Observe that we cannot use Markov's inequality on the expected cost incurred in a stage directly because the time at which a stage ends is a random variable.

Let $\bar{x}$ denote the optimal solution to the convex paging instance $\mathbb{I}$. Here $\bar{x}_i$ denotes the number of evictions of page i by this solution. Let $\bar{x}^{(s)}$ denote the solution $\bar{x}$ at the beginning of stage s (of the current phase). We now give a crucial definition:

Definition 4.6. We say that a stage s (in the current phase) is a *bad* stage if it ends with condition (18) and

$$\langle w^{(s)}, \Delta \bar{\mathbf{x}}^{(\mathbf{s})} \rangle \leq \frac{1}{4\beta} \left(\frac{g(\text{opt})}{n} - 4\gamma_n \sum_i w_i^{(s)} \right) \tag{21}$$

A stage is said to be good if it is not a bad stage.

We now show that a stage is good with constant probability. For technical reasons, if $T < T^*$, we add *null* stages at the end so that there are exactly T^* stages. These null stages are classified as good stages.

Claim 4.7. *Let $s \leq T^*$ be a stage (in the current phase). Condition on the coin tosses (of $\mathbb{B}$) in the first $s-1$ stages. Then stage s is good with probability at least 3/4 (where the probability is over the coin tosses of $\mathbb{B}$ in stage s).*

Proof. Let t_s be the start time of stage s (which is deterministic since we have conditioned on the first $s-1$ stages). Consider the weighted paging instance $\mathbb{I}^{(s)}$ obtained from the original request instance $\mathbb{I}$ by considering requests from time t_s onward (with weights of pages given by $w^{(s)}$). Let $A^{(s)}$ denote the r.h.s. of (21). Let t_e be the first time after t_s such that $\langle w^{(s)}, \Delta \bar{\mathbf{x}}^{(\mathbf{s})} \rangle$ reaches $A^{(s)}$ (if no such time exists, then t_e is the last time in the input sequence).

Consider running the algorithm $\mathbb{B}$ on $\mathbb{I}^{(s)}$ till time t_e (without worrying about whether any of the conditions for ending of a stage gets satisfied). By definition of $\mathbb{B}$, the expected weighted paging cost during this stage if at most $\beta \cdot A^{(s)} + \gamma_n \sum_i w_i^{(s)}$. Using Markov's inequality, it follows that with probability at least 3/4, the total weighted paging cost of $\mathbb{B}$ during $[t_s, t_e]$ is at most

$$4\beta A^{(s)} + 4\gamma_n \sum_i w_i^{(s)} = \frac{g(\text{opt})}{n}.$$

Assume that this event, call it $\mathcal{E}$, happens. We now claim that stage s must end beyond t_e. Indeed, otherwise condition (18) is not satisfied, and hence, stage s must be a good stage. □

As a corollary, we get

Corollary 4.8. *Fix an integer s, $1 \leq s \leq T^\star$. Then the probability that less than $0.6s$ of the first s stages are good is at most $e^{-0.04s}$.*

Proof. We apply Azuma's inequality (see e.g. [16]). For a stage s, let Y_s be the indicator random variable which is 1 if Y_s is good. Then Claim 4.7 shows that $Z_s := (Y_1 + \ldots + Y_s) - 0.75s$ is a submartingale. Therefore, using Azuma's inequality,

$$\Pr[Z_s \leq -0.15s] \leq e^{-0.04s}.$$

Thus, $(Y_1 + \ldots + Y_s) \leq 0.6s$ with probability at most $e^{-0.04s}$. □

4.2.3 Bounding the number of stages. Let $\mathcal{E}$ denote the event that among the first T^* stages, at least $0.6T^*$ are good. Corollary 4.8 shows that $\Pr[\mathcal{E}] \geq 1 - e^{-p}$ (recall that $T^* \geq c'p$ for a large enough constant c'). For rest of this section we assume that $\mathcal{E}$ happens and $g(\mathbf{z}_{\text{init}}^{(1)}) \leq 1/n \cdot g(\text{opt})$. Our goal is to show that the algorithm terminates in the current phase:

Theorem 4.9. *Assuming that $g(\mathbf{z}_{\text{init}}^{(1)}) \leq 1/n \cdot g(\text{opt})$ and the event $\mathcal{E}$ occurs, the current phase ends in less than T^* stages.*

We now prove this theorem. Suppose, for the sake of contradiction, that the current phase runs till stage T^* (i.e., all these stages are non-null). Let S_3^g denote the set of good stages that end with condition (18), and let $T_3^g := |S_3^g|$. We show that a constant fraction of stages are in S_3^g:

Claim 4.10. $T_3^g \geq 0.05 \cdot T^*$.

Proof. The fact that event $\mathcal{E}$ occurs implies that

$$T_3^g \geq 0.6T^* - (T_1 + T_2) \geq 0.6T^* - (T_1 + n).$$

It follows from Lemma 4.4 that $2T_1 \leq T^* + 2np$. The desired result now follows from the above inequality and the fact that $T^* \gg np$. □

The definition of a stage in S_3^g implies that

$$\sum_{s\in S_3^g} \langle w^{(s)}, \Delta\bar{\mathbf{x}}^{(\mathbf{s})}\rangle \geq \frac{T_3^g\, g(\text{opt})}{4\beta n} - \frac{1}{\beta}\sum_{s\leq T^*}\sum_i w_i^{(s)}\gamma_n. \tag{22}$$

We first bound the LHS of the above inequality.

Claim 4.11. $\sum_{s\leq T^*}\langle w^{(s)}, \Delta\bar{\mathbf{x}}^{(\mathbf{s})}\rangle \leq 5\varepsilon T^*\cdot g(\text{opt})$.

Proof. We have that

$$\sum_{s\leq T^*}\langle w^{(s)}, \Delta\bar{\mathbf{x}}^{(\mathbf{s})}\rangle \leq \langle w^{(T^*)}, \bar{x}^{(T^*+1)}\rangle \leq \langle \nabla g(\mathbf{z}_{\text{final}}^{(T^*)}), \bar{x}^{(T^*+1)}\rangle$$
$$\leq g(\text{opt}) + p\cdot g(\mathbf{z}_{\text{final}}^{(T^*)})$$
$$\leq \left(5p^2 + \frac{4\varepsilon p T^*}{n}\right)\cdot g(\text{opt}) \leq 5\varepsilon T^*\cdot g(\text{opt}),$$

where the first inequality follows from monotonicity of the weights, the second inequality follows from Claim 3.1, the third inequality follows from Corollary 4.5, and the last one follows from the fact that $T^* = c_1\beta p^2 n \geq 5(c\beta p)pn = {}^{(5pn)}/_{\varepsilon}$ (using (15)). □

The proof of the following technical result follows along the same lines as that of Claim 3.3. (Since $T^\star = c_1 p^2\beta n$, we do not need any additional assumptions on T^* here.)

Claim 4.12. *For any page i,*

$$\sum_{s\leq T^*} w_i^{(s)}\gamma_n \leq \frac{1}{c_1}\left(\sum_{s\leq T^*} w_i^{(s)}((\mathbf{x}_{\text{final}}^{(s)})_i - (\mathbf{x}_{\text{init}}^{(s)})_i) + \frac{g(\text{opt})}{n}\right).$$

We use this to bound the second term in the RHS of (22).

Corollary 4.13.

$$\frac{1}{\beta}\cdot\sum_{s\leq T^*}\sum_i w_i^{(s)}\gamma_n \leq 2\varepsilon T^*\cdot g(\text{opt}).$$

Proof. Using Claim 4.12, (19) and the fact that $\gamma_n \geq 1$, we see that

$$(c_1-1)\sum_{s\leq T^*}\sum_i w_i^{(s)}\gamma_n \leq g(\text{opt}) + \frac{T^* g(\text{opt})}{n} \leq \frac{2T^* g(\text{opt})}{n}.$$

The desired result now follows from the fact that $c_1 \gg c$ (as in (15)) and the definition of ε. □

Substituting the results in Claim 4.11 and Corollary 4.13 in (22), we see that

$$5\varepsilon T^* \geq \frac{T_3^g}{4\beta n} - 2\varepsilon T^*.$$

But this is a contradiction using Claim 4.10 and the fact that $\varepsilon = {}^1/_{(c\beta n)}$ for a large enough constant c. This proves Theorem 4.9.

4.2.4 Putting it together. Now we calculate the total cost incurred during a phase.

Claim 4.14. *Assume that $g(\mathbf{z}_{\text{init}}^{(1)}) \leq {}^1/_2\cdot g(\text{opt})$. Let $\mathbf{x}_{\text{final}}^{(T^*)}$ be the actual solution maintained by the algorithm at the end of T^* stages in this phase. Then,*

$$(O(p\beta))^p\cdot g(\text{opt}) + \gamma_{p,n}\, g(\mathbf{1}),$$

for some $\gamma_{p,n}$ that depends only on p and n, and not on the sequence.

Proof. It follows from (17) and the fact that we may count an extra +1 for each x_i at the end of each stage that

$$\mathbf{x}_{\text{final}} \leq \mathbf{x}_{\text{init}}^{(1)} + (1/\varepsilon)\cdot\mathbf{z}_{\text{final}} + (\lambda+2)T\cdot\mathbf{1},$$

where the first term accounts for the initialization, the second term captures that difference in scales between $\mathbf{x}$ and $\mathbf{z}$, and the last term accounts for the discrepancies due to the lag and the boundary effects. Using our initial value for $\mathbf{x} = \eta\mathbf{1}$, the choice of $\varepsilon = {}^1/_{cp\beta}$, and $T^* = c_1\beta p^2 n$, we get

$$\mathbf{x}_{\text{final}} \leq (cp\beta)\cdot\mathbf{z}_{\text{final}} + \underbrace{[2c_1\beta p^2 n(\lambda+2)+\eta]}_{\gamma=\gamma_{p,n}}\cdot\mathbf{1},$$

Using Lemma 2.1, Corollary 3.5, and the definition of ε, we get

$$g(\mathbf{x}_{\text{final}}) \leq (2cp\beta)^p\, g(\mathbf{z}_{\text{final}}) + (2\gamma)^p\, g(\mathbf{1})$$
$$\leq (2cp\beta)^p\,(4p + 4c_1cp^2)\, g(\text{opt}) + (2\gamma)^p\, g(\mathbf{1}).$$ □

Now suppose $g(\text{opt}) \geq \max\{\gamma_{p,n}\, g(\mathbf{1}), 2\, g(\mathbf{z}_{\text{init}}^{(1)})\}$. The above claim shows that if $\mathbf{x}^h$ denotes the solution produced during a particular phase h, then $g(\mathbf{x}^h)$ is $(O(\beta p))^p$. Now, if $\mathbf{x}^{\leq h}$ denotes the solution produced till the end of phase h, i.e., $\mathbf{x}^1 + \ldots + \mathbf{x}^h$, we have

$$g(\mathbf{x}^{\leq h}) \leq h^p((g(\mathbf{x}^1) + \ldots g(\mathbf{x}^h)) \leq h^p\cdot(O(\beta p))^p\cdot g(\text{opt}),$$

where we have used the fact that

$$g(\mathbf{x}^{\leq h}) = g\left(h\cdot\frac{\mathbf{x}^1+\ldots+\mathbf{x}^h}{h}\right)$$
$$\leq h^p\cdot g\left(\frac{\mathbf{x}^1+\ldots+\mathbf{x}^h}{h}\right) \leq h^p(g(\mathbf{x}^1)+\ldots+g(\mathbf{x}^h)).$$

The first inequality above follows from (2) and the last one follows from the convexity and the monotonicity of g.

Theorem 4.9 shows that the probability of h phases is at most e^{-hp}. Thus, we see that the expected cost of the solution is

$$\sum_{h\geq 1}\frac{h^p}{e^{hp}}\,(O(\beta p))^p\cdot g(\text{opt}) = (O(\beta p))^p\cdot g(\text{opt}).$$

As in Section 3.2, we can remove the assumption that the algorithm knows $g(\text{opt})$ and that there is a lower bound on $g(\text{opt})$. (see the full version for details). This completes the proof of Theorem 1.1.

5 Lower Bounds for Randomized Algorithms

In this section, we prove Theorem 1.2, that is restated here.

Theorem 1.2 (Randomized Lower Bound). *There exists a constant $C > 0$ such that for any $p \leq C\ln n$, any randomized algorithm (that is even allowed to produce a fractional solution) for ℓ_p-norm paging has an $\Omega(p\log k)$ competitive ratio against an optimal integer solution. This result implies an $\Omega(\log n\log k)$ lower bound for randomized min-max paging.*

The Instance. The total number of pages is n and the cache size is k, where $n > (k+1)^2$. We assume wlog that $n = 2^r\cdot(k+1)$ for some integer $r := \lg\frac{n}{k+1}$, and that k is odd. The request sequence is partitioned into $r+1 = \Omega(\log n)$ *epochs*, indexed $e = 0, 1, 2, \ldots, r$. In each epoch e, a subset A_e of $\frac{n}{2^e}$ pages is *active* and the rest of the

pages are *inactive*. The set A_e is defined recursively. In epoch 0, all n pages are active; i.e., A_0 is the set of all n pages. To define the set A_{e+1} from the set A_e, we need to introduce the notion of *blocks* of pages. The active pages in any epoch are (arbitrarily) partitioned into blocks of $k+1$ pages each. Note that A_e contains $2^{r-e} = \frac{n}{2^e(k+1)}$ blocks. We denote the bth block of the eth epoch by B_{eb}. For every b, a uniform random subset of $\frac{k+1}{2}$ pages from block B_{eb} is added to A_{e+1}, i.e., these pages are active in epoch $e+1$; the remaining $\frac{k+1}{2}$ pages in block B_{eb} are inactive in epoch $e+1$.

Next, we define the request sequence for the instance. The overall request sequence is ordered by epochs $e = 0, 1, 2, \ldots, r$ and for a fixed epoch e, by the blocks in A_e in arbitrary order. Overloading notation, we call the request sequence for an epoch of pages as an epoch, and the request sequence for a block of pages as a block. We now describe the requests in the bth block of the eth epoch. These requests are entirely for pages in block B_{eb} and are partitioned into N *phases*. (Here N is a large number whose value we will make precise later.) Morally, we would like each phase to comprise a minimal sequence of requests spanning all the $k+1$ distinct pages in block B_{eb} (similar to the usual definition of a phase for the standard caching problem). For technical reasons, we need to modify this definition slightly. We define a phase as the minimal sequence of requests that contains all the $\frac{k+1}{2}$ pages from block $B_{eb} \setminus A_{e+1}$. In other words, the phase ends when we have seen at least one request for every pages in block B_{eb} that is not active in the next epoch. Finally, we generate each individual request in a phase by choosing a page in block B_{eb} uniformly at random.

The procedure for generating this input request sequence is described formally in Algorithm 1. For each block B_{eb} during an epoch e, we select a random subset A_{eb} of size $|B_{eb}|/2$ (line 1.9), which we call the *static* pages of B_{eb}, because they will be passed on to the next epoch (and therefore $A_{e+1} \leftarrow \cup_b A_{eb}$. The rest of the pages, i.e., $B_{eb} \setminus A_{eb}$, are called *dynamic* pages (of this epoch). This completes the description of the input sequence.

We now give an overview of the lower bound analysis for the random input sequence generated using Algorithm 1. The formal details of the analysis are deferred to the full version.

Optimal Solution. The benchmark solution (we call it the optimal solution) is defined via two rules. The first rule is about the static pages A_{eb} of a block B_{eb}. Note that these pages shall be requested in later epochs. For any page $i \in A_{eb}$, it stays in the cache for the entire duration of the (requests in) block B_{eb}. Thus, these pages are evicted only once in epoch e. Since $|A_{eb}| = (k+1)/2$, the remaining $(k-1)/2$ slots in the cache are available for the dynamic pages during this block.

The second rule is about managing the dynamic pages $B_{eb} \setminus A_{eb}$ to serve all the page requests in the sequence corresponding to the block B_{eb}. For each phase H_ℓ in this block, we ensure that the first $(k-1)/2$ pages requested in this phase are in the cache at the beginning of phase H_ℓ. The last page request in H_ℓ causes a page eviction. Thus, there is only page eviction in each phase. However, one needs to be careful about which page to evict at the end of each phase. Indeed, we may be dealing with the ℓ_∞ metric and evicting the same page at the end of each phase could lead to high ℓ_∞ eviction cost. We show that a carefully designed round-robin eviction scheme ensures that each active page in a block B_{eb} is evicted $O(N/k)$ times with high probability during the N phases of this block. Since each page appears as an active page in only the last epoch where it is active, this shows that with high probability, each page is evicted $O(N/k + r) = O(N/k)$ times. Formally, we show (in the full version):

Algorithm 1: GenerateRequestSequence(k, n).

```
1.1  Input: Cache size k and total number of pages n. Assume n
     is of the form 2^r (k + 1).
1.2  Set A_0 ← [n].        (Active Pages)
1.3  for e = 0, ..., r do
1.4      (Generate request sequence for epoch e)
1.5      Partition A_e into r_e := 2^{r-e} blocks, each of size k + 1.
1.6      Let the blocks be B_{e1}, ..., B_{er_e}.
1.7      for b = 1, ..., r_e do
1.8          (Generate request sequence for block b in epoch e)
1.9          A_{eb} ← random subset of size (k+1)/2 from B_{eb}.
1.10         for ℓ = 1, ..., N do
1.11             (Generate request sequence for phase ℓ in this
                 block)
1.12             repeat
1.13                 Request a randomly chosen page from B_{eb}.
1.14             until all the pages in B_{eb} \ A_{eb} have been
                   requested at least once during this phase.
1.15     A_{e+1} ← ∪_b A_{eb}        (Active pages for next epoch)
```

Claim 5.1. *Let X_i be the random variable denoting the number of evictions of page i in the solution constructed by the adversary as above. Then $\mathbb{E}[\max_i X_i] = O(N/k)$. Further, for any $p \le \log n$, $\mathbb{E}[(\sum_i X_i^p)^{1/p}] = n^{1/p} \cdot O(N/k)$.*

A Deterministic (Fractional) Algorithm's Solution. Fix any deterministic (fractional) algorithm $\mathcal{A}$. Consider a block B_{eb} in an epoch e. By standard coupon collector arguments, each phase H_ℓ in this block has $\Omega(k \log k)$ page requests with high probability. Since each request in H_ℓ selects a random page from B_{eb}, which has $(k+1)$ pages, the algorithm incurs eviction cost of $\frac{1}{k+1}$ in expectation for each request. Let $X_e(i)$ denote the number of evictions of a page i in epoch e. Then $(1/|A_e|) \cdot \sum_{i \in A_e} \mathbb{E}[X_e(i)] = \Omega((N/k) \log k)$. Since the sets A_e are randomly selected, the expected eviction cost on a random page in A_e is $\Omega(e(N/k) \log k)$. Using this, we show (in the full version):

Claim 5.2. *Let X_i be the evictions incurred by page i in algorithm $\mathcal{A}$ on the instance given in Algorithm 1. Then, $\mathbb{E}[\max_i X(i)] = (N/k) \cdot \Omega(\log n \log k)$ and $\mathbb{E}[(\sum_i (X(i))^p)^{1/p}] = n^{1/p} \cdot (N/k) \cdot \Omega(p \log k)$.*

Combining Claim 5.2 and Claim 5.1, we get Theorem 1.2.

6 Closing Remarks

In this paper we showed how to reduce convex paging problems (for a wide class of convex functions) to weighted (linear) paging problems, for both randomized and deterministic settings, and also for integer as well as fractional optimization. A crucial aspect of our work is the black-box nature of the reduction, so that we can

take algorithms for the well-studied linear cases and lift them to the convex case, without having to reinvent the wheel (e.g., the clever rounding approaches known in linear settings). This allows us to give tight algorithms for min-max paging and ℓ_p-norm paging, thereby closing gaps in results given by previous works.

Our work suggests several interesting future directions. Can we extend these techniques and/or results to other classes of online covering problems? The analogous paradigm for fractional problems – obtaining online algorithms to covering programs with non-linear convex objectives by employing a local linear approximation – has been applied to a broad class of problems such as mixed linear programming, ℓ_p-norm set cover, machine activation, ℓ_p-norm scheduling, capacitated facility location, and more [3]. In fact, the same framework is also useful for appropriately defined dual online packing problems such as variants of social welfare maximization. Can we extend our algorithmic paradigm to obtain new algorithms for *integral* online covering problems in other domains beyond caching? What kinds of results can be obtained for *integral* online packing problems using this framework?

Acknowledgments

AG and DP were supported in part by NSF grants CCF-2422926 and CCF-2224718, and CCF-1955703 and CCF-2329230 respectively. DP would also like to acknowledge the support of Google Research and the Simons Institute for the Theory of Computing at UC Berkeley for hosting him during his sabbatical from Duke University in 2023-24, when a part of this research was conducted. AG thanks Google Research for their support.

References

[1] Adi Avidor, Yossi Azar, and Jirí Sgall. 2001. Ancient and New Algorithms for Load Balancing in the l_p Norm. *Algorithmica* 29, 3 (2001), 422–441.

[2] Baruch Awerbuch, Yossi Azar, Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. 1995. Load Balancing in the L_p Norm. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 383–391.

[3] Yossi Azar, Niv Buchbinder, T.-H. Hubert Chan, Shahar Chen, Ilan Reuven Cohen, Anupam Gupta, Zhiyi Huang, Ning Kang, Viswanath Nagarajan, Joseph Naor, and Debmalya Panigrahi. 2016. Online Algorithms for Covering and Packing Problems with Convex Objectives. In *Proceedings of the 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 148–157.

[4] Yossi Azar and Amir Epstein. 2005. Convex programming for scheduling unrelated parallel machines. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*. 331–337.

[5] Yossi Azar, Amir Epstein, and Leah Epstein. 2003. Load Balancing of Temporary Tasks in the ℓ_p Norm. In *Approximation and Online Algorithms, First International Workshop, (WAOA) (Lecture Notes in Computer Science, Vol. 2909)*. 53–66.

[6] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. 2012. A Primal-Dual Randomized Algorithm for Weighted Paging. *J. ACM* 59, 4 (2012), 19:1–19:24.

[7] Nikhil Bansal and Kirk Pruhs. 2003. Server scheduling in the L_p norm: a rising tide lifts all boats. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*. 242–250.

[8] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. 2001. A unified approach to approximating resource allocation and scheduling. *J. ACM* 48, 5 (2001), 1069–1090.

[9] Laszlo A. Belady. 1966. A Study of Replacement Algorithms for Virtual-Storage Computer. *IBM Systems Journal* 5, 2 (1966), 78–101.

[10] Ioannis Caragiannis. 2008. Better bounds for online load balancing on unrelated machines. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, Shang-Hua Teng (Ed.). 972–981.

[11] Ashish Chiplunkar, Monika Henzinger, Sagar Sudhir Kale, and Maximilian Vötsch. 2023. Online Min-Max Paging. In *Proceedings of the 34th ACM-SIAM Symposium on Discrete Algorithms, (SODA)*. 1545–1565.

[12] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. 1991. New Results on Server Problems. *SIAM J. Discrete Math.* 4, 2 (1991), 172–181.

[13] Edith Cohen and Haim Kaplan. 1999. LP-based Analysis of Greedy-dual-size. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 879–880.

[14] Nikhil R. Devanur and Zhiyi Huang. 2018. Primal Dual Gives Almost Optimal Energy-Efficient Online Algorithms. *ACM Trans. Algorithms* 14, 1 (2018), 5:1–5:30.

[15] Nikhil R. Devanur and Kamal Jain. 2012. Online matching with concave returns. In *Proceedings of the 44th Symposium on Theory of Computing Conference, (STOC)*. 137–144.

[16] Devdatt P. Dubhashi and Alessandro Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press.

[17] Matthias Englert and Harald Räcke. 2009. Oblivious Routing for the L_p-norm. In *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 32–40.

[18] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. 1991. Competitive Paging Algorithms. *J. Algorithms* 12, 4 (1991), 685–699.

[19] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. 2012. Online Primal-Dual For Non-linear Optimization with Applications to Speed Scaling. In *Workshop on Approximation and Online Algorithms (Lecture Notes in Computer Science, Vol. 7846)*. 173–186.

[20] Sungjin Im, Nathaniel Kell, Janardhan Kulkarni, and Debmalya Panigrahi. 2019. Tight Bounds for Online Vector Scheduling. *SIAM J. Comput.* 48, 1 (2019), 93–121.

[21] Sungjin Im and Benjamin Moseley. 2011. Online Scalable Algorithm for Minimizing k-norms of Weighted Flow Time On Unrelated Machines. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 95–108.

[22] Thomas Kesselheim, Marco Molinaro, and Sahil Singla. 2023. Online and Bandit Algorithms Beyond ℓ_p Norms. In *Proceedings of the 34th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Nikhil Bansal and Viswanath Nagarajan (Eds.). 1566–1593.

[23] Ishai Menache and Mohit Singh. 2015. Online Caching with Convex Costs: Extended Abstract. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 46–54.

[24] Qifan Pu, Haoyuan Li, Matei Zaharia, Ali Ghodsi, and Ion Stoica. 2016. FairRide: Near-Optimal, Fair Cache Sharing. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 393–406.

[25] Daniel Dominic Sleator and Robert Endre Tarjan. 1985. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM* 28, 2 (1985), 202–208.

[26] Shanjiang Tang, Qifei Chai, Ce Yu, Yusen Li, and Chao Sun. 2020. Balancing Fairness and Efficiency for Cache Sharing in Semi-external Memory System. In *Proceedings of the 49th International Conference on Parallel Processing (ICPP)*. ACM, 13:1–13:11.

Received 2024-11-04; accepted 2025-02-01