Online Min-Max Paging

Ashish Chiplunkar
Indian Institute of Technology Delhi
ashishc@iitd.ac.in *

Monika Henzinger
Department of Computer Science
University of Vienna
monika.henzinger@univie.ac.at

Sagar Sudhir Kale
Department of Computer Science
University of Vienna
sagar.kale@univie.ac.at

Maximilian Vötsch
Department of Computer Science,
UniVie Doctoral School Computer Science DoCS
University of Vienna
maximilian.voetsch@univie.ac.at

Abstract

Motivated by fairness requirements in communication networks, we introduce a natural variant of the online paging problem, called *min-max* paging, where the objective is to minimize the maximum number of faults on any page. While the classical paging problem, whose objective is to minimize the total number of faults, admits k-competitive deterministic and $O(\log k)$-competitive randomized algorithms, we show that min-max paging does not admit a $c(k)$-competitive algorithm for any function c. Specifically, we prove that the randomized competitive ratio of min-max paging is $\Omega(\log(n))$ and its deterministic competitive ratio is $\Omega(k \log(n)/\log(k))$, where n is the total number of pages ever requested.

We design a fractional algorithm for paging with a more general objective – minimize the value of an n-variate differentiable convex function applied to the vector of the number of faults on each page. This gives an $O(\log(n)\log(k))$-competitive fractional algorithm for min-max paging. We show how to round such a fractional algorithm with at most a k factor loss in the competitive ratio, resulting in a deterministic $O(k \log(n)\log(k))$-competitive algorithm for min-max paging. This matches our lower bound modulo a $\text{poly}(\log(k))$ factor. We also give a randomized rounding algorithm that results in a $O(\log^2 n \log k)$-competitive algorithm.

1 Introduction

Paging is a decades-old, classical computer science problem. Suppose a computer process working on n *pages* of data has access to two levels of memory: a fast memory, called the *cache*, that can hold a small amount k of pages, and a slow memory containing all n pages. Typically, k is much smaller than n, and initially, all pages are in slow memory. Whenever the process accesses a page, it is read from the cache; if it is not already in the cache, a *page fault* occurs, and the page must be brought into the cache, which

*Ashish Chiplunkar is partially supported by the Pankaj Gupta New Faculty Fellowship.

M. Henzinger and M. Vötsch: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564 "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and from the Austrian Science Fund (FWF) project "Fast Algorithms for a Reactive Network Layer (ReactNet)", P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024.

possibly necessitates *evicting* another page from the cache to make room. This is called *serving* the request. In the *online* setting, each request must be served before the algorithm sees the subsequent request. The goal is to minimize the *total number of page faults* incurred while serving a sequence of requests.

The paging problem has found new applications in communication networks, where caching is ubiquitous and is used to minimize energy usage, communication latency, and network traffic. Consider, for example, TCP connections that are kept alive on a router [CKZ99] or optical links in reconfigurable data center topologies [FS19, BFMS21]. Every user application prefers to have an active connection, as re-establishing a TCP connection or link takes time and slows down communication or computation. Another example is content on web pages that is cached in a content delivery network, such as Akamai. In practice, the cache servers in these networks rely on dynamic, eviction-based algorithms for managing cache contents that solve the so-called content placement problem [TKR21]. Web pages in the cache have a clear advantage as they can be served faster to the user than web pages that must be re-fetched from the server. Ideally, all applications (of the same priority) and all web pages should be treated equally. This motivates us to propose the study of a *fair* variant of paging, which we call *min-max paging*. Its goal is to minimize the number of page faults on *any* page, i.e. to minimize the *maximum number of page faults of any single page*.

In the *online* setting, the page requests are revealed one by one without knowledge of the future, so the description of how to serve each request must depend only on the request sequence thus far and the current cache contents. Naturally, for many problems, an online algorithm cannot output an optimal solution to a given instance – something an *offline* algorithm having access to the entire input can produce. The sub-optimality of an online algorithm is usually measured using competitive analysis. Informally, we say that an online algorithm has a competitive ratio of c if, on every problem instance, it produces a solution with (expected) cost at most c times the cost of an optimal solution. For the classic online paging problem the competitive ratio has been well-studied: It is $O(k)$ for deterministic algorithms [ST85, KMRS86] and $H_k \approx 0.577 + \ln(k)$ for randomized algorithms [FKL$^+$91, MS91].

To the best of our knowledge, the min-max paging problem has not been studied before. While an efficient offline algorithm for the classical paging problem is known, we neither have an efficient offline algorithm for min-max paging nor a proof of NP-hardness. In this paper, we focus on the min-max paging problem in the *online* setting and give both upper and lower bounds on its competitive ratio.

Our results We first propose an algorithm for the *fractional paging problem* with objective function f, where pages can be held in the cache fractionally, subject to having a total volume of at most k pages at all times. The objective function f is an arbitrary function from an appropriately defined subclass of convex functions applied to the *fault vector*. Here, the fault vector refers to the n-dimensional vector of the number of faults incurred on each page, where n is the number of pages. We use the theory of convex programming and properties of f to analyze our algorithm and establish the following bound.

Theorem 1 (Stated formally as Theorem 19)**.** *For the fractional paging problem with objective function f, there exists a $(2q\log(k+1))^q$-competitive algorithm, provided f grows no faster than a degree-q polynomial.*

In particular, when instantiating f to be the q'th power of the q-norm, we get the following bound:

Theorem 2 (Stated formally as Theorem 20)**.** *For the fractional paging problem with the objective of minimizing the ℓ_q-norm of the fault-vector, there exists a $2q\log(k+1)$-competitive algorithm.*

Note that the above theorem does not give a sensible result for the ℓ_∞-norm, which is the objective function we are interested in. However, using the fact that the ℓ_∞-norm of an n-dimensional vector is well-approximated by its $\ell_{\log(n)}$-norm, we get the following result.

Theorem 3 (Stated formally as Theorem 22)**.** *For the fractional paging problem with the objective of minimizing the ℓ_∞-norm of the fault-vector (a.k.a. fractional min-max paging), there exists an $O(\log(n)\log(k))$-competitive algorithm.*

Next, we propose two approaches for rounding solutions of fractional min-max paging algorithms online and obtain the following two results. Note that the bound of the latter result is better than the former in the $k = \omega(\log n)$ regime, and it also rules out a lower bound linear in k for randomized algorithms.

Theorem 4 (Stated formally as Corollary 24)**.** *There exists an $O(k\log(k)\log(n))$-competitive* deterministic *algorithm for min-max paging.*

Theorem 5. *There exists an $O(\log^2 n \log k)$-competitive* randomized *integral algorithm for min-max paging.*

We complement the above upper bounds by the following impossibility results.

Theorem 6 (Stated formally as Theorem 3)**.** *Every deterministic algorithm for min-max paging is $\Omega(k \log(n)/\log(k))$-competitive.*

Theorem 7 (Stated formally as Theorem 9)**.** *Every algorithm for min-max paging is $\Omega(\log(n))$-competitive.*

Note that we only have a $O(\log^2 k)$ discrepancy between our deterministic bounds, i.e., the bounds are tight up to a polylogarithmic in k factor. Moreover, our lower bounds show that min-max paging is fundamentally more difficult than classical paging and its several generalizations (see Section 6), which admit competitive ratios independent of n, the total number of pages.

We now present some intuition why algorithms for the classical paging problem and a simple algorithm for min-max paging fail to achieve anything better than a trivial competitive ratio for min-max paging. Algorithms for the classical paging problem are oblivious to the number of faults a single page has incurred while processing the sequence σ up to a given point in time t. Consider the Least Recently Used (LRU) algorithm, which evicts the page whose last request was before the requests to other pages in the cache. Let $p_0 \in P = \{p_0, p_1, \ldots, p_n\}$ and assume that $n = |P| - 1 = mk$ is a large multiple of k. The sequence $\sigma = (p_0, p_1, p_2, \ldots, p_k, p_0, p_{k+1}, p_{k+2}, \ldots, p_{2k}, p_0, p_{2k+1}, \ldots, p_{mk}, p_0)$ will cause the LRU algorithm to fault $m+1$ times on page p_0, while the optimal algorithm faults exactly once per page. We pair each request to p_0 with requests to a set of k pages. After processing these k requests, LRU will have ejected p_0, so the next request to p_0 will result in a page fault, yielding in a competitive ratio of $\Omega(n/k)$.

Another obvious strategy is to greedily keep the k pages which have incurred the most faults thus far in the cache. In this case, there also exists a request sequence for which the strategy is no better than $\frac{n}{k}$-competitive. For simplicity's sake, let us assume that $k = 2$. Then the request sequence is constructed as follows: (1) Request p_1, p_2, p_3 in this order N times, where N is a parameter. (2) Request p_4, p_5 until the algorithm includes *both* of them into the cache. (3) Request p_4, p_5, p_6 in this order N times. (4) Repeat steps 2 and 3 with pages p_7, p_8, and p_9 next, then with pages p_{10}, p_{11}, and p_{12}, and so on.

After step 3, the greedy algorithm will hold two pages of cost $(r+1)N$, where r is the number of times we have repeated steps 2 and 3. In step 2, we request a set of new pages, and the greedy algorithm will fault on them until they reach cost $(r+1)N$. During step 3, the algorithm will fault N times on each page, making it so that it has cost $(r+2)N$ on the pages introduced in step 2. Meanwhile, the cost of the optimal offline algorithm is no greater than N, obtained by immediately adding the pages of step 2 to the cache.

Our techniques Our lower bound of $\Omega(k\log(n)/\log(k))$ is established by generalizing the above construction, using the following approach: The adversary fixes a sequence of requests over a set of n pages, which can be served while keeping the number of faults on any page small. The core idea is to successively reduce the set of pages that we request in the future in such a way that the algorithm *cannot predict*

which pages will stop being requested. A clairvoyant adversary processes the sequence so that she initially incurs a small number of faults on pages that will be requested many times in the future. This causes the adversary to have roughly uniform cost over all pages, while the algorithm has one page on which it has faulted many times.

To design an online algorithm one could try to use standard techniques to transform a max-based objective function into a linear program and solve the corresponding linear program online. However, this does not work as all known online algorithms for linear programs only work with *exclusively packing or exclusively covering constraints* and can not handle a mix of constraints, except for [ABFP13], which cannot handle *box constraints,* i.e., an upper bound on the variables as required for paging.

Thus, to solve the online min-max paging problem, we solve a more general problem: We give a $O((q \log k)^q)$-competitive algorithm for a *fractional paging problem, which minimizes a convex, differentiable function with q-bounded growth* and *an upper bound constraint (i.e., a box constraint) on each variable.* A function with q-bounded growth behaves like a polynomial function of degree q. To the best of our knowledge, this problem has not been studied before, and no non-trivial online algorithm is known.

As our cost function is not linear, the combinatorial technique of potential functions used for server problems with linear cost functions breaks down. Informally, a potential function captures the advantage accumulated by the adversary at any time, which she can use to make the algorithm "pay" more than herself in the future. The potential function is a function on the *state space* of the problem, where the state of the algorithm, at any time, fully determines its future behavior. The state space is usually a small set when the objective is linear. On the contrary, in the case of min-max paging, a state must capture the vector of faults accumulated on each page and its current cache, and there can be multiple fault vectors for the same current cache, which makes the state space blow up with every request, thus, making the use of potential functions challenging, messy, and inelegant.

Instead, we build on the work of Azar et al. [ABC+16], which minimizes a convex cost function with linear constraints of *row sparsity* ρ. Their approach requires the variables $x_{p,j}$ to be unbounded, and for q-bounded growth functions, it gives an $O((q \log \rho)^q)$-competitive algorithm.

We also draw on ideas from Bansal et al. [BBN12b], which studied the weighted paging problem with linear cost functions. They first compute a fractional solution using a primal-dual approach and then show how to round it. As they have a linear cost function, they can show that the rate of increase of the primal, i.e., the fractional algorithm's cost, is proportional to the rate of increase of the dual. In our setting, the cost function is not linear, and we have to use the theory of duality of convex programs and conjugate duals. To do so, we extend their analysis to the convex program setting, which requires solving various technical hurdles. This results in a $(2q \log(k+1))^q$-competitive algorithm for fractional paging with any convex, differentiable function with q-bounded growth and box constraints. Furthermore, for norm-objective functions, more specifically for q-norms, we achieve a competitive ratio of $2q \log k$. Since the cost function of min-max paging is the ℓ_∞-norm of the vector of page-wise costs, we approximate it by $\ell_{\log n}$-norm, resulting in a $2e \log n \log(k+1)$-competitive algorithm for fractional min-max paging.

We round our solution deterministically using for every page p a threshold for x_p of $1 - 1/k$, resulting in the upper bound of $O(k \log n \log k)$. It might be tempting to apply the randomized rounding algorithm of [BBN12b] directly, but it does not apply as it crucially uses the fact that the cost of the algorithm is the sum of the fractional values of all pages. Instead, we adapt the scheme of [BNT21] from the weighted paging setting to the min-max setting. Specifically, this requires to "charge" the cost of each rounding step to each individual page, as opposed to the sum of the changes in the fractional solution over all pages. This charging to individual pages has not been done before in online paging and might be interesting in other settings.

In Section 2, we give all definitions. In Section 3, we show our lower bounds, in Section 4, we present and analyze our algorithm for paging with convex objective functions. In Section 5 we round the fractional algorithm to obtain an $O(k \log(n) \log(k))$-competitive deterministic and $O(\log^2(n) \log(k))$-competitive

randomized algorithms for (integral) min-max paging. All omitted proofs are given in the appendix.

2 Preliminaries

The problems in this paper are studied in the online setting, where an adversary fixes a request sequence σ ahead of time, and the requests in this sequence are presented to an algorithm one by one. When the algorithm receives a new request from the sequence, it can only use its knowledge of the requests seen thus far to make a decision. In particular, the algorithm does not have any knowledge of future requests.

In this setting, we use *competitive analysis* [ST85] to measure the quality of an algorithm. In competitive analysis, we study the *competitive ratio* of an online algorithm, which compares the worst-case ratio between the cost of the algorithm and the cost of an optimum offline solution over all possible σ.

More formally, for a deterministic algorithm ALG, the competitive ratio of ALG is the smallest $c \in \mathbb{R}$, such that for all instances σ of an online minimization problem, we have

$$\mathrm{ALG}(\sigma) \leqslant c \cdot \mathrm{OPT}(\sigma) + d,$$

where $\mathrm{ALG}(\sigma)$ is the cost of the algorithm, $\mathrm{OPT}(\sigma)$ is the cost of the optimum offline solution, and d is some constant independent of σ. We will call an algorithm fulfilling the above definition a c-competitive algorithm. If ALG is a randomized algorithm, then the competitive ratio is defined as the smallest $c \in \mathbb{R}$ such that

$$\mathbb{E}[\mathrm{ALG}(\sigma)] \leqslant c \cdot \mathrm{OPT}(\sigma) + d.$$

We study a variant of the paging problem called *min-max paging*. In any paging problem the request sequence σ is made up of requests to a set of pages $P = \{p_1, p_2, \ldots, p_n\}$ of size n. We will assume that σ is of finite length, denoted by T. The algorithm is given a cache C of size k, which always is a subset of P and is empty when the algorithm begins processing σ.

When page p is requested during round t, we must add p to the cache C if it is not already contained in C. If adding the page causes C to be of size $k+1$, we must evict a page other than p from the cache before we are allowed to process the next request. The situation where a request to page p arrives while p is not in C is called a *page fault*.

Whenever a page fault occurs, we incur some cost. The objective of the classical paging problem is to minimize the total number of page faults. In the case of min-max paging, the objective is to minimize the maximum number of page faults occurring for any page. More precisely, if we let $x_{p,j}$ be a zero-one variable, which denotes that a page fault occurs upon the j-th request to page p, then we seek to minimize

$$\max_{p \in P} \sum_{j} x_{p,j},$$

where the summation is over all requests to p. We can think of this as minimizing the ℓ_∞-norm of the vector $\vec{c}(\sigma) = (\sum_j x_{p_1,j}, \ldots, \sum_j x_{p_n,j})^\top$, whereas the classical paging problem is equivalent to minimizing the ℓ_1-norm of $\vec{c}(\sigma)$.

In Section 4 we solve a fractional version of the paging problem for convex objective functions $f(x)$, where x is the vector consisting of the variables $x_{p,j}$, under the assumption that $f(x)$ is well behaved. Of particular interest is the case where $f(x) = \|\vec{c}(\sigma)\|_q$, i.e. the ℓ_q-norm. We refer to this case as *q-paging*. For details refer to Section 4.

Remark 8. Paging problems are studied in the eviction cost model, where fetching a page incurs no cost, and the algorithm pays for evicting a page, and in the fetching cost model, where evicting a page comes without an associated cost, and the algorithm pays for fetching a page. For min-max paging, the cost of

these models differs by at most 1. Said difference occurs on the set of pages contained in the cache at time T that the algorithm does not have to evict anymore.

Because of this equivalence, we use both models interchangeably in this paper. The lower bounds of Section 3 use the fetching cost model, and the upper bounds of Section 4 use the eviction cost model, as the choice of the respective model simplifies the proofs.

3 Lower Bounds

We show a deterministic lower bound of $\Omega(k(\log n)/\log k)$ and a randomized lower bound of $\Omega(\log n)$ (for $k = 2$) on the competitive ratio for min-max paging. Our lower bounds are based on a simple construction that is cleanly demonstrated with $k = 2$ and can be generalized for $k \geqslant 2$. The interested reader will find complete proof for the deterministic lower bound in the appendix in Section A.

Any deterministic algorithm for min-max paging with cache size k is at least $\frac{k-1}{2}\log_{k+1} n$-competitive, where n is the number of pages.

Theorem 9. *The randomized competitive ratio of min-max paging is $\Omega(\log n)$, where n is the number of pages when the cache size is $k = 2$.*

By Yao's principle, it suffices to exhibit a probability distribution on input instances, forcing every deterministic online algorithm to perform a factor $\Omega(\log n)$ worse in expectation than the optimum cost. Let $n = 3^m$ for some large integer m. Our adversarial strategy takes a parameter $N \gg n$ and is defined as follows.

Algorithm 1 An adversarial strategy for min-max paging

```
Let L_m = {p_0^m, ..., p_{n-1}^m} be a set of n = 3^m pages.
for ℓ = m to 1 do
    L_{ℓ-1} ← ∅.
    for i = 0 to 3^{ℓ-1} − 1 do
        Give N requests to each of p_{3i}^ℓ, p_{3i+1}^ℓ, p_{3i+2}^ℓ in a round-robin manner.
        p_i^{ℓ-1} ← a uniformly random page from {p_{3i}^ℓ, p_{3i+1}^ℓ, p_{3i+2}^ℓ}.
        Add p_i^{ℓ-1} to L_{ℓ-1}.
```

We call each iteration of the outer for-loop a *layer* and each of the inner for-loop a *phase*. We number the layers $m, m-1, \ldots, 1$.

Lemma 10. *The adversary can serve all requests while faulting at most $m + N$ times on every page with probability one.*

Proof. Consider an arbitrary phase of an arbitrary layer ℓ. Let q be the page added to $L_{\ell-1}$ at the end of the phase, and let q_1, q_2 be the other two pages requested in the phase. On the first request to q, the adversary will add q to its cache and keep it there until the end of the phase. It uses the remaining cache slot to serve all requests to q_1 and q_2. Thus, the adversary faults only once on q and N times on q_1 and q_2 each.

Consider an arbitrary page p. In all phases where p is requested except the last one, the adversary faults only once on p (p is the page q in the above argument). In the last phase, the adversary faults N times on p. Every layer contains at most one phase in which p is requested. Since the number of layers is m, the algorithm faults at most $m + N$ times on p. □

To analyze the algorithm's performance, let the random variable X_i^ℓ be the number of the algorithm's faults on the randomly chosen page p_i^ℓ at the beginning of layer ℓ.

Lemma 11. *For every layer ℓ and every $i \in \{0,\ldots,3^\ell - 1\}$ we have $\mathbb{E}[X_i^\ell] \geqslant (m-\ell)\cdot N/2$.*

Proof. We prove the claim by reverse induction on ℓ. Recall the numbering of phases and observe that $X_i^m = 0$ for all $i \in \{0,\ldots,n-1\}$. Thus, the claim is true for $\ell = m$. Assuming as induction hypothesis that for every $i \in \{0,\ldots,3^\ell - 1\}$ we have $\mathbb{E}[X_i^\ell] \geqslant (m-\ell)\cdot N/2$, we prove that for every $j \in \{0,\ldots,3^{\ell-1}-1\}$ we have $\mathbb{E}[X_j^{\ell-1}] \geqslant (m-\ell+1)\cdot N/2$.

In any phase, since the cache size is 2 and three pages are requested in a round-robin manner N times each, the total number of faults is at least $3N/2$. This is evident if we consider the behavior of the optimal algorithm for (usual) paging that always evicts the page needed farthest in the future. Consider the j'th phase of layer ℓ, and recall that $p_j^{\ell-1}$ is defined at the end of this phase. The total number of faults in this phase is at least $3N/2$, and these faults are distributed over the three pages, $p_{3j}^\ell, p_{3j+1}^\ell, p_{3j+2}^\ell$. Since $p_j^{\ell-1}$ is uniformly random among these three pages, the expected number of faults on $p_j^{\ell-1}$ during layer ℓ is at least $N/2$. Again, since $p_j^{\ell-1}$ is uniformly random among $\{p_{3j}^\ell, p_{3j+1}^\ell, p_{3j+2}^\ell\}$, by linearity of expectation we have,

$$\mathbb{E}[X_j^{\ell-1}] \geqslant \frac{\mathbb{E}[X_{3j}^\ell] + \mathbb{E}[X_{3j+1}^\ell] + \mathbb{E}[X_{3j+2}^\ell]}{3} + \frac{N}{2}.$$

By the induction hypothesis, each of $\mathbb{E}[X_{3j}^\ell]$, $\mathbb{E}[X_{3j+1}^\ell]$, $\mathbb{E}[X_{3j+2}^\ell]$ is at least $(m-\ell)\cdot N/2$. Thus, $\mathbb{E}[X_j^{\ell-1}] \geqslant (m-\ell+1)\cdot N/2$, as required. □

Having proven Lemma 10 and Lemma 11, we are ready to prove the claimed lower bound.

Proof of Theorem 9. By Lemma 10, the cost of the adversary's solution to the random instance generated by the adversarial strategy is $m+N$ with probability one. Note that at the end of the adversarial strategy, we are left with the singleton set L_0 containing the page p_0^0. The number of faults of the algorithm on page p_0^0 is a lower bound on the algorithm's cost with probability one. Thus, the algorithm's expected cost is at least the expectation of the number of algorithm's faults on p_0^0. By Lemma 11, this quantity is $\mathbb{E}[X_0^0] \geqslant mN/2$. Thus, the ratio of the algorithm's expected cost to the adversary's cost is at least $mN/(2\cdot(m+N))$, which approaches $m/2 = (\log_3 n)/2$ as $N \to \infty$. Thus, the competitive ratio of any randomized algorithm for min-max paging is at least $(\log_3 n)/2 = \Omega(\log n)$. □

4 A Fractional Algorithm for General Paging

We study a general class of convex objective functions for the paging problem to arrive at a competitive algorithm for min-max paging. Let $x \in \mathbb{R}^T$ be the vector consisting of the variables $x_{p,j}$ in order of appearance in σ. The objective functions $f : \mathbb{R}^T \to \mathbb{R}$ which we consider in this section have the following properties: (1) $f(0) = 0$; (2) $f(x)$ is a monotonically increasing function in x; (3) $\nabla f(x)$ is monotonically increasing in each coordinate; and (4) $f(x)$ has *q-bounded growth*, i.e. there exists a positive integer q such that for all $x \in \mathbb{R}_+^T$, $\langle \nabla f(x), x\rangle \leqslant q f(x)$. In particular, any polynomial function of x of degree q will fulfill these requirements.

We formulate the general paging problem as an online convex program. Given a convex function $f : \mathbb{R}_+^n \to \mathbb{R}$ and a matrix $A \in \mathbb{R}^{m\times n}$, a general (offline) convex programming problem is to minimize $f(x)$ subject to $Ax \geqslant \mathbf{1}$ and $x \geqslant \mathbf{0}$.

In online convex programming, the rows of the constraint matrix A are revealed one by one, corresponding to the request sequence σ. Upon receiving the tth row A_t of the constraint matrix, the task of the algorithm is to increase the variables x until the constraint $A_t x \geqslant 1$ is fulfilled. The algorithm is never allowed to decrease any of the variables in x.

In the fractional convex program for paging, we denote by p_t the page requested in round t. Furthermore, we let $r(p,t)$ indicate the number of requests to page p up to and including round t, and let $t(p,j)$ be the round during which the page p is requested for the j'th time. As each round corresponds uniquely to a pair (p,j), we have $\sum_p r(p,t) = T$. We let $B(t) = \{p \in P | r(p,t) \geqslant 1\}$ be the set of distinct pages encountered up to, and including, round t. The variables $x_{p,j}$ can now take values in the interval $[0,1]$ and indicate the fraction of the page the algorithm has removed from the cache between the j'th and $j+1$'st times it was requested. Using this notation, the convex program for general paging looks as follows:

$$\begin{array}{lll} \text{minimize} & f(x) & \\ \text{subject to} & \sum_{p \in P \setminus \{p_t\}} x_{p,r(p,t)} \geqslant |B(t)| - k & \forall t \in [T] \\ & 0 \leqslant x_{p,j} \leqslant 1 & \forall p \in [n], j \in [r(p,T)] \end{array} \tag{1}$$

By using a convex objective function, this formulation generalizes prior work on online paging, including weighted paging [BBN12b]. Crucially, the box constraint $0 \leqslant x_{p,j} \leqslant 1$ means that the online convex programming framework of [ABC+16] can not be used to solve this program.

At the beginning of round t, we are given a new variable $x_{p_t,r(p_t,t)}$, which is initialized to 0 along with the constraint $\sum_{p \in B(t) \setminus \{p_t\}} x_{p,r(p,t)} \geqslant |B(t)| - k$. This constraint ensures that after each round t, at least $|B(t)| - k$ fractional page mass has been ejected, or, equivalently, at most k fractional page mass is inside the cache. We observe that the variable $x_{p,j}$ will only appear in the constraints corresponding to rounds $t \in \{t(p,j)+1, t(p,j)+2, \ldots t(p,j+1)-1\}$, i.e. the variable $x_{p,j}$ does not appear in round $t(p,j)$ when it is requested. This is because we are not allowed to increase $x_{p,j}$ during this round, as page p is required to be fully inside the cache in round $t(p,j)$, in order to serve the request.

In order to define a dual for the convex program 1, we will need the following definition:

Definition 12. Given a request sequence σ of length T consisting of pages from the set P, we can uniquely, up to relabeling of pages, define a constraint matrix $A \in \{0,1\}^{T \times T}$ as

$$A_{t,(p,j)} = \begin{cases} 1 & \text{if } t \in [t(p,j)+1, t(p,j+1)-1] \\ 0 & \text{otherwise.} \end{cases}$$

In round t, we can determine all non-zero entries, as they only depend on the variables encountered up to round t. Additionally, we can implicitly set the columns corresponding to future variables to 0. If we order both the columns and rows by order of appearance, then the constraint matrix will be lower triangular, see Figure 1 in the appendix.

Definition 13. Let $f : \mathbb{R}_+^T \to \mathbb{R}$ be a convex function. The fenchel dual $f^* : \mathbb{R}_+^T \to \mathbb{R}$ of f is defined as $f^*(y) = \sup_{w \in \mathbb{R}_+^T} (\langle w, y \rangle - f(w))$, where $\langle w, y \rangle = \sum_{i=0}^T w_i \cdot y_i$ denotes the Euclidean scalar product.

We need the following property of the Fenchel dual in the analysis of our algorithm:

Property 14. *The Fenchel dual $f^*(y) : \mathbb{R}_+^T \to \mathbb{R}$ of a convex function f is monotonically increasing in y.*

The dual will consist of two sets of T variables each, denoted by y_t and $z_{p,j}$, respectively. We let y be the vector consisting of the y_t ordered increasingly in t and z being the vector consisting of the $z_{p,j}$ ordered the same way as x.

We will use the following *conjugate dual* $D(y,z)$ for our primal-dual algorithm. For the convex primal (1), the conjugate dual is:

$$\begin{array}{lll} \text{maximize} & D(y,z) = \sum_{t=1}^T (|B(t)| - k) y_t - \sum_{p,j} z_{p,j} - f^*(A^\top y - z), & \\ \text{subject to} & 0 \leqslant y_t & \forall t \in [T] \\ & 0 \leqslant z_{p,j} & \forall p \in [n], j \in [r(p,T)], \end{array}$$

This dual differs from the dual used in [BBN12b] by the inclusion of the Fenchel dual term $f^*(A^\top y - z)$ and from the dual used in [ABC+16] by the use of non-uniform coefficients for the y_t variables and the inclusion of the variables $z_{p,j}$. The dual will only be used to obtain a lower bound on OPT for the analysis of our algorithm, and it does not influence the primal solution the algorithm produces. It remains to show that the stated dual fulfills this property for the convex program (1):

Lemma 15 (Weak Duality). *For any feasible $x \in [0,1]^T$ and $y, z \in \mathbb{R}^T_+$, we have*

$$f(x) \geqslant \sum_{t \in [T]} (|B(t)| - k) y_t - \sum_{p,j} z_{p,j} - f^*(A^\top y - z).$$

Our online algorithm, given in Algorithm 2 uses a continuous time τ, which is 0 initially and increases throughout the algorithm. Let $\tau(t)$ denote the value of τ when we finish processing the t'th constraint and let $\tau(0) = 0$. As all variables are 0 at creation and increase at a rate dependent on τ, we use $x_{p,j}(\tau)$, $y_t(\tau)$, $z_{p,j}(\tau)$ to denote the values of the variables $x_{p,j}$, y_t, $z_{p,j}$ respectively at time τ. Let $t(\tau)$ be the unique t such that $\tau \in [\tau(t-1), \tau(t))$. The algorithm uses parameters r and $s_{p,j}(\tau)$ fixed later. Note that $s_{p,j}(\tau)$ depends on the value of τ, p and j and, thus, is not a constant parameter.

Our algorithm maintains dual variables y and z such that $A^\top y - z \leqslant \delta \nabla f(x)$ is approximately fulfilled, i.e. , $A^\top y - z \leqslant r\delta \nabla f(x)$ for some constant $r \geqslant 1$, which will then appear in the competitive ratio. We observe that if $f(x) = c^\top x$, the gradient is c and we get $A^\top y - z \leqslant c$, which is the dual constraint in the linear program for weighted paging. The reason why this point-wise upper bound is necessary is because, together with Property 14, it allows us to upper-bound the convex conjugate term $f^*(A^\top y - z)$ in $D(y,z)$ in terms of the primal function $f(x)$. For general w the conjugate $f^*(w)$ may be arbitrarily large as it is a convex function in w.

Algorithm 2 A fractional algorithm for min-max paging

Require: $r > 0$ and $s_{p,j}(\tau) > 0$, $s_{p,j}(\tau)$ monotonically decreasing in τ

1: $\tau \leftarrow 0$
2: **for** each round $t \in [T]$ **do**
3: let p_t be the page requested in this round
4: $x_{p_t,r(p,t)}(\tau) \leftarrow 0$, $y_t(\tau) \leftarrow 0$, $z_{p_t,r(p,t)}(\tau) \leftarrow 0$
5: $\frac{dy_t(\tau)}{d\tau} \leftarrow r$
6: $\frac{dx_{p,r(p,t)}(\tau)}{d\tau} \leftarrow \begin{cases} s_{p,r(p,t)}(\tau)\left(x_{p,r(p,t)}(\tau) + \frac{1}{k}\right) & \text{if } x_{p,r(p,t)}(\tau) < 1 \\ 0 & \text{otherwise} \end{cases}$
7: $\frac{dz_{p,r(p,t)}(\tau)}{d\tau} \leftarrow \begin{cases} r & \text{if } x_{p,r(p,t)}(\tau) = 1 \\ 0 & \text{otherwise} \end{cases}$
8: Increase τ, $y_t(\tau)$, $x_{p,r(p,t)}(\tau)$ and $z_{p,r(p,t)}(\tau)$ for all $p \in B(t) \setminus \{p_t\}$ simultaneously as per the above differential equations until $\sum_{p \in B(t) \setminus \{p_t\}} x_{p,r(p,t)}(\tau) \geqslant |B(t)| - k$

Next, to bound the conjugate term in the dual, it is necessary to obtain a bound on the dual "constraints" $A^\top y - z$, which we obtain by relating the constant growth of y_t and $z_{p,j}$ to the exponential growth of the $x_{p,j}$:

Lemma 16. *Let $\bar{x}$ denote the value of x after processing the complete request sequence σ, and similarly for $\bar{y}$ and $\bar{z}$. If $s_{p,j}(\tau)$ is monotonically decreasing in τ, then*

$$\bar{x}_{p,j} \geqslant \frac{1}{k}\left(\exp\left(\frac{s'_{p,j}}{r}\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t - \bar{z}_{p,j}\right)\right) - 1\right), \tag{2}$$

where $s'_{p,j}$ is the minimum value that $s_{p,j}(\tau)$ takes on during the execution of the algorithm.

The following is an immediate consequence of the previous lemma and the fact that $x_{p,j} \leqslant 1$:

Lemma 17. *The $x(\tau(t))$ produced by Algorithm 2 throughout its execution are feasible for the primal for all $t \in [T]$ and the vector $(A^\top \bar{y} - \bar{z})_{p,j} \leqslant \frac{r}{s'_{p,j}} \ln(k+1)$.*

The conjugate of a convex function with bounded growth can be bounded in terms of the original function and q, using the following lemma:

Lemma 18 ([ABC+16])**.** *Let $f : \mathbb{R}^T_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ be a monotone, convex, differentiable function satisfying $f(0) = 0$. If there is a $q > 1$ such that $\langle \nabla f(x), x \rangle \leqslant q f(x)$, then for any $0 < \gamma < 1$, $y \in \mathbb{R}^T_{\geqslant 0}$, $f^*(\gamma y) \leqslant \gamma^{\frac{q}{q-1}} \cdot f^*(y)$ and $f^*(\gamma \nabla f(y)) \leqslant \gamma^{\frac{q}{q-1}}(q-1) f(y)$.*

Theorem 19. *Let $f(x)$ be a convex function satisfying the requirements stated at the beginning of this section, and let σ be any request sequence. If we set $s_{p,j}(\tau) = \frac{\partial f(x)}{\partial x_{p,j}}^{-1}$ and $r = \frac{1}{\ln(k+1)(2q\ln(k+1))^{q-1}}$, then Algorithm 2 produces a $(2q\log(k+1))^q$-competitive solution $\bar{x}$ for fractional paging with objective function $f(x)$ in an online manner.*

Proof. By weak duality, it suffices to show that the primal is no larger than $O((q\log(k+1))^q)$ times the dual, which is a lower bound on the cost of an optimal solution x^* by weak duality. We will bound the primal and the dual growth rates for each round t. It suffices to only consider the case $A_t x(\tau) < |B(t)| - k$, as otherwise, the round is finished, and nothing needs to be done.

The processing of round t begins at time $\tau(t-1)$ and will last until $\tau(t)$, so we assume that $\tau \in (\tau(t-1), \tau(t)]$ for the remainder of this proof. Let $C_t(\tau) = \{(p,j) \mid t(p,j) < t < t(p,j+1) \text{ and } x_{p,j}(\tau) < 1\}$ be the set of indices of variables $x_{p,j}(\tau)$ in round t which correspond to a page that is (partially) in the cache, i.e., the indices of the variables corresponding to the latest request of a given p, which are increasing and have not been fully removed from the cache. Similarly, let $D_t(\tau) = \{(p,j) \mid t(p,j) < t < t(p,j+1) \text{ and } x_{p,j}(\tau) = 1\}$ be the set of indices of the variables $x_{p,j}(\tau)$ which have been fully removed from the cache since they have been last requested and which correspond to the latest request to a given page p. Note that $|C_t(\tau)| + |D_t(\tau)| = |B(t)| - 1$, as the sets $C_t(\tau)$ and $D_t(\tau)$ are disjoint and include a variable for each page except the page p_t. While processing the t-th constraint, we have, by the choice of $s_{p,j}(\tau)$, and the fact that $x_{p,j}(\tau)$ is constant if $(p,j) \in D_t(\tau)$:

$$G_1 := \frac{df(x)}{d\tau} = \sum_{p,j} \frac{\partial f(x)}{\partial x_{p,j}} \frac{\partial x_{p,j}(\tau)}{\partial \tau} = \sum_{(p,j)\in C_t(\tau)} \frac{\partial f(x)}{\partial x_{p,j}} \left(s_{p,j}(\tau) \left(x_{p,j}(\tau) + \frac{1}{k} \right) \right) \tag{3}$$

$$= \sum_{(p,j)\in C_t(\tau)} \left(x_{p,j}(\tau) + \frac{1}{k} \right) \leqslant |B(t)| - k - |D_t(\tau)| + \frac{|C_t(\tau)|}{k}. \tag{4}$$

The first equality is due to the chain rule for vector-valued functions. The second equality uses the definition of $\frac{\partial x_{p,j}(\tau)}{\partial \tau}$ and the fact that $x_{p,j}$ does not change for $(p,j) \notin C_t(\tau)$. And, the last inequality follows from the fact that the variables in $D_t(\tau)$ are all equal to 1.

Note that only the y_t corresponding to round t may increase during round t. For the linear term $\sum_t (|B(t)| - k) y_t(\tau) - \sum_{p,j} z_{p,j}(\tau)$ in the dual, it holds that in round t

$$G_2 := \frac{d}{d\tau} \left(\sum_t (|B(t)| - k) y_t(\tau) - \sum_{p,j} z_{p,j}(\tau) \right) = (|B(t)| - k) r - \sum_{(p,j)\in D_t(\tau)} r = r(|B(t)| - k - |D_t(\tau)|). \tag{5}$$

We note that the right-hand side of Equation (5) is r-times the first term of the right-hand side of Equation (4). Furthermore, $\frac{|C_t(\tau)|}{k} \leqslant |C_t(\tau)| - k + 1 = |B(t)| - 1 - |D_t(\tau)| - k + 1 = \frac{1}{r} G_2$, since $|C_t(\tau)| \geqslant k$. By adding together Equation (5) and the last inequality, we obtain

$$G_1 \leqslant |B(t)| - k - |D_t(\tau)| + |C_t(\tau)|/k \leqslant G_2/r + G_2/r = 2G_2/r. \tag{6}$$

Since both the primal and the linear term of the dual initially have value 0 at time $\tau = 0$, their overall competitive ratio after processing all elements will be $\frac{2}{r}$. Thus for the choice of $r(\delta) = \frac{\delta}{\ln(k+1)}$, where δ is a parameter which we will optimize later, from Equation (6) it follows that $\sum_t (B(t) - k)\bar{y}_t - \sum_{p,j} \bar{z}_{p,j} \geqslant \frac{\delta}{2\ln(k+1)} f(\bar{x})$. Plugging $s_{p,j}(\tau) = \frac{\partial f(x)}{\partial x_{p,j}}^{-1}$ and $r(\delta) = \frac{\delta}{\ln(k+1)}$ into the second statement of Claim 17, we obtain that $A^T y - z \leqslant \delta \nabla f(\bar{x})$, which allows us to bound the conjugate term of the dual as

$$f^*(A^T \bar{y} - \bar{z}) \leqslant f^*(\delta \nabla f(\bar{x})) \leqslant \begin{cases} \delta^{\frac{q}{q-1}} \cdot (q-1) \cdot f(\bar{x}) & \text{if } q > 1, \\ 0 & \text{if } q = 1, \end{cases}$$

where the first inequality is due to Property 14 and the second inequality uses Lemma 18. Hence the relationship between the final value $D(\bar{y}, \bar{z})$ of the dual and the final value $f(\bar{x})$ of the primal is

$$D(\bar{y}, \bar{z}) = \sum_t (|B(t)| - k)\bar{y}_t - \sum_{p,j} \bar{z}_{p,j} - f^*(A^T \bar{y} - \bar{z}) \geqslant \left(\frac{\delta}{2\ln(1+k)} - \delta^{\frac{q}{q-1}} \cdot (q-1) \right) \cdot f(\bar{x}).$$

The term $h(\delta) = \left(\frac{\delta}{2\ln(k+1)} - \delta^{\frac{q}{q-1}} \cdot (q-1) \right)$ is a polynomial in δ, which governs our competitive ratio. The best competitive ratio is obtained if we find $\delta \in (0,1)$ such that $h(\delta)$ is maximized. We find a local maximum at $\delta^* = \frac{1}{(2q\ln(k+1))^{q-1}}$, yielding $h(\delta^*) = \frac{1}{(2q\ln(k+1))^q}$. By rearranging and weak duality (Lemma 15) we obtain

$$f(\bar{x}) \leqslant (2q\ln(k+1))^q D(\bar{y}, \bar{z}) \leqslant (2q\ln(k+1))^q f(x^*),$$

where x^* is an optimal solution. □

The ℓ_q-norm does not lie in our class of objective functions, as a coordinate of $\nabla f(x)$ can decrease while we increase all coordinates of x, hence we can not apply Theorem 19 straight away.

Theorem 20. *Let $q \in [1, \infty)$. Then there exists a $2q\log(k+1)$-competitive algorithm for fractional q-paging with a cache of size k.*

Proof. Let us fix $q \in [1, \infty)$. We apply Theorem 19 with the target function $f(x) = \sum_{p \in P} \left(\sum_{j=1}^{r(p,T)} x_{p,j} \right)^q$, which is the qth power of the ℓ_q-norm. This produces a solution $\bar{x}$, which is $(2q\log(k+1))^q$-competitive for the paging problem with target function $f(x)$.

Let $g : \mathbb{R} \to \mathbb{R}$ be a monotone function, then a solution x to the paging problem with target function $f(x)$ will also be a feasible solution to the paging problem with target function $g(f(x))$. In particular, as g preserves the standard ordering on the reals, an optimal solution to paging with target function $f(x)$ will remain an optimal solution to the problem with target function $g(f(x))$.

If we let $g(y) = y^{\frac{1}{q}}$ and we let x^* be an optimal solution to the paging problem with target function $f(x)$, then $g(f(x))$ will be the ℓ_q-norm and we find that $g(f(\bar{x})) \leqslant g((2q\log(k+1))^q f(x^*)) = 2q\log(k+1) g(f(x^*))$. □

Remark 21. Note that if the gradient of $f(x)$ is 0 at $x = 0$, then we start the algorithm at $\epsilon \cdot \mathbf{1}$ for a small $\epsilon > 0$ instead, which can be chosen sufficiently small, so it does not influence the competitive ratio.

We use Theorem 20 to show that we can obtain a $2e\log(n)\log(k+1)$-competitive fractional solution for ∞-paging by reducing it to $\log n$-paging.

Theorem 22. *There exists a $2e\log(n)\log(k+1)$-competitive algorithm for fractional min-max paging.*

Proof. Let x^* be the optimal solution to the ∞-paging problem for the request sequence σ. We denote the cost of this solution by OPT_∞. Let $\bar{x}$ denote the fractional solution obtained using Algorithm 2. By Theorem 20 and $\|x\|_\infty \leqslant \|x\|_{\log n} \leqslant e\|x\|_\infty$, we know that this solution has cost

$$\max_{p\in P}\sum_{j=1}^{r(p,T)}\bar{x}_{p,j} \leqslant \left(\sum_{p\in P}\left(\sum_{j=1}^{r(p,T)}\bar{x}_{p,j}\right)^{\log n}\right)^{\frac{1}{\log n}} \leqslant 2\log(n)\log(k+1)\cdot\mathrm{OPT}_{\log n} \leqslant 2e\log(n)\log(k+1)\cdot\mathrm{OPT}_\infty,$$

where $\mathrm{OPT}_{\log n}$ denotes the cost of an optimal solution to the $\log n$-paging problem with input σ. This implies that $\bar{x}$ is a $2e\log(n)\log(k+1)$-competitive solution for ∞-paging. □

5 Rounding Fractional Solutions Online

5.1 An $O(k\log(n)\log(k))$-competitive Deterministic Algorithm

This section shows how to round a fractional solution for min-max paging to an integral solution online. The rounding procedure is deterministic and, when coupled with a fractional min-max paging algorithm, gives a deterministic min-max paging algorithm.

Theorem 23. *If there exists an α-competitive algorithm for fractional min-max paging with cache size k, then there exists a (αk)-competitive deterministic algorithm for min-max paging with cache size k.*

Proof. Without loss of generality, we assume that the fractional min-max paging algorithm is lazy. That is, it loads a page only when the page is requested. Indeed, an arbitrary solution can be converted into a lazy solution online without increasing the cost by delaying page loads as much as possible.

The deterministic integral algorithm maintains the following invariant: it always has a page p in its cache whenever the fractional algorithm has more than a $1-1/k$ fraction of p in its cache. We observe that the fractional algorithm must always fully have at least one page in its cache: the most recently requested page. Therefore, at any time, the number of pages p such that the fractional algorithm contains more than a $1-1/k$ fraction of p is less than $1+(k-1)/(1-1/k) = k+1$, and therefore, this number is at most k.

Consider an arbitrary request to some page p. If the integral algorithm already has p in its cache, it ignores the request, whereas the fractional algorithm possibly serves the request by evicting some pages fractionally. On the other hand, suppose the integral algorithm does not already have p in its cache, then this implies that the fractional algorithm has at most a $1-1/k$ fraction of p in its cache. After the fractional algorithm brings p into its cache, the integral algorithm must have a page q in its cache such that the fractional algorithm has at most a $1-1/k$ fraction of q in its cache. (Otherwise, the fractional algorithm has more than a $1-1/k$ fraction of $k+1$ pages in its cache, namely, the k pages in the integral algorithm's cache and the page p, thus contradicting the observation from the last paragraph.) The integral algorithm replaces one such page q by p to serve the request and thus, maintains the invariant. In this process, the integral and the fractional algorithms incur 1 and at least $1/k$ faults, respectively, on page p.

Thus, at the end of the request sequence, for every page p, the number of faults of the integral algorithm on p is at most k times the number of faults of the fractional algorithm on p. Thus, the cost of the integral algorithm is at most k times the cost of the fractional algorithm. Since the latter is at most α times the cost of the optimum, the cost of the integral algorithm is at most αk times the cost of the optimum solution. □

Corollary 24. *There exists a $2ek\log(n)\log(k+1)$-competitive deterministic algorithm for min-max paging.*

Proof. Follows from Theorem 22 and Theorem 23. □

It is noteworthy that the trick in the proof of Theorem 23 can also be used for the derandomization of randomized algorithms. Specifically, suppose an α-competitive randomized algorithm exists for min-max paging. Then there also exists a fractional one with the same competitive ratio. Thus, by Theorem 23, there exists a αk-competitive deterministic algorithm for min-max paging.

5.2 An $O(\log^2(n)\log(k))$-competitive Randomized Algorithm.

Using a more sophisticated rounding approach, we obtain a randomized algorithm whose competitive ratio no longer depends linearly on k, in exchange for an additional $\log(n)$ factor. This result rules out a lower bound of $\Omega(k)$. This algorithm is of interest in the regime where $\log(n) \leqslant k$, which is often the case in applications.

We can obtain a randomized algorithm for min-max paging by using the rounding scheme for weighted paging of Bansal et al. [BNT21]. The simplified rounding scheme is presented in Algorithm 3. Each online rounding step only depends on the previous, and current fractional cache states $x(t-1)$ and $x(t)$ as well as the previous integral cache state and on a parameter β, which indicates how aggressively we eject pages from the cache. *The rounding scheme works for any caching scheme that fulfills the condition that (1) at any time t, for any page $p \neq p_r$, $x_p(t) - x_p(t-1) \geqslant 0$ and (2) the total fraction of pages evicted upon any request is at most 1.* Algorithm 2 indeed has these properties, so we can use the rounding scheme as long as we can relate the rounding costs to our target function, even though we solve a different paging problem than they do.

Let x be a fractional solution produced by Algorithm 2. After processing round t, the algorithm will produce a fractional value $x_p(t)$ for each page, indicating the fraction of page p in the cache in this round. In other words, the process of solving the fractional problem online produces, whenever Algorithm 2 finishes processing a round at time $\tau(t)$, the vector

$$x(t) = \begin{bmatrix} x_{p_1,r(p,t)}(\tau(t)) \\ \vdots \\ x_{p_n,r(p,t)}(\tau(t)) \end{bmatrix}.$$

We let $y_p(t) = \min\{\beta \cdot x_p(t), 1\}$ be the solution in which every coordinate is scaled up by a factor of β. The factor β governs how much more aggressively pages are evicted from the cache.

Algorithm 3 may evict pages and incur costs in two separate places. The first type we need to account for is the cost incurred via the random evictions of pages in the for-loop in lines 4-5 of the algorithm. The second type is the cost incurred by fixing the cache size in lines 6-7 if no page was evicted in the for-loop. We will bound these costs separately and combine them in our upper bound.

For the first type, it is easy to see that the cost incurred for evicting a page in lines 4-5 depends only on the sequence of fractional values $y_p(1), y_p(2), \ldots, y_p(T)$ that this page takes on and it is independent of the values $y_{p'}(t)$ for all t and $p' \neq p$. In particular, the probability $y_{p,j}$ that a page is evicted in lines 4-5,

Algorithm 3 The randomized rounding scheme of [BNT21] adapted to our problem.

procedure ROUND($x(t), x(t-1), C(t-1)$)
  if $p_t \notin C(t-1)$ **then** ▷ *Add the page p_t to the cache, if it is not already in it.*
    $C(t-1) \leftarrow C(t-1) \cup \{p_t\}$
  for $p \in C(t-1) \setminus \{p_t\}$ **do**
    Evict p from $C(t-1)$ independently with probability $\frac{y_p(t)-y_p(t-1)}{1-y_p(t-1)}$
  if $|C(t-1)| > k$ **then**
    Evict an arbitrary page $p \neq p_t$ from $C(t-1)$
  $C(t) \leftarrow C(t-1)$

between its j-th and $j+1$-st request is

$$\begin{aligned}
\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \Pr[\text{page } p \text{ is evicted in round } t] &= \sum_{t=t(p,j)+1}^{t(p,j+1)-1} \frac{y_p(t)-y_p(t-1)}{1-y_p(t-1)} \Pr[\text{page } p \text{ is not evicted until round } t] \\
&= \sum_{t=t(p,j)+1}^{t(p,j+1)-1} \frac{y_p(t)-y_p(t-1)}{1-y_p(t-1)} \prod_{t'=t(p,j)+1}^{t-1} 1 - \frac{y_p(t')-y_p(t'-1)}{1-y_p(t'-1)} \\
&= \sum_{t=t(p,j)+1}^{t(p,j+1)-1} \frac{y_p(t)-y_p(t-1)}{1-y_p(t-1)} \prod_{t'=t(p,j)+1}^{t-1} \frac{1-y_p(t')}{1-y_p(t'-1)} \\
&= \sum_{t=t(p,j)+1}^{t(p,j+1)-1} \frac{y_p(t)-y_p(t-1)}{1-y_p(t(p,j))} = \frac{y_p(t(p,j+1)-1)}{1-y_p(t(p,j))} = y_p(t(p,j+1)-1).
\end{aligned}$$

The second equation holds because of the independence of the probability of eviction in different rounds; the fourth holds because it is a telescoping product, and the last equation holds as $y_p(t(p,j)) = 0$. Let $Y_{p,j}$ be a Bernoulli random variable that is 1 with probability $y_p(t(p,j+1)-1)$ and let $Y_p = \sum_j Y_{p,j}$ be the sum of all $Y_{p,j}$ for fixed p. We let these variables track the expected cost of evictions for each page. By linearity of expectation, we immediately see that

$$\mathbb{E}[Y_p] = \mathbb{E}\left[\sum_{j=1}^{r(p,T)} Y_{p,j}\right] = \sum_{j=1}^{r(p,T)} \Pr[p \text{ is evicted between request } j \text{ and } j+1] = \sum_{j=1}^{r(p,T)} y_{p,j} \leqslant \beta \sum_{j=1}^{r(p,T)} x_{p,j}.$$

It follows that

$$\max_p \mathbb{E}[Y_p] \leqslant \beta \max_p \sum_{j=1}^{r(p,T)} x_{p,j},$$

where the right-hand side is β times the cost of the fractional solution x. It remains to relate the left side of this inequality with $\mathbb{E}[\max_p Y_p]$.

Lemma 25. *Let $Y_{p,j}$ be Bernoulli random variables which are 1 with probability $y_{p,j}$. Let $Y_p = \sum_j Y_{p,j}$ and assume there are n such sums, then*

$$\mathbb{E}[\max_p Y_p] \leqslant e \cdot \max_p E[Y_p] + \log(n).$$

Proof. Let $Y = \max_p Y_p$. Using Jensen's inequality, we get the first inequality in the following chain of inequalities:

$$\begin{aligned}
\exp(\mathbb{E}[Y]) &\leqslant \mathbb{E}[\exp(Y)] = \mathbb{E}[\max_p \exp(Y_p)] \\
&\leqslant \sum_p \mathbb{E}[\exp(Y_p)] = \sum_p \prod_{j=1}^{r(p,T)} \mathbb{E}[\exp(Y_{p,j})] \\
&= \sum_p \prod_{j=1}^{r(p,T)} (1 - y_{p,j} + e y_{p,j}) \leqslant \sum_p \prod_{j=1}^{r(p,T)} e^{e \cdot y_{p,j}} \\
&= \sum_p e^{e \cdot \sum_{j=1}^{r(p,T)} y_{p,j}} \leqslant n \cdot \max_p e^{e \cdot \mathbb{E}[Y_p]}.
\end{aligned}$$

The first equality follows as exp is a monotone function, and the second equality follows by the independence of the $Y_{p,j}$. After taking logarithms, we obtain

$$\mathbb{E}[Y] \leqslant e \cdot \max_p \mathbb{E}[Y_p] + \log(n),$$

which yields the desired result. □

Therefore, by the above lemma, the expected cost of the first type of costs is bounded by

$$\mathbb{E}[\max_p Y_p] \leqslant O(1) \cdot \beta \max_p \sum_{j=1}^{r(p,T)} x_{p,j}. \tag{7}$$

Lemma 26 ([BNT21])**.** *Let x be a fractional solution for a general paging problem. The expected cost of resets is at most $16ke^{-\beta/4} \cdot \sum_{p \in P} \sum_{j=1}^{r(p,T)} x_{p,j}$.*

By choosing $\beta = 4\log(nk)$ in Lemma 26, the expected total cost of resets for the solution y becomes

$$16ke^{-\log(nk)} \sum_{p \in P} \sum_{j=1}^{r(p,T)} y_{p,j} \leqslant \frac{16}{n} \sum_{p \in P} \sum_{j=1}^{r(p,T)} y_{p,j} \leqslant \frac{16\beta}{n} \sum_{p \in P} \sum_{j=1}^{r(p,T)} x_{p,j} \leqslant 16\beta \max_{p \in P} \sum_{j=1}^{r(p,T)} x_{p,j} \tag{8}$$

where the last inequality follows due to the fact that the average cost per page $\frac{1}{n} \sum_{p \in P} \sum_{j=1}^{r(p,T)} x_{p,j}$ is a lower bound on the maximum cost of a single page in the solution x.

Crucially, Lemma 26 depends on the following helper lemma:

Lemma 27. *Given a fractional solution x to the paging problem, we can find a fractional solution x^* in which every variable is a multiple of $\delta = \frac{1}{4k}$, and the cost of which is no more than 3 times the cost of x.*

Taking the bounds on the two types of costs, namely Equations (7) and (8), we have shown the following:

Theorem 5. *There exists an $O(\log^2 n \log k)$-competitive* randomized *integral algorithm for min-max paging.*

6 Further related work

Sleator and Tarjan [ST85] defined the framework of online algorithms and competitive analysis, and paging is one of the earliest problems studied in the online setting. Several deterministic algorithms, such as "Least Recently Used" (LRU) and "First In First Out" (FIFO), among others, are known to achieve the optimal deterministic competitive ratio of k [ST85], where k is the maximum number of pages that can be inside the cache at any point in time. The randomized competitive ratio is known to be H_k, where the upper bound is due to Achlioptas et al. [ACN00] and the lower bound is due to Fiat et al. [FKL+91].

Several practical generalizations of the paging problem have been studied and they are known to have a deterministic competitive ratio of k [CKPV90, You98] and randomized competitive ratio $\Theta(\log k)$ [BBN12b, BBN12a]. These include weighted paging – where pages have arbitrary loading costs, the *bit model* – where pages have arbitrary sizes and loading cost proportional to size, the *fault model* – where pages have arbitrary sizes but unit loading cost, and generalized paging – where pages have arbitrary loading costs as well as sizes. Interestingly, all these results are robust in the sense that they all extend to the resource-augmentation setting, where the adversary has fewer servers than the algorithm. It is noteworthy that the line of work in search of a randomized algorithm for these paging variants by Bansal, Buchbinder, and Naor led to the development of the online primal-dual framework for designing fractional algorithms for online problems, whose solutions can often be rounded to an integral solution online.

A simple-looking but intriguing generalization of paging is the k-server problem defined by Manasse, McGeogh, and Sleator [MMS88], which concerns moving k mobile servers on a metric space to serve requests while minimizing total movement. (The paging problem is the k-server problem on the uniform metric over the set of pages.) While Manasse et al. [MMS88] proved a lower bound of k on the deterministic competitive ratio for every metric space with more than k points, the existence of a k-competitive algorithm is still unknown, and this is popularly called the k-server conjecture. The best-known k-server algorithm that works for all metrics called the *Work Function Algorithm* by Koutsoupias and Papadimitriou [KP95], achieves a competitive ratio of $2k-1$. For randomized algorithms, surprisingly, neither a better upper bound than the deterministic $2k-1$ nor a better lower bound of $\Omega(\log k)$ arising from paging is known. Koutsoupias [Kou09] presents a more comprehensive discussion on the k-server problem.

References

[ABC+16] 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. Online algorithms for covering and packing problems with convex objectives. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 148–157. IEEE Computer Society, 2016.

[ABFP13] Yossi Azar, Umang Bhaskar, Lisa Fleischer, and Debmalya Panigrahi. Online mixed packing and covering. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 85–100. SIAM, 2013.

[ACN00] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.

[BBN12a] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. *SIAM J. Comput.*, 41(2):391–414, 2012.

[BBN12b] Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4), aug 2012.

[BFMS21] Marcin Bienkowski, David Fuchssteiner, Jan Marcinkowski, and Stefan Schmid. Online dynamic b-matching: With applications to reconfigurable datacenter networks. *ACM SIGMETRICS Performance Evaluation Review*, 48(3):99–108, 2021.

[BNT21] Nikhil Bansal, Joseph Naor, and Ohad Talmon. Efficient online weighted multi-level paging. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 94–104, 2021.

[CKPV90] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. In *SODA*, pages 291–300. SIAM, 1990.

[CKZ99] Edith Cohen, Haim Kaplan, and Uri Zwick. Connection caching. In *Proceedings of the thirty-first annual ACM symposium on Theory of Computing*, pages 612–621, 1999.

[FKL+91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[FS19] Klaus-Tycho Foerster and Stefan Schmid. Survey of reconfigurable data center networks: Enablers, algorithms, complexity. *ACM SIGACT News*, 50(2):62–79, 2019.

[KMRS86] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 244–254, 1986.

[Kou09] Elias Koutsoupias. The k-server problem. *Comput. Sci. Rev.*, 3(2):105–118, 2009.

[KP95] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

[MMS88] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333. ACM, 1988.

[MS91] Lyle A. McGeoch and Daniel D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(1-6):816–825, jun 1991.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[TKR21] Srujan Teja Thomdapu, Palash Katiyar, and Ketan Rajawat. Dynamic cache management in content delivery networks. *Computer Networks*, 187:107822, 2021.

[You98] Neal E. Young. On-line file caching. In *SODA*, pages 82–86. ACM/SIAM, 1998.

A Further Details for Lower Bounds

This section shows the complete details for the deterministic lower bound for min-max paging. We begin with a proof of the lower bound for $k = 2$, which neatly highlights the core construction lying at the heart of the lower bound.

Lemma 28. *Any deterministic algorithm* ALG *for min-max paging with cache size* $k = 2$ *is at least* $\Omega(\log n)$*-competitive.*

Proof. Suppose $n = 3^\ell$ and let the set of pages be $\{p_1, p_2, \ldots, p_n\}$. We construct a bad request sequence σ for ALG in ℓ layers. Each layer is further divided into phases. Let $N \gg n$ be a large integer parameter.

We call the number of page faults incurred by page p up to round t the *cost* of p at t. Similarly, the *cost of the min-max paging algorithm* ALG is the maximum cost over all pages $p \in P$.

We now iteratively construct the adversarial sequence σ, going layer by layer.

Layer 1 will use all pages, that is the set $\{p_1, p_2, \ldots, p_n\}$.

- In the first phase, we request pages p_1, p_2, and p_3 in such a way that ALG faults on every request, such a cruel sequence composed of $k+1$ pages exists for every deterministic algorithm for paging. We stop this phase once the cost of the algorithm becomes N, which must happen before sending $3N$ requests. Without loss of generality, we assume that the cost of p_1 first reaches N, and hence the costs of p_2 and p_3 are $< N$. These costs are the same as the number of requests to the respective pages because the algorithm always faults.
- In the second phase, we repeat this step with pages p_4, p_5, and p_6. Without loss of generality, we assume that the cost of p_4 is N.
- Repeat this process until the set of pages $\{p_1, p_2, \ldots, p_n\}$ is exhausted.

In all phases, the adversary always keeps the lowest numbered page, that is pages $p_1, p_4, p_7, \ldots$, respectively, in the cache, incurring a cost of only 1 on them, while the cost of the other pages is at most N. We *promote* pages $p_1, p_4, p_7, \ldots$ to Layer 2.

Layer 2 with universe of pages $\{p_1, p_4, p_7, \ldots, p_{n-2}\}$ is constructed exactly in the same way as Layer 1. After this layer, the cost of ALG is $2N$, whereas the cost of the adversary is $\leqslant N$, with costs of pages $p_1, p_{10}, p_{19}, \ldots$ having cost $\leqslant 2$, and we *promote* them to Layer 3, and so on.

After phase i, the number of pages in the universe becomes $n/3^i$, the cost of ALG becomes iN, whereas the adversary's cost is always $\leqslant \max\{N, i\}$. This gives us the desired lower bound using $\log_3(n)$ layers by choosing $N \geqslant \ell$. □

Generalizing the above construction The idea behind the lower bound of $\Omega(k(\log n)/\log k)$ is as follows. We generalize the construction above by using $n = (k+1)^\ell$ pages and $\ell = \log_{k+1} n$ layers. In each phase, we use $k+1$ pages and force the cost of the algorithm on one of these pages to increase by N. The adversary's cost increases by at most 1 on the page she will promote to the next layer. By using a smarter *offline* algorithm, the cost of the adversary increases by at most $O(N/k)$ on the k pages of this phase that will not be promoted. So, in the end, the adversary's cost is $O(\ell + N/k)$, whereas the cost of the algorithm is $\Omega(N\ell)$. We obtain the desired lower bound by choosing $N \geqslant ck\ell$ for a large enough constant c.

For any paging algorithm ALG and any request sequence σ, we define $\text{cost}(\text{ALG}, \sigma, p, t)$ to be the the number of page faults incurred on page p after processing the first t requests of σ. Furthermore, we define

$$\text{cost}(\text{ALG}, \sigma) = \max_{p \in P} \text{cost}(\text{ALG}, \sigma, p, T),$$

to be the overall cost incurred by ALG while processing request sequence σ.

The optimal offline algorithm OPT for min-max paging is not known to us, so we use Algorithm 4 (GreedyLFD) to obtain an upper bound on the cost of OPT. Intuitively, this algorithm avoids increasing its maximum cost for as long as possible by greedily keeping the most expensive pages in its cache. As GreedyLFD is an offline algorithm, it has access to the complete request sequence σ and can always eject the page that is *furthest in the future*. That is, in round t, it ejects the page $p \in C$ whose next occurrence

comes last in the remainder of σ after t. If a page does not occur in the remainder of σ, it is treated as being infinitely far in the future, and the algorithm will always prefer to eject this page over one that will still occur in σ.

Algorithm 4 The offline algorithm GreedyLFD

```
1:  procedure GreedyLFD(σ)
2:      C ← ∅                                          ▷ Initialize the cache.
3:      t ← 1
4:      for p ∈ P do
5:          c_p ← 0 ▷ Counter variables for the number of faults on page p. At any point in time c_p = cost(GreedyLFD, σ, p, t).
6:      while t < T do
7:          if the tth element, say p_t, of σ is not in C then
8:              c_p ← c_p + 1
9:              if |C| < k then
10:                 C ← C ∪ {p}
11:             else
12:                 S ← {q ∈ C | c_q < max_r c_r} ▷ Obtain the set of pages whose cost is less than the current maximum.
13:                 if S = ∅ then                      ▷ This happens if all pages in C are of the same cost.
14:                     S ← C
15:                 Evict q ∈ S which next occurs farthest in the future.
16:         t ← t + 1
```

Lemma 29. *Let σ be a request sequence for min-max paging using $k+1$ unique pages. Then*

$$\text{cost}(GreedyLFD, \sigma) \leqslant \frac{2(\text{len}(\sigma) - 2k - 1)}{2k + k(k+1)} + 2.$$

Proof. We fix σ to be an arbitrary request sequence of length T using pages from the set $P = \{p_1, p_2, \ldots, p_{k+1}\}$. Let

$$t_i = \min\left\{ t \in [T] \mid \max_{p \in P} \text{cost}(GreedyLFD, \sigma, p, t) \geqslant i \right\}$$

be the first time GreedyLFD faults on a page for the ith time. We note that at time t_i, there is only one page of cost i.

Furthermore, we note that $t_1 = 1$, as the algorithm starts with an empty cache and $t_2 \geqslant 2k+1$, as the algorithm will fault on the first $k+1$ distinct pages it encounters, and it will then eject the page of cost 1 which occurs farthest in the future. As there are k pages in its cache, at least one page will not occur in the next $k-1$ time steps, so the shortest sequence that can cause GreedyLFD to fault two times on a single page is of length $2k+1$.

We now show $t_{i+1} \geqslant t_i + k + \frac{k(k+1)}{2}$ for all $i \geqslant 2$. Let us fix $i \geqslant 2$ and assume we are currently at time t_i. This means that there exists some page $p \in P$ for which $\text{cost}(GreedyLFD, \sigma, p, t_i) = i$ and it is the only page of cost i. In order to make room for p, GreedyLFD will evict a page of cost at most $i-1$ from its cache. As there are k such pages in the cache at time t_i, at least one of them will not occur for the next $k-1$ requests. As there are only $k+1$ pages in total, this means that the next page fault occurs at time $t_i + k$ or later.

In general, when the jth page of cost i is added to GreedyLFD's cache at time $t_{i,j}$, there are $k-j+1$ pages of cost at most $i-1$ in its cache, and so the next page fault will not occur until time $t_{i,j} + k - j + 1$, which gives a lower bound on $t_{i,j+1}$. Note that $t_{i,1} = t_i$

As GreedyLFD's cost can only increase to $i+1$ once it evicts a page of cost i, we find that t_{i+1} must occur after k pages of cost i have been added to its cache. GreedyLFD can choose which of the k pages of cost i to evict, so we find $t_{i+1} \geqslant t_{i,k} + k$. This yields

$$t_{i+1} \geqslant t_{i,k} + k \geqslant t_{i,k-1} + k + 1 \geqslant t_{i,k-2} + k + 1 + 2 \geqslant \ldots \geqslant t_i + k + \sum_{i=1}^{k} i.$$

By expanding the recurrence for $i \geqslant 2$, we find that

$$_i \geqslant (i-2)\left(k + \frac{k(k+1)}{2}\right) + 2k + 1.$$

Using this expression, we derive an upper bound on $\text{cost}(GreedyLFD, \sigma)$, by finding the minimum i for which $\text{len}(\sigma) \leqslant t_i$. From our expression we find that $t_i \geqslant \text{len}(\sigma)$ if $i \geqslant \frac{2(T-2k-1)}{2k+k(k+1)} + 2$, and so

$$\text{cost}(GreedyLFD, \sigma) \leqslant \frac{2(\text{len}(\sigma) - 2k - 1)}{2k + k(k+1)} + 2.$$

□

Lemma 30. *Any deterministic algorithm ALG for min-max paging with cache size k is at least $\frac{k}{2}$-competitive.*

Proof. Let $n = k+1$, that is $P = \{p_1, p_2, \ldots, p_{k+1}\}$. We initialize ALG and GreedyLFD with empty caches. Once ALG's cache is full, at any time step t, there is always one page that is not present in the cache. The adversary's strategy is always to request this page. We call this the *cruel* strategy. Since ALG is deterministic, the adversary always knows which page ALG will evict from its cache if a page fault occurs, so such a sequence must always exist.

Algorithm 5 An algorithm to generate an adversarial sequence for ALG of length $T \geqslant k+1$, on which ALG faults T times.

```
output pages p_1, p_2, ..., p_{k+1}
i ← k + 1
while i < T do
    p ← the page p_i which is currently not in the cache of ALG;
    output p
    i ← i + 1
```

Let σ be a sequence of length T generated by the cruel strategy of Algorithm 5. We note that σ causes a page fault at every step, so the total number of page faults incurred by ALG will be T. By a simple averaging argument, there must be at least one page that has incurred $\frac{T}{k+1}$ page faults and so $\frac{T}{k+1} \leqslant cost(ALG, \sigma)$. On the other hand, by Lemma 29, GreedyLFD will incur a cost of at most $\frac{2(T-2k-1)}{2k+k(k+1)} + 2$ while processing σ. This immediately yields

$$\text{cost}(\text{OPT}, \sigma) \leqslant \text{cost}(GreedyLFD, \sigma) \leqslant \frac{2(T-2k-2)}{2k+k(k+1)} + 2 \leqslant \frac{2T}{k(k+1)} + 2 \leqslant \frac{2}{k}\text{cost}(ALG, \sigma) + 2,$$

and so the competitive ratio is $\frac{cost(ALG,\sigma)}{\text{cost}(OPT,\sigma)} \geqslant \frac{k}{2}$, since the constant vanishes as the cost grows large. □

Algorithm 6 An adversarial strategy for min-max paging

Let $L_m = \{p_0^m, \ldots, p_{n-1}^m\}$ be a set of $n = (k+1)^m$ pages.
for $\ell = m$ to 1 **do**
    $L_{\ell-1} \leftarrow \emptyset$.
    for $i = 0$ to $(k+1)^{\ell-1} - 1$ **do**
        Use the cruel strategy of Algorithm 5 on the set of $k+1$ pages $\{p^\ell_{(k+1)i}, p^\ell_{(k+1)i+1}, \ldots, p^\ell_{(k+1)i+k}\}$ until one page, say p', has incurred N page faults since the start of this loop.
        $p_i^{\ell-1} \leftarrow p'$
        $L_{\ell-1} \leftarrow L_{\ell-1} \cup \{p_i^{\ell-1}\}$

Finally, we strengthen the lower bound to $\Omega(\frac{k}{\log k} \log n)$, introducing a dependence on the number of pages. We do this using the strategy presented in Algorithm 6.

Intuitively, our strategy consists of splitting the $n = (k+1)^m$ pages into $(k+1)^{m-1}$ disjoint sets of $k+1$ variables. We then present the algorithm ALG with a cruel sequence for each set until one of the pages reaches cost N. We repeat this process layer by layer until we obtain one final page. ALG will have faulted mN times on this page, while OPT will have faulted no more than roughly $\frac{N}{k} + m$ times on any page.

Any deterministic algorithm for min-max paging with cache size k is at least $\frac{k-1}{2} \log_{k+1} n$-competitive, where n is the number of pages.

Proof. We use the strategy defined in Algorithm 6 to generate our request sequence. We observe that at the beginning of iteration ℓ of the outer for-loop in Algorithm 6, ALG will have faulted $(m-\ell)N$ times on each page in $\{p^\ell_0, \ldots, p^\ell_{(k+1)^\ell - 1}\}$, because we only add a page to the next level once it has incurred N faults during the current level. Hence, once ALG has processed the complete sequence provided by Algorithm 6, it has cost mN, witnessed by page p_0^0.

On the other hand, while processing the ith set of variables in the inner loop, the optimal offline algorithm OPT will keep page p' in its cache. When p' is requested for the first time in this iteration of the loop, OPT will fault once on p'. Afterward, p' will remain in the cache of OPT until the current iteration of the inner loop finishes, incurring no more cost.

This shows that OPT will have cost $m - \ell$ for each page $p^\ell_0, \ldots, p^\ell_{(k+1)^\ell - 1}$ at the beginning of the ℓth iteration of the outer loop.

During an iteration of the inner loop, we use the remaining $k-1$ slots in OPT's cache, which are not occupied by p', to process the remaining k pages in each iteration. As the cruel sequence causes ALG to fault on every request and we end it as soon as one page has faulted N times, each of the remaining pages may be requested $N-1$ times. It follows that the sub-sequence σ' of the cruel sequence, defined on the remaining k pages, is of length at most $k(N-1)$. By Lemma 29, we get $\text{cost}(\text{OPT}, \sigma') \leqslant \frac{2(k(N-1)-2(k-1)-1)}{2(k-1)+(k-1)k} + 2 \leqslant \frac{2(N-1)}{k-1} + 2$.

Thus we find that for any page p, the total cost consists of the level it is raised to plus the cost incurred while processing σ' on its last level and thus $\text{cost}(OPT, \sigma, p, T) \leqslant m + \frac{2(N-1)}{k-1} + 2$ and so

$$\frac{\text{cost}(ALG, \rho)}{\text{cost}(OPT, \rho)} \geqslant \frac{mN}{m + 2 + \frac{2(N-1)}{k-1}} = \frac{(k-1)mN}{(k-1)(m+2) + 2(N-1)}.$$

As N grows large, the right hand side will converge to $\frac{(k-1)m}{2} = \frac{k-1}{2} \log_{k+1} n$. □

B Deferred Proofs

This section contains some deferred proofs from the paper.

Property 14. *The Fenchel dual $f^*(y) : \mathbb{R}_+^T \to \mathbb{R}$ of a convex function f is monotonically increasing in y.*

Proof. Indeed, let $c \in \mathbb{R}_+^T$, then

$$f^*(y) = \sup_{w \in \mathbb{R}_+^T} \langle y, w \rangle - f(w) \leqslant \sup_{w \in \mathbb{R}_+^T} \langle y, w \rangle + \langle c, w \rangle - f(w) = f^*(y + c),$$

as $\langle c, w \rangle$ is always non-negative. □

Lemma 15 (Weak Duality)**.** *For any feasible $x \in [0,1]^T$ and $y, z \in \mathbb{R}_+^T$, we have*

$$f(x) \geqslant \sum_{t \in [T]} (|B(t)| - k) y_t - \sum_{p,j} z_{p,j} - f^*(A^\top y - z).$$

Proof. Let $b = \begin{bmatrix} B(1) - k \\ \vdots \\ B(T) - k \end{bmatrix}$. As x is feasible, it must satisfy $Ax \geqslant b$, which gives $b - Ax \leqslant \mathbf{0}$, and similarly from $x \leqslant \mathbf{1}$, we get $x - \mathbf{1} \leqslant \mathbf{0}$. As all entries are negative, taking the inner product of these vectors with the non-negative vectors y and z, respectively, will yield a negative number. Hence, we get the first inequality in the following chain of inequalities:

$$\begin{aligned}
f(x) &\geqslant f(x) + y^\top ((|B(t)| - k)\mathbf{1} - Ax) + z^\top (x - \mathbf{1}) \\
&= \sum_{t=1}^{T} (|B(t)| - k) y_t - \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} z_{p,j} - (y^\top Ax - z^\top x - f(x)) \\
&= \sum_{t=1}^{T} (|B(t)| - k) y_t - \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} z_{p,j} - (\langle A^\top y - z, x \rangle - f(x)) \\
&\geqslant \sum_{t=1}^{T} (|B(t)| - k) y_t - \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} z_{p,j} - \sup_{w \in \mathbb{R}_+^T} (\langle A^\top y - z, w \rangle - f(w)) \\
&= \sum_{t=1}^{T} (|B(t)| - k) y_t - \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} z_{p,j} - f^*(A^\top y - z).
\end{aligned}$$

All equalities are obtained via simple rearranging of terms, and the final inequality is due to the definition of the supremum. □

Lemma 16. *Let $\bar{x}$ denote the value of x after processing the complete request sequence σ, and similarly for $\bar{y}$ and $\bar{z}$. If $s_{p,j}(\tau)$ is monotonically decreasing in τ, then*

$$\bar{x}_{p,j} \geqslant \frac{1}{k} \left(\exp\left(\frac{s'_{p,j}}{r} \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t - \bar{z}_{p,j} \right) \right) - 1 \right), \tag{2}$$

where $s'_{p,j}$ is the minimum value that $s_{p,j}(\tau)$ takes on during the execution of the algorithm.

Proof. First, observe that $x_{p,j}$ starts increasing at time $\tau(t(p,j))$, the time at which we finish processing the j'th request to p. $x_{p,j}$ keeps increasing until one of the two events happens: $x_{p,j}$ reaches 1, or we get the next request to p, after which it remains constant till the end. Let τ^{end} denote the time at which either of these events happens.

For $\tau \in [\tau(t(p,j)), \tau^{\text{end}})$, we have

$$\frac{dx_{p,j}(\tau)}{d\tau} = s_{p,j}(\tau) \cdot \left(x_{p,j}(\tau) + \frac{1}{k}\right) \geqslant s'_{p,j} \cdot \left(x_{p,j}(\tau) + \frac{1}{k}\right),$$

and therefore,

$$\frac{1}{s'_{p,j}} \frac{d}{d\tau} \ln\left(x_{p,j}(\tau) + \frac{1}{k}\right) = \frac{1}{s'_{p,j} \cdot (x_{p,j}(\tau) + 1/k)} \cdot \frac{dx_{p,j}(\tau)}{d\tau} \geqslant 1.$$

Integrating over the interval $[\tau(t(p,j)), \tau^{\text{end}})$, we get,

$$\frac{1}{s'_{p,j}} \ln\left(\frac{\bar{x}_{p,j} + 1/k}{1/k}\right) \geqslant \tau^{\text{end}} - \tau(t(p,j)). \tag{9}$$

Next, observe that the variable y_t starts increasing from 0 at the uniform rate r at time $\tau(t-1)$ and stops increasing at time $\tau(t)$. Thus, $\bar{y}_t = r \cdot (\tau(t) - \tau(t-1))$. Summing over all t from $t(p,j)+1$ to $t(p,j+1)-1$, we get,

$$\frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t\right) = \tau(t(p,j+1)-1) - \tau(t(p,j)). \tag{10}$$

Finally, consider the variable $z_{p,j}$. If $x_{p,j}$ stops increasing because the next request to p arrives, then $\bar{z}_{p,j} = 0$ and $\tau^{\text{end}} = \tau(t(p,j+1)-1)$. Using Equation (10), we get,

$$\frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t - \bar{z}_{p,j}\right) = \tau(t(p,j+1)-1) - \tau(t(p,j)) = \tau^{\text{end}} - \tau(t(p,j)).$$

On the other hand, if $x_{p,j}$ stops increasing because it reaches 1, then $z_{p,j}$ starts increasing from 0 at the uniform rate r at time τ^{end}, and stops increasing at time $\tau(t(p,j+1)-1)$. Thus, $\bar{z}_{p,j} = r \cdot (\tau(t(p,j+1)-1) - \tau^{\text{end}})$. Again, using Equation (10), we get,

$$\frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t - \bar{z}_{p,j}\right) = (\tau(t(p,j+1)-1) - \tau(t(p,j))) - (\tau(t(p,j+1)-1) - \tau^{\text{end}}) = t^{\text{end}} - \tau(t(p,j)).$$

Thus, in either case, we have,

$$\frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t - \bar{z}_{p,j}\right) = \tau^{\text{end}} - \tau(t(p,j)). \tag{11}$$

From Inequality (9) and Equation (11), we get,

$$\frac{1}{s'_{p,j}} \ln\left(\frac{\bar{x}_{p,j} + 1/k}{1/k}\right) \geqslant \tau^{\text{end}} - \tau(t(p,j)) = \frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t - \bar{z}_{p,j}\right).$$

Rearranging, we get,

$$\bar{x}_{p,j} \geqslant \frac{1}{k}\left(\exp\left(\frac{s'_{p,j}}{r}\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t - \bar{z}_{p,j}\right)\right) - 1\right),$$

as required. □

Lemma 17. *The $x(\tau(t))$ produced by Algorithm 2 throughout its execution are feasible for the primal for all $t \in [T]$ and the vector $(A^\top \bar{y} - \bar{z})_{p,j} \leqslant \frac{r}{s'_{p,j}} \ln(k+1)$.*

Proof. The first statement follows immediately from the definition of the algorithm and the fact that we can always fulfill the primal constraints, for example, by setting all the variables to 1.

To show the second statement, we note that $\bar{x} \leqslant \mathbf{1}$, which with Equation (2) gives us

$$\frac{1}{k}\left(\exp\left(\frac{s'_{p,j}}{r}\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t - \bar{z}_{p,j}\right)\right) - 1\right) \leqslant \bar{x}_{p,j} \leqslant 1.$$

Taking the left- and right-hand sides gives us, after rearranging and taking logarithms,

$$\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{y}_t - \bar{z}_{p,j} \leqslant \frac{r}{s'_{p,j}} \ln(k+1),$$

where the left hand side is $(A^T \bar{y} - \bar{z})_{p,j}$ as a 1 only appears in the (p,j)th column of A from row $t(p,j)+1$ through row $t(p,j+1)-1$. □

Lemma 27. *Given a fractional solution x to the paging problem, we can find a fractional solution x^* in which every variable is a multiple of $\delta = \frac{1}{4k}$, and the cost of which is no more than 3 times the cost of x.*

Proof. We define our rounded solution x^* as

$$x^*_{p,j} = \begin{cases} 0 & \text{if } x_{p,j} < \frac{1}{8k}, \\ \min\{\frac{1}{4k}\lceil 8k x_{p,j}\rceil, 1\} & \text{otherwise.} \end{cases}$$

That is, we round every variable $x_{p,j}$ of value less than $\frac{1}{8k}$ to 0, and every variable of greater value will be doubled and then rounded up to the nearest multiple of $\frac{1}{4k}$.

As each variable's value is at most doubled and then rounded up, the fractional cost for min-max paging can at most triple, as each variable in x^* is no larger than 3 times the corresponding variable in x and the objective function $f(x)$ in min-max paging satisfies $f(cx) = cf(x)$ for any $c \in \mathbb{R}$. It remains to show that the solution x^* is feasible.

We note that the only variables whose value in x^* can decrease during the rounding are those of value less than $\frac{1}{8k}$. The total number of such variables other than p_t in a feasible solution is at most k, as otherwise.

$$\sum_{B(t)\setminus\{p_t\}} x_{p,r(p,t)} \leqslant n - 1 - k + \frac{1}{8k}k \leqslant n - k.$$

Hence we note that the total contribution L of these variables to the constraint of round t is at most $\frac{1}{8}$.

As $n-k$ is an integer, and the fractional algorithm stops ejecting pages as soon as the constraint is satisfied, we know that $\sum_{p\in B(t)\setminus\{p_t\}} x_{p,r(p,t)}$ is integral.

	…	$x_{p_1,7}$	$x_{p_2,9}$	$x_{p_3,3}$	$x_{p_1,8}$	$x_{p_4,5}$	$x_{p_2,10}$	$x_{p_5,3}$	$x_{p_1,9}$	…
⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
$t(p_1,7)$	…	0	0	0	0	0	0	0	0	…
$t(p_2,9)$	…	1	0	0	0	0	0	0	0	…
$t(p_3,3)$	…	1	1	0	0	0	0	0	0	…
$t(p_1,8)$	…	0	1	1	0	0	0	0	0	…
$t(p_4,5)$	…	0	1	1	1	0	0	0	0	…
$t(p_2,10)$	…	0	0	1	1	1	0	0	0	…
$t(p_5,3)$	…	0	0	1	1	1	1	0	0	…
$t(p_1,9)$	…	0	0	1	0	1	1	1	0	…
⋮	…	0	0	1	0	1	1	1	1	…
⋮		⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Figure 1: An example of the structure of a constraint matrix of the paging problem, corresponding to the sub-sequence of requests $p_1, p_2, p_3, p_1, p_4, p_2, p_5, p_1$ starting at time $t' = t(p_1, 7)$. The matrix is lower triangular, due to the columns being in order of the appearance of variables. In this example, we assume that the pages p_1, p_2, p_3, p_4 and p_5 have been requested $6, 8, 2, 4$, and 2 times before the appearance of this sequence respectively.

We further note that the contribution of all variables that are rounded up in this bound must therefore be at least $n - k - \frac{1}{8}$

If any variable $x_{p,j}$ in x fulfills $\frac{1}{8} \leqslant x_{p,j} \leqslant \frac{7}{8}$, then the rounding step will increase this variable by at least $\frac{1}{8}$, which compensates the value lost by rounding down all small variables.

Otherwise, if we let $A = \{x_{p,r(p,t)} \in B(t) \setminus \{p_t\} \mid x_{p,r(p,t)} \leqslant \frac{1}{8}\}$ and $\sum_{x \in A} x \geqslant \frac{1}{8}$, then the doubling of these variables compensates for the rounding down of small variables.

Finally, if neither of these is the case, the variables of size greater than $\frac{7}{8}$ must sum up to at most $n - k - L$, where L denotes the amount of mass lost by rounding the small variables to 0, and at least $n - k - 2L > n - k - 1$. The sum of the rounded-up variables is an integer, as all variables of value greater than $\frac{1}{2}$ are rounded to 1, so we gain at least L from rounding up the large variables. □