No-regret Caching via Online Mirror Descent

Tareq Si Salem, Giovanni Neglia, Stratis Ioannidis

No-Regret Caching via Online Mirror Descent

TAREQ SI SALEM*, Inria, Université Côte d'Azur, France
GIOVANNI NEGLIA*, Inria, Université Côte d'Azur, France
STRATIS IOANNIDIS*, Northeastern University, USA

We study an online caching problem in which requests can be served by a local cache to avoid retrieval costs from a remote server. The cache can update its state after a batch of requests and store an arbitrarily small fraction of each file. We study no-regret algorithms based on Online Mirror Descent (OMD) strategies. We show that bounds for the regret crucially depend on the diversity of the request process, provided by the diversity ratio R/h, where R is the size of the batch, and h is the maximum multiplicity of a request in a given batch. We characterize the optimality of OMD caching policies w.r.t. regret under different diversity regimes. We also prove that, when the cache must store the entire file, rather than a fraction, OMD strategies can be coupled with a randomized rounding scheme that preserves regret guarantees, even when update costs cannot be neglected. We provide a formal characterization of the rounding problem through optimal transport theory, and moreover we propose a computationally efficient randomized rounding scheme.

CCS Concepts: • **Theory of computation → Caching and paging algorithms**.

Additional Key Words and Phrases: Randomized algorithms, Gradient methods, Adversarial analysis

ACM Reference Format:
Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2023. No-Regret Caching via Online Mirror Descent. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 8, 4, Article 11 (August 2023), 30 pages. https://doi.org/10.1145/3605209

1 INTRODUCTION

Caches are deployed at many different levels in computer systems: from CPU hardware caches to operating system memory caches, from application caches at clients to CDN caches deployed as physical servers in the network or as cloud services like Amazon's ElastiCache [3]. They aim to provide faster service to the user and/or to reduce the computation/communication load on other system elements, like hard disks, file servers, etc.

The ubiquity of caches has motivated extensive research on the performance of existing caching policies, as well as on the design of new policies with provable guarantees. To that end, most prior work has assumed that caches serve requests generated according to a stochastic process, ranging from the simple, memory-less independent reference model [17] to more complex models trying to capture temporal locality effects and time-varying popularities (e.g., the shot-noise model [58]). An alternative modeling approach is to consider an *adversarial* setting. Assuming that the sequence of requests is generated by an adversary, an online caching policy can be compared to the optimal offline policy that views the sequence of requests in advance. Caching was indeed one of the first problems studied by Sleator and Tarjan in the context of the competitive analysis of online algorithms [57]. In competitive analysis, the metric of interest is the *competitive ratio*, i.e., the worst-case ratio between the costs incurred by the online algorithm and the optimal offline *dynamic*

Authors' addresses: Tareq Si Salem, tareq.si-salem@inria.fr, tareq.si-salem@inria.fr, Inria, Université Côte d'Azur, Sophia Antipolis, France; Giovanni Neglia, giovanni.neglia@inria.fr, giovanni.neglia@inria.fr, Inria, Université Côte d'Azur, Sophia Antipolis, France; Stratis Ioannidis, ioannidis@ece.neu.edu, ioannidis@ece.neu.edu, Northeastern University, Boston, USA.

algorithm. This line of work led to the study of metrical task systems [9], a popular research area in the algorithms community [34].

Recently, Paschos et al. [48, 49] proposed studying caching as an online convex optimization (OCO) problem [26]. OCO considers again an adversarial setting, but the metric of interest is the *regret*, i.e., the difference between the costs incurred over a time horizon T by the algorithm and by the optimal offline *static* solution. Online algorithms whose regret grows sublinearly with T are called *no-regret* algorithms, as their time-average regret becomes negligible for large T. Paschos et al. proposed a no-regret caching policy based on the classic online gradient descent method (OGD), under the assumption that (1) the cache can store arbitrarily small fractions of each file (the so-called fractional setting), and (2) the cache state is updated after each request.

In this paper, we extend and generalize the analysis of Paschos et al. in three different directions:

(1) We assume the cache can update its state after processing a batch of $R \geq 1$ requests. This is of interest both in high-demand settings, as well as in cases when updates are infrequent, because they are costly w.r.t. either computation or communication.
(2) We consider a family of caching policies based on online mirror descent (OMD); OGD, employed by Paschos et al., is a special instance of this family.
(3) We also depart from the fractional setting, extending our analysis to the case when the cache can only store entire files (the integral setting).

Batching is a generalization from the point of view of the practical application to caching: online algorithms applied to caching have considered until now a single request $R = 1$ [7, 42, 47, 49], whereas in this work, we consider a more general operation, and we recover the basic one for $R = 1$. In particular, OCO learning algorithms applied to caching suffer from a time complexity that is dependent on the catalog size [26], which can be extremely large. Therefore, despite their theoretical guarantees, their computational overhead is difficult to justify if requests are processed individually, especially when cache updates are costly and can then occur only sporadically. However, this difficulty can be overcome through batching, where a batch includes the requests arriving between two consecutive cache updates. Batching amortizes the computational cost of the different policies, reducing the cost *per request* by the batch size R. Moreover, the batch size R could simply be a characteristic of the caching system instead of being a design choice.

Our contributions are summarized as follows. First, applying the analysis of OMD by Bubeck [11] to the caching setting, we show that the $O\left(\sqrt{T}\right)$ regret of OGD observed by Paschos et al. in the fractional setting extends to general OMD caching policies. We also show that constants in regret bounds depend on the diversity of the request process. In particular, the regret depends on the *diversity ratio* R/h, where R is the size of the batch, and h is the maximum multiplicity of a request in a given batch. Second, we characterize the optimality of OMD caching policies w.r.t. regret under different diversity regimes. We observe that, for a large region of possible values of the diversity ratio, the optimum is either OGD or OMD with a neg-entropy mirror map (OMD_{NE}). In particular, OGD is optimal in the *low diversity* regime, while OMD_{NE} is optimal in the *high diversity* regime. Third, OMD algorithms include a gradient update followed by a projection to guarantee that the new solution is in the feasible set (e.g., it does not violate the cache capacity constraints). The projection is often the most computationally expensive step of the algorithm. We show that efficient polynomial algorithms exist both for OGD (slightly improving the algorithm in [49]) and for OMD_{NE}. Finally, OMD algorithms work in a continuous space, and are therefore well-suited for the fractional setting originally studied by Paschos et al. Still, we show that, if coupled with opportune rounding techniques, they can also be used when the cache can only store a file in its entirety, while preserving their regret guarantees. To the best of our knowledge, this is the first paper to provide a formal characterization of the randomized rounding problem

in caching, wherein the objective is to maintain the regret guarantees for the expected service cost, while minimizing the update costs. This characterization casts the rounding problem as an optimal transport problem in Sec. 6.1. Moreover, we further prove that an opportune modification of Madow's sampling [8, 28, 39, 47] enables to guarantee sublinear expected update costs.

The remainder of this paper is organized as follows. After an overview of the related work in Sec. 2, we introduce our model assumptions in Sec. 3 and provide technical background on gradient algorithms in Sec. 4. Section 4.3 presents our main results on the regret of OMD caching policies and their computational complexity. A discussion about extending the model to include cache update costs, in Sec. 5, is required to introduce the integral setting in Sec. 6. Finally, numerical results are presented in Sec. 7.

2 RELATED WORK

The caching problem has been extensively studied in the literature under different assumptions on the request process. When the requests occur according to a given stochastic process, the analysis leads usually to complex formulas even in simple settings. For example, even the hit ratio of a single cache managed by the LRU eviction policy under the independent reference model is hard to precisely characterize [20, 32]. The characteristic time approximation (often referred to as Che's approximation) significantly simplifies this analysis by assuming that a file, in absence of additional requests for it, stays in the cache for a random time sampled independently from requests for other files. Proposed by Fagin [19] and rediscovered and popularized by Che et al. [14], the approximation has been justified formally by several works [22, 29, 30] and has allowed the study of a large number of existing [23] and new [24, 35] caching policies. It also applies to networked settings [1, 6, 16, 21] and to more general utilities beyond the hit ratio [18, 44], all under stochastic requests.

Online caching policies based on gradient methods have also been studied in the stochastic request setting, leading to Robbins-Monro/stochastic approximation algorithms (see, e.g., [27, 28]). Though related to OCO, guarantees are very different than the regret metric we study here. Many works have also explored the offline, network-wide static allocation of files, presuming demand is known [10, 51, 55]. We differ from the work above, as we consider adversarial requests.

Caching under adversarial requests has been studied since Sleator and Tarjan's seminal paper [57] through the competitive ratio metric. An algorithm is said to be α-competitive when its competitive ratio is bounded by α over all possible input sequences. The problem has been generalized by Manasse et al. [40] under the name *k-server problem*, and further generalized by Borodin et al. under the name *metrical task systems (MTS)* [9]. The literature on both the k-server and MTS problems is vast. A recent trend is to apply continuous optimization techniques to solve these combinatorial problems. Bansal et al. [4] study the k-server problem on a weighted star metric space. In the same spirit, Bubeck et al. [12] use the framework of continuous online mirror descent to provide an $o(k)$-competitive algorithm for the k-server problem on hierarchically separated trees. In this paper, we focus on regret rather than competitive ratio as the main performance metric. Andrew et al. [2] give a formal comparison between competitive ratio and regret and prove that there is an intrinsic incompatibility between the two: no algorithm can have both sub-linear regret and a constant competitive ratio. At the same time, they propose an algorithm with sub-linear regret and slowly increasing competitive ratio.

Online convex optimization (OCO) was first proposed by Zinkevich [60], who showed that projected gradient descent attains sublinear regret bounds in the online setting. OCO generalizes previous online problems like the experts problem [38], and has become widely influential in the learning community [26, 53]. To the best of our knowledge, Paschos et al. [48, 49] were the first to apply the OCO framework to caching. Besides proposing OGD for the single cache, they extended it to a simple networked scenario, where users have access to a set of parallel caches

	Notational Conventions		
$[n]$	Set of integers $\{1, 2, \dots, n\}$	$\mathcal{R}_{R,h}$	Set of possible adversarial requests
	Caching	$\boldsymbol{r}_t$	Batch of request at timeslot t
$\mathcal{N}$	Catalog set with size $\lvert\mathcal{N}\rvert = N$	$f_{\boldsymbol{r}_t}$	Cost received at timeslot t
k	Cache capacity	$\mathrm{UC}_{\boldsymbol{r}_t}$	Update cost of the cache at timeslot t
$\mathcal{X}$	Set of fractional cache states	$\boldsymbol{w}$ / $\boldsymbol{w}'$	Service / update costs in $\mathbb{R}_+^N$
$\mathcal{X}_\delta = \mathcal{X} \cap [\delta, 1]^N$	The δ-interior of $\mathcal{X}$		**Online Learning**
$\mathcal{Z} = \mathcal{X} \cap \{0, 1\}^N$	Set of integral cache states	T	The time horizon
$\boldsymbol{x}_t$	Fractional cache state at timeslot t	η	Learning rate
$\boldsymbol{\zeta}_t$	Integral cache state at timeslot t	$\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})$	Update cost at timeslot t
$\boldsymbol{z}_t$	Random integral cache state at timeslot t	$\mathrm{Regret}_T(\mathcal{A})$	Regret of policy $\mathcal{A}$ over T
$\boldsymbol{x}_*$	Optimal cache allocation in hindsight	$\text{E-Regret}_T(\mathcal{A}, \Xi)$	Extended regret of policy $\mathcal{A}$ over T
R	Number of files' requests in a batch	$\Phi(\boldsymbol{x})$	Mirror map
h	Maximum multiplicity of a requested file	$D_\Phi(\boldsymbol{x}, \boldsymbol{y})$	Bregman divergence associated to Φ
		$\Pi_{\mathcal{B}}^{\Phi}(\boldsymbol{y})$	The projection onto $\mathcal{B}$ under D_Φ

Table 1. Notation Summary

that store pseudo-random linear combinations of the files. They proposed no-regret algorithms in both settings. Bhattacharjee et al. [7] extended this work proving tighter lower bounds for the regret and proposing new caching policies for the networked setting that do not require file coding; Mukhopadhyay and Sinha [42] accounted for switching costs due to file retrievals. Our work drops assumption A2 and A6 stated by Bhattacharjee et al. [7] under both fractional and integral caching settings, because we account for the update cost associated to changing the cache state, and moreover, we permit in our caching model to have multiple requests be processed in a single timeslot $R \geq 1$. In particular, in Sec. 4, only assumptions A3–A5 are needed for the proposed algorithms OGD and $\mathrm{OMD}_{\mathrm{NE}}$, and in Sec. 6, we also require assumption A1, i.e., the cache can fetch files that are not necessarily requested in the previous timeslot. Paria and Sinha [47] studied integral caching over bipartite network topologies. They employ a randomized rounding scheme (Madow's sampling [39]) which is also the starting point for our rounding scheme (Online Rounding in Alg. 3), however, they only provide update cost guarantees under a strong stochastic regularity assumption over the request process. In this work, an opportune modification of the Madow's sampling scheme, motivated by an optimal transport [50] formulation of the randomized rounding problem, guarantees sublinear update cost even under adversarial requests. Li et al. [36], building on our proposed randomized rounding scheme, studied integral caching networks under arbitrary topology and adversarial requests. We depart from these works in considering OMD algorithms, a more general request process, and allowing for integral cache states obtained through randomized rounding.

This work is an extension of our previous work [56]. In particular, (1) we analyze and derive regret bounds for a family of OMD algorithms (q-norm mirror maps), and (2) we extend our analysis to the integral caching setting.

3 SYSTEM DESCRIPTION

Remote Service and Local Cache. We consider a system in which requests for files are served either remotely or by an intermediate cache of finite capacity; a cache miss incurs a file-dependent remote retrieval cost. Formally, we consider a sequence of requests for files of equal size from a catalog $\mathcal{N} = \{1, 2, \dots, N\}$. These requests can be served by a remote server at cost $w_i \in \mathbb{R}_+$ per request for file $i \in \mathcal{N}$. This cost could be, e.g., an actual monetary cost for using the network infrastructure, or a quality of service cost incurred due to fetching latency. Costs may vary across files, as each file may be stored at a different remote location. We denote by $\boldsymbol{w} = [w_i]_{i \in \mathcal{N}} \in \mathbb{R}_+^N$ the vector of costs and assume that $\boldsymbol{w}$ is known.

A local cache of finite capacity is placed in between the source of requests and the remote server(s). The local cache's role is to reduce the costs incurred by satisfying requests locally. We

denote by $k \in \{1, 2, \ldots, N\}$ the capacity of the cache. The cache is allowed to store fractions of files (this assumption will be removed in Sec. 6). We assume that time is slotted, and denote by $x_{t,i} \in [0, 1]$ the fraction of file $i \in \mathcal{N}$ stored in the cache at timeslot $t \in \{1, 2, \ldots, T\}$. The cache state is then given by vector $\boldsymbol{x}_t = [x_{t,i}]_{i \in \mathcal{N}} \in \mathcal{X}$, where $\mathcal{X}$ is the capped simplex determined by the capacity constraint, i.e., $\mathcal{X} = \left\{\boldsymbol{x} \in [0, 1]^N : \sum_{i=1}^N x_i = k\right\}$.

Requests. We assume that a batch of multiple requests may arrive within a single timeslot. The number of requests (i.e., the batch size) at each timeslot is given by $R \in \mathbb{N}$. A file may be requested multiple times (e.g., by different users, whose aggregated requests form the stream reaching the cache) within a single timeslot. We denote by $r_{t,i} \in \mathbb{N}$ the *multiplicity* of file $i \in \mathcal{N}$, i.e., the number of requests for i, at time t, and by $\boldsymbol{r}_t = [r_{t,i}]_{i \in \mathcal{N}} \in \mathbb{N}^N$ the vector of such requests, representing the entire batch. We also assume that the maximum multiplicity of a file in a batch is bounded by $h \in \mathbb{N}$. As a result, $\boldsymbol{r}_t$ belongs to set $\mathcal{R}_{R,h} = \left\{\boldsymbol{r} \in \{0, \ldots, h\}^N : \sum_{i=1}^N r_i = R\right\}$.

Intuitively, the ratio $\frac{R}{h}$ defines the diversity of request batches in a timeslot. For example, when $\frac{R}{h} = 1$, all R requests are concentrated on a single file. When $\frac{R}{h} = N$, requests are spread evenly across the catalog $\mathcal{N}$. In general, $\frac{R}{h}$ is a lower bound for the number of distinct files requested in the batch. For that reason, we refer to $\frac{R}{h}$ as the *diversity ratio*.[1] We note that our request model generalizes the setting by Paschos et al. [49], which can be seen as the case $R = h = 1$, i.e., the batch contains only one request per timeslot. We make no additional assumptions on the request arrival process; put differently, we operate in the adversarial online setting, where a potential adversary may select an arbitrary request sequence $\{\boldsymbol{r}_t\}_{t=1}^T$ in $\mathcal{R}_{R,h}$ to increase system costs.

Service Cost Objective. When a request batch $\boldsymbol{r}_t$ arrives, the cache incurs the following cost:

$$f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) = \sum_{i=1}^N w_i r_{t,i}(1 - x_{t,i}). \tag{1}$$

In other words, for each file $i \in \mathcal{N}$, the system pays a cost proportional to the file fraction $(1 - x_{t,i})$ missing from the local cache, weighted by the file cost w_i and by the number of times $r_{t,i}$ file i is requested in the current batch $\boldsymbol{r}_t$.

The cost objective (1) captures several possible real-life settings. First, it can be interpreted as a QoS cost paid by each user for the additional delay to retrieve part of the file from the server. Second, assuming that the R requests arrive and are served individually (e.g., because they are spread-out within a timeslot), Eq. (1) can represent the load on the servers or on the network to provide the missing part of the requested files. Our model also applies when all requests for the same file are aggregated and served simultaneously by a single fetch operation. In this case, $r_{t,i}$ in Eq. (1) should be interpreted as the indicator variable denoting if file i was requested; correspondingly, R then indicates the total number of *distinct* files requested, and $h = 1$.

Online Caching Algorithms and Regret. Cache files are determined online as follows. The cache has selected a state $\boldsymbol{x}_t \in \mathcal{X}$ at the beginning of a timeslot.[2] The request batch $\boldsymbol{r}_t$ arrives, and the linear cost $f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$ is incurred; the state is subsequently updated to $\boldsymbol{x}_{t+1}$. Formally, the cache state is determined by an online policy $\mathcal{A}$, i.e., a sequence of mappings $\{\mathcal{A}_t\}_{t=1}^{T-1}$, where for every $t \geq 1$, $\mathcal{A}_t : (\mathcal{R}_{R,h} \times \mathcal{X})^t \to \mathcal{X}$ maps the sequence of past request batches and decisions $\{(\boldsymbol{r}_s, \boldsymbol{x}_s)\}_{s=1}^t$ to the next state $\boldsymbol{x}_{t+1} \in \mathcal{X}$. We assume that the policy starts from a feasible state $\boldsymbol{x}_1 \in \mathcal{X}$.

We measure the performance of an online algorithm $\mathcal{A}$ in terms of regret, i.e., the difference between the total cost experienced by a policy $\mathcal{A}$ over a time horizon T and that of the best static

[1] This definition of diversity is consistent with other notions of diversity, such as, e.g., the entropy; indeed the diversity ratio provides a lower bound on the entropy of the normalized batch vector $\frac{\boldsymbol{r}_t}{R}$, as $E\left(\frac{\boldsymbol{r}_t}{R}\right) \geq \log\left(\frac{R}{h}\right)$ [37, Lemma 3], where $E(\boldsymbol{p}) = -\sum_i p_i \log(p_i)$ is the entropy function.

[2] We neglect the cost associated with the initial population of the cache since it is a fixed one-time cost.

state $\boldsymbol{x}_*$ in hindsight. Formally,

$$\text{Regret}_T(\mathcal{A}) = \sup_{\{\boldsymbol{r}_1,\boldsymbol{r}_2,\ldots,\boldsymbol{r}_t\}\in\mathcal{R}_{R,h}^T} \left\{\sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) - \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_*)\right\}, \tag{2}$$

where $\boldsymbol{x}_* = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x})$ is the optimal static cache state (in hindsight). Note that, by taking the supremum in Eq. (2), we indeed measure regret in the adversarial setting, i.e., against an adversary that potentially picks requests in $\mathcal{R}_{R,h}$ trying to jeopardize cache performance.

Update Costs. An online algorithm $\mathcal{A}$ updating the cache state at timeslot t may require moving a portion of a file from a remote server to the cache to implement this update. The update cost of the online algorithm is not explicitly modeled in our cost and regret (Eqs. (1) and (2), respectively). We postpone the discussion of such cost in Sec. 5. For the moment we observe that updates come "for free" for files requested in the current timeslot. The main algorithms studied in this paper (OGD and OMD_{NE}) implement cache updates by fetching parts of files that have been requested in the previous timeslot. As a result, to implement these updates we can piggyback the traffic created to serve the user, and the cost of this traffic is already accounted for in our service cost model (1). As a result, the update cost is zero (see also Proposition 5.1). We note that this property does not hold for randomized integral caching policies in Sec. 6, which may require to store files that have not been requested.

4 FRACTIONAL CACHING AND GRADIENT-BASED ALGORITHMS

Inspired by offline minimization, it is natural to design a policy that, upon seeing $\boldsymbol{r}_t$, selects as $\boldsymbol{x}_{t+1}$ the state that would have minimized (on hindsight) the aggregate cost up to time t (i.e., $\sum_{t'=1}^t f_{\boldsymbol{r}_{t'}}(\boldsymbol{x})$). Unfortunately, such a policy has poor regret:

Proposition 4.1. *The aggregate cost minimization policy is a policy $\mathcal{A}$ that selects for every timeslot $t \in [T-1]$ the state $\boldsymbol{x}_{t+1} = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{t'=1}^t f_{\boldsymbol{r}_{t'}}(\boldsymbol{x})$. This policy has linear (worst-case) regret, i.e.,* $\text{Regret}(\mathcal{A}) = \Omega(T)$.

The proof follows the same argument of Shalev-Shwartz [53, Example 2.2]. A more conservative approach, that indeed leads to sublinear regret, is to take gradual steps, moving in the direction of a better decision according to the latest cost; we present algorithms of this nature in this section.

4.1 Online Gradient Descent (OGD)

In OGD, introduced by Paschos et al. [49] for online caching, the cache is initialized with a feasible state $\boldsymbol{x}_1 \in \mathcal{X}$ and updated as follows. Upon receiving a request batch $\boldsymbol{r}_t$, the cost $f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$ is incurred and the next state becomes:

$$\boldsymbol{x}_{t+1} = \Pi_{\mathcal{X}}\left(\boldsymbol{x}_t - \eta \nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)\right), \quad \text{for all } t \in [T-1], \tag{3}$$

where $\Pi_{\mathcal{X}}(\cdot)$ is the Euclidean projection onto $\mathcal{X}$, that ensures feasibility, and $\eta \in \mathbb{R}_+$ is called the learning rate. Note that the state $\boldsymbol{x}_{t+1}$ obtained according to Eq. (3) is indeed a function of $\{(\boldsymbol{r}_t, \boldsymbol{x}_t)\} \subset \{(\boldsymbol{r}_s, \boldsymbol{x}_s)\}_{s=1}^t$ for every $t \geq 1$; hence, OGD is indeed an online caching policy as defined in Sec. 3. Paschos et al. [49] show that OGD attains sub-linear regret when $R = h = 1$; more specifically:

Theorem 4.2.*([49, Theorem 2]) When $R = h = 1$, the regret of OGD is bounded as follows:*

$$\text{Regret}_T(\text{OGD}) \leq \|\boldsymbol{w}\|_\infty \sqrt{\min(2k, 2(N-k))T}. \tag{4}$$

In other words, OGD attains an $O\left(\sqrt{T}\right)$ regret when $R = h = 1$. In this paper, we study a broader class of gradient descent algorithms that include OGD as a special case. As we will see below (see

Algorithm 1 Online mirror descent (OMD_Φ)

Require: $\boldsymbol{x}_1 = \underset{\boldsymbol{x}\in\mathcal{X}\cap\mathcal{D}}{\arg\min}\,\Phi(\boldsymbol{x})$, $\eta \in \mathbb{R}_+$

for $t \leftarrow 1, 2, \ldots, T$ **do** ▷ Incur a cost $f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$, and receive a gradient $\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x})$

  $\hat{\boldsymbol{x}}_t \leftarrow \nabla\Phi(\boldsymbol{x}_t)$ ▷ Map primal point to dual point

  $\hat{\boldsymbol{y}}_{t+1} \leftarrow \hat{\boldsymbol{x}}_t - \eta\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$ ▷ Take gradient step in the dual space

  $\boldsymbol{y}_{t+1} \leftarrow (\nabla\Phi)^{-1}(\hat{\boldsymbol{y}}_{t+1})$ ▷ Map dual point to a primal point

  $\boldsymbol{x}_{t+1} \leftarrow \Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\boldsymbol{y}_{t+1})$ ▷ Project new point onto feasible region $\mathcal{X}$

end for

Thm. 4.8), the regret attained by OGD is not necessarily the tightest possible when $R \neq 1 \neq h$; broadening the class of algorithms we consider allows us to improve upon this bound.

4.2 Online Mirror Descent (OMD)

OMD [26, Sec. 5.3] is the online version of the mirror descent (MD) algorithm [5] for convex optimization of a fixed, known function. The main premise behind mirror descent is that variables and gradients live in two distinct spaces: the *primal space*, for variables, and the *dual space*, for gradients. The two are linked via a function known as a *mirror map*. Contrary to standard gradient descent, updates using the gradient occur on the dual space; the mirror map is used to invert this update to a change on the primal variables. For several constrained optimization problems of interest, mirror descent leads to faster convergence compared to gradient descent [11, Sec. 4.3]. OMD arises by observing that MD is agnostic to whether the gradients are obtained from a *fixed* function, or a sequence revealed adversarially.

OMD for Caching. Applied to our caching problem, OMD takes the form summarized in Algorithm 1. In our case, both the primal and dual spaces are $\mathbb{R}^N$. To disambiguate between the two, we denote primal points by $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$ and dual points by $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}} \in \mathbb{R}^N$, respectively. Formally, OMD is parameterized by (1) a fixed learning rate $\eta \in \mathbb{R}_+$, and (2) a differentiable map $\Phi : \mathcal{D} \to \mathbb{R}$, strictly convex over $\mathcal{D}$ and ρ-strongly convex over $\mathcal{X} \cap \mathcal{D}$, where $\mathcal{X}$ is included in the closure of $\mathcal{D}$; that is

$$\mathcal{X} \subseteq \text{closure}(\mathcal{D}). \tag{5}$$

Function Φ is called the *mirror map*, that links the primal to the dual space.

Given η and Φ, an OMD iteration proceeds as follows. After observing the request batch $\boldsymbol{r}_t$ and incurring the cost $f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$, the current state $\boldsymbol{x}_t$ is first mapped from the primal to the dual space via:

$$\hat{\boldsymbol{x}}_t = \nabla\Phi(\boldsymbol{x}_t). \tag{6}$$

Then, a regular gradient descent step is performed *in the dual space* to obtain an updated dual point:

$$\hat{\boldsymbol{y}}_{t+1} = \hat{\boldsymbol{x}}_t - \eta\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t). \tag{7}$$

This updated dual point is then mapped back to the primal space using the inverse of mapping $\nabla\Phi$, i.e.:

$$\boldsymbol{y}_{t+1} = (\nabla\Phi)^{-1}(\hat{\boldsymbol{y}}_{t+1}). \tag{8}$$

The resulting primal point $\boldsymbol{y}_{t+1}$ may lie outside the constraint set $\mathcal{X}$. To obtain the final feasible point $\boldsymbol{x}_{t+1} \in \mathcal{X}$, a projection is made using the Bregman divergence associated with the mirror map Φ; that is, instead of the orthogonal projection used in OGD, the final cache state becomes:

$$\boldsymbol{x}_{t+1} = \Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\boldsymbol{y}_{t+1}), \tag{9}$$

where $\Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\,\cdot\,)$ is the Bregman projection, which we define formally below, in Definition 4.3.

Together, steps (6)–(9) define OMD. Note that, as it was the case for OGD, $\boldsymbol{x}_{t+1}$ is a function of $\{(\boldsymbol{r}_t, \boldsymbol{x}_t)\} \subset \{(\boldsymbol{r}_s, \boldsymbol{x}_s)\}_{s=1}^t$, hence OMD is indeed an online algorithm. Two additional technical assumptions on Φ and $\mathcal{D}$ must hold for steps (8) and (9) to be well-defined.[3] First, the gradient of Φ must diverge at the boundary of $\mathcal{D}$; this, along with strict convexity, ensures the existence and uniqueness of the Bregman projection in (9). Second, the image of $\mathcal{D}$ under the gradient of Φ should take all possible values, that is $\nabla\Phi(\mathcal{D}) = \mathbb{R}^N$; this, along again with strict convexity, ensures that $\nabla\Phi$ is one-to-one and onto, so its inverse exists and Eq. (8) is well-defined.

Setting $\Phi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_2^2$ and $\mathcal{D} = \mathbb{R}^N$ yields the identity mapping $\nabla\Phi(\boldsymbol{x}) = \boldsymbol{x}$, for all $\boldsymbol{x} \in \mathcal{D}$. Furthermore, the Bregman divergence associated with this map is just the Euclidean distance $D_\Phi(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2$. Thus, this Euclidean version of OMD is equivalent to OGD, and OMD can be seen as a generalization of the OGD to other mirror maps.

To conclude our description of OMD, we define the Bregman projection [33].

Definition 4.3. The Bregman projection denoted by $\Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}} : \mathbb{R}^N \to \mathcal{X} \cap \mathcal{D}$, is defined as

$$\Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\boldsymbol{y}) = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathcal{X}\cap\mathcal{D}} D_\Phi(\boldsymbol{x}, \boldsymbol{y}), \qquad \text{where} \qquad D_\Phi(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x}) - \Phi(\boldsymbol{y}) - \nabla\Phi(\boldsymbol{y})^T(\boldsymbol{x} - \boldsymbol{y}) \tag{10}$$

is the Bregman divergence associated with the mirror map Φ.

4.3 Analysis of Online Mirror Descent Algorithms

We present our main results regarding the application of OMD under several different mirror maps to the online caching problems. We will be concerned with both (1) the regret attained, and (2) computational complexity issues, particularly pertaining to the associated Bregman projection. Our key observation is that *the regret of different algorithms is significantly influenced by demand diversity, as captured by the diversity ratio* $\frac{R}{h}$. In particular, our analysis allows us to characterize regimes of the diversity ratio in which OGD outperforms other mirror maps, and vice versa.

4.4 q-Norm Mirror Maps

A natural generalization of the OGD algorithm to a broader class of OMD algorithms is via q-norm mirror maps, whereby:

$$\Phi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_q^2, \quad \text{where } q \in (1, 2], \text{ and } \quad \mathcal{D} = \mathbb{R}^N. \tag{11}$$

It is easy to verify that Φ and $\mathcal{D}$, defined as above, satisfy all technical requirements set in Sec. 4.2 on a mirror map and its domain. We define $\text{OMD}_{q\text{-norm}}$ to be the OMD Algorithm 1 with Φ and q given by Eq. (11). Note that this map generalizes OGD, which corresponds to the special case $q = 2$. In what follows, we denote by $\|\cdot\|_p$ the dual norm of $\|\cdot\|_q$. Then, $p \in [2, \infty)$ is such that $\frac{1}{p} + \frac{1}{q} = 1$. Note that sometimes $\text{OMD}_{q\text{-norm}}$ is referred to as a *p-norm algorithm* [53].

4.4.1 Regret Analysis. We begin by providing a regret bound for $\text{OMD}_{q\text{-norm}}$ algorithms:

Theorem 4.4. *For* $\eta = \sqrt{\frac{(q-1)k^2\left(k^{-\frac{2}{p}} - N^{-\frac{2}{p}}\right)}{\|\boldsymbol{w}\|_\infty^2 h^2 \left(\frac{R}{h}\right)^{\frac{2}{p}} T}}$, *the regret of* $\text{OMD}_{q\text{-norm}}$ *over* $\mathcal{X}$ *satisfies:*

$$\text{Regret}_T(\text{OMD}_{q\text{-norm}}) \leq \|\boldsymbol{w}\|_\infty hk \left(\frac{R}{h}\right)^{\frac{1}{p}} \sqrt{\frac{1}{q-1}\left(k^{-\frac{2}{p}} - N^{-\frac{2}{p}}\right) T}. \tag{12}$$

[3] All hold for the algorithms we consider in Sec. 4.3.

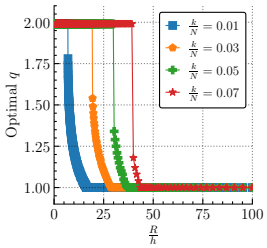


Fig. 1. Numerical characterization of $q^* \in [1, 2]$ as a function of the diversity ratio R/h, for different cache capacities k expressed as fractions of the catalog size ($N = 100$). Given R/h, the optimal q^* is determined as the value in $[1, 2]$ that minimizes the upper-bound in Eq. (12). Higher values of R/h represent more diverse requests. Under small diversity, OGD is optimal; as diversity increases, mirror maps for which $q < 2$ attain a more favorable upper bound than OGD.

The proof can be found in Appendix A.2. We use an upper bound on the regret of general OMD from [11, Theorem 4.2] and relate it to our setting; in doing so, we bound the diameter of $\mathcal{X}$ w.r.t. Bregman divergence under Φ as well as the dual-norm $\|\cdot\|_p$ of the gradients $\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$.

Comparing Theorem 4.4 to Theorem 4.2, we see that both attain an $O\left(\sqrt{T}\right)$ regret. A natural question to ask when comparing the two bounds is whether there are cases where $\text{OMD}_{q\text{-norm}}$ with $q \neq 2$ outperforms OGD (i.e., $\text{OMD}_{2\text{-norm}}$). The constants in the r.h.s. of Eq. (12) depend on the diversity ratio $\frac{R}{h}$; this, in turn, affects which is the optimal q, i.e., the one that minimizes the bound in Eq. (12). Let $q^* = \arg\inf_{q\in(1,2]} \mathsf{ub}(q)$ be the optimal q, where $\mathsf{ub} : (1, 2] \to \mathbb{R}_+$ is the upper bound in Eq. (12). Note that $q^* \in [1, 2]$. Figure 1 shows q^* as a function of the diversity ratio, for different values of cache capacity k. We observe that OGD ($q = 2$) is optimal for lower diversity regimes and larger caches; when diversity $\frac{R}{h}$ increases or cache capacity k decreases, values $q < 2$ become optimal. The transition from $q^* = 2$ to $q^* = 1$ is sharp, and becomes sharper as k increases.

4.4.2 Optimality Regimes. Motivated by these observations, we turn our attention to formally characterizing the two regimes under which optimality transitions from $q^* = 2$ to $q^* = 1$. We first determine the upper bound on the regret for these two regimes. Indeed, by setting $q = 2$ in Theorem 4.4, we obtain the following bound, generalizing Theorem 4.2 to the case $R/h > 1$:

Corollary 4.5. *For* $\eta = \sqrt{\frac{k\left(1-\frac{k}{N}\right)}{\|\boldsymbol{w}\|_\infty^2 hRT}}$ *the regret of OGD, satisfies:*

$$\text{Regret}_T(\text{OGD}) \leq \|\boldsymbol{w}\|_\infty \sqrt{hRk\,(1 - k/N)\,T}. \tag{13}$$

This a direct consequence of Theorem 4.4 by replacing $q = 2$ in Eq. (12). We note that, in this result, we tighten the bound of Paschos et al. [49]: for $R = h = 1$, the bound in Eq. (13) is smaller than the one in Theorem 4.2 by at least a $\sqrt{2}$ factor.

We also characterize the limiting behavior of $\text{OMD}_{q\text{-norm}}$ as q converges to 1.

Corollary 4.6. *As* q *converges to* 1*, the upper bound on* $\text{OMD}_{q\text{-norm}}$ *regret given by Eq.* (12) *converges to:*

$$\|\boldsymbol{w}\|_\infty\, hk\sqrt{2\log\left(N/k\right)T}. \tag{14}$$

The proof can be found in Appendix A.3. This limit is precisely the bound on the regret attained under the neg-entropy mirror map (see Theorem 4.10 below). Armed with Corollaries 4.5 and 4.6, we can formally characterize the regimes in which either of the two strategies become dominant:

Theorem 4.7. *The regret bound for* $\text{OMD}_{q\text{-norm}}$ *in Eq.* (12) *is minimized for* $q = 2$*, when* $\frac{R}{h} \leq k$.

In other words, when the diversity ratio is smaller than the cache size, it is preferable to update the cache via OGD. The proof, in Appendix A.4, establishes that the upper bound in Eq. (12) is monotonically decreasing w.r.t q in the specified interval $\frac{R}{h} \leq k$. Our next result characterizes then the neg-entropy (q converges to 1) mirror map outperforms OGD:

Theorem 4.8. *The limit, as q converges to 1, of the* $\text{OMD}_{q\text{-norm}}$ *regret bound in Eq.* (14) *is smaller than the corresponding bound for OGD (*$\text{OMD}_{q\text{-norm}}$ *with $q = 2$) when $\frac{R}{h} > 2\sqrt{Nk}$.*

The proof is provided in Appendix A.5. We stress that Theorem 4.8 implies the sub-optimality of OGD in the regime $\frac{R}{h} > 2\sqrt{Nk}$. The experiments in Fig. 1 suggest the bound in Theorem 4.8 is quite tight: for example for $k = 7$ the bounds suggest $q = 1$ should be optimal when R/h exceeds $2\sqrt{100 \times 7} \approx 52.9$, while experiments show that it is optimal when R/h exceeds 45. On the contrary, we observe that the bound in Theorem 4.7 seems to be loose and the transitions we observe in Fig. 1 are sharper than what one would predict from the bounds.

4.4.3 Dual-Primal Update and Bregman Projection. Having characterized the regret of $\text{OMD}_{q\text{-norm}}$ algorithms, we turn our attention to implementation issues. The map to the dual space and back in Eq. (6) and Eq. (8) (Lines 2 and 4 in Algorithm 1), have the following expression [25], respectively:

$$\hat{x}_{t,i} = (\nabla\Phi(\boldsymbol{x}_t))_i = \text{sign}(x_{t,i})|x_{t,i}|^{q-1}/\|\boldsymbol{x}_t\|_q^{q-2}, \quad \text{for all } \; i \in \mathcal{N}, \tag{15}$$

$$y_{t+1,i} = \left((\nabla\Phi)^{-1}(\hat{\boldsymbol{y}}_{t+1})\right)_i = \text{sign}(\hat{y}_{t+1,i})|\hat{y}_{t+1,i}|^{p-1}/\left\|\hat{\boldsymbol{y}}_{t+1}\right\|_p^{p-2}, \quad \text{for all } \; i \in \mathcal{N}. \tag{16}$$

Finally, for all $q \in (1, 2]$ the Bregman projection in Eq. (9) (Line 5 in Algorithm 1) involves solving a convex optimization problem, in general. For the OGD Algorithm however ($q = 2$) the projection is the usual Euclidean projection. The following theorem holds:

Theorem 4.9. *The Euclidean projection requires $O\left(N^2\right)$ operations per iteration, for general values of R and h, and only $O(N)$ operations, when $\frac{R}{h} = 1$.*

For general values of R and h the Euclidean projection is performed using the projection algorithm by Wang and Lu [59] in $O\left(N^2\right)$ time. Specifically when $\frac{R}{h} = 1$, only a single coefficient is updated through the gradient step (Lines 2–4 in Algorithm 1) per iteration, and Paschos et al. [49] provide an algorithm that performs the projection in $O(N)$ time.[4]

4.5 Neg-Entropy Mirror Map

To conclude this section, we turn our attention to the neg-entropy mirror map that, as discussed earlier, attains the same regret performance as $\text{OMD}_{q\text{-norm}}$ as q converges to 1. Beyond its improved performance in terms of regret in the high diversity ratio regime, the neg-entropy mirror map comes with an additional computational advantage: the Bregman projection admits a highly efficient implementation.

Formally, OMD under the neg-entropy mirror map uses:

$$\Phi(\boldsymbol{x}) = \sum_{i=1}^{N} x_i \log(x_i), \text{ and } \mathcal{D} = \mathbb{R}^N_{>0}. \tag{17}$$

Note that, as per the requirements in Sec. 4.2, $\mathcal{X} \subseteq \text{closure}(\mathcal{D})$. Also, $\nabla\Phi$ indeed diverges at the boundary of $\mathcal{D}$, and $\nabla\Phi(\mathcal{D}) = \mathbb{R}^N$, as

$$\frac{\partial\Phi(\boldsymbol{x})}{\partial x_i} = 1 + \log(x_i), \quad \text{for all } i \in \mathcal{N}. \tag{18}$$

We refer to the resulting algorithm as OMD_{NE}.

[4]To be precise, the projection algorithm as presented in [49] requires at each iteration a preliminary step with complexity $O(N \log(N))$ to sort a vector of size N, followed by $O(N)$ steps. However, it is possible to replace sorting by $O(\log(N))$ binary search and insertion operations reducing the complexity to $O(N)$ per iteration.

Algorithm 2 Neg-Entropy Bregman projection onto the capped simplex

Require: N; k; $\|\boldsymbol{y}\|_1$; P; Partially sorted $y_N \geq \cdots \geq y_{N-k+1} \geq y_i, \forall i \leq N-k$
▷ $\boldsymbol{y}$ is the intermediate cache state, and P is a scaling factor initialized to 1
$y_{N+1} \leftarrow +\infty$
for b $\in \{N, \ldots, N-k+1\}$ **do**
  $m_b \leftarrow (k+b-N) \,/\, \left(\|\boldsymbol{y}\|_1 - \sum_{i=b+1}^{N} y_i P\right)$
  if $y_b m_b P < 1 \leq y_{b+1} m_b P$ **then**
    ▷ Appropriate b is found
    for i $\geq$ b+1 **do**
      $y_i \leftarrow 1/(m_b P)$
    end for
    $P \leftarrow m_b P$
    return $\boldsymbol{y}P$ ▷ $\boldsymbol{y}P$ is the result of the projection
  end if
end for

4.5.1 Regret Analysis. We first characterize the regret of OMD_{NE}:

Theorem 4.10. *For* $\eta = \sqrt{\frac{2\log(N/k)}{\|\boldsymbol{w}\|_\infty^2 h^2 T}}$, *the regret of* OMD_{NE} *satisfies:*

$$\text{Regret}_T(\text{OMD}_{\text{NE}}) \leq \|\boldsymbol{w}\|_\infty hk\sqrt{2\log(N/k)}. \tag{19}$$

The proof, in Appendix A.7, is similar to the proof of Theorem 4.4. Using again the general bound of the regret of OMD algorithms in Bubeck [11, Theorem 4.2], we bound the diameter of $\mathcal{X}$ w.r.t. to the Bregman divergence as well as the dual norm $\|\cdot\|_\infty$ of gradients $\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$. Crucially, we observe that OMD_{NE} indeed attains the same regret bound as the one in Corollary 4.6, namely, the bound on $\text{OMD}_{q\text{-norm}}$ when q converges to 1. This immediately implies the advantage of OMD_{NE} over OGD in high diversity ratio regimes, as described in Sec. 4.4.2 and Theorem 4.8.

4.5.2 Dual-Primal Update and Bregman Projection. As $\nabla\Phi(\boldsymbol{x})$ is given by Eq. (18), the inverse mapping is given by $\left((\nabla\Phi)^{-1}(\hat{\boldsymbol{y}}_{t+1})\right)_i = \exp(\hat{y}_{t,i} - 1)$. Hence, the map to the dual space and back in Eq. (6)–Eq. (8) (Lines 2–4 in Algorithm 1) can be concisely written as:

$$y_{t+1,i} = \exp\left(\hat{x}_{t,i} - \eta\frac{\partial f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)}{\partial x_i} - 1\right) = \exp\left(\log(x_{t,i}) - \eta\frac{\partial f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)}{\partial x_i}\right) = x_{t,i}\, e^{-\eta\frac{\partial f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)}{\partial x_i}}, \text{ for all } i \in \mathcal{N}. \tag{20}$$

In other words, OMD under the neg-entropy mirror map adapts the cache state via *a multiplicative rule* (namely, the one implied by the above equation), as opposed to the additive rule of OGD (see Eq. (3)). In Theorem A.2 we prove that $\text{OMD}_{q\text{-norm}}$ when q converges to 1 also adapts the cache state via a multiplicative update rule; moreover, it is equivalent to OMD_{NE} over the simplex. This justifies why the regret bounds for the two algorithms in Eq. (14) and Eq. (19) are identical.

Finally, the projection algorithm onto the capped simplex can be implemented in $O(N + k\log(k))$ time for arbitrary R and h values using a waterfilling-like algorithm. The full procedure is presented in Algorithm 2. The algorithm receives as input the top-k elements of $\boldsymbol{y}$, sorted in descending order. It then identifies via a linear search which elements exceed an appropriate threshold and set them to one. The other elements are scaled by a constant factor to satisfy the capacity constraint. The following theorem holds:

Theorem 4.11. *Algorithm 2 returns the projection* $\Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\boldsymbol{y})$ *onto the capped simplex* $\mathcal{X}$ *under the neg-entropy* Φ. *It requires* $O(N + k\log(k))$ *operations per iteration, for general values of R and h, and only* $O(k)$ *operations, when* $\frac{R}{h} = 1$.

The proof is given in Appendix A.8. To prove this theorem, we characterize the KKT conditions of the minimization problem. Then we show that these conditions can be checked in $O(k)$ time. Finally, we show how maintaining $\boldsymbol{y}$ in a partially sorted list across iterations leads to the reported complexity results. Theorem 4.11 implies that OMD_{NE} *has significant computational savings when compared to OGD* (cf. Theorem 4.9), both when $\frac{R}{h} = 1$ and for general values of R and h.

5 UPDATE COST

The model presented in Sec. 3 can be extended by adding the cost to update the cache state after the batch of R requests has been served. This cost may quantify the additional load on the server or on the network. This update cost is often called *movement cost* [11] or *switching cost* [2]. As the state changes from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$, the cache evicts part of the file i if $x_{t+1,i} < x_{t,i}$ and stores additional bytes of it if $x_{t+1,i} > x_{t,i}$. We make the following assumptions:

(1) Evictions do not engender update costs, as the cache can perform them autonomously;
(2) Insertions of (part of) files which have been requested do not engender update costs, as these files have already been retrieved by the cache in their entirety to satisfy the requests.
(3) Insertions of (part of) files which have not been requested incur a cost proportional to the fraction of file retrieved.

We can then define the update cost at time slot t as

$$\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}) = \textstyle\sum_{i \notin \mathrm{supp}(\boldsymbol{r}_t)} w'_i \max\left\{0, x_{t+1,i} - x_{t,i}\right\}, \tag{21}$$

where $\mathrm{supp}(\boldsymbol{r}_t) = \left\{i \in \mathcal{N} : r_{t,i} \neq 0\right\}$ denotes the support of $\boldsymbol{r}_t$, i.e., the set of files that have been requested during the t-th timeslot, and $w'_i \in \mathbb{R}_+$ is the cost to retrieve the whole file i, and can in general be different from the cost w_i appearing in (1).

If the update cost is introduced in the model, the *extended* regret can be defined as follows:

$$\text{E-Regret}_T(\mathcal{A}) = \sup_{\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_t\} \in \mathcal{R}_{R,h}^T} \left\{\textstyle\sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) + \mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}) - \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_*)\right\} \tag{22}$$

$$\leq \sup_{\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_t\} \in \mathcal{R}_{R,h}^T} \left\{\textstyle\sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) - \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_*)\right\} + \sup_{\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_t\} \in \mathcal{R}_{R,h}^T} \left\{\textstyle\sum_{t=1}^T \mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})\right\}. \tag{23}$$

Equation (23) shows that the regret of an arbitrary online algorithm can be bounded by considering the regret we have derived so far (Eq. (2)), ignoring update costs, and subsequently accounting for an additional term corresponding to the update. Note that the optimal static allocation does not incur any update cost. Equation (23) implies that any policy with $O\left(\sqrt{T}\right)$ regret and $O\left(\sqrt{T}\right)$ update cost in expectation has also $O\left(\sqrt{T}\right)$ extended regret.

One of the reasons why we did not introduce directly the update cost is that, in the fractional setting, OMD update cost is zero both for the Euclidean (OGD) and the neg-entropy ($\mathrm{OMD}_{\mathrm{NE}}$) mirror maps. Formally, we have:

Proposition 5.1. *For any request batch $\boldsymbol{r}_t$ received at time slot $t \in [T]$, the update of fractional cache state from $\boldsymbol{x}_t \in \mathcal{X}$ to $\boldsymbol{x}_{t+1} \in \mathcal{X}$ obtained by* $\mathrm{OMD}_{\mathrm{NE}}$ *or* OGD *has no cost, i.e.,* $\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}) = 0$.

The proof is provided in Appendix A.9. In fact, the gradient step increases the fraction $x_{t,i}$ only for files i that have been requested, and the projection step reduces the fraction for all other files in order to satisfy the capacity constraint. It follows that $x_{t+1,i} - x_{t,i} > 0$ if and only if $i \in \mathrm{supp}(\boldsymbol{r}_t)$, and thus $\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}) = 0$. Hence, the $O\left(\sqrt{T}\right)$ regret guarantees we proved in the previous sections for OGD and $\mathrm{OMD}_{\mathrm{NE}}$ extend to the more general definition in (22). In the next section, we show that update costs *cannot be neglected* when caches are forced to store files in their entirety.

6 INTEGRAL CACHING

In the previous sections, we assumed that the cache can store arbitrarily small chunks of a file, and this allowed us to design no-regret policies that employ fractional caching. However, this assumption can be too strong in some applications. For example, when the catalog is composed of small-sized files, the discreteness of chunks sizes cannot be neglected; moreover, the metadata needed for

each chunk can cause memory and computational overheads. These observations motivate us to study the case when the cache can only store the entire file. We refer to this setting as the *integral* caching. Formally, we restrict the cache states to belong to the set $\mathcal{Z} = \left\{\boldsymbol{\zeta} \in \{0,1\}^N : \sum_{i \in \mathcal{N}} \zeta_i = k\right\}$. Note that the set $\mathcal{Z}$ is a restriction of the set of fractional caching states $\mathcal{X}$ to its corners, i.e., $\mathcal{Z} = \mathcal{X} \cap \{0,1\}^N$; thus, we maintain the same definition of the requests and the service cost objective in Sec. 3. In this setting, we allow policies to be randomized. This extension turns out to be necessary in order to have a sublinear regret policy; formally, we have:

Proposition 6.1. *Any deterministic policy restricted to select integral cache states in $\mathcal{Z}$ has the following lower bound on its regret:* $\text{Regret}_T(\mathcal{A}) \geq k\,(1 - k/N)\,T$.

To prove the proposition, we show that an adversary can exploit the deterministic nature of the policy by continuously requesting the files that are not stored in the cache. We provide the proof in Appendix B.1.

We thus turn our attention to randomized policies. In particular, we focus on a special class of randomized policies, constructed by (1) a fractional online caching policy $\mathcal{A}$, i.e., of the type we have studied so far (see Sec. 3), combined with (2) a randomized rounding scheme Ξ, that maps fractional caching states to integral ones. In particular, for every $t \geq 1$ the randomized rounding scheme $\Xi : \mathcal{X}^t \times \mathcal{Z}^{t-1} \times [0,1] \to \mathcal{Z}$ maps the previous fractional cache states $\{\boldsymbol{x}_s\}_{s=1}^{t-1} \in \mathcal{X}^{t-1}$, the current fractional cache state $\boldsymbol{x}_t \in \mathcal{X}$, the previous random cache states $\{\boldsymbol{z}_s\}_{s=1}^{t-1} \in \mathcal{Z}^{t-1}$, and a source of randomness[5] $\xi_t \in [0,1]$ to a new random cache state $\boldsymbol{z}_t \in \mathcal{Z}$ where

$$\mathbb{E}[\boldsymbol{z}_t] = \boldsymbol{x}_t. \tag{24}$$

Note that the rounding function takes into account not only the current fractional state $\boldsymbol{x}_t$, which determines its expectation, but also the past fractional and integral states $(\{(\boldsymbol{z}_s, \boldsymbol{x}_s)\}_{s=1}^{t-1})$; this is in fact instrumental in attaining a sublinear extended regret (see Theorems 6.3 and B.1 below).

We extend the definitions of the regret and the extended regret as follows:

$$\text{Regret}_T(\mathcal{A}, \Xi) = \sup_{\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_t\} \in \mathcal{R}_{R,h}^T} \left\{\mathbb{E}\left[\sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{z}_t)\right] - \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{z}_*)\right\}, \tag{25}$$

and

$$\text{E-Regret}_T(\mathcal{A}, \Xi) = \sup_{\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_t\} \in \mathcal{R}_{R,h}^T} \left\{\mathbb{E}\left[\sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{z}_t) + \text{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] - \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{z}_*)\right\}, \tag{26}$$

where the expectation is taken over the random choices of the rounding scheme Ξ, and

$$\boldsymbol{z}_* = \arg\min_{\boldsymbol{z} \in \mathcal{Z}} \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{z}) \tag{27}$$

is the optimal static integral cache state (in hindsight). By restricting our focus to such randomized policies, we obtain a regret that is equal to the fractional caching policy's regret. Formally, we have:

Proposition 6.2. *Any randomized caching policy constructed by an online policy $\mathcal{A}$ combined with a randomized rounding scheme Ξ has the same regret as $\mathcal{A}$, i.e.,* $\text{Regret}_T(\mathcal{A}, \Xi) = \text{Regret}_T(\mathcal{A})$, *given by* (25) *and* (2)*, respectively.*

The result follows from the linearity of the cost functions and the expectation operator; moreover, the static optimum can always be selected to be integral from the integrality of the capacity constraint and linearity of the objective function. The proof is provided in Appendix. B.2.

Proposition 6.2 thus implies that regret guarantees for a fractional policy $\mathcal{A}$ readily transfer to the integral regime, when coupled with rounding Ξ. Unfortunately, when considering the extended

[5]In this section, we assume the adversary is oblivious [26, Sec. 5.5], i.e., he selects the request process adversarially ahead of time, independently of the decisions of the online learner.

Algorithm 3 Online Rounding

procedure Online Rounding($\boldsymbol{x} \in \mathcal{X}$, $\xi \in [0,1]$)
  $\mathcal{I}_0 = \emptyset$
  for $i = 1, 2, \ldots, N$ **do**
    $\mathcal{I}_i \leftarrow \begin{cases} \mathcal{I}_{i-1} \cup \{i\} & \text{if} \quad \sum_{j=1}^{i} x_j \geq \xi + |\mathcal{I}_{i-1}|, \\ \mathcal{I}_{i-1} & \text{otherwise.} \end{cases}$
  end for
  return $\boldsymbol{z} \leftarrow \sum_{i \in I_N} \boldsymbol{e}_i$
end procedure

▷ In *online independent rounding*, Online Rounding is called with arguments $(\boldsymbol{x}_t, \xi_t)$, where $\boldsymbol{x}_t$ are provided by algorithm $\mathcal{A}$ and $\{\xi_t\}_{t=1}^{T-1}$ are i.i.d., sampled u.a.r. from $[0,1]$.
▷ In *online coupled rounding*, a ξ is sampled once u.a.r. from $[0,1]$; then, Online Rounding is called with arguments $(\boldsymbol{x}_t, \xi)$, i.e., using the same ξ for all $\boldsymbol{x}_t$ provided by algorithm $\mathcal{A}$.
▷ Both return an integral r.v. $\boldsymbol{z}_t$ s.t. $\mathbb{E}[\boldsymbol{z}_t] = \boldsymbol{x}_t$, with the expectation being over $\{\xi_t\}_{t=1}^{T-1}$ and ξ, respectively.

regret (Eq. (26)) instead, naïve rounding policies can arbitrarily evict and fetch objects to the cache causing large update costs (see Theorem 6.3). Thus, unless rounding is carefully designed, we may fail to have sublinear regret guarantees when accounting for update costs. In the next section, we show how a randomized rounding scheme Ξ can be selected to avoid incurring large update costs.

6.1 Rounding Schemes and Extended Regret

6.1.1 Online Independent Rounding. If we consider a fractional caching state $\boldsymbol{x}_t \in \mathcal{X}$, then a random integral caching state $\boldsymbol{z}_t \in \mathcal{Z}$ with the marginal $\mathbb{E}[\boldsymbol{z}_t] = \boldsymbol{x}_t$ exists and can be sampled in polynomial time (see, e.g., [8, 28, 39]). Thus, a rounding scheme Ξ can be constructed with such a strategy that takes as input the current fractional cache state $\boldsymbol{x}_t$ ignoring the previous fractional cache states $\{\boldsymbol{x}_s\}_{s=1}^{t-1} \in \mathcal{X}^{t-1}$, and previous random cache states $\{\boldsymbol{z}_s\}_{s=1}^{t-1} \in \mathcal{Z}^{t-1}$. We provide pseudocode for this procedure in Algorithm 3.[6] Because at any time t the random cache states are sampled independently from previous random cache states, we refer to this rounding as *online independent rounding*. Unfortunately, when considering the extended regret (26), any caching policy coupled with this rounding scheme loses its $O\left(\sqrt{T}\right)$ regret guarantee. Formally, we have the following:

Theorem 6.3. *Any randomized caching policy constructed by an online policy $\mathcal{A}$ combined with online independent rounding as a randomized rounding scheme Ξ has linear (worst-case) extended regret, i.e.,* $\text{E-Regret}_T(\mathcal{A}, \Xi) = \Omega(T)$.

The proof is provided in Appendix B.3. Online independent rounding causes frequent cache updates, as it samples a new state from $\boldsymbol{z}_t$ ignoring the previous state $\boldsymbol{\zeta}_{t-1}$ sampled from $\boldsymbol{z}_{t-1}$. Intuitively, imposing dependence (coupling) between the two consecutive random states may significantly reduce the expected update cost.

6.1.2 Online Coupled Rounding. To address this issue, our proposed *online coupled rounding* scheme is described also in Algorithm 3, using however the same randomization source across all timeslots. In particular, the coupling across states comes from the use of the same uniform random variable ξ. A consequence of this coupling is that the next integral state can be computed efficiently and leads to small movement costs. Note that Algorithm 3 does not necessarily find an optimal coupling, still it yields a sublinear update cost, and thus preserves the sublinearity of the regret. This is formally expressed in the following Theorem:

Theorem 6.4. *Consider a randomized caching policy constructed by an OMD policy $\mathcal{A}$ with sublinear regret (i.e., configured with a learning rate $\eta = \Theta\left(1/\sqrt{T}\right)$) combined with online coupled*

[6] Algorithm 3 provides a linear-time variant of the algorithms proposed in [8, 28]. The algorithm samples an integral caching state without constructing a distribution and its support. This sampling scheme is also known as Madow's sampling [39].

rounding Ξ *in Algorithm 3 (fixed* $\xi_t = \xi$ *for* $t \in [T]$*). The expected movement cost of the random integral cache states is* $\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] = O\left(\sqrt{T}\right)$*. Moreover, the extended regret is sublinear* $\text{E-Regret}_T(\mathcal{A}, \Xi) = O\left(\sqrt{T}\right)$.

We provide the proof in Appendix B.5. In summary, any OMD policy combined with online coupled rounding yields $O\left(\sqrt{T}\right)$ extended regret in the integral caching setting. The computational complexity of online coupled rounding is $O(N)$ (see also Fig. 10).

6.1.3 Online Optimally-Coupled Rounding. It is possible in general to reduce the update cost of online coupled rounding. In particular, minimizing the expected update cost over all joint distributions of the random variables $\boldsymbol{z}_t$ and $\boldsymbol{z}_{t+1}$ leads to an optimal transport problem [50]. For completeness, we describe this rounding scheme here, though (1) it does not reduce the extended regret guarantee attained by online coupled rounding (up to multiplicative constants), and (2) it has an increased computational cost.

Formally, at each time t the random variables $\boldsymbol{z}_t$ with marginal $\boldsymbol{x}_t$ can be constructed by sampling from a distribution $\boldsymbol{p}_t$ with $O(N)$ support $\left\{\boldsymbol{\zeta}_t^1, \boldsymbol{\zeta}_t^2, \ldots, \boldsymbol{\zeta}_t^{|\boldsymbol{p}_t|}\right\}$, where $p_{t,i} = \mathbb{P}(\boldsymbol{z}_t = \boldsymbol{\zeta}_t^i)$ for $i \in [|\boldsymbol{p}_t|]$. The decomposition can be performed in $O(kN\log(N))$ steps [28]. We denote the joint probability $\mathbb{P}\left(\boldsymbol{z}_{t+1} = \boldsymbol{\zeta}_{t+1}^j, \boldsymbol{z}_t = \boldsymbol{\zeta}_t^i\right)$ by the flow $f_{i,j}$ for all $(i,j) \in [|\boldsymbol{p}_t|] \times [|\boldsymbol{p}_{t+1}|]$. The optimal transport problem can be described by the following linear program:

$$\boldsymbol{f} = \underset{[f_{i,j}]_{(i,j)\in[|\boldsymbol{p}_t|]\times[|\boldsymbol{p}_{t+1}|]}}{\arg\min} \mathbb{E}\left[\mathrm{UC}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] = \sum_{i=1}^{|\boldsymbol{p}_t|}\sum_{j=1}^{|\boldsymbol{p}_{t+1}|} \mathrm{UC}_{\boldsymbol{r}_t}\left(\boldsymbol{\zeta}_t^i, \boldsymbol{\zeta}_{t+1}^j\right) f_{i,j}$$
$$\text{s.t.} \quad \sum_{j=1}^{|\boldsymbol{p}_{t+1}|} f_{i,j} = p_{t,i}, \quad \sum_{i=1}^{|\boldsymbol{p}_t|} f_{i,j} = p_{t+1,j}, \quad f_{i,j} \in [0,1], \forall (i,j) \in [|\boldsymbol{p}_t|] \times [|\boldsymbol{p}_{t+1}|].$$

We solve the above linear program to obtain a minimum-cost flow $\boldsymbol{f}$. If the random state at time t is $\boldsymbol{\zeta}_t^i$, then we select the new random state to be $\boldsymbol{\zeta}_{t+1}^j$ with (conditional) probability $\mathbb{P}\left(\boldsymbol{z}_{t+1} = \boldsymbol{\zeta}_{t+1}^j \mid \boldsymbol{z}_t = \boldsymbol{\zeta}_t^i\right) = \frac{f_{i,j}^*}{p_i^t}$. Such coupling ensures that the expected update cost is minimized. When we combine this rounding scheme with a no-regret fractional policy we obtain sublinear extended regret (26):

Corollary 6.5. *Consider an OMD policy* $\mathcal{A}$ *configured with learning rate* $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$ *combined with online optimally-coupled rounding* Ξ*. The obtained randomized integral caching policy has sublinear extended regret, i.e.,* $\text{E-Regret}(\mathcal{A}, \Xi) = O\left(\sqrt{T}\right)$.

The corollary follows from Theorem 6.4, because online coupled rounding constructs a feasible transportation flow (see Fig. 11 for an illustration) that gives sublinear update costs, and the optimal flow can only have lower update costs. The naïve implementation of the optimal transport problem has $O\left(N^3\right)$ time complexity, but several efficient approximations exist in the literature [50] at the expense of losing the established guarantee.

7 NUMERICAL EXPERIMENTS

7.1 Experimental setup

7.1.1 Datasets. Throughout all experiments, we assume equal costs per file, i.e., $w_i = w_i' = 1, \forall i \in \mathcal{N}$. The learning rate η^* denotes the learning rate value specified in Corollary 4.5 and in Theorem 4.10 for OGD and OMD_{NE}, respectively. Note that all the parameters assumed known to the algorithm can be learned through the following meta-algorithm: one can execute in parallel multiple OMD_{NE}

and OGD algorithms configured for different R/h values in $\{\Delta, 2\Delta, \ldots, N\}$ and frame an expert problem to learn the best expert (policy). The meta problem is a standard prediction with expert advice problem, and can be tackled with well understood and computationally efficient learning algorithms [26, 41, 53]. In what follows, we distinguish the number of batches in the trace (B) and the time horizon (T).

We generate the following synthetic datasets, summarized in Table 2.

Fixed Popularity. Requests are i.i.d. and sampled from a catalog of $N = 10^5$ files according to a *Zipf* distribution with exponent $\alpha = 0.8$. Each batch counts a single request ($R = 1$). We set set the time horizon as $T = 10^5$. The cache capacity is $k = 100$. The total number of requests is the product of the requests in each batch (R) and the number of batches (B), both values are reported in Table 2.

Batched Fixed Popularity. Request are generated as above from a Zipf distribution with exponent α, but are now grouped in batches of $R = 5 \times 10^3$ requests. We take different exponents $\alpha \in \{0.1, 0.2, 0.7\}$ for traces *Batched Fixed Popularity* (1), (2), and (3), respectively, in Table 2. The parameter α controls the diversity of the files in the request batches. If $\alpha = 0$, then each file is requested with equal probability, corresponding to $\frac{R}{h} \to N$ (high diversity). As we increase α, the requests become more concentrated; this corresponds to $\frac{R}{h} \to 1$ (low diversity). Table 2 shows the value of h observed in each trace. In all cases, we select catalog size $N = 10^4$, cache size $k \in \{25, 125, 250\}$, and time horizon $T = 10^4$.

Transient Popularity. We also generate two non-stationary request traces. In these traces, we reset the popularity distribution periodically.

In the first scenario (*Partial Popularity Change* traces), we still have batches of $R = 5 \times 10^3$ requests sampled from a catalog of $N = 10^4$ files according to a Zipf distribution with parameter $\alpha \in \{0.1, 0.3, 0.4\}$ for traces (1), (2), and (3), respectively. But now the popularities of a subset of files is modified every 10^3 time slots. In particular the 5% most popular files become the 5% least popular ones and vice versa. We want to model a situation where the cache knows the timescale over which the request process changes and which files are affected (but not how their popularity changes). Correspondingly, the time horizon is also set to $T = 10^3$ and, at the end of each time horizon, the cache redistributes uniformly the cache space currently allocated by those files. The cache size is $k = 50$.

In the second scenario (*Global Popularity Change* trace) each batch counts only a single request ($R = 1$) sampled from a catalog of $N = 10^4$ files according to a Zipf distribution with exponent $\alpha = 0.8$. Every 5×10^4 time slots (or requests in this case) the popularity of each files change: file $i \in \{1, \ldots, N\}$ assumes the popularity of file $j = (1 + (i + N/4) \mod N)$. The cache size is $k = 200$. We also generate the *Downscaled Global Popularity Change* trace as a downscaled version of *Global Popularity Change* trace, where the catalog size is reduced to $N = 25$, the cache size to $k = 4$, and the number of requests to 9×10^3. The learning rate is set to $\eta = 0.01$.

Akamai Trace. We consider also a real file request trace collected from Akamai Content Delivery Network (CDN) [43]. The trace spans 1 week, and we extract from it about 8.5×10^7 requests for the $N = 10^4$ most popular files. We group requests in batches of size $R = 5 \times 10^3$, and we consider a time horizon $T = 100$ time slots corresponding roughly to 1 hour. The cache size is $k = 25$.

7.1.2 Online Algorithms. Starting with the gradient based algorithms, we implemented OMD_{NE} with the projection defined in Algorithm 2. We implemented two different projection algorithms for OGD: the one by Paschos et al. [49] for the setting $\frac{R}{h} = 1$, and the one by Wang and Lu [59] for the general setting $\frac{R}{h} > 1$.

In addition, we implemented four caching eviction policies: LRU, LFU, W-LFU, and FTPL. LRU and LFU evict the least recently used and least frequently used file, respectively. While LFU estimates file popularities considering all requests seen in the past, W-LFU [31] is an LFU variant that only

Trace	B	T	N	R	h
Fixed Popularity	1.5×10^5	1.5×10^5	10^4	1	1
Batched Fixed Popularity (1)	10^4	10^4	10^4	5×10^3	2
Batched Fixed Popularity (2)	10^4	10^4	10^4	5×10^3	5
Batched Fixed Popularity (3)	10^4	10^4	10^4	5×10^3	87
Partial Popularity Change (1)	5×10^3	10^3	10^4	5×10^3	2
Partial Popularity Change (2)	5×10^3	10^3	10^4	5×10^3	6
Partial Popularity Change (3)	5×10^3	10^3	10^4	5×10^3	10
Global Popularity Change	1.5×10^5	1.5×10^5	10^4	1	1
Downscaled Global Popularity Change	9×10^3	9×10^3	25	1	1
Akamai CDN	1.7×10^4	10^2	10^3	5×10^4	380

Table 2. Trace Summary

Performance metric	Definition	Range
Normalized Average Cost	$\mathrm{NAC}(\mathcal{A}) = \frac{1}{Rt} \sum_{s=0}^{t} f_{\boldsymbol{r}_s}(\boldsymbol{x}_s)$	$[0, 1]$
Normalized Moving Average Cost	$\mathrm{NMAC}(\mathcal{A}) = \frac{1}{R \min(\tau, t)} \sum_{s=t-\min(\tau,t)}^{t} f_{\boldsymbol{r}_s}(\boldsymbol{x}_s)$	$[0, 1]$
Time Average Regret	$\mathrm{TAR}(\mathcal{A}) = \frac{1}{t} \left(\sum_{s=1}^{t} f_{\boldsymbol{r}_s}(\boldsymbol{x}_s) - \sum_{s=1}^{t} f_{\boldsymbol{r}_s}(\boldsymbol{x}_*) \right)$	$[0, R]$
Cumulative Update Cost	$\mathrm{CUC}(\mathcal{A}) = \sum_{s=1}^{t} \mathrm{UC}_{\boldsymbol{r}_s}(\boldsymbol{x}_s, \boldsymbol{x}_{s+1})$	$[0, \infty)$

Table 3. Performance Metrics. All are better if lower.

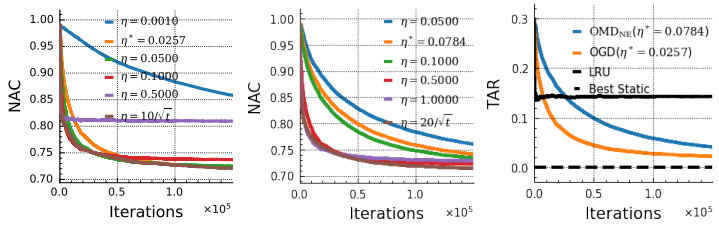
considers requests during a recent time window W, which we set equal to $T \times R$ in our experiments. The policies LRU, LFU, and W-LFU are allowed to process individual requests. FTPL is a no-regret policy proposed by Mukhopadhyay and Sinha [42], which, roughly speaking, behaves as a LFU policy whose request counters are perturbed by some Gaussian noise. Finally, we define *Best Static* to be the optimal static allocation $\boldsymbol{x}^*$, i.e., the configuration storing the k most popular files as we consider $w_i = 1, \forall i \in \mathcal{N}$. We also define *Best Dynamic* to be the caching policy that stores the k most popular files at any time for the synthetic traces (for which the instantaneous popularity is well defined). The optimality of such policy is formally studied in [45].

7.1.3 Online Rounding. We also implemented the three rounding schemes described in Sec. 6: (a) the online independent rounding in Algorithm 3, (b) the online coupled rounding in Algorithm 3, and (c) the online optimally-coupled rounding. The rounding schemes are combined with OGD configured with learning rate $\eta = 0.01$ under the *Downscaled Global Popularity Change* trace.

7.1.4 Performance Metrics. We measure performance w.r.t. four metrics defined in Table 3. The Normalized Average Cost $\mathrm{NAC}(\mathcal{A}) \in [0, 1]$ corresponds to the time-average cost over the first t time slots, normalized by the batch size R. The Normalized Moving Average Cost $\mathrm{NMAC}(\mathcal{A}) \in [0, 1]$ is computed similarly, using a moving average instead over a time window $\tau > 0$; we use $\tau = 500$ in our experiments. We also consider the Time Average Regret $\mathrm{TAR}(\mathcal{A}) \in [0, R]$, which is precisely the time average regret over the first t time slots. Finally, when studying rounding algorithms, we also measure and report the Cumulative Update Cost $\mathrm{CUC}(\mathcal{A}) \in [0, \infty)$.

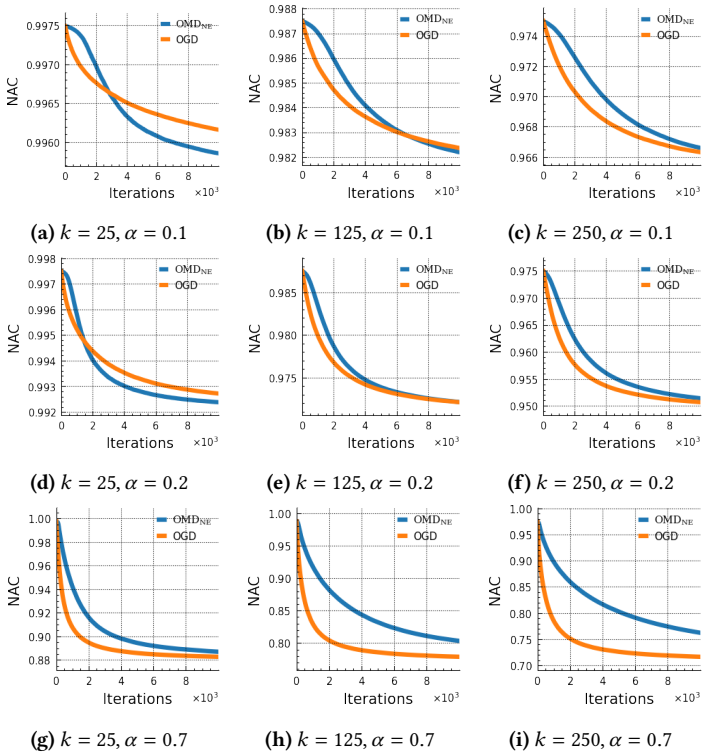
7.2 Results

7.2.1 Stationary Requests. Figures 2 (a) and 2 (b) show the performance w.r.t. NAC of OGD and $\mathrm{OMD}_{\mathrm{NE}}$, respectively, under different learning rates η on the *Fixed Popularity* trace. We observe that both algorithms converge slower under small learning rates, but reach a final lower cost, while larger learning rates lead to faster convergence, albeit to higher final cost. This may motivate the adoption of a diminishing learning rate, that combines the best of the two options, starting large to enable fast convergence, and enabling eventual fine-tuning (as it is also advocated by the theory of stochastic approximation [52]). We show curves corresponding to a diminishing learning rate both for OGD and $\mathrm{OMD}_{\mathrm{NE}}$, and indeed they achieve the smallest costs. The learning rate η^* gives



(a) NAC of OGD **(b)** NAC of OMD_{NE} **(c)** Time-Average Regret

Fig. 2. NAC of the different caching policies over the *Fixed Popularity* trace. Subfigures (a) and (b) show the performance of OGD and OMD_{NE} respectively under different learning rates. For small learning rates the algorithms both converge slower but more precisely, while for larger learning rates they converge faster, but to a higher cost. Subfigure (c) shows the time average regret of the two gradient algorithms. When the regret is sub-linear, the time average regret converges to 0 as $T \to \infty$.



(a) $k = 25, \alpha = 0.1$ **(b)** $k = 125, \alpha = 0.1$ **(c)** $k = 250, \alpha = 0.1$

(d) $k = 25, \alpha = 0.2$ **(e)** $k = 125, \alpha = 0.2$ **(f)** $k = 250, \alpha = 0.2$

(g) $k = 25, \alpha = 0.7$ **(h)** $k = 125, \alpha = 0.7$ **(i)** $k = 250, \alpha = 0.7$

Fig. 3. NAC of OMD_{NE} and OGD evaluated under different cache sizes and diversity regimes. We use traces *Batched Fixed Popularity* (1), (2), and (3) corresponding to different diversity regimes. Figures from left to right correspond to different cache sizes $k \in \{25, 125, 250\}$, while figures from top to bottom correspond to different exponent values $\alpha \in \{0.1, 0.2, 0.7\}$. OMD_{NE} outperforms OGD in the more diverse regimes and for small cache sizes, while OGD outperforms for large cache sizes and concentrated requests.

the tightest worst-case regrets for OGD and OMD_{NE}, as stated in Theorems 4.5 and 4.10. While this learning rate is selected to protect against any (adversarial) request sequence, it is not too pessimistic: Figures 2 (a) and 2 (b) show it performs well when compared to other learning rates.

Figure 2 (c) shows the time-average regret TAR of OGD and OMD_{NE} over the *Fixed Popularity* trace. As both algorithms have sub-linear regret, their time average regret goes to 0 for $T \to \infty$. Note how instead LRU exhibits a constant time average regret.

7.2.2 Effect of Diversity. Figure 3 shows the NAC performance of OMD_{NE} and OGD on the traces *Batched Fixed Popularity* (1), (2), and (3) under different cache capacities k and exponent values α. We observe that OMD_{NE} outperforms OGD in the more diverse regimes ($\alpha \in \{0.1, 0.2\}$). This is

more apparent for smaller cache sizes k. In contrast, OGD outperforms OMD_{NE} when requests are less diverse ($\alpha = 0.7$); again, this is more apparent for larger cache size k. These observations agree with Theorems 4.8 and 4.7 in Sec. 4.4.2: high diversity and small cache sizes indeed favor OMD_{NE}.

7.2.3 Robustness to Transient Requests. Figure 4 shows the normalized average cost of OMD_{NE} and OGD over the *Partial Popularity Change* traces, evaluated under different diversity regimes. Dashed lines indicate the projected performance in the stationary setting (if request popularities stay fixed). Across the different diversity regimes, we find the OMD_{NE} is more robust to popularity changes. In (a), (b) and (c) OMD_{NE} outperforms OGD in the non-stationary popularity setting: we observe a wider performance gap as compared to the stationary setting.

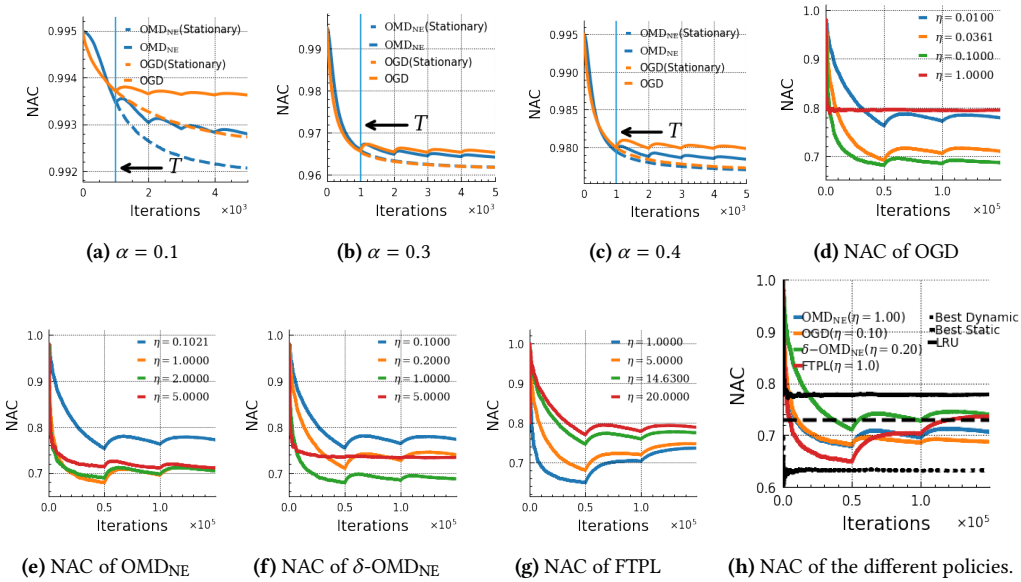
Figure 4 (d) and (e) show the normalized average cost over the *Global Popularity Change* trace for the policies OGD and OMD_{NE}, respectively. We observe in Figure 4 (b) the NAC of OMD_{NE} performance degrades after each popularity change. This is a limitation due to the multiplicative nature of OMD_{NE}. When the algorithm learns that a file, say it i, is not important, it can set $x_{t,i}$ arbitrarily close to 0. If, suddenly, this content becomes popular, then OMD_{NE} adapts slowly, due to its multiplicative nature—remember Eq. (20). This is shown in Figure 4 (e). We can overcome this limitation by requiring all state variables to be larger than some small $\delta > 0$; OMD_{NE} is then limited to $\mathcal{X}_\delta$, the δ interior of the capped simplex $\mathcal{X}$. More precisely, the δ interior of the capped simplex is defined as $\mathcal{X}_\delta \triangleq \mathcal{X} \cap [\delta, 1]^N$. In Figure 4 (f), we use $\delta = 10^{-4}$. This parameter indeed prevents the algorithm from driving the fractional allocations arbitrary close to 0, improving its adaptability. In Figure 4, we show the performance of FTPL [7]; we observe that this policy fails to adapt to popularity changes. Both our mirror descent algorithms outperform competitors (Fig. 4(h)).

7.2.4 Akamai Trace. Figure 5 shows that the two gradient algorithms, OMD_{NE} and OGD, perform similarly over the Akamai Trace w.r.t. NMAC; OGD is slightly better in parts of the trace. Overall, these algorithms consistently outperform LFU, W-LFU, LRU, and FTPL. Note that these caching policies process requests individually, while OMD_{NE} and OGD adapt slower, *freezing* their state for the entire batch size ($R = 5000$). Nevertheless, OMD_{NE} and OGD still perform better. Despite the observation that OMD_{NE} falls behind OGD in some parts of the trace, note that in many scenarios OMD_{NE} remains an attractive choice because it provides similar performance to OGD at a much lower computation cost (cf. Theorems 4.9 and 4.11).

7.2.5 Randomized Rounding. Figure 6 shows the cumulative update cost for the online independent rounding, the online coupled rounding, and the online optimally-coupled rounding algorithms over the *Downscaled Global Popularity Change* trace. All the rounding algorithms exhibit the same service cost in expectation. The update cost of online coupled rounding and the online optimally-coupled rounding is small, in the order of the learning rate η; moreover, we observe that online optimally-coupled rounding yields lower update costs than the online coupled rounding. In contrast, online independent rounding incurs a significantly larger update cost.

Figure 7 shows the fractional and (rounded) integral cache states under *Downsampled Global Popularity Change* trace. Online independent rounding indeed leads to more frequent updates than online coupled rounding, while the latter maintains a more stable cache configuration by coupling the consecutive states and avoiding unnecessary updates.

7.2.6 Computational Cost. Figure 8 shows the time taken by both policies OMD and OMD_{NE} to perform 500 iterations over the *Fixed Popularity* trace (Fig. 8 (a)), and the time taken to perform 50 iterations over the *Batched Fixed Popularity (2)* trace (Fig. 8 (b)). We observe that OMD_{NE} is at least 15 times faster in computing cache states on average.



(a) $\alpha = 0.1$ **(b)** $\alpha = 0.3$ **(c)** $\alpha = 0.4$ **(d)** NAC of OGD

(e) NAC of $\mathrm{OMD}_{\mathrm{NE}}$ **(f)** NAC of δ-$\mathrm{OMD}_{\mathrm{NE}}$ **(g)** NAC of FTPL **(h)** NAC of the different policies.

Fig. 4. Subfigures (a)–(c) show NAC of OGD and $\mathrm{OMD}_{\mathrm{NE}}$ evaluated under different diversity regimes when 10% of the files change popularity over time. We use traces *Partial Popularity Change* (1), (2), and (3) corresponding to the different diversity regimes. The diversity regimes are selected, such that, in the stationary setting (dashed line): (a) $\mathrm{OMD}_{\mathrm{NE}}$ outperforms OGD, (b) $\mathrm{OMD}_{\mathrm{NE}}$ has similar performance to OGD and (c) $\mathrm{OMD}_{\mathrm{NE}}$ performs slightly worse than OGD. $\mathrm{OMD}_{\mathrm{NE}}$ is consistently more robust to partial popularity changes than OGD. Subfigures (d)–(h) show the NAC of the different caching policies evaluated on the *Global Popularity Change* trace, where file popularity changes every 5×10^4 iterations. While OGD adapts seamlessly to popularity changes (d), multiplicative updates can make $\mathrm{OMD}_{\mathrm{NE}}$ less reactive (e), unless $\mathrm{OMD}_{\mathrm{NE}}$ is forced to operate over the δ-interior of $\mathcal{X}$ (f) ($\delta = 10^{-4}$). Finally, our mirror descent policies outperform competitors (h).

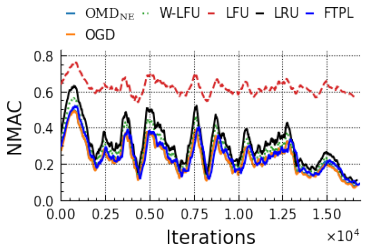
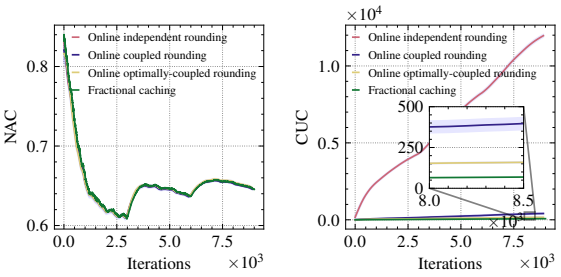


Fig. 5. NMAC of the different caching policies evaluated on the *Akamai Trace*. $\mathrm{OMD}_{\mathrm{NE}}$ and OGD provide consistently the best performance compared to W-LFU, LFU, LRU, and FTPL. OGD performs slightly better than OMD in some parts of the trace.

8 CONCLUSIONS

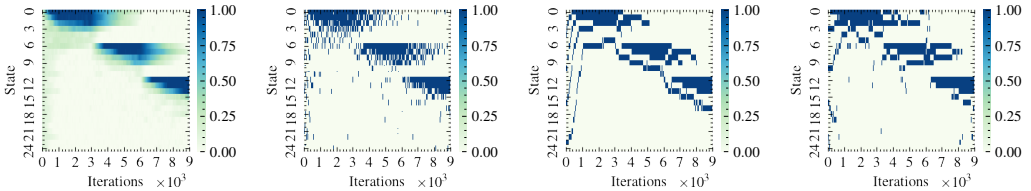
We study no-regret caching algorithms based on OMD with q-norm and neg-entropy mirror maps. We find that batch diversity impacts regret performance; a key finding is that OGD is optimal in low-diversity regimes, while $\mathrm{OMD}_{\mathrm{NE}}$ is optimal under high diversity. With an appropriately designed rounding scheme, our $O\left(\sqrt{T}\right)$ bound on the regret for general OMD algorithms extends to integral caches as well, despite the need to account for update costs in this setting.

Our numerical experiments indicate that the gap between the regimes in which OGD and $\mathrm{OMD}_{\mathrm{NE}}$ are optimal, w.r.t. the diversity ratio, is narrow; this suggests that our characterization of the two regimes can be further improved. Also $\mathrm{OMD}_{q\text{-norm}}$ algorithms for arbitrary values of $q \in (1, 2)$ deserves more investigation to 1) devise strongly polynomial, efficient algorithms for their Bregman projection, 2) characterize their update costs, and 3) compare their performance with $\mathrm{OMD}_{\mathrm{NE}}$.



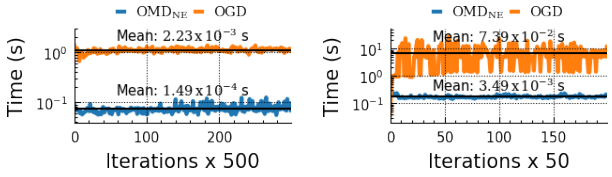
(a) Normalized average cost **(b)** Cumulative update cost

Fig. 6. Costs associated with the rounded integral caching over the *Downscaled Global Popularity Change* trace. The normalized average costs shown in (a) are the same for the online independent rounding, the online coupled rounding and the online optimally-coupled rounding. The cumulative update cost of the online coupled rounding and online optimally-coupled rounding in (b) is of the same order as in the fractional setting, while the online independent rounding in (b) gives a much higher update cost. The reported values are averaged over 20 experiments, and the blue shaded area represents 10% scaling of the standard deviation.



(a) Fractional cache states **(b)** Online independent rounding **(c)** Online coupled rounding **(d)** Online optimally-coupled rounding

Fig. 7. Online rounding of fractional caching states. Visually, we see that the online independent rounding has more frequent updates than the online coupled rounding. This leads to large update costs. The online coupled rounding and the online optimally-coupled rounding prevents the cache to perform unnecessary updates.



(a) $R/h = 1$ **(b)** $R/h > 1$

Fig. 8. Runtime per iteration of OMD_{NE} and OGD. Subfigure (a) shows the runtime per 500 iterations over the *Fixed Popularity* trace. Subfigure (b) shows the runtime per 50 iterations over the *Batched Fixed Popularity (2)* trace.

Acknowledgements. This research was supported in part by the French Government through the "Plan de Relance" and "Programme d'investissements d'avenir" and by Inria under the exploratory action MAMMALS. The authors gratefully acknowledge support from the National Science Foundation (grants 2107062 and 2112471).

REFERENCES

[1] Sara Alouf, Nicaise Choungmo Fofack, and Nedko Nedkov. 2016. Performance Models for Hierarchy of Caches: Application to Modern DNS Caches. *Performance Evaluation* 97 (2016), 57–82.

[2] Lachlan Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. 2013. A Tale of Two Metrics: Simultaneous Bounds on Competitiveness and Regret. *SIGMETRICS Performance Evaluation Review* 41, 1 (June 2013), 329–330.

[3] AWS. 2021. Amazon Web Service ElastiCache. https://aws.amazon.com/elasticache/

[4] Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. 2012. A Primal-Dual Randomized Algorithm for Weighted Paging. *Journal of the ACM (JACM)* 59, 4, Article 19 (Aug. 2012).

[5] Amir Beck and Marc Teboulle. 2003. Mirror Descent and Nonlinear Projected Subgradient Methods for Convex Optimization. *Operations Research Letters* 31, 3 (2003), 167–175.

[6] Daniel S. Berger, Philipp Gland, Sahil Singla, and Florin Ciucu. 2014. Exact Analysis of TTL Cache Networks. *Performance Evaluation* 79 (2014), 2–23.

[7] Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. 2020. Fundamental Limits on the Regret of Online Network-Caching. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2, Article 25 (June 2020).

[8] B. Blaszczyszyn and A. Giovanidis. 2015. Optimal Geographic Caching In Cellular Networks. In *ICC*. 3358–3363.

[9] Allan Borodin, Nathan Linial, and Michael E. Saks. 1992. An Optimal On-Line Algorithm for Metrical Task System. *Journal of the ACM (JACM)* 39, 4 (Oct. 1992), 745–763.

[10] Sem Borst, Varun Gupta, and Anwar Walid. 2010. Distributed Caching Algorithms for Content Distribution Networks. In *2010 Proceedings IEEE INFOCOM*. IEEE, 1–9.

[11] Sébastien Bubeck. 2015. Convex Optimization: Algorithms and Complexity. *Foundations and Trends in Machine Learning* 8, 3–4 (Nov. 2015), 231–357.

[12] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Mądry. 2018. K-Server via Multiscale Entropic Regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing* (Los Angeles, CA, USA) *(STOC 2018)*. Association for Computing Machinery, New York, NY, USA, 3–16.

[13] Nicolo Cesa-Bianchi and Gabor Lugosi. 2006. *Prediction, Learning, and Games*. Cambridge University Press, USA.

[14] Hao Che, Ye Tung, and Z. Wang. 2002. Hierarchical Web Caching Systems: Modeling, Design and Experimental Results. *IEEE Journal on Selected Areas in Communications* (Sep 2002).

[15] Christophe Chesneau and Yogesh J. Bagul. 2020. New Sharp Bounds for the Logarithmic Function. *Electronic Journal of Mathematical Analysis and Applications* 8, 1 (2020), 140–145.

[16] Weibo Chu, Mostafa Dehghan, John C.S. Lui, Don Towsley, and Zhi-Li Zhang. 2018. Joint Cache Resource Allocation and Request Routing for In-network Caching Services. *Computer Networks* 131 (2018), 1–14.

[17] Edward Grady Coffman and Peter J. Denning. 1973. *Operating Systems Theory*. Vol. 973. Prentice-Hall, Inc.

[18] Mostafa Dehghan, Laurent Massoulie, Don Towsley, Daniel Sadoc Menasche, and Y. C. Tay. 2019. A Utility Optimization Approach to Network Cache Design. *IEEE/ACM Transactions on Networking* 27, 3 (June 2019), 1013–1027.

[19] Ronald Fagin. 1977. Asymptotic Miss Ratios over Independent References. *J. Comput. System Sci.* 14, 2 (1977), 222–250.

[20] Philippe Flajolet, Danièle Gardy, and Loÿs Thimonier. 1992. Birthday Paradox, Coupon Collectors, Caching Algorithms and Self-Organizing Search. *Discrete Applied Mathematics* 39, 3 (1992), 207–229.

[21] Nicaise Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. 2014. Performance Evaluation of Hierarchical TTL-based Cache Networks. *Computer Networks* 65 (2014), 212–231.

[22] Christine Fricker, Philippe Robert, and James Roberts. 2012. A Versatile and Accurate Approximation for LRU Cache Performance. In *Proceedings of the 24th International Teletraffic Congress*. 8.

[23] Michele Garetto, Emilio Leonardi, and Valentina Martina. 2016. A Unified Approach to the Performance Analysis of Caching Systems. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 1, 3, Article 12 (May 2016), 12:1–12:28 pages.

[24] Nicolas Gast and Benny Van Houdt. 2017. TTL Approximations of the Cache Replacement Algorithms LRU (m) and h-LRU. *Performance Evaluation* 117 (2017), 33–57.

[25] Claudio Gentile and Nick Littlestone. 1999. The Robustness of the p-Norm Algorithms. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory* (Santa Cruz, California, USA) *(COLT '99)*. Association for Computing Machinery, New York, NY, USA, 1–11.

[26] Elad Hazan. 2016. Introduction to Online Convex Optimization. *Foundations and Trends® in Optimization* 2, 3–4 (Aug. 2016), 157–325.

[27] Stratis Ioannidis, Laurent Massoulié, and Augustin Chaintreau. 2010. Distributed Caching over Heterogeneous Mobile Networks. In *Proceedings of the ACM SIGMETRICS*. 311–322.

[28] Stratis Ioannidis and Edmund Yeh. 2016. Adaptive Caching Networks with Optimality Guarantees. *SIGMETRICS Performance Evaluation Review* 44, 1 (2016), 113–124.

[29] Predrag R. Jelenkovic. 1999. Asymptotic Approximation of the Move-to-Front Search Cost Distribution and Least-Recently Used Caching Fault Probabilities. *The Annals of Applied Probability* 9, 2 (1999), 430–464.

[30] Bo Jiang, Philippe Nain, and Don Towsley. 2018. On the Convergence of the TTL Approximation for an LRU Cache under Independent Stationary Request Processes. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 3, 4 (2018).

[31] George Karakostas and Dimitrios N. Serpanos. 2002. Exploitation of Different Types of Locality for Web Caches. In *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications*. IEEE, 207–212.

[32] W. F. King. 1972. Analysis of Paging Algorithms. In *Proceedings of the IFIP congress on Information Processing*, Vol. 71. 485–490.

[33] Krzysztof C. Kiwiel. 1997. Proximal Minimization Methods with Generalized Bregman Functions. *SIAM Journal on Control and Optimization* 35, 4 (1997), 1142–1168.

[34] Elias Koutsoupias. 2009. The k-server Problem. *Computer Science Review* 3, 2 (May 2009), 105–118.
[35] Emilio Leonardi and Giovanni Neglia. 2018. Implicit Coordination of Caches in Small Cell Networks Under Unknown Popularity Profiles. *IEEE Journal on Selected Areas in Communications* 36, 6 (June 2018), 1276–1285.
[36] Yuanyuan Li, Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2021. Online Caching Networks with Adversarial Guarantees. 5, 3, Article 35 (dec 2021), 39 pages.
[37] Jun-Lin Lin. 2013. On the Diversity Constraints for Portfolio Optimization. *Entropy* 15, 11 (2013), 4607–4621.
[38] N. Littlestone and M. K. Warmuth. 1994. The Weighted Majority Algorithm. *Information and computation* 108, 2 (1994), 212–261.
[39] William G. Madow and Lillian H. Madow. 1944. On the Theory of Systematic Sampling. *Annals of Mathematical Statistics* 15, 1 (March 1944), 1–24.
[40] Mark Manasse, Lyle McGeoch, and Daniel Sleator. 1988. Competitive Algorithms for On-Line Problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, USA) *(STOC '88)*. Association for Computing Machinery, New York, NY, USA, 322–333.
[41] H Brendan McMahan. 2017. A survey of algorithms and analysis for adaptive online learning. *The Journal of Machine Learning Research* 18, 1 (2017), 3117–3166.
[42] Samrat Mukhopadhyay and Abhishek Sinha. 2021. Online Caching with Optimal Switching Regret. In *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 1546–1551.
[43] Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. 2017. Access-Time-Aware Cache Algorithms. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 2, 4, Article 21 (Nov. 2017).
[44] Giovanni Neglia, Damiano Carra, and Pietro Michiardi. 2018. Cache Policies for Linear Utility Maximization. *IEEE/ACM Transactions on Networking* 26, 1 (Feb. 2018), 302–313.
[45] Nitish K. Panigrahy, Philippe Nain, Giovanni Neglia, and Don Towsley. 2021. A New Upper Bound on Cache Hit Probability for Non-Anticipative Caching Policies. *SIGMETRICS Performance Evaluation Review* 48, 3 (March 2021), 138–143.
[46] Rodrigo Paredes and Gonzalo Navarro. 2006. Optimal Incremental Sorting. In *2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 171–182.
[47] Debjit Paria and Abhishek Sinha. 2021. LeadCache: Regret-Optimal Caching in Networks. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 4435–4447.
[48] Georgios Paschos, George Iosifidis, Giuseppe Caire, et al. 2020. Cache Optimization Models and Algorithms. *Foundations and Trends® in Communications and Information Theory* 16, 3–4 (2020), 156–345.
[49] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis. 2019. Learning to Cache With No Regrets. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 235–243.
[50] Gabriel Peyré, Marco Cuturi, et al. 2019. Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning* 11, 5-6 (2019), 355–607.
[51] Konstantinos Poularakis, George Iosifidis, Vasilis Sourlas, and Leandros Tassiulas. 2016. Exploiting Caching and Multicast for 5G Wireless Networks. *IEEE Transactions on Wireless Communications* 15, 4 (2016), 2995–3007.
[52] Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics* 22, 3 (1951), 400–407.
[53] Shai Shalev-Shwartz. 2012. Online Learning and Online Convex Optimization. *Foundations and Trends in Machine Learning* 4, 2 (Feb. 2012), 107–194.
[54] Shai Shalev-Shwartz and Yoram Singer. 2007. *Online Learning: Theory, Algorithms, and Applications*. Ph. D. Dissertation. Hebrew University.
[55] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire. 2013. FemtoCaching: Wireless Content Delivery Through Distributed Caching Helpers. *IEEE Transactions on Information Theory* 59, 12 (2013), 8402–8413.
[56] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2021. No-Regret Caching via Online Mirror Descent. In *ICC 2021 - IEEE International Conference on Communications*. 1–6.
[57] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM* 28, 2 (Feb. 1985), 202–208.
[58] Stefano Traverso et al. 2013. Temporal Locality in Today's Content Caching: Why It Matters and How to Model It. *ACM SIGCOMM Computer Communication Review* 43, 5 (Nov. 2013), 5–12.
[59] Weiran Wang and Canyi Lu. 2015. Projection onto the Capped Simplex. *preprint arXiv:1503.01002* (2015).
[60] Martin Zinkevich. 2003. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning* (Washington, DC, USA) *(ICML'03)*. AAAI Press, 928–935.

A FRACTIONAL CACHING AND GRADIENT-BASED ALGORITHMS

A.1 Online Mirror Descent

THEOREM A.1. *([11, Theorem 4.2]) Let (1) the map $\Phi : \mathcal{D} \to \mathbb{R}$ be a mirror map (see Sec. 4.2) ρ-strongly convex w.r.t a norm $\|\cdot\|$ over $\mathcal{S} \cap \mathcal{D}$ ($\mathcal{S}$ is a convex set), (2) the cost functions $f_t : \mathcal{S} \to \mathbb{R}$ be convex with bounded gradients (i.e., $\|\nabla f_t(\boldsymbol{x})\|_* \leq L, \forall \boldsymbol{x} \in \mathcal{S}$) for every $t \in [T]$, where $\|\cdot\|_*$ is the dual norm of $\|\cdot\|$, (3) and the Bregman divergence $D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1)$ be bounded by D^2 for $\boldsymbol{x} \in \mathcal{S}$ where $\boldsymbol{x}_1 = \arg\min_{\boldsymbol{x} \in \mathcal{S} \cap \mathcal{D}} \Phi(\boldsymbol{x})$. Then Algorithm 1 satisfies*

$$\sum_{t=1}^{T} f_t(\boldsymbol{x}_t) - \sum_{t=1}^{T} f_t(\boldsymbol{x}) \leq D^2/\eta + \eta L^2/(2\rho T), \qquad \text{for } \boldsymbol{x} \in S. \tag{28}$$

We remark that Theorem A.1 is expressed differently in [11], where $f_t = f, \forall t \in [T]$ (fixed cost function). Nonetheless, as observed in [11, Sec. 4.6] the bound obtained in Eq. (28) holds as long as the dual norms of the gradients are bounded by L.

A.2 Proof of Theorem 4.4

The map $\Phi(x) = \frac{1}{2}\|x\|_q^2, q \in (1, 2]$ is $\rho = q - 1$ strongly convex w.r.t $\|\cdot\|_q$ over $\mathcal{D} = \mathbb{R}^N$ a direct result from [54, Lemma 17], and the dual norm of $\|\cdot\|_q$ is $\|\cdot\|_p$ (Hölder's inequality).Take $\mathcal{S} = \mathcal{X}$. The minimum value of $\Phi(x)$ over $\mathcal{X}$ is achieved when we spread the capacity mass k over the decision variable, i.e., $x_i = \frac{k}{N}, i \in \mathcal{N}$. If we select $\boldsymbol{x}_1$ to be the minimizer of $\Phi(\boldsymbol{x})$, then we have $\nabla\Phi^T(\boldsymbol{x}_1)(\boldsymbol{x} - \boldsymbol{x}_1) \geq 0, \forall \boldsymbol{x} \in \mathcal{X}$ [26, Theorem 2.2], so we obtain $D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1) = \Phi(\boldsymbol{x}) - \Phi(\boldsymbol{x}_1) - \nabla\Phi(\boldsymbol{x}_1)^T(\boldsymbol{x} - \boldsymbol{x}_1) \leq \Phi(\boldsymbol{x}) - \Phi(\boldsymbol{x}_1)$. Moreover, it is easy to check that Φ is maximized at a sparse point $\boldsymbol{x}_* \in \mathcal{X} \cap \{0, 1\}^N$; thus, we have $D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1) \leq \Phi(\boldsymbol{x}_*) - \Phi(\boldsymbol{x}_1)$. By replacing $\boldsymbol{x}_1$ and $\boldsymbol{x}_*$ with their values in the previous equation we get $\Phi(\boldsymbol{x}_1) = \frac{1}{2}\left(\left(\frac{k}{N}\right)^q N\right)^{\frac{2}{q}} = \frac{1}{2}k^2 N^{-\frac{2}{p}}$, and $\Phi(\boldsymbol{x}_*) = \frac{1}{2}k^{\frac{2}{q}} = \frac{1}{2}k^2 k^{-\frac{2}{p}}$. Thus, we have $D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1) \leq \frac{1}{2}k^2\left(k^{-\frac{2}{p}} - N^{-\frac{2}{p}}\right) = D^2$. Note that the maximum of $\|\boldsymbol{r}\|_p$ is achieved when $\frac{R}{h}$ components are set to h, then the following bound holds on the gradients $\max_{r \in \mathcal{R}} \|\nabla f_{\boldsymbol{r}}(\boldsymbol{x})\|_p \leq \max_{\boldsymbol{r} \in \mathcal{R}} \|\boldsymbol{w}\|_\infty \|\boldsymbol{r}\|_p = \|\boldsymbol{w}\|_\infty h\left(\frac{R}{h}\right)^{\frac{1}{p}} = L_p$. The gradients are bounded in the dual norm $\|\nabla f_r(\boldsymbol{x}_t)\|_p \leq L_p, \forall \boldsymbol{r} \in \mathcal{R}$. The final bound follows by Theorem A.1, plugging the constants D and L in (28), and selecting the learning rate that achieves the tightest bound $\eta = \sqrt{(q-1)k^2\left(k^{-\frac{2}{p}} - N^{-\frac{2}{p}}\right) / \left(\|w\|_\infty^2 h^2 \left(\frac{R}{h}\right)^{\frac{2}{p}} T\right)}$. □

A.3 Proof of Corollary 4.6

Taking $\alpha = \frac{k}{N}$ and $\beta = \frac{Nh}{R}$, we can rewrite (12) to have $\text{Regret}_T(\text{OMD}_{q\text{-norm}}) \leq \|\boldsymbol{w}\|_\infty R\beta^{\frac{1}{q}}\sqrt{\frac{\alpha^{2/q} - \alpha^2}{q-1}T}$. We take the limit $q \to 1$, to obtain the upper bound

$$\text{Regret}_T(\text{OMD}_{1\text{-norm}}) \leq \lim_{q \to 1} \|\boldsymbol{w}\|_\infty R\beta^{\frac{1}{q}}\sqrt{\frac{\alpha^{2/q} - \alpha^2}{q-1}T} = \|\boldsymbol{w}\|_\infty R\beta\sqrt{\left[\alpha^{2/q}\right]'_{q=1} T}$$

$$= \|\boldsymbol{w}\|_\infty R\beta\sqrt{\left[-2q^{-2}\alpha^{2/q}\log(\alpha)\right]_{q=1} T} = \|\boldsymbol{w}\|_\infty R\alpha\beta\sqrt{2\log(\alpha^{-1})T} = \|\boldsymbol{w}\|_\infty hk\sqrt{2\log\left(\frac{N}{k}\right)T}. \quad \square$$

A.4 Proof of Theorem 4.7

We take the simplified version of the regret of the general class of q-norm mirror maps in Eq. (12), select $\alpha = \frac{k}{N}$ and $\beta = \frac{Nh}{R}$, so we get $\text{Regret}_T(\text{OMD}_{q\text{-norm}}) \leq \|\boldsymbol{w}\|_\infty R\phi(q)\sqrt{T}$, where $\phi(q) \triangleq \beta^{\frac{1}{q}}\sqrt{\frac{\alpha^{2/q} - \alpha^2}{q-1}}$. The tightest regret bound is achieved with q^* that minimizes $\phi(q)$. We have

$$\phi'(q) = -\alpha^2\beta^{\frac{1}{q}}\frac{q^2\left(\alpha^{2/q-2} - 1\right) + 2(q-1)\left(\alpha^{2/q-2}\log(\alpha) + \left(\alpha^{2/q-2} - 1\right)\log(\beta)\right)}{2q^2(q-1)^2\sqrt{\frac{\alpha^{2/q} - \alpha^2}{q-1}}}. \tag{29}$$

We study the sign $(-J(q))$ of the derivative of the minimizer in Eq (29)

$$J(q) = q^2\left(\alpha^{2/q-2}-1\right) + 2(q-1)\left(\alpha^{2/q-2}\log(\alpha) + \left(\alpha^{2/q-2}-1\right)\log(\beta)\right) \tag{30}$$

$$\geq 2q(1-q)\log(\alpha) + 2(q-1)\left(2\tfrac{1-q}{q}\log(\alpha)\log(\alpha\beta) + \log(\alpha)\right) \tag{31}$$

$$\geq 2q(q-1)\left((1-q)\log(\alpha) + 2\tfrac{1-q}{q}\log(\alpha)\log(\alpha\beta)\right). \tag{32}$$

Note that $(1-q)\log(\alpha) \geq 0$ and $\frac{1-q}{q}\log(\alpha) \geq 0$. We take $\frac{R}{h} \leq k$, this gives $\alpha\beta \geq 1$ and $J(q) \geq 0$ implying $\text{sign}(\phi'(q)) = -\text{sign}(J(q)) = -1$. We conclude that $\phi(q)$ is a decreasing function of $q \in (1,2]$ when $\frac{R}{h} \leq k$; therefore, the minimum is obtained at $q = 2$ for $\frac{R}{h} \leq k$. □

A.5 Proof of Theorem 4.8

We have the following regret upper bound for the q-norm mirror map, as $q \to 1$ from Corollary 4.6: $\text{Regret}_T(\text{OMD}_{\text{1-norm}}) \leq \|\boldsymbol{w}\|_\infty hk\sqrt{2\log\left(\frac{N}{k}\right)T}$. In [15], it is proved that the log function satisfies $\log(u+1) \leq \frac{u}{\sqrt{u+1}}, u \geq 0$. We take $u = \frac{N}{k} - 1$, and note that $N \geq k > 0$, so we get $u \geq 0$. We have the following $\log\left(\frac{N}{k}\right) \leq \frac{N-k}{\sqrt{Nk}} = \sqrt{\frac{N}{k}}\left(1 - \frac{k}{N}\right)$. Thus, the upper bound in Corollary 4.6 Eq. (14) can be loosened to obtain $\text{Regret}_T(\text{OMD}_{\text{1-norm}}) \leq \|\boldsymbol{w}\|_\infty kh\sqrt{2\log\left(\frac{N}{k}\right)T} \leq \|\boldsymbol{w}\|_\infty \sqrt{2\sqrt{Nk}h^2k\left(1-\frac{k}{N}\right)}$. If we take $\frac{R}{h} \geq 2\sqrt{Nk}$, then this upper bound is tighter than the upper bound on the regret of OGD in Corollary 4.5. □

A.6 OMD_{NE} and OMD with q-norm Mirror Map Correspondence

Theorem A.2. *The algorithm* $\text{OMD}_{\text{1-norm}}$ *defined as the limitting algorithm obtain by taking q converges to 1 of* $\text{OMD}_{q\text{-norm}}$ *with learning rate* $\eta_q = \eta(q-1)k$, *intermediate states* $\boldsymbol{y}_t^{(q)}$, *and fractional states* $\boldsymbol{x}_t^{(q)}$ *for* $t \geq 1$ *is equivalent to* OMD_{NE} *configured with learning rate* $\eta \in \mathbb{R}_+$ *over the simplex (capped simplex* $\mathcal{X}$ *with* $k = 1$*), when both policies are initialized with same state in* $\mathbb{R}^N_{>0} \cap \mathcal{X}$. *Moreover,* $\text{OMD}_{\text{1-norm}}$ *has a multiplicative update rule over the capped simplex.*

Proof. Let $\boldsymbol{g}_t = \nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$ be the gradient of the cost function at time slot t. From lines 2–3 in Algorithm 1 and Eq. (15) we obtain the following $\hat{y}_{t,i}^{(q)} = (\nabla\Phi(\boldsymbol{x}_t))_i + \eta_q g_{t,i} = \text{sign}(x_{t,i})\frac{|x_{t,i}|^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta_q g_{t,i}$ for a given $q \in (1,2]$. The algorithm guarantees that $\boldsymbol{x}_t \in \mathcal{X} \cap \mathbb{R}^N_{>0}$, then $x_i > 0, \forall i \in \mathcal{N}$. So, we get $\hat{y}_{t,i}^{(q)} = \frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta_q g_{t,i}, \forall i \in \mathcal{N}$. Note that $-\eta g_{t,i}$ is non-negative. The numbers p and q are conjugate numbers (see Sec. 4.4) and satisfy $q - 1 = \frac{1}{p-1}$. We use Eq. (16) to get the expression of $y_{t+1,i}^{(q)}$ as

$$\frac{\left(\frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta_q g_{t,i}\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}}\left(\frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta_q g_{t,i}\right)^p\right)^{\frac{p-2}{p}}} = \frac{\left(\frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta(q-1)kg_{t,i}\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}}\left(\frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta(q-1)kg_{t,i}\right)^p\right)^{\frac{p-2}{p}}} = \frac{x_{t,i}\|\boldsymbol{x}_t\|_q^{(2-q)}\left(1-\eta(q-1)kg_{t,i}\frac{\|\boldsymbol{x}_t\|_q^{q-2}}{(x_{t,i})^{q-1}}\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}}(x_{t,i})^{(q-1)p}\left(1-\eta(q-1)kg_{t,i}\frac{\|\boldsymbol{x}_t\|_q^{q-2}}{(x_{t,i})^{q-1}}\right)^p\right)^{\frac{p-2}{p}}}.$$

We rewrite the above expression solely in terms of p, and taking the limit for q converges to 1 is equivalent to let p diverges to $+\infty$, so we have for all $i \in \mathcal{N}$

$$y_{t+1,i} \triangleq \lim_{p\to+\infty} y_{t+1,i}^{\left(\frac{p}{p-1}\right)} = x_{t,i}k\left(\lim_{p\to+\infty}\frac{\left(1-\frac{\eta g_{t,i}}{p-1}\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}}x_{t,i}\left(1-\frac{\eta g_{t,i}}{p-1}\right)^p\right)^{\frac{p-2}{p}}}\right) = x_{t,i}k\frac{\exp\left(-\eta g_{t,i}\right)}{\sum_{i\in\mathcal{N}}x_{t,i}\exp\left(-\eta g_{t,i}\right)}. \tag{33}$$

The intermediate state of $\text{OMD}_{\text{1-norm}}$ in Eq. (33) is a multiplicative update rule identical to the update rule of OMD_{NE} ($y_{t+1,i} = x_{t,i}\, e^{-\eta g_{t,i}}$ in Eq. (20)) with an additional multiplicative factor $\frac{k}{\sum_{i\in\mathcal{N}} x_{t,i}\exp\left(-\eta g_{t,i}\right)}$. For $k = 1$, the intermediate state of $\text{OMD}_{\text{1-norm}}$ in Eq. (33) is feasible (i.e., $\boldsymbol{x}_{t+1} = \boldsymbol{y}_{t+1}$ and the projection has no effect). On the other hand, the neg-entropy projection in the case of $k = 1$ is just a normalization of the intermediate states (i.e., $x_{t+1,i} = \frac{x_{t,i}\, e^{-\eta g_{t,i}}}{\sum_{i\in\mathcal{N}} x_{t,i}\exp\left(-\eta g_{t,i}\right)}$ for $i \in \mathcal{N}$). Thus, the states obtained by the two algorithms coincide. □

A.7 Proof of Theorem 4.10

The neg-entropy mirror map is $\rho = \frac{1}{k}$-strongly convex w.r.t the norm $\|\cdot\|_1$ over $\mathcal{X} \cap \mathcal{D}$ [53, Example 2.5]. The dual norm of $\|\cdot\|_1$ is $\|\cdot\|_\infty$. By taking $p \to \infty$ of L_p in the Proof of Theorem 4.4 we can consider as bound for the gradient in Eq. (28)

$$L = \|\boldsymbol{w}\|_\infty h. \tag{34}$$

The initial state $\boldsymbol{x}_1$ with $x_{1,i} = k/N, \forall i \in \mathcal{N}$ is the minimizer of Φ, and we have $\Phi(x) \leq 0, \forall \boldsymbol{x} \in X$. Thus

$$D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1) \leq \Phi(\boldsymbol{x}) - \Phi(\boldsymbol{x}_1) \leq -\Phi(\boldsymbol{x}_1) = -\sum_{i=1}^N \frac{k}{N}\log\left(\frac{k}{N}\right) = k\log\left(\frac{N}{k}\right) = D^2. \tag{35}$$

The bound follows by Theorem A.1, plugging (34) and (35) in (28), and selecting the learning rate that gives the tightest upper bound, that is $\eta = \sqrt{\frac{2\log\left(\frac{N}{k}\right)}{\|w\|_\infty^2 h^2 T}}$. □

A.8 Proof of Theorem 4.11

We adapt the Euclidean projection algorithm in [59]. Finding the projection $\mathbf{x} = \Pi^\Phi_{\mathcal{X}\cap\mathcal{D}}(\mathbf{y})$ corresponds to solving a convex problem as $D_\Phi(\mathbf{x}, \mathbf{y})$ is convex in $\mathbf{x}$ and $\mathcal{X} \cap \mathcal{D}$ is a convex set. Without loss of generality, we assume the components of $\mathbf{x} = \Pi^\Phi_{\mathcal{X}\cap\mathcal{D}}(\mathbf{y})$ to be in non-decreasing order. Let $b \in \mathcal{N}$ be the index of the largest component of $\mathbf{x}$ smaller than 1. The KKT conditions lead to conclude that if the components of $\mathbf{y}$ are ordered in ascending order, so are the components of $\mathbf{x}$. In particular, the smallest b components of $\mathbf{x}$ can be obtained as $x_i = y_i e^\gamma$ and $y_b e^\gamma < 1 \leq y_{b+1} e^\gamma$, where γ is the Lagrangian multiplier associated with the capacity constraint. If b is known, then it follows from the capacity constraint that $m_b \triangleq e^\gamma = \frac{k+b-N}{\sum_{i=1}^b y_i} = \frac{k+b-N}{\|\mathbf{y}\|_1 - \sum_{i=b+1}^N y_i}$. We observe that necessarily $b \in \{N-k+1, \ldots, N\}$. In fact, we cannot have $b \leq N-k$. If $b \leq N-k$, we get $\sum_{i=N-k+1}^N x_i \geq k$ and the capacity constraint implies that $x_i = 0, \forall i \leq b$, but we must have $x_i > 0$ since $\mathbf{x} \in \mathcal{X} \cap \mathcal{D}$ and $\mathcal{D} = R^N_{>0}$. We can then find the value of b, but checking which number in $\{N-k+1, \ldots, N\}$ satisfies $y_b e^\gamma < 1 \leq y_{b+1} e^\gamma$. Note that this operation only requires the largest k components of $\mathbf{y}$. The projection corresponds to setting the components $y_{b+1}, \ldots, y_N$ to 1 and multiply the other $N-b$ components by m_b. In order to avoid updating all components at each step, we can simply set the components x_i for $i > b$ (those that should be set equal to 1) to $\frac{1}{m_b}$. Then, at any time t, we can recover the value of $x_{t,i}$, multiplying the i-th component of the vector $\mathbf{x}$ by $P = \prod_{s=1}^t m_{b,s}$, where $m_{b,s}$ is the returned m_b from the Bregman projection at time step s. For general values of R and h, the projection step takes $O(k)$ steps per iteration and a partial sort is required to maintain top-k components of $\mathbf{y}_t$ sorted; this can be done using partial sorting in $O(N + k\log(k))$ [46]. When $R = h = 1$, Alg. 1 leads to only a single state coordinate update, and requires $O(\log(k))$ steps to maintain top-k components of $\mathbf{x}_t$ sorted online. □

A.9 Proof of Proposition 5.1

Every time slot $t \in [T]$, we obtain an intermediate cache state $\boldsymbol{y}_{t+1}$ through lines 2–4 in Algorithm 1 as $\boldsymbol{y}_{t+1} = (\nabla\Phi^{-1})(\nabla\Phi(\boldsymbol{x}_t) - \eta\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t))$. Let $\{\boldsymbol{x}_t\}_{t=1}^T$ and $\{\boldsymbol{x}'_t\}_{t=1}^T$ be fractional cache states obtained by OGD and OMD_{NE}, respectively, and $\{\boldsymbol{y}_t\}_{t=1}^T$ and $\{\boldsymbol{y}'_t\}_{t=1}^T$ be their intermediate fractional cache

states. In the case of OGD, or equivalently OMD configured with mirror map $\Phi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_2^2$, the intermediate fractional states have the same components as the previous cache state for files are not requested $\boldsymbol{y}_{t,i} = \boldsymbol{x}_{t,i}$ for every $i \notin \text{supp}(\boldsymbol{r}_t)$. Similarly, we also have $\boldsymbol{y}'_{t,i} = \boldsymbol{x}'_{t,i}$ for OMD_{NE} for every $i \notin \text{supp}(\boldsymbol{r}_t)$. The Euclidean projection algorithm onto the capped simplex [59] can only set a component of the intermediate fractional cache state to one if it exceeds it, and the remaining components are either set to zero or reduced by a constant amount $\Delta = \frac{N-b-k+\sum_{j=a+1}^{b} y_{t+1,j}}{b-a}$, where a is the number of components set to zero and b is the number of components strictly less than one, and $\Delta \geq y_{t+1,a}$ (a KKT condition in [59]) and in turn $\Delta \geq 0$ because $y_{t+1,i} \geq 0$ for any $i \in \mathcal{N}$. Therefore, all the components $i \notin \text{supp}(\boldsymbol{r}_t)$ of the resulting state are decreased or at most kept unchanged. Similarly, the neg-entropy Bregman projection onto the capped simplex sets some components to one if they exceed it, and the remaining components are scaled by a constant m_b. In our caching setting we have $y_{t+1,i} \geq x_{t,i}$ for $i \in \mathcal{N}$ in turn $\|\boldsymbol{y}_{t+1}\|_1 \geq k$, thus the equality constraint $\|\boldsymbol{x}\|_1 = k$ in the projection can be replaced by $\|\boldsymbol{x}\|_1 \leq k$. From the KKT dual feasibility condition we obtain $-\gamma \geq 0$ and $m_b = e^{\gamma} \leq 1$. Thus, we have $x_{t+1,i} \leq x_{t,i}$ for every $i \notin \text{supp}(\boldsymbol{r}_t)$. We conclude that the update cost is zero for both policies, i.e., $\text{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}) = \sum_{i \notin \text{supp}(\boldsymbol{r}_t)} w'_i \max(0, x_{t+1,i} - x_{t,i}) = 0$. □

B INTEGRAL CACHING

B.1 Proof of Proposition 6.1

Consider equal service costs $w_i = 1$ for any i in $\mathcal{N}$. A deterministic policy denoted by $\mathcal{A}$ selects an integral cache state $\boldsymbol{x}_t$ from $\mathcal{Z}$ for every time slot t, and the adversary can select a request batch $\boldsymbol{r}_t$ based on the selected state. Let $\boldsymbol{r}_t = [\mathbb{1}_{\{x_{t,i} \neq 1\}}]_{i \in \mathcal{N}}$ be the request batch selected by the adversary at time t, so the cost incurred at any time slot t is $f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) = N - k$, and the total cost incurred by $\mathcal{A}$ for the time horizon T is $\sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) = (N-k)T$. For a fixed integral cache state $\boldsymbol{x} \in \mathcal{Z}$,

$$\sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x}) = \sum_{t=1}^{T} \sum_{i=1}^{N} (1 - x_i)(1 - x_{t,i}) = T(N - 2k) + \sum_{i=1}^{N} x_i \sum_{t=1}^{T} x_{t,i}. \tag{36}$$

The best static cache state $\boldsymbol{x}_*$ is given by $\boldsymbol{x}_* = \arg\min_{\boldsymbol{x} \in \mathcal{Z}} \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x}) = \arg\min_{\boldsymbol{x} \in \mathcal{Z}} \sum_{i=1}^{N} x_i \sum_{i=1}^{T} x_{t,i}$. The maximum value of $\sum_{i=1}^{N} x_{*,i} \sum_{t=1}^{T} x_{t,i}$ is achieved when $\sum_{t=1}^{T} x_{t,i} = \sum_{t=1}^{T} x_{t,j} = Tk/N$ for every $i, j \in \mathcal{N}$, and in this case $\boldsymbol{x}_*$ can be arbitrary in $\mathcal{Z}$. Thus, the cost incurred by the static optimum is upper bounded by $\sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x}_*) = T(N-2k) + \sum_{i=1}^{N} x_{*,i} \sum_{t=1}^{T} x_{t,i} \leq T(N-2k) + \frac{Tk^2}{N}$. The regret of $\mathcal{A}$ over time horizon T is lower bounded by

$$\text{Regret}_T(\mathcal{A}) = \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) - \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x}_*) \geq T(N-k) - T(N-2k) - T\frac{k^2}{N} = k\left(1 - \frac{k}{N}\right)T. \tag{37}$$

We conclude that the regret of any deterministic policy $\mathcal{A}$ is $\Omega(T)$ compared to a static optimum selecting the best state in $\mathcal{Z}$; therefore, it also has $\Omega(T)$ regret compared to a static optimum selecting the best fractional state in $\mathcal{X}$, which includes $\mathcal{Z}$. □

B.2 Proof of Proposition 6.2

Proof. The expected service cost incurred when sampling the integral caching states $\boldsymbol{z}_t$ from $\boldsymbol{x}_t$ at each time t, by the linearity of $f_{\boldsymbol{r}_t}$ is $\mathbb{E}\left[\sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{z}_t)\right] = \sum_{t=1}^{T} \mathbb{E}\left[f_{\boldsymbol{r}_t}(\boldsymbol{z}_t)\right] = \sum_{t=1}^{T} f_{\boldsymbol{r}_t}\left(\mathbb{E}\left[\boldsymbol{z}_t\right]\right) = \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$. The best static configuration $\boldsymbol{x}_*$ in the fractional setting can always be selected to be integral; this is because the objective and constraints are linear, so integrality follows from the fundamental theorem of linear programming. Hence, the expected regret for the service cost coincides with the regret of the fractional caching policy. □

B.3 Proof of Theorem 6.3

Proof. We consider the catalog $\mathcal{N} = \{1, 2\}$, cache capacity $k = 1$, and equal service and update costs $w_i = w'_i = 1$ for $i \in \mathcal{N}$. A policy $\mathcal{A}$ selects the states $\{\boldsymbol{x}_t\}_{t=1}^{T} \in \mathcal{X}^T$. The randomized states obtained by Ξ are $\{\boldsymbol{z}_t\}_{t=1}^{T}$; thus, we have $\boldsymbol{z}_t = [1, 0]$ w.p. $x_{t,1}$, and $\boldsymbol{z}_t = [0, 1]$ w.p. $x_{t,2}$. An

adversary selects the request batch as $\boldsymbol{r}_t = [\mathbb{1}_{\{i=i_t\}}]_{i\in\mathcal{N}}$ aiming to greedily maximize the cost of the cache, where $i_t \triangleq \arg\min_{i\in\mathcal{N}}\{x_{t,i}\}$. The expected service cost at time t is $\mathbb{E}\left[f_{\boldsymbol{r}_t}(\boldsymbol{z}_t)\right] = 1 - x_{t,i_t}$, and the expected update cost is $\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t,\boldsymbol{z}_{t+1})\right] = x_{t,i_t}(1-x_{t+1,i_t})$. An update cost is incurred when i_t is requested and the state changes from $z_{t,i_t}=1$ to $z_{t+1,i_t}=0$; we pay a unitary cost due to fetching a single file that is not requested with probability $\mathbb{P}\left(z_{t,i_t}=1, z_{t+1,i_t}=0\right)$, and this gives the first equality. We use independence of the random variables $\boldsymbol{z}_t$ and $\boldsymbol{z}_{t+1}$ to obtain the second equality. A fixed state $\boldsymbol{x} = \left[\frac{1}{2},\frac{1}{2}\right]$ incurs a cost of $\frac{1}{2}$ for every timeslot $t \in [T]$. We define the instantaneous extended regret w.r.t. the fixed state $\boldsymbol{x}$ for every timeslot $t \in [T]$ as $a_t = \mathbb{E}\left[f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) + \mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t,\boldsymbol{z}_{t+1})\right] - f_{\boldsymbol{r}_t}(\boldsymbol{x}) = \mathbb{E}\left[f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) + \mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t,\boldsymbol{z}_{t+1})\right] - \frac{1}{2}$. Observe here that looking for a minimizer over $\mathcal{X}$ or over $\mathcal{Z}$ is equivalent. Because $\boldsymbol{x}$ is not necessarily the minimizer of the aggregate service cost $\sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x})$, we can lower bound the extended regret (26) as $\text{E-Regret}_T(\mathcal{A},\Xi) \geq \sum_{t=1}^{T} a_t$. Without loss of generality, assume that T is even so $\sum_{t=1}^{T} a_t = \sum_{k=1}^{T/2}(a_{2k}+a_{2k-1})$, and we have

$$\begin{aligned}\sum_{t=1}^{T} a_t &= \sum_{k=1}^{T/2}\left(1 - x_{2k-1,i_{2k-1}} - x_{2k,i_{2k}} + x_{2k-1,i_{2k-1}}(1-x_{2k,i_{2k-1}}) + x_{2k,i_{2k}}(1-x_{2k+1,i_{2k}})\right)\\ &\geq \sum_{k=1}^{T/2}\left(1 - x_{2k-1,i_{2k-1}} - x_{2k,i_{2k}} + x_{2k-1,i_{2k-1}}(1-x_{2k,i_{2k-1}})\right).\end{aligned}$$

From the definition of i_{2k} we have $x_{2k,i_{2k}} \leq (1-x_{2k,i_{2k-1}})$ for any $k \in [T/2]$. Thus,

$$\begin{aligned}&\sum_{t=1}^{T} a_t \geq \sum_{k=1}^{T/2}\left(1 - x_{2k-1,i_{2k-1}} - x_{2k,i_{2k}} + x_{2k-1,i_{2k-1}}x_{2k,i_{2k}}\right) = \sum_{k=1}^{T/2} 1 - x_{2k-1,i_{2k-1}} - x_{2k,i_{2k}}\left(1 - x_{2k-1,i_{2k-1}}\right)\\ &\geq \sum_{k=1}^{T/2}\left(1 - x_{2k-1,i_{2k-1}} - \tfrac{1}{2}\left(1 - x_{2k-1,i_{2k-1}}\right)\right) = \sum_{k=1}^{T/2}\left(\tfrac{1}{2} - \tfrac{1}{2}x_{2k-1,i_{2k-1}}\right) \geq \tfrac{T}{8}.\end{aligned}$$

The second and third inequalities are obtained using $x_{t,i_t} \leq \frac{1}{2}$ for every timeslot $t \in [T]$; a direct result from the definition of i_t. We combine the above lower bound with $\text{E-Regret}_T(\mathcal{A},\Xi) \geq \sum_{t=1}^{T} a_t$ to obtain $\text{E-Regret}_T(\mathcal{A},\Xi) \geq \frac{T}{8}$. □

B.4 Family of Coupling Schemes with Sublinear Update Cost

The following theorem provides a sufficient condition for the sublinearity of the expected total update cost of the random cache states $\{\boldsymbol{z}_t\}_{t=1}^{T}$ obtained through a rounding scheme Ξ from the input fractional states $\{\boldsymbol{x}_t\}_{t=1}^{T}$.

Theorem B.1. *Consider an OMD Algorithm and a joint distribution of $(\boldsymbol{z}_t,\boldsymbol{z}_{t+1})$ that satisfy (a) $\mathbb{E}[\boldsymbol{z}_t] = \boldsymbol{x}_t$ and $\mathbb{E}[\boldsymbol{z}_{t+1}] = \boldsymbol{x}_{t+1}$, and (b) $\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t,\boldsymbol{z}_{t+1})\right] = O\left(\|\boldsymbol{x}_{t+1}-\boldsymbol{x}_t\|_1\right)$. This algorithm incurs an expected service cost equal to the service cost of the fractional sequence. Moreover, if $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$, the algorithm has also $O\left(\sqrt{T}\right)$ expected update cost and then $O\left(\sqrt{T}\right)$ extended regret.*

Proof. Consider that the sequence $\{\boldsymbol{x}_t\}_{t=1}^{T}$ is generated by an OMD algorithm, configured with a ρ-strongly convex mirror map Φ w.r.t a norm $\|\cdot\|$. Assume that we can find a joint distribution of $(\boldsymbol{z}_t,\boldsymbol{z}_{t+1})$ satisfying $\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t,\boldsymbol{z}_{t+1})\right] = O\left(\|\boldsymbol{x}_{t+1}-\boldsymbol{x}_t\|_1\right)$, where $\mathbb{E}[\boldsymbol{z}_t] = \boldsymbol{x}_t$ and $\mathbb{E}[\boldsymbol{z}_{t+1}] = \boldsymbol{x}_{t+1}$. Then there exists a constant $\gamma_1 > 0$, such that $\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t,\boldsymbol{z}_{t+1})\right] \leq \gamma_1\|\boldsymbol{x}_{t+1}-\boldsymbol{x}_t\|_1$. Moreover, there exists $\gamma_2 > 0$, such that $\gamma\|\boldsymbol{x}_{t+1}-\boldsymbol{x}_t\|_1 \leq \gamma_1\gamma_2\|\boldsymbol{x}_{t+1}-\boldsymbol{x}_t\|$, and this gives $\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t,\boldsymbol{z}_{t+1})\right] \leq \gamma_1\gamma_2\|\boldsymbol{x}_{t+1}-\boldsymbol{x}_t\|$. As Φ is ρ-strongly convex w.r.t. the norm $\|\cdot\|$, it holds

$$\begin{aligned}&D_\Phi(\boldsymbol{x}_t,\boldsymbol{y}_{t+1}) = \Phi(\boldsymbol{x}_t) - \Phi(\boldsymbol{y}_{t+1}) + \nabla\Phi(\boldsymbol{x}_t)^T(\boldsymbol{y}_{t+1}-\boldsymbol{x}_t) + (\nabla\Phi(\boldsymbol{x}_t) - \nabla\Phi(\boldsymbol{y}_{t+1}))^T(\boldsymbol{x}_t-\boldsymbol{y}_{t+1})\\ &\leq -\frac{\rho}{2}\|\boldsymbol{x}_t-\boldsymbol{y}_{t+1}\|^2 + \eta g_t^T(\boldsymbol{x}_t-\boldsymbol{y}_{t+1}) \leq -\frac{\rho}{2}\|\boldsymbol{x}_t-\boldsymbol{y}_{t+1}\|^2 + \eta\|\boldsymbol{x}_t-\boldsymbol{y}_{t+1}\|L \leq \frac{\eta^2L^2}{2\rho}\end{aligned} \tag{38}$$

The above inequalities are obtained using the strong convexity of Φ and the update rule, Cauchy-Schwarz inequality, and the inequality $ax - bx^2 \leq \max_x ax - bx^2 = a^2/4b$ as in the last step in the

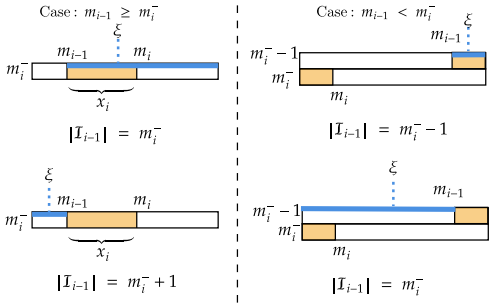


Fig. 9. The support and distribution of the random variable $|\mathcal{I}_{i-1}|$ for $m_{i-1} \geq m_i^-$ (left) and $m_{i-1} < m_i^-$ (right). The blue shaded area represents the probability that $|\mathcal{I}_{i-1}|$ takes the value indicated below the corresponding image. This figure can be viewed as a subset of the rows in Fig. 10.

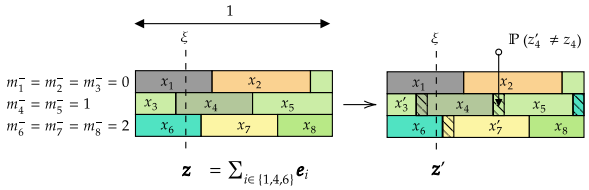


Fig. 10. The random integral cache configuration obtained by calling Online Rounding given the fractional cache state $\boldsymbol{x}$ and ξ (left). When ξ is kept fixed, the probability over the initial choice of ξ the random integral cache configuration rounded from a new fractional state $\boldsymbol{x}' = \boldsymbol{x} - \delta\boldsymbol{e}_3 + \delta\boldsymbol{e}_7$ is different is illustrated by the dashed areas (right).

proof of [11, Theorem 4.2], respectively. We have $\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\| \leq \sqrt{\frac{2}{\rho} D_\Phi(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})} \leq \sqrt{\frac{2}{\rho} D_\Phi(\boldsymbol{x}_t, \boldsymbol{y}_{t+1})} \overset{(38)}{\leq} \sqrt{2\eta^2 \frac{L^2}{2\rho^2}} \leq \frac{L\eta}{\rho}$. The first inequality is obtained using the strong convexity of Φ, and the second using the generalized Pythagorean inequality[13, Lemma 11.3]. We combine $\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\| \leq L\eta/\rho$ and $\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] \leq \gamma_1\gamma_2 \|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|$ to obtain $\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] \leq \gamma_1\gamma_2 \|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\| \leq \gamma_1\gamma_2 \frac{L\eta}{\rho}$. The total update cost is $\sum_{t=1}^{T-1} \mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] \leq \gamma_1\gamma_2 \frac{L\eta}{\rho} T$. When OMD has a fixed learning rate $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$, we obtain $\sum_{t=1}^{T-1} \mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] = O\left(\sqrt{T}\right)$. The expected service cost is $\mathbb{E}\left[\sum_{t=1}^{T} f_t(\boldsymbol{z}_t)\right] = \sum_{t=1}^{T} f_t(\boldsymbol{x}_t) = O(T)$; the first equality is obtained from the linearity of the expectation operator and the function $f_{\boldsymbol{r}_t}$, and the second equality is obtained using the bound in Eq. (28) with $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$. □

B.5 Proof of Theorem 6.4

Lemma B.2 and Lemma B.4 guarantee that Algorithm 3 used with an OMD algorithm satisfies the hypothesis of Theorem B.1 and, hence, provides sublinear extended regret.

Lemma B.2. *The random integral cache state $\boldsymbol{z}$ obtained by calling Algorithm 3 with fractional cache configuration input $\boldsymbol{x} \in \mathcal{X}$ satisfies $\boldsymbol{z} \in \mathcal{Z}$ and $\mathbb{E}_\xi[\boldsymbol{z}] = \boldsymbol{x}$.*

Proof. We employ the shorthand notation $m_i = \sum_{j=1}^{i} x_j$ and $m_i^- = \lfloor m_i \rfloor$. The choice of ξ (see Fig. 10) defines k different thresholds $\xi, \xi + 1, \ldots, \xi + k - 1$. For each threshold, we select the first item, whose accumulated mass exceeds the threshold. As $m_N = k$, we are guaranteed to exceeds all k thresholds, and as $x_i \leq 1$, we are guaranteed to select one item for each threshold. Therefore $\boldsymbol{z}$ belongs to $\mathcal{Z}$. From Algorithm 3 for any $i \in \mathcal{N}$ we have $\mathbb{P}(z_i = 1) = \mathbb{P}(\mathcal{I}_i \setminus \mathcal{I}_{i-1} = \{i\}) = \mathbb{P}(m_i \geq \xi + |\mathcal{I}_{i-1}|)$.

If $m_{i-1} \geq m_i^-$, see Fig. 9 (left), then $|\mathcal{I}_{i-1}| \in \{m_i^-, m_i^- + 1\}$ and $\mathbb{P}(m_i \geq \xi + |\mathcal{I}_{i-1}|) = \frac{m_i - m_{i-1}}{(m_i^- + 1 - m_{i-1})} \cdot (m_i^- + 1 - m_{i-1}) + 0 \cdot (m_{i-1} - m_i^-) = x_i$. If $m_{i-1} < m_i^-$, see Fig. 9 (right), then $|\mathcal{I}_{i-1}| \in \{m_i^- - 1, m_i^-\}$ and $\mathbb{P}(m_i \geq \xi + |\mathcal{I}_{i-1}|) = 1 \cdot (m_i^- - m_{i-1}) + \frac{m_i - m_i^-}{m_{i-1} - m_i^- + 1} \cdot (m_{i-1} - m_i^- + 1) = m_i - m_{i-1} = x_i$. □

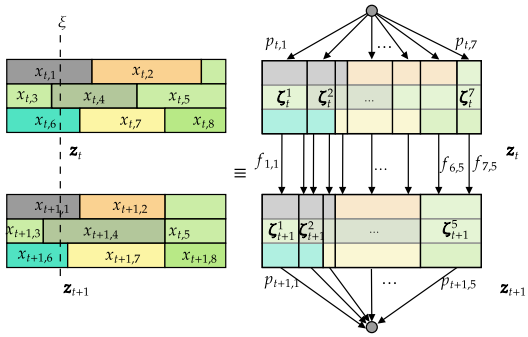


Fig. 11. Coupling induced by Online Rounding Algorithm 3 when ξ is fixed. The flow $f_{i,j}$ is the joint probability $\mathbb{P}(\boldsymbol{z}_{t+1} = \zeta_{t+1,j}, \boldsymbol{z}_t = \zeta_{t,i})$, so that the next state is $\zeta_{t+1,j}$ and the previously selected state is $\zeta_{t,i}$.

Lemma B.3. *Consider a fractional cache configuration $\boldsymbol{x}'$ obtained by an elementary mass movement of δ from $u \in \mathcal{N}$ to $v \in \mathcal{N}$ for configuration $\boldsymbol{x} \in \mathcal{X}$, i.e., $\boldsymbol{x}' = \boldsymbol{x} - \delta \boldsymbol{e}_u + \delta \boldsymbol{e}_v$. Algorithm 3 outputs the random integral cache configurations $\boldsymbol{z}$, and $\boldsymbol{z}'$, given the input fractional cache states $\boldsymbol{x}$ and $\boldsymbol{x}'$, respectively. The random integral configurations satisfy $\mathbb{E}_\xi \left[\|\boldsymbol{z}' - \boldsymbol{z}\|_{1,\boldsymbol{w}'} \right] \leq 2kN \|\boldsymbol{w}'\|_\infty \delta$, where $\|\boldsymbol{x}\|_{1,\boldsymbol{w}'} \triangleq \sum_{i \in \mathcal{N}} |x_i| w_i'$, and note that $\mathrm{UC}_{\boldsymbol{r}_t} (\boldsymbol{z}_t, \boldsymbol{z}_{t+1}) \leq \|\boldsymbol{z}_t - \boldsymbol{z}_{t+1}\|_{1,\boldsymbol{w}'}$.*

Proof. The probability that a random integral cache state $\boldsymbol{z}'$ is changed w.r.t $\boldsymbol{z}$ can be upper bounded as $\mathbb{P}(\boldsymbol{z} \neq \boldsymbol{z}') \leq \sum_{i \in \mathcal{N}} \mathbb{P}(z_i \neq z_i')$. Consider w.l.g that $u < v$, then $\mathbb{P}(z_i \neq z_i') = 0$ for $i \in \mathcal{N} \setminus \{u, u+1, \ldots, v\}$, and $\mathbb{P}(z_i \neq z_i') = \delta$ for $i \in \{u, u+1, \ldots, v\}$. We obtain $\mathbb{P}(\boldsymbol{z} \neq \boldsymbol{z}') \leq \delta(v-u+1)$ (e.g., see Fig. 10). More generally, for $u \neq v$ we have $\mathbb{P}(\boldsymbol{z} \neq \boldsymbol{z}') \leq \delta(|v-u|+1)$. We conclude that

$$\mathbb{E}\left[\|\boldsymbol{z}' - \boldsymbol{z}\|_{1,\boldsymbol{w}'} \right] \leq \max_{(\boldsymbol{z},\boldsymbol{z}') \in \mathcal{Z}^2} \|\boldsymbol{z}' - \boldsymbol{z}\|_{1,\boldsymbol{w}'} \cdot \mathbb{P}(\boldsymbol{z} \neq \boldsymbol{z}') \leq 2k \|\boldsymbol{w}'\|_\infty (|v-u|+1)\delta \leq 2kN \|\boldsymbol{w}'\|_\infty \delta. \quad \square$$

Lemma B.4. *The expected movement cost of the random integral cache states generate by Algorithm 3 is $\mathbb{E}_\xi \left[\mathrm{UC}_{\boldsymbol{r}_t} (\boldsymbol{z}_t, \boldsymbol{z}_{t+1}) \right] = O\left(\|\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\|_1 \right)$, when ξ is sampled once u.a.r. from the interval $[0,1]$ and then fixed for $t \in [T]$.*

Proof. The general fractional movement caused by a policy $\mathcal{A}$ changes the cache state from fractional state $\boldsymbol{x}_t \in \mathcal{X}$ to $\boldsymbol{x}_{t+1} \in \mathcal{X}$, and we denote by $\mathcal{J} = \left\{ i \in \mathcal{N} : x_{t+1,i} - x_{t,i} > 0 \right\}$ the set of components that have a fractional increase. We have $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \sum_{j \in \mathcal{J}} \phi_j \boldsymbol{e}_j - \sum_{i \in \mathcal{N} \setminus \mathcal{J}} \phi_i \boldsymbol{e}_i$. where $\phi_i, i \in \mathcal{N}$ is the absolute fractional change in component i of the cache. Remark that we have $\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_{1,\boldsymbol{w}'} = \sum_{i \in \mathcal{N}} w_i' \phi_i$. From the capacity constraint we know that $\sum_{i \in \mathcal{N} \setminus \mathcal{J}} \phi_i = \sum_{j \in \mathcal{J}} \phi_j$. If we want to decompose this general fractional change to elementary operations, then we need to find a flow $\left[\delta_{i,j} \right]_{(i,j) \in (\mathcal{N} \setminus \mathcal{J}) \times \mathcal{J}}$ that moves $\sum_{j \in \mathcal{J}} \phi_j$ mass from the components in $\mathcal{N} \setminus \mathcal{J}$ to to those in $\mathcal{J}$. This requires at most $N-1$ elementary operations. We define the map $\nu : \mathcal{N}^2 \to \mathbb{N}$ that provides an order on the sequence of elementary operations. Let $\boldsymbol{z}^{\nu(i,j)}$ be the random cache state that could have been sampled after the $\nu(i,j)$-th elementary operation where $\mathbb{E}\left[\boldsymbol{z}^{\nu(i,j)} \right] = \boldsymbol{x}^{\nu(i,j)}$, and the total number of operations is denoted by $|\nu| \leq N-1$. Note that by definition $\boldsymbol{z}^{|\nu|} = \boldsymbol{z}_{t+1}$, and we take $\boldsymbol{z}^0 = \boldsymbol{z}_t$. For each of these operations we pay in expectation at most $2kN \|\boldsymbol{w}'\|_\infty \delta_{i,j}$ update cost from Lemma B.3. Then the total expected movement cost is:

$$\mathbb{E}_\xi \left[\mathrm{UC}_{\boldsymbol{r}_t} (\boldsymbol{z}_t, \boldsymbol{z}_{t+1}) \right] \leq \mathbb{E}_\xi [\|\boldsymbol{z}_{t+1} - \boldsymbol{z}_t\|_{1,\boldsymbol{w}'}] = \mathbb{E}_\xi \left[\left\| \sum_{l=0}^{|\nu|-1} \left(\boldsymbol{z}^{l+1} - \boldsymbol{z}^l \right) \right\|_{1,\boldsymbol{w}'} \right] \leq \sum_{l=0}^{|\nu|-1} \mathbb{E}_\xi \left[\left\| \boldsymbol{z}^{l+1} - \boldsymbol{z}^l \right\|_{1,\boldsymbol{w}'} \right]$$
$$\leq \sum_{i \in \mathcal{N} \setminus \mathcal{J}} \sum_{j \in \mathcal{J}} 2k \|\boldsymbol{w}'\|_\infty N \delta_{i,j} = 2kN \|\boldsymbol{w}'\|_\infty \sum_{i \in \mathcal{N} \setminus \mathcal{J}} \phi_i \leq kN \|\boldsymbol{w}'\|_\infty \|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_1 .$$

The update cost is thus $O\left(\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_1 \right)$ in expectation. $\square$