Learning to Cache With No Regrets

Georgios S. Paschos*, Apostolos Destounis*, Luigi Vigneri*†, George Iosifidis‡
*France Research Center, Huawei Technologies, Paris
†IOTA Foundation, Berlin, Germany
‡School of Computer Science and Statistics, Trinity College Dublin, Ireland

***Abstract*—This paper introduces a novel caching analysis that, contrary to prior work, makes no modeling assumptions for the file request sequence. We cast the caching problem in the framework of *Online Linear Optimization* (OLO), and introduce a class of *minimum regret* caching policies, which minimize the losses with respect to the *best static configuration in hindsight* when the request model is unknown. These policies are very important since they are robust to popularity deviations in the sense that they learn to adjust their caching decisions when the popularity model changes. We first prove a novel lower bound for the regret of any caching policy, improving existing OLO bounds for our setting. Then we show that the *Online Gradient Ascent* (OGA) policy guarantees a regret that matches the lower bound, hence it is universally optimal. Finally, we shift our attention to a network of caches arranged to form a bipartite graph, and show that the *Bipartite Subgradient Algorithm* (BSA) has no regret.**

I. Introduction

We study the performance of caching systems from a new perspective: *we seek a caching policy that optimizes the system's performance under any distribution of file request sequence*. This not only has huge practical significance as it tackles the caching policy design problem in its most general form, but also reveals a novel connection between caching and online linear optimization [1]–[3]. This, in turn, paves the way for a new mathematical framework enabling the principled design of caching policies with performance guarantees.

A. Background and Related Work

Due to its finite capacity a cache can typically host only a small subset of the file library, and hence a *caching policy* must decide which files should be stored. The main performance criterion for a caching policy is the so-called *cache hit ratio*, i.e., the portion of file requests the cache can satisfy locally. Several policies have been proposed in the past with the aim to maximize the cache hit ratio. For instance, the Least-Recently-Used (LRU) policy inserts in the cache the newly requested file and evicts the one that has not been requested for the longest time period; while the Least-Frequently-Used (LFU) policy evicts the file that is least frequently requested. These policies (and variants) were designed empirically, and one might ask: *under what conditions do they achieve high hit ratios*?

The answer depends on the properties of the file request sequence. For instance, we know that *(i)* for stationary requests, LFU achieves the highest hit ratio [4]; *(ii)* a more sophisticated *age-based-threshold* policy maximizes the hit ratio when the requests follow the Poisson Shot Noise model [5]; and *(iii)* LRU has the highest competitive hit ratio [6] for the adversarial model [7], [8] which assumes that the requests are set by an *adversary* aiming to degrade the system's performance. However, the highest competitive hit ratio is achieved by any *marking* policy [9] – including FIFO – suggesting that this metric is perhaps "too strong" to allow a good policy classification. Evidently, to decide which policy to use, it is necessary to know the underlying file request model, which in practice is *a priori* unknown and, possibly, time-varying. *This renders imperative the design of a universal caching policy that will work provably well for all request models.*

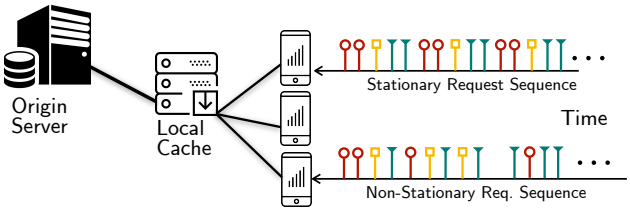


Fig. 1. File requests must be served by a local cache (hit) or the origin server (miss). A caching policy observes past requests and decides which files to cache in order to increase the hit ratio.

This is even more important in the emerging edge caching architectures that use small local servers or caches attached to wireless stations. These caches receive a low number of requests and therefore, inevitably, "see" request processes with highly non-stationary popularity [5], [10], [11]. Prior works employ random replacement models [11] or inhomogeneous Poisson processes [5], [12] to model the requests in these systems. However, such multi-parametric models are challenging to fit and rely on strong assumptions about the popularity evolution. Other notable approaches learn the instantaneous popularity model with no prior assumptions. For instance [13], [14] employ *Q-learning*, and [15] leverages a scalable prediction method; but contrary to our approach, they assume the popularity evolution is stationary across time. The design of adaptive paging policies is systematically studied in [16]–[18] as an online learning problem. These works, however, consider only the basic paging problem, i.e., hit ratio maximization when caching entire files in a single cache.

Caching networks (CNs) on the other hand, are hitherto under-explored; yet very important as most often caches are organized in networks. In CNs one needs to jointly decide which cache will satisfy a request (routing) and which files will be evicted (caching). The works [19], [20] proposed joint routing and caching policies for bipartite and general network graphs, respectively, and [21] extended them to CNs

This work was supported by Science Foundation Ireland under Grant No. 17/CDA/4760. The opinions expressed in this paper are of the authors alone, and do not represent an official position of Huawei Technologies.

with elastic storage. However all these works assume that file popularity is stationary. On the other hand, [22] proposed the multi-LRU (mLRU) strategy, and [23] proposed the "lazy rule" extending q-LRU to provide local optimality guarantees under stationary requests. *It transpires that we currently lack a principled method to design caching policies with provable performance guarantees, under general popularity conditions, for single or networks of caches. This is exactly our focus.*

B. Methodology and Contributions

We introduce a model-free caching policy along the lines of Online Linear Optimization (OLO). For the single cache case we obtain matching lower and upper regret bounds, proving that the online gradient ascent achieves order optimal performance, and we then extend these results to CNs. We assume that file requests are drawn from a general distribution, which is equivalent to caching versus an adversary selecting the requests. At each slot, in order, *(i)* the caching policy decides which file parts to cache; *(ii)* the adversary introduces the new request; and *(iii)* a file-dependent utility is obtained proportional to the fraction of the request that was served by the cache. This generalizes the criterion of cache hit ratio and allows us to build policies that, e.g., optimize delay or provide preferential treatment to files. In this setting, we seek to find a caching policy with minimum *regret*, i.e., minimum utility loss over an horizon T, compared to the best cache configuration when the request sample path is known.

We prove that well-known caching policies such as LRU and LFU have $\Omega(T)$ regret, hence there exist request patterns for which these policies fail to learn a good caching configuration (losses increase with time). In contrast, we propose the *Online Gradient Ascent* (OGA) policy, and prove its regret is at most $O(\sqrt{CT})$, for a cache that can store C out of the N total files. This shows that OGA eventually (as $T \to \infty$) amortizes the losses under any request model, even under denial-of-service attacks. Additionally we prove a *novel lower bound* which is tighter than the general OLO lower bound. For the case of online hit ratio maximization, we find that any policy must have at least $\Omega(\sqrt{CT})$ regret. Combining the two results we conclude that *(i)* the regret of online caching is exactly $\Theta(\sqrt{CT})$, and *(ii)* OGA is a universally optimal policy. Interestingly, OGA can be seen as a regularized LFU, or a slightly modified LRU as we explain in Section IV-C.

We extend our model to a network of caches, arranged in the form of a bipartite graph – a setting known as femtocaching [24]. We provide the *Bipartite Subgradient Algorithm* (BSA) caching strategy that achieves a regret $O(\sqrt{\deg\, JCT})$, where J is the number of caches and deg the maximum network node degree. Our contributions can be summarized as follows:

- *Machine Learning (ML) approach*: we provide a fresh ML angle into caching policy design and performance analysis. To the best of our knowledge this is the first time OLO is used to provide optimality bounds in caching problem. Moreover, we reverse-engineer the standard caching LRU/LFU-type of policies, by drawing connections with OLO, and provide directions for improving them.
- *Universal single-cache policy*: The proposed OGA policy is universally optimal, i.e., provides zero loss versus the best caching configuration under any request model. An important projection algorithm is provided to reduce complexity and enable operation in large caches.
- *Universal bipartite caching*: We consider a general bipartite CN and design the *online joint caching and routing BSA policy*. Our approach hits the sweet spot of complexity versus utility for CNs: offers rigorous performance results, while it is applicable to fairly complicated settings.
- *Trace-driven Evaluation*: We employ a battery of tests evaluating our policies with several request patterns. We find that OGA outperforms LRU/LFU by 20% in different scenarios, while BSA beats lazy-LRU by 45.8%.

II. System model

We study a system with a library of files $\mathcal{N}=\{1,2,\ldots,N\}$ of equal size and a cache that fits $C<N$ of them. The system evolves in time slots, and in each slot t a single request is made for $n\in\mathcal{N}$, denoted with the event $x_t^n=1$. Vector $x_t=(x_t^n, n\in\mathcal{N})$ represents the t-slot request, chosen from the set:

$$\mathcal{X}=\left\{x\in\{0,1\}^N \;\middle|\; \sum_{n=1}^{N} x^n=1\right\}.$$

The instantaneous file popularity is determined by the probability distribution $P(x_t)$ (with support $\mathcal{X}$), which is allowed to be unknown and arbitrary; and the same holds for the joint distribution $P(x_1,\ldots,x_T)$ that describes the file popularity evolution. This generic model captures all studied request sequences in the literature, including stationary (i.i.d. or otherwise), non-stationary, and adversarial models.

The cache is managed with the caching configuration variable $y_t\in[0,1]^N$, that denotes the fraction of file n cached in slot t.[1] Taking into account the cache size C, the set $\mathcal{Y}$ of admissible caching configurations is:

$$\mathcal{Y}=\left\{y\in[0,1]^N \;\middle|\; \sum_{n=1}^{N} y^n\leq C\right\}. \tag{1}$$

Definition 1 (Caching Policy)**.** *A caching policy σ is a (possibly randomized) rule that at slot $t=1,\ldots,T$ maps past observations $x_1,\ldots,x_{t-1}$ and configurations $y_1,\ldots,y_{t-1}$ to the configuration $y_t(\sigma)\in\mathcal{Y}$ of slot t.*

We denote with w^n the utility obtained when file n is requested and found in the cache (also called a *hit*). This file-dependent utility can be used to model bandwidth economization from cache hits [25], QoS improvement [24], or any other cache-related benefit. We will also be concerned with the special case $w^n=w, n\in\mathcal{N}$, i.e., the *cache hit ratio* maximization. A cache configuration y_t accrues in slot t a utility $f\big(x_t,y_t\big)$ determined as follows:

$$f\big(x_t,y_t\big)=\sum_{n=1}^{N} w^n x_t^n y_t^n.$$

[1] Why caching of file fractions makes sense? Large video files are composed of chunks stored independently, see literature of *partial caching* [25]. Also, the fractional variables may represent caching probabilities [2], [26], or coded equations of chunks [24]. For practical systems, the fractional y_t^n should be rounded, which will induce a small application-specific error.

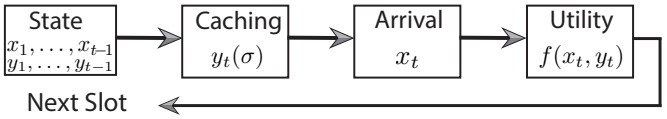


Fig. 2. A caching decision is taken; the adversary selects the request; the caching utility is realized; and the system state is updated for next t.

Let us now cast caching as an online linear optimization problem. This requires the following conceptual innovations. Since we allow the request sequence to follow any arbitrary probability distribution, we may equivalently think of x_t as being set by an *adversary* that aims to degrade the performance of the policy. Going a step further, we can equivalently interpret that the adversary selects at each slot t the utility function of the system $f_t(y)$ from a family of linear functions, $f_t(y) \equiv f(x_t, y)$. Finally, note that y_t is set at the beginning of slot t, before the adversary selects x_t, Fig. 2. The above place our problem squarely in the OLO framework.

Given the adversarial nature of our model, the ability to extract useful conclusions depends crucially on the selected performance metric. Differently from the competitive ratio approach of [6], we introduce a new metric that compares how our caching policy fares against the *best static policy in hindsight*. This metric is often used in the literature of machine learning [2], [3] with the name worst-case static regret. In particular, we define the *regret of policy* σ as:

$$R_T(\sigma) = \max_{P(x_1,\ldots,x_T)} \mathbb{E}\left[\sum_{t=1}^{T} f_t\big(y^*\big) - \sum_{t=1}^{T} f_t\big(y_t(\sigma)\big)\right]$$

where T is the horizon; the maximization is over the admissible adversary distributions; the expectation is taken w.r.t. the possibly randomized x_t and $y_t(\sigma)$; and $y^* \in \arg\max_{y\in\mathcal{Y}} \sum_{t=1}^{T} f_t(y)$ is the best static configuration in hindsight, i.e., a benchmark policy that knows the sample path $x_1, \ldots, x_T$. Intuitively, measuring the utility loss of σ over y^* constrains the power of the adversary: radical request pattern changes will challenge σ but also induce huge losses in y^*. This comparison allows us to discern policies that can learn good caching configurations from those that fail.

We seek a caching policy that minimizes the regret by solving the problem $\inf_\sigma R_T(\sigma)$, known as *Online Linear Optimization* (OLO) [2]. The analysis in OLO aims to discover how the regret scales with horizon T. A caching policy with sublinear regret $o(T)$ produces average losses $R_T(\sigma)/T \to 0$ with respect to the best configuration with hindsight, hence it learns to adapt the cache configuration without any prior knowledge about the request distribution. Our problem is further compounded due to the cache size dimension. Namely, apart from optimizing the regret w.r.t. T, it is of practical importance to consider also the dependence on N (or C).

Regret of Standard Policies. Having introduced this new caching optimization formulation, it is interesting to characterize the worst case performance of LRU and LFU policies. Recall that LRU caches the C most recently requested files, while LFU calculates for each file the request frequency $h_t^n = \frac{1}{t-\tau^n}\sum_{i=\tau^n}^{t} x_i^n$, where τ^n is the slot when file n was requested for the first time, and caches the C most frequent files. The following proposition describes their performance under arbitrary requests where, for simplicity, we assumed $w^n = w, \forall n$ (hit ratio maximization).

Proposition 1. *The regret of* LRU*,* LFU *satisfies:*

$$R_T(\text{LRU}) = R_T(\text{LFU}) \geq wC\left(\frac{T}{C+1} - 1\right).$$

Proof: Assume that the adversary chooses the periodic request sequence $\{1, 2, \ldots, C+1, 1, 2, \ldots\}$. For any $t > C$, since the requested file is the $C+1$ least recent file, it is not in the LRU cache, and no utility is received. Hence, LRU can achieve at most wC utility from the first C slots. However, a static policy with hindsight achieves at least $wTC/(C+1)$ by caching the first C files. The same rationale can be used for LFU by noticing that due to the structure of the periodic arrivals, the least frequent file is also the least recent one. ■

The $\Omega(T)$ performance of standard caching policies is poor, yet this is rather expected since they are designed to perform well only under certain models (LRU for requests with temporal locality; LFU for stationary requests), and they are known to under-perform in other cases, e.g., LRU in stationary, and LFU in decreasing popularity patterns. Low regret performance means that there exist request distributions for which the policy fails to "learn" what is the best configuration. *Remarkably, we show below that there exist universal caching policies which ensure low regret under any request model.*

III. Regret Lower Bound

A regret lower bound is a powerful theoretical result that provides the fundamental limits of how fast any algorithm can learn to cache, much like the upper bound of the channel capacity. Regret lower bounds in OLO have been previously derived for different action sets: for N-dimensional unit ball centered at the origin in [27], and N-dimensional hypercube in [28]. In our case, however, the above results do not apply since $\mathcal{Y}$ in (1) is a capped simplex, i.e., the intersection of a box and a simplex inequality. Therefore, we need the following new regret lower bound tailored to the online caching problem.

Theorem 1 (Regret Lower Bound)**.** *The regret of any caching policy σ satisfies:*

$$R_T(\sigma) > \sum_{i=1}^{C} \mathbb{E}\big[Z_{(i)}\big]\sqrt{T}, \quad \text{as } T \to \infty,$$

where $Z_{(i)}$ is the i-th maximum element of a Gaussian random vector with zero mean and covariance matrix $\mathbf{\Sigma}(w)$ given by (5)*.*
Furthermore, assume $C < N/2$ and define ϕ any permutation of $\mathcal{N}$ and Φ the set of all such permutations:

$$R_T(\sigma) > \frac{\max_{\phi\in\Phi}\sum_{k=1}^{C}\sqrt{w^{\phi(2(k-1)+1)} + w^{\phi(2k)}}}{\sqrt{2\pi\sum_{n=1}^{N} 1/w^n}}\sqrt{T}$$

In the important special case of hit rate maximization, where each file n is $w^n = w$, the above bound simplifies to:

Corollary 1. *Fix $\gamma \triangleq C/N$, $w^n = w$, $\forall n$, and $C < N/2$. Then, the regret of any caching policy σ satisfies:*

$$R_T(\sigma) > w\sqrt{\frac{\gamma}{\pi}}\sqrt{CT}, \quad \text{as } T \to \infty.$$

Before providing the proof, a few remarks are in order. Our bound is tighter than the classical $\Omega(\sqrt{(\log N)T})$ of OLO in the literature [27], [28], which is attributed to the difference of sets $\mathcal{X}, \mathcal{Y}$. In our proof we provide technical lemmas that are also useful, beyond caching, for the regret analysis of capped simplex sets. In next section we will design a caching policy that achieves regret $O(\sqrt{CT})$, establishing that the regret of online caching is in fact $\Theta(\sqrt{CT})$.

Proof of Theorem 1: To find a lower bound, we will analyze a specific adversary x_t. In particular, we will consider an i.i.d. x_t such that file n is requested with probability

$$\mathbb{P}(x_t = \mathbf{e}_n) = \frac{1/w^n}{\sum_{i=1}^N 1/w^i}, \quad \forall n, t,$$

where $\mathbf{e}_n$ is a vector with element n equal to 1 and the rest zero. With such a choice of x_t, any causal caching policy yields an expected utility at most $CT/\sum_{n=1}^N (1/w^n)$, since

$$\mathbb{E}\left[\sum_{t=1}^T f_t(y_t(\sigma))\right] = \sum_{t=1}^T \sum_{n=1}^N w^n \mathbb{P}(x_t = \mathbf{e}_n)\, y_t^n(\sigma) \qquad (2)$$
$$= \sum_{t=1}^T \frac{1}{\sum_n 1/w^n} \sum_{n=1}^N y_t^n(\sigma) \leq \frac{CT}{\sum_n 1/w^n},$$

independently of σ. To obtain a regret lower bound we show that a static policy with hindsight can exploit the knowledge of the sample path $x_1, \ldots, x_T$ to achieve a higher utility than (2). Specifically, defining ν_t^n the number of times file n is requested in slots $1, \ldots, t$, the best static policy will cache the C files with highest products $w^n \nu_T^n$. In the following, we characterize how this compares against the average utility of (2) by analyzing the order statistics of an Gaussian vector.

For i.i.d. x_t we may rewrite regret as the expected difference between the best static policy in hindsight and (2):

$$R_T = \mathbb{E}\left[\max_{y \in \mathcal{Y}} y^T \sum_{t=1}^T w \odot x_t\right] - \frac{CT}{\sum_n 1/w^n}, \qquad (3)$$

where $w \odot x_t = [w^1 x_t^1, w^2 x_t^2, ..., w^N x_t^N]^T$ is the Hadamard product between the weights and request vector. Further, (3) can be rewritten as a function:

$$R_T = \mathbb{E}[g_{N,C}(\overline{z}_T)] = \mathbb{E}\left[\max_{b \in \overset{\circ}{\mathcal{Y}}} \left[b^T \overline{z}_T\right]\right],$$

where, *(i)* $\overset{\circ}{\mathcal{Y}}$ is the set of all possible integer caching configurations (and therefore $g_{N,C}(.)$ is the sum of the maximum C elements of its argument); and *(ii)* the process $\overline{z}_T$ is the vector of utility obtained by each file after the first T rounds, centered around its mean:

$$\overline{z}_T = \sum_{t=1}^T w \odot x_t - w \odot \frac{T}{\sum_{n=1}^N 1/w^n} w^{-1}$$
$$= \sum_{t=1}^T \left(z_t - \frac{1}{\sum_{n=1}^N 1/w^n} \mathbf{1}_N\right) \qquad (4)$$

where $z_t = w \odot x_t$ are i.i.d. random vectors, with distribution

$$\mathbb{P}\left(z_t = w^i \mathbf{e}_i\right) = \frac{1/w^i}{\sum_{n=1}^N 1/w^n}, \forall t, \forall i,$$

and, therefore, mean $\mathbb{E}[z_t] = \frac{1}{\sum_{n=1}^N 1/w^n} \mathbf{1}_N$.[2] A key ingredient in our proof is the limiting behavior of $g_{N,C}(\overline{z}_T)$:

Lemma 1. *Let Z be a Gaussian vector $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma}(w))$, where $\mathbf{\Sigma}(w)$ is given in* (5)*, and $Z_{(i)}$ its $i-$th largest element. Then*

$$\frac{g_{N,C}(\overline{z}_T)}{\sqrt{T}} \xrightarrow[T\to\infty]{\text{distr.}} \sum_{i=1}^C Z_{(i)}.$$

Proof: Observe that $\overline{z}_T$ is the sum of T uniform i.i.d. zero-mean random vectors, where the covariance matrix can be calculated using (4): $\mathbf{\Sigma}(w) =$

$$= \mathbb{E}\left[\left(z_1 - \frac{1}{\sum_{n=1}^N 1/w^n}\mathbf{1}_N\right)\left(z_1 - \frac{1}{\sum_{n=1}^N 1/w^n}\mathbf{1}_N\right)^T\right]$$
$$= \frac{1}{\sum_{n=1}^N 1/w^n} \begin{cases} w_i - \frac{1}{\sum_{n=1}^N 1/w^n}, i = j \\ -\frac{1}{\sum_{n=1}^N 1/w^n}, i \neq j \end{cases}, \qquad (5)$$

where the second equality follows from the distribution of z_t and some calculations.[3] Due to the Central Limit Theorem:

$$\frac{\overline{z}_T}{\sqrt{T}} \xrightarrow[T\to\infty]{\text{distr.}} Z. \qquad (6)$$

Since $g_{N,C}(x)$ is continuous, (6) and the Continuous Mapping Theorem [29] imply

$$\frac{g_{N,C}\left(\overline{z}_T\right)}{\sqrt{T}} = g_{N,C}\left(\frac{\overline{z}_T}{\sqrt{T}}\right) \xrightarrow[T\to\infty]{\text{distr.}} g_{N,C}\left(Z\right),$$

and the proof is completed by noticing that $g_{N,C}(x)$ is the sum of the maximum C elements of its argument. ∎

An immediate consequence of Lemma 1, is that

$$\frac{R_T}{\sqrt{T}} = \frac{\mathbb{E}[g_{N,C}(\overline{z}_T)]}{\sqrt{T}} \xrightarrow{T\to\infty} \mathbb{E}\left[\sum_{i=1}^C Z_{(i)}\right] = \sum_{i=1}^C \mathbb{E}\left[Z_{(i)}\right] \qquad (7)$$

and the first part of the Theorem is proved.

To prove the second part, we remark that the RHS of (7) is the expected sum of C maximal elements of vector Z, and hence larger than the expected sum of any C elements of Z. In particular, we will compare with the following: Fix a permutation $\bar{\phi}$ over all N elements, partition the first $2C$

[2] Above we have used the notation $w^{-1} = \left[1/w^1, 1/w^2, ..., 1/w^N\right]^T$.

[3] For the benefit of the reader, we note that Z has no well-defined density (since $\mathbf{\Sigma}(w)$ is singular). For the proof, we only use its distribution.

elements in pairs by combining 1-st with 2-nd, ..., i-th with i+1-th, $2C$-1-th with $2C$-th, and then from each pair choose the maximum element and return the sum. We then obtain:

$$\mathbb{E}\left[\sum_{i=1}^{C} Z_{(i)}\right] \geq \mathbb{E}\left[\sum_{i=1}^{C} \max\left[Z^{\bar{\phi}(2(i-1)+1)}, Z^{\bar{\phi}(2i)}\right]\right] = \sum_{i=1}^{C} \mathbb{E}\Big[\max\left[Z^{\bar{\phi}((2(i-1)+1)}, Z^{\bar{\phi}(2i)}\right]\Big],$$

where the second step follows from the linearity of the expectation, and the expectation is taken over the marginal distribution of a vector with two elements of Z. We now focus on $\max\left[Z^k, Z^\ell\right]$ for (any) two fixed k, ℓ. We have that

$$(Z^k, Z^\ell)^T \sim \mathcal{N}\left(\mathbf{0}, \mathbf{\Sigma}(w_k, w_\ell)\right)$$

where $\mathbf{\Sigma}(w^k, w^\ell) =$

$$= \frac{1}{\sum_{n=1}^{N} 1/w^n} \begin{bmatrix} w^k - \frac{1}{\sum_{n=1}^{N} 1/w^n} & -\frac{1}{\sum_{n=1}^{N} 1/w^n} \\ -\frac{1}{\sum_{n=1}^{N} 1/w^n} & w^\ell - \frac{1}{\sum_{n=1}^{N} 1/w^n} \end{bmatrix}.$$

From [30] we then have:

$$\mathbb{E}\big[\max\left[Z^k, Z^\ell\right]\big] = \sqrt{\frac{1}{\sum_{n=1}^{N} 1/w^n}} \frac{1}{\sqrt{2\pi}} \sqrt{w^k + w^\ell},$$

therefore:

$$\mathbb{E}\left[\sum_{i=1}^{C} Z_{(i)}\right] \geq \frac{1}{\sqrt{2\pi}} \frac{\sum_{i=1}^{C} \sqrt{w^{\bar{\phi}((2(i-1)+1)} + w^{\bar{\phi}(2i)}}}{\sqrt{\sum_{n=1}^{N} 1/w^n}}, \quad (8)$$

for all $\bar{\phi}$. The result follows noticing that the tightest bound is obtained by maximizing (8) over all permutations. ∎

IV. Online Gradient Ascent

Gradient-based algorithms are widely used in resource allocation mechanisms due to their attractive scalability properties. Here, we focus on the online variant and show that, despite its simplicity, it achieves the best possible regret.

A. Algorithm Design and Properties

Recall that the utility in slot t is described by the linear function $f_t(y_t) = \sum_{n=1}^{N} w^n x_t^n y_t^n$. The gradient ∇f_t at y_t is an N-dimensional vector with coordinates:

$$\frac{\partial f_t}{\partial y_t^n} = w^n x_t^n, \; n = 1, \ldots, N.$$

Definition 2 (OGA)**.** *The* Online Gradient Ascent *(OGA) caching policy adjusts the decisions ascending in the direction of the gradient:*

$$y_{t+1} = \Pi_{\mathcal{Y}}\left(y_t + \eta_t \nabla f_t\right),$$

where η_t is the stepsize, and $\Pi_{\mathcal{Y}}(z) \triangleq \arg\min_{y \in \mathcal{Y}} \|z - y\|$ is the Euclidean projection of the argument vector z onto $\mathcal{Y}$, and $\|.\|$ the Euclidean norm.

The projection step is discussed in detail next. We emphasize that OGA bases the decision y_{t+1} on the caching configuration y_t and the most recent request x_t. Therefore, it is a very simple causal policy that does not require memory for storing the entire state (full history of x and y).

Let us now discuss the regret performance of OGA. We define first the set diameter $diam(\mathcal{S})$ to be the largest Euclidean distance between any two elements of set $\mathcal{S}$. To determine the diameter, we inspect two vectors $y_1, y_2 \in \mathcal{Y}$ which cache entire and totally different files and obtain:

$$diam(\mathcal{Y}) = \begin{cases} \sqrt{2C} & \text{if } 0 < C \leq N/2, \\ \sqrt{2(N-C)} & \text{if } N/2 < C \leq N. \end{cases}$$

Also, let L be an upper bound of $\|\nabla f_t\|$, we have $L \leq \max_n(\sum_n w^n x_t^n) \leq \max_n(w^n) \equiv w^{(1)}$.

Theorem 2 (Regret of OGA)**.** *Fix stepsize $\eta_t = \frac{diam(\mathcal{Y})}{L\sqrt{T}}$, the regret of* OGA *satisfies:*

$$R_T(\text{OGA}) \leq diam(\mathcal{Y}) L \sqrt{T}.$$

Proof: Using the non-expansiveness property of the Euclidean projection [31] we can bound the distance of the algorithm iteration from the best static policy in hindsight:

$$\|y^{t+1} - y^*\|^2 \triangleq \|\Pi_{\mathcal{Y}}\left(y_t + \eta_t \nabla f_t\right) - y^*\|^2 \leq \|y_t + \eta_t \nabla f_t - y^*\|^2 = \|y_t - y^*\|^2 + 2\eta_t \nabla {f_t}^T (y_t - y^*) + \eta_t^2 \|\nabla f_t\|^2,$$

where we expanded the norm. If we fix $\eta^t = \eta$ and sum telescopically over horizon T, we obtain:

$$\|y_T - y^*\|^2 \leq \|y_1 - y^*\|^2 + 2\eta \sum_{t=1}^{T} \nabla {f_t}^T (y_t - y^*) + \eta^2 \sum_{t=1}^{T} \|\nabla f_t\|^2.$$

Since $\|y_T - y^*\|^2 \geq 0$, rearranging terms and using $\|y_1 - y^*\| \leq diam(\mathcal{Y})$ and $\|\nabla f_t\| \leq L$:

$$\sum_{t=1}^{T} \nabla {f_t}^T (y^* - y_t) \leq \frac{diam(\mathcal{Y})^2}{2\eta} + \frac{\eta T L^2}{2}. \quad (9)$$

For f_t convex it holds $f_t(y_t) \geq f_t(y) + \nabla {f_t}^T (y_t - y)$, $\forall y \in \mathcal{Y}$, and with equality if f_t is linear. Plugging these in the OGA regret expression ($\max$ operator is removed) we get:

$$R_T(OGA) = \sum_{t=1}^{T} (f_t(y^*) - f_t(y_t)) = \sum_{t=1}^{T} \nabla {f_t}^T (y_t - y^*) \overset{(9)}{\leq} \frac{diam(\mathcal{Y})^2}{2\eta} + \frac{\eta T L^2}{2},$$

and for $\eta = diam(\mathcal{Y})/L\sqrt{T}$ we obtain the result. ∎

Using the above values of L and $diam(\mathcal{Y})$ we obtain:

$$R_T(OGA) \leq w^{(1)} \sqrt{2CT}, \quad \text{for } C < N/2\,.$$

Corollary 2 (Regret of Online Caching)**.** *Fix $C/N = \gamma$, $w^n = w$, for all n, and assume $C < N/2$, the regret of online caching satisfies:*

$$w\sqrt{\frac{\gamma}{\pi}} \sqrt{CT} \leq \min_{\sigma} R_T(\sigma) \leq w\sqrt{2}\sqrt{CT} \quad \text{as } T \to \infty.$$

Corollary 2 follows from Corol. 1 and Theorem 2. We conclude that disregarding $\sqrt{2\pi/\gamma}$ (amortized by T) OGA achieves the best possible regret and thus fastest learning rate.

B. Projection Algorithm

We explain next the Euclidean projection $\Pi_{\mathcal{Y}}$ used in OGA, which can be written as a constrained quadratic program:

$$\Pi_{\mathcal{Y}}(z) \triangleq \arg\min_{y \geq \mathbf{0}} \sum_{n=1}^{N} (z^n - y^n)^2 \quad \text{s.t.} \sum_{n=1}^{N} y^n \leq C \ \text{ and } y^n \leq 1, \ \forall n \in \mathcal{N}. \tag{10}$$

In practice N is expected to be large, and hence we require a fast algorithm. Let us introduce the Lagrangian:

$$L(y, \rho, \mu, \kappa) = \sum_{n=1}^{N} (z^n - y^n)^2 + \rho\Big(\sum_{n=1}^{N} y^n - C\Big) + \sum_{n=1}^{N} \mu_n (y^n - 1) - \sum_{n=1}^{N} \kappa_n y_n,$$

where ρ, μ, κ are the Lagrangian multipliers. The KKT conditions of (10) ensure that the values of y^n at optimality will be partitioned into three sets $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$:

$$\mathcal{M}_1 = \{n \in \mathcal{N} : y^n = 1\}, \ \ \mathcal{M}_2 = \{n \in \mathcal{N} : y^n = z^n - \rho/2\}, \\ \mathcal{M}_3 = \{n \in \mathcal{N} : y^n = 0\}, \tag{11}$$

where $\rho = 2\big(|\mathcal{M}_1| - C + \sum_{n \in \mathcal{M}_2} z^n\big)/|\mathcal{M}_2|$ follows from the tightness of the simplex constraint. It suffices for the projection to determine a partition of files into these sets. Note that given a candidate partition, we can check in linear time whether it satisfies all KKT conditions (and only the unique optimal partition does). Additionally, one can show that the ordering of files in z is preserved at optimal y, hence a known approach is to search exhaustively over all possible ordered partitions, which takes $O(N^2)$ steps [32]. For our problem, we propose Algorithm 1, which exploits the property that all elements of z satisfy $z^n \leq 1$ except at most one (hence also $|\mathcal{M}_1| \in \{0, 1\}$), and computes the projection in $O(N \log N)$ steps (where the term $\log N$ comes from sorting z). In our simulations each loop is visited at most two times, and the OGA simulation takes comparable time with LRU.

Finally, the projection algorithm provides insight into OGA functionality. In a slot where file n' has been requested, OGA will increase $y^{n'}$ according to the stepsize, and then decrease all other variables $y_n > 0, n \neq n'$ symmetrically until the simplex constraint is satisfied.

C. Performance and Relation to Other Policies

Fig. 3(a) showcases the hit ratio of OGA for different choices of fixed step sizes, where it can be seen that larger steps lead to faster but more inaccurate convergence. The horizon-optimal step is given in Theorem 2 as $\eta^* = w\sqrt{2C/T}$, and plugging in $C = 1000$, $w = 1$ and $T = 2 \cdot 10^5$, we obtain $\eta^* = 0.1$; indeed we see that our experiments verify this.

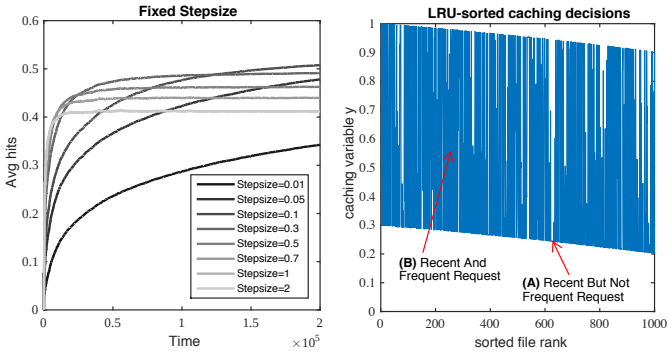
The online gradient descent (similar to OGA) is identical to the well-known *Follow-the-Leader* (FtL) policy with a Euclidean regularizer $\frac{1}{2\eta_t}\|y\|$, cf. [2], where FtL chooses in slot t the configuration that maximizes the average utility in slots $1, 2, \ldots, t-1$. We may observe that FtL applied here would cache the files with the highest frequencies, hence it is identical to LFU (when the frequency starts counting from $t = 0$). Hence, OGA can be seen as a regularized version of a utility-LFU policy, where additionally to largest frequencies we smoothen the decisions by a Euclidean regularizer.

Furthermore, OGA for $\eta = 1$, $w^n = w$ bears similarities to LRU, since recent requests enter the cache at the expense of older requests. Since the Euclidean projection drops some chunks from each file, we expect least recent requests to drop first in OGA. The difference is that OGA evicts files gradually, chunk by chunk, and not in one shot. Fig. 3(b) shows the values $y^n(OGA)$ for all files in the LRU cache (the $C = 1K$ most recently used). This reveals that the two policies make strongly correlated decisions, but OGA additionally "remembers" which of the recent requests are also infrequent (e.g., see point (A) in Fig. 3(b)), and decreases accordingly the y^n values (as LFU would have done).

Finally, in Fig. 4 we compare the performance of OGA to LRU, LFU, and the best in hindsight static configuration. We perform the comparison for catalogues of 10K files, with

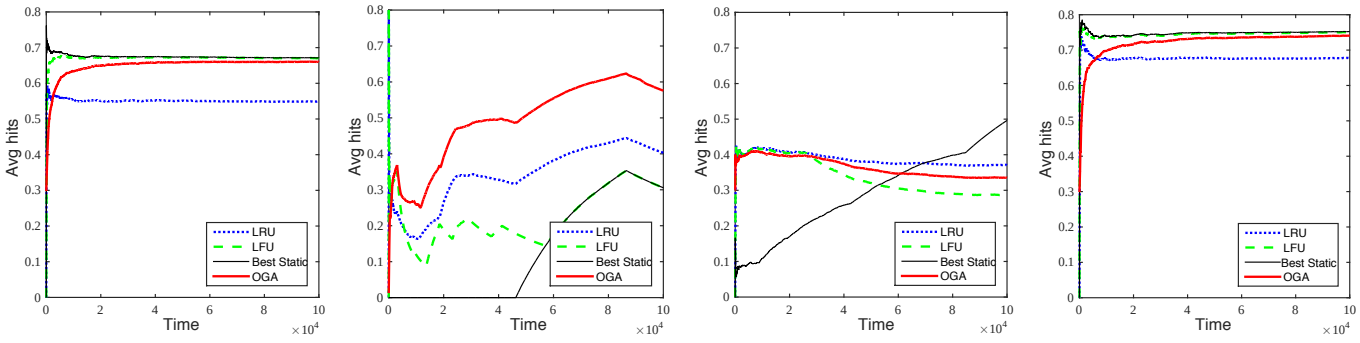
Algorithm 1 Projection on Capped Simplex

Require: C; sorted $z^{(1)} \geq \cdots \geq z^{(N)}$
Ensure: $y = \Pi_{\mathcal{Y}}(z)$
$\mathcal{M}_1 \leftarrow \emptyset, \mathcal{M}_2 \leftarrow \{1, \ldots, N\}, \mathcal{M}_3 \leftarrow \emptyset$
repeat
  $\rho \leftarrow 2(|\mathcal{M}_1| - C + \sum_{n \in \mathcal{M}_2} z^n)/|\mathcal{M}_2|$
  $y^n \leftarrow \begin{cases} 1 & n \in \mathcal{M}_1, \\ z^n - \rho/2 & n \in \mathcal{M}_2, \\ 0 & n \in \mathcal{M}_3 \end{cases}$
  $\mathcal{S} \leftarrow \{n \in \mathcal{N} : y^n < 0\}$
  $\mathcal{M}_2 \leftarrow \mathcal{M}_2 \setminus \mathcal{S}, \mathcal{M}_3 \leftarrow \mathcal{M}_3 \cup \mathcal{S}$
until $\mathcal{S} = \emptyset$
if $y^1 > 1$ **then**
  $\mathcal{M}_1 \leftarrow \{1\}, \mathcal{M}_2 \leftarrow \{2, \ldots, N\}, \mathcal{M}_3 \leftarrow \emptyset$
  Repeat 2-7
end if % KKT conditions are satisfied.



(a) Fixed stepsize (b) Sorted based on LRU

Fig. 3. Single cache with $N = 10^4$, $C = 10^3$. (a) Smaller stepsizes converge slower, but more accurately. (b) For each file in the LRU cache, we show the respective OGA caching variable.



(a) CDN aggregation (IRM model) (b) Youtube videos (model [5]) (c) Web browsing (trace [33]) (d) Ephemeral Torrents (model [11])

Fig. 4. Average hits under different request models [34]; (a) i.i.d. Zipf, (b) Poisson Shot Noise [5], (c) web browsing dataset [33], (d) random replacement [11]; Parameters: $\gamma = 0.3, T = 2 \times 10^5, \eta = 0.1$.

a cache that fits 3K files, and we use four different request models: *(a)* an i.i.d. Zipf model that represents requests in a CDN aggregation point [34]; *(b)* a Poisson shot noise model that represents ephemeral YouTube video requests [5]; *(c)* a dataset from [33] with actual web browsing requests at a university campus; and *(d)* a random replacement model from [11] that represents ephemeral torrent requests. We observe that OGA performance is always close to the best among LFU and LRU. The benefits from the second best policy here is as high as 16% over LRU and 20% over LFU.

V. Bipartite Online Caching

We extend now our study to a network of caches reachable by the user population via a weighted bipartite graph. Bipartite caching was first used for small cell networks (see the seminal femtocaching model [24]), and subsequently also to model a variety of wired and wireless caching problems where the bipartite graph links represent delay, cost or other QoS metrics when users accessing different caches [35].

A. Bipartite Caching Model

Our caching network (CN) comprises a set of user locations $\mathcal{I} = \{1, 2, \ldots, I\}$ served by a set of caches $\mathcal{J} = \{1, 2, \ldots, J\}$, each with capacity C_j, $j \in \mathcal{J}$. We use parameters:

$$\boldsymbol{\ell} = \big(\ell_{ij} \in \{0,1\} : i \in \mathcal{I}, j \in \mathcal{J}\big),$$

to denote whether cache j is reachable from location i, or not ($\ell_{ij} = 0$). This includes the general case where a location is connected to multiple caches (e.g., consider base stations with overlapping coverage). We also maintain the origin server as a special node (indexed with $j=0$), which contains the entire library, and serves the requests for files not found in the caches. The request process is a sequence of vectors with element $x_t^{n,i} = 1$ if a request for file n arrives at $i \in \mathcal{I}$ in slot t. At each slot t there is only one request $\sum_{n,i} x_t^{n,i} = 1$.

We perform caching using the standard model of *Maximum Distance Separable* (MDS) codes [24], where each stored chunk is a pseudo-random linear combination of original file chunks, and a user requires a fixed number of such chunks to decode the file. Furthermore, we will populate the caches with different random chunks such that, following from the MDS properties, the collected chunks will be linearly independent with high probability (and therefore complement each other for decoding). This results in the following model: the caching decision vector y_t has $N \times J$ elements, and each element $y_t^{n,j} \in [0,1]$ denotes the amount of random equations of file n stored at j. The set of eligible caching vectors is convex:

$$\mathcal{Y}_{\mathcal{J}} = \left\{ y \in [0,1]^{N \times J} \;\middle|\; \sum_{n=1}^{N} y^{n,j} \leq C_j, \; j \in J \right\}.$$

The caching policy σ can be defined as follows:

$$\sigma : (x_1, x_2, \ldots, x_{t-1}, y_1, y_2, \ldots, y_{t-1}) \longrightarrow y_t \in \mathcal{Y}_{\mathcal{J}}\,. \quad (12)$$

Since each location i might be connected to multiple caches, we use *routing variables* $z_t^{n,i,j}$ to describe how a request is served at each location. The caching and routing decisions are coupled and constrained: *(i)* a request cannot be routed to an unreachable cache, *(ii)* we cannot route from a cache more data than it has, and *(iii)* each request must be fully routed. Hence, the eligible routing decisions under y_t are:

$$\mathcal{Z}(y_t) = \left\{ z \in [0,1]^{N \times I \times J} \;\middle|\; \begin{array}{l} \sum_{j \in \mathcal{J} \cup \{0\}} z_t^{n,i,j} = x_t^{n,i}, \;\; \forall n, i \\ z_t^{n,i,j} \leq \ell_{ij} y^{n,j}, \;\; \forall n, i, j \in \mathcal{J} \end{array} \right\},$$

where $z_t^{n,i,0}$ (the routing to origin) is unconstrained, and hence the set $\mathcal{Z}(y_t)$ is non-empty for all $y_t \in \mathcal{Y}_{\mathcal{J}}$ (i.e. $\sum_{j \in \mathcal{J} \cup \{0\}} z_t^{n,i,j} = x_t^{n,i}$ can always be satisfied). Naturally, we assume that $z_t \in \mathcal{Z}(y_t)$ is decided after the requests arrive, and hence also after the caching policy.

Finally, we introduce utility weights $w^{n,i,j}$ to denote the obtained benefit by retrieving a unit fraction of a file from cache j instead of the origin, and trivially $w^{n,i,0} \equiv 0$. Apart from file importance, these weights also model the locality of caches within the geography of user locations–for example a cache might have higher benefit for certain locations and lower for others. In sum, the total utility accrued in slot t is:

$$f_t(y_t) = \max_{z \in \mathcal{Z}(y_t)} \sum_{n=1}^{N} \sum_{i=1}^{I} \sum_{j=1}^{J} w^{n,i,j} x_t^{n,i} z_t^{n,i,j}, \quad (13)$$

where the index t is used to remind us that f_t is affected by the adversary's decision $x_t^{n,i}$. The regret of policy σ is:

$$R_T(\sigma) = \max_{P(x_1,\ldots,x_T)} \mathbb{E}\left[\sum_{t=1}^{T} f_t(y^*) - \sum_{t=1}^{T} f_t(y_t(\sigma))\right],$$

where y^* is the best static configuration in hindsight (factoring the associated routing).

Next we establish the concavity of our objective function $f_t(y)$ for each slot t. Since there is only one request at each slot t, we can simplify the form of $f_t(y)$. Let $\hat{n}, \hat{i}$ be the file and location where the request in t arrives. Then $f_t(y_t)$ is zero except for $x_t^{\hat{n},\hat{i}}$. Denoting with $\mathcal{J}^*$ the set of reachable caches from $\hat{i}$, and simplifying the notation, (13) reduces to:

$$f(y) = \max_{z \geq 0} \sum_{j \in \mathcal{J}^*} w^j z^j \tag{14}$$

$$\text{s.t.} \sum_{j \in \mathcal{J}^*} z^j \leq 1 \tag{15}$$

$$z^j \leq \begin{cases} y^j & j \in \mathcal{J}^* \\ 0 & j \notin \mathcal{J}^*. \end{cases} . \tag{16}$$

Lemma 2. *The function $f(y)$ is concave in its domain $\mathcal{Y}_{\mathcal{J}}$.*

Proof: Let us consider two feasible caching vectors $y_1, y_2 \in \mathcal{Y}_{\mathcal{J}}$, our goal is to show that:

$$f(\lambda y_1 + (1-\lambda)y_2) \geq \lambda f(y_1) + (1-\lambda) f(y_2), \ \forall \lambda \in [0,1].$$

We begin by denoting with z_1 and z_2 the routing maximizers of (14) for vectors y_1, y_2 respectively. Immediately, it is $f(y_i) = \sum_j w^j z_i^j, \ i = 1, 2$. Next, consider a candidate vector $y_3 = \lambda y_1 + (1-\lambda) y_2$ for some $\lambda \in [0,1]$. We first show that the routing $z_3 = \lambda z_1 + (1-\lambda) z_2$ is a feasible routing for y_3, i.e., that $z_3 \in \mathcal{Z}(y_3)$; by the feasibility of z_1, z_2, we have:

$$\sum_j z_3^j = \sum_j (\lambda z_1^j + (1-\lambda) z_2^j) = \lambda \sum_j z_1^j + (1-\lambda) \sum_j z_2^j = 1,$$

which proves z_3 satisfies (15). Further, for all j, it is:

$$z_3^j = \lambda z_1^j + (1-\lambda) z_2^j \leq \lambda y_1^j + (1-\lambda) y_2^j = y_3^j,$$

which proves that z_3 also satisfies (16); hence $z_3 \in \mathcal{Z}(y_3)$. It follows that $f(y_3) \equiv \max_{z \in \mathcal{Z}(y_3)} \sum_j w^j z^j \geq \sum_j w^j z_3^j$. Combining:

$$f(\lambda y_1 + (1-\lambda) y_2) = f(y_3) \geq \sum_j w^j z_3^j = \tag{17}$$

$$\lambda \sum_j w^j z_1^j + (1-\lambda) \sum_j w^j z_2^j = \lambda f(y_1) + (1-\lambda) f(y_2). \blacksquare$$

B. Bipartite Subgradient Algorithm (BSA)

Since $f(y)$ is concave, our plan is to design a universal online bipartite caching policy by extending OGA. Note, however, that $f(y)$ is not necessarily differentiable everywhere (hence the gradient might not exist at y_t), and hence we need to find a subgradient direction at each slot, as explained next.

Consider the partial Lagrangian of (14):

$$L(y, z, \alpha, \beta) = \sum_j w^j z^j + \alpha\big(\sum_j z^j - 1\big) + \sum_j \beta^j (z^j - y^j),$$

and define the function $\Lambda(y, \beta) = L(y, z^*, \alpha^*, \beta) \equiv \min_{\alpha \geq 0} \max_{z \geq 0} L(y, z, \alpha, \beta)$. From strong duality we obtain:

$$f(y) = \min_{\beta \geq 0} \Lambda(y, \beta). \tag{18}$$

Lemma 3 (Supergradient)**.** *Let $\beta^*(y) \triangleq \arg\min_{\beta \geq 0} \Lambda(y, \beta)$ be the vector of optimal multipliers of* (16)*. Define:*

$$g^j = \begin{cases} \beta^{j,*}(y) & j \in \mathcal{J}^* \\ 0 & j \notin \mathcal{J}^*, \end{cases}$$

The vector g is a supergradient of f at y, i.e., it holds $f(y) \geq f(y') - g^T(y' - y), \ \forall y' \in \mathcal{Y}_{\mathcal{J}}$.

Proof: We have

$$\begin{aligned} f(y) &\overset{(18)}{=} \min_{\beta \geq 0} \Lambda(y, \beta) \equiv \Lambda(y, \beta^*(y)) \\ &\overset{(a)}{=} \Lambda(y', \beta^*(y)) - \beta^*(y)^T (y' - y) \\ &\overset{(b)}{\geq} \Lambda(y', \beta^*(y')) - \beta^*(y)^T (y' - y) \\ &= f(y') - \beta^*(y)^T (y' - y) \end{aligned}$$

where (a) is obtained directly by the form of the Lagrangian $L(\cdot)$ where only one term depends on y, and in (b) we have used that y' minimizes $\Lambda(;, \beta^*(y'))$. ■

Now that we have found a method to calculate the supergradient, we can extend OGA as follows:

Definition 3 (BSA)**.** *The Bipartite Subgradient Algorithm caching policy adjusts the decisions with the supergradient:*

$$y_{t+1} = \Pi_{\mathcal{Y}_{\mathcal{J}}} (y_t + \eta_t g_t),$$

where η_t is the step, g_t is given in Lemma 3, and $\Pi_{\mathcal{Y}_{\mathcal{J}}}(z) \triangleq \arg\min_{y \in \mathcal{Y}_{\mathcal{J}}} \|z - y\|$ is the Euclidean projection of the argument vector z onto $\mathcal{Y}_{\mathcal{J}}$, performed with Algorithm 1.

Theorem 3 (Regret of BSA)**.** *Let $C_j = C < N/2$. The regret of* BSA *satisfies:*

$$R_T(\text{BSA}) \leq w^{(1)} \sqrt{2 \text{deg} JCT}.$$

Proof: Replacing the gradient with the supergradient and repeating the steps of Theorem 2 proof, we arrive at:

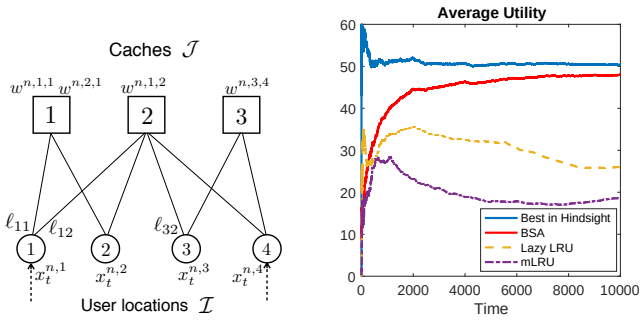
$$\sum_{t=1}^{T} g_t^T (y^* - y_t) \leq \frac{diam(\mathcal{Y}_{\mathcal{J}})^2}{2\eta} + \frac{\eta T L^2}{2}, \tag{19}$$

where it holds $f_t(y_t) \geq f_t(y) + g_t^T (y_t - y), \forall y \in \mathcal{Y}_{\mathcal{J}}$. Plugging these in the BSA regret expression we get:

$$\begin{aligned} R_T(\text{BSA}) &= \sum_{t=1}^{T} (f_t(y^*) - f_t(y_t)) = \sum_{t=1}^{T} g_t^T (y_t - y^*) \\ &\overset{(19)}{\leq} \frac{diam(\mathcal{Y}_{\mathcal{J}})^2}{2\eta} + \frac{\eta T L^2}{2}, \end{aligned}$$

where, $diam(\mathcal{Y}_{\mathcal{J}}) = \sqrt{\sum_{j \in \mathcal{J}} C_j} = \sqrt{2CJ}$ (for $C_j = C < N/2$) and L is an upper bound on the supergradient norm:

$$\|g\| \leq \sqrt{\sum_{j : \ell_{ij} = 1} (w^{(1)} - w^{(J)})^2} = w^{(1)} \sqrt{\text{deg}} \triangleq L,$$



(a) Bipartite scenario (b) Simulation results

Fig. 5. (a) A bipartite caching example with 3 caches and 4 user locations. (b) Average utility of different caching policies.

where deg is the maximum right degree and $w^{(1)}$ the maximum utility, and we used the optimal constant stepsize. ■

The functionality of BSA is similar to OGA. In a slot where file n' has been requested from location i', BSA will increase variables $y^{n',j}$, where j is reachable by i', and then each cache j will perform a local projection step, decreasing all other variables $y^{n,j} > 0$ until the local cache constraint is satisfied. The difference from OGA lies only in how we compute the $y^{n',j}$ increase. This now depends on the corresponding subgradient element, which is proportional to the hypothetical utility gain if we would increase $y^{n',j}$ by δy and we would launch another request for n' from the same location i'.

We have performed extensive simulations of BSA and selected to present a representative scenario in Fig. 5(a). The bipartite graph is composed of 3 caches with $C = 10$, and 4 user locations. The caches have utilities (1,2,100), and a good caching policy should place popular content on cache 3. The network is fed with stationary Zipf requests from a catalogue of 100 files, and each request arrives at a user location uniformly at random. We compare BSA to the best static policy in hindsight (we may verify BSA has no regret), and literature heuristics mLRU from [22] and q-LRU with the lazy rule from [23] (for $q = 1$). Over time, BSA learns what files are popular and increases their cache allocation at the high utility cache, while adjusting accordingly the less popular files in other caches. Doing so, BSA outperforms lazy-LRU, which was the best heuristic in our simulations, by 45.8%.

VI. Conclusions

We established a connection between caching policies such as LRU, LFU and their variants, and the framework of OLO. Our analysis sheds light from a different angle to these established (but based on intuition) policies, and hence allows us not only to characterize their performance but also to re-design them. To this end, we proposed the online gradient caching algorithm and proved that it has universally optimal performance; and followed a similar OLO-based approach for bipartite caching networks where routing decisions are also considered. We believe our work opens a very exciting area, paving the road for the principled analysis and design of dynamic caching policies for single or networks of caches.

References

[1] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *ICML*, 2003.

[2] S. Shalev-Shwartz, *Online Learning and Online Convex Optimization*. Now Publishers Inc., 2012.

[3] E. Belmega, P. Mertikopoulos, R. Negrel, and L. Sanguinetti, "Online convex optimization and no-regret learning: Algorithms, guarantees and applications," arXiv:1804.04529, Tech. Rep., 2018.

[4] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in *ITC*, 2012.

[5] S. Traverso *et al.*, "Temporal locality in today's content caching: Why it matters and how to model it," *ACM SIGCOMM*, 2013.

[6] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, pp. 202–208, 1985.

[7] L. A. Belady, "A study of replacement algorithms for virtual storage computers," in *IBM Systems Journal*, 1966.

[8] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," in *IBM Systems Journal*, 1970.

[9] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, 1998.

[10] M. Leconte *et al.*, "Placing dynamic content in caches with small population," in *IEEE INFOCOM*, 2016.

[11] S.-E. Elayoubi and J. Roberts, "Performance and cost effectiveness of caching in mobile access networks," in *ACM ICN*, 2015.

[12] F. Olmos, B. Kauffmann, A. Simonian, and Y. Carlinet, "Catalog dynamics: Impact of content publishing and perishing on the performance of a LRU cache," in *ITC*, 2014.

[13] S. O. Somuyiwa, A. Gyorgy, and D. Gunduz, "A reinforcement-learning approach to proactive caching in wireless networks," *IEEE JSAC*, vol. 36, no. 6, 2018.

[14] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5G using reinforcement learning of space-time popularities," *IEEE J. of S. Top. in Signal Proc.*, pp. 180–190, 2018.

[15] S. Li, J. Xu, M. Schaar, , and W. Li, "Trend-aware video caching through online learning," *IEEE Trans. Multimedia*, pp. 2503–2516, 2016.

[16] G. Sascha *et al.*, "Regret minimization for online buffering problems using the weighted majority algorithm." in *COLT*, 2010.

[17] M. Englert, H. Röglin, J. Spönemann, and B. Vöcking, "Economical caching," *ACM Trans. on Computation Theory*, pp. 4:1–4:21, 2013.

[18] T. Lykouris and S. Vassilvitskii, "Competitive caching with machine learning advice," in *ICML*, 2018.

[19] S. Ioannidis and E. Yeh, "Jointly optimal routing and caching for arbitrary network topologies," in *ACM ICN*, 2017.

[20] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation algorithms for mobile data caching in small cell networks," *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3665–3677, 2014.

[21] J. Kwak, G. Paschos, and G. Iosifidis, "Dynamic cache rental and content caching in elastic wireless cdns," in *Proc. of WiOpt*, 2018.

[22] A. Giovanidis and A. Avranas, "Spatial multi-LRU: Distributed caching for wireless networks with coverage overlaps," *arXiv:1612.04363*, 2016.

[23] E. Leonardi and G. Neglia, "Implicit coordination of caches in small cell networks under unknown popularity profiles," *IEEE JSAC*, vol. 36, no. 6, 2018.

[24] N. Golrezaei *et al.*, "Femtocaching: Wireless video content delivery through distributed caching helpers," in *IEEE INFOCOM*, 2012.

[25] L. Maggi *et al.*, "Adapting caching to audience retention rate: Which video chunk to store?" *Computer Comm.*, pp. 159–171, 2018.

[26] B. Blaszczyszyn and A. Giovanidis, "Optimal geographic caching in cellular networks," *arXiv preprint arXiv:1409.7626*, 2014.

[27] J. Abernethy *et al.*, "Optimal strategies and minimax lower bounds for online convex games," in *COLT*, 2008.

[28] E. Hazan, "Efficient algorithms for online convex optimization and their applications," Ph.D. dissertation, Princeton University, 2006.

[29] P. Billingsley, *Convergence of Probability Measures*. Wiley&Sons, 1999.

[30] C. E. Clark, "The greatest of a finite set of random variables," *Oper. Res.*, pp. 145–162, 1961.

[31] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.

[32] W. Wang and C. Lu, "Projection onto the capped simplex," *arXiv preprint arXiv:1503.01002*, 2015.

[33] M. Zink *et al.*, "Watch global, cache local: Youtube network traffic at a campus network: measurements and implications," in *SPIE*, 2008.

[34] C. Fricker *et al.*, "Impact of traffic mix on caching performance in a content-centric network," in *IEEE INFOCOM Workshops*, 2012.

[35] G. Paschos *et al.*, "The role of caching in future communication systems and networks," *IEEE JSAC*, vol. 36, no. 6, 2018.