Optimistic No-regret Algorithms for Discrete Caching

NARAM MHAISEN, Delft University of Technology, The Netherlands
ABHISHEK SINHA, Tata Institute of Fundamental Research, India
GEORGIOS PASCHOS, Amazon, Luxembourg
GEORGE IOSIFIDIS, Delft University of Technology, The Netherlands

We take a systematic look at the problem of storing whole files in a cache with limited capacity in the context of optimistic learning, where the caching policy has access to a prediction oracle (provided by, e.g., a Neural Network). The successive file requests are assumed to be generated by an adversary, and no assumption is made on the accuracy of the oracle. In this setting, we provide a universal lower bound for prediction-assisted online caching and proceed to design a suite of policies with a range of performance-complexity trade-offs. All proposed policies offer sublinear regret bounds commensurate with the accuracy of the oracle. Our results substantially improve upon all recently-proposed online caching policies, which, being unable to exploit the oracle predictions, offer only $O(\sqrt{T})$ regret. In this pursuit, we design, to the best of our knowledge, the first comprehensive optimistic Follow-the-Perturbed leader policy, which generalizes beyond the caching problem. We also study the problem of caching files with different sizes and the bipartite network caching problem. Finally, we evaluate the efficacy of the proposed policies through extensive numerical experiments using real-world traces.

CCS Concepts: • Networks → Network performance analysis.

Additional Key Words and Phrases: online algorithms; optimistic learning; caching; regret bounds.

ACM Reference Format:
Naram Mhaisen, Abhishek Sinha, Georgios Paschos, and George Iosifidis. 2022. Optimistic No-regret Algorithms for Discrete Caching. Proc. ACM Meas. Anal. Comput. Syst. 6, 3, Article 48 (December 2022), 28 pages. https://doi.org/10.1145/3570608

1 INTRODUCTION

This paper addresses the discrete caching (prefetching) problem: choose files to replicate in a local cache in order to maximize the probability that a new file request is served locally. Hitting the cache speeds up CPU, optimizes user experience in CDN’s [8], and enhances the performance of wireless networks [63]. With the perpetual growth of Internet traffic fueled by new services such as AR/VR [18], caching policies that learn fast to maximize cache hits can mitigate the increasing costs of information transportation [54], and similar benefits can be expected for embedded and other computing systems [23]. This work aspires to advance our theoretical understanding of this fundamental problem and proposes new provably-optimal and computationally-efficient caching algorithms using a new modeling and solution approach based on optimistic learning.

1.1 Motivation

Common caching policies store the newly requested files and employ the Least-Recently-Used (LRU) [32], Least-Frequently-Used (LFU) [37] and other similar rules to evict files when the cache capacity is exhausted. Under certain statistical assumptions on the request trace, such policies...
where we characterize the policy’s performance by using the static regret $\Delta$. Then, a policy can be selected only with access to future requests. Here, the learner makes no statistical assumptions and receives a reward of $f_t(x_t)$ for cache hits. The reward is revealed only after committing $x_t$, which naturally matches the dynamic caching operation where the cached files are decided before the next request arrives. Here, the learner makes no statistical assumptions and can follow any distribution, even one that is handpicked by an adversary. In the optimistic framework, the learner does not only consider its hit or miss performance so far when deciding $x_t$, but also the predictor’s performance and output (Fig. 1 right). As customary in the online learning literature, we characterize the policy’s performance by using the static regret metric:

$$R_T\{\{x_t\}_{T}\} \triangleq \sup_{\{f_t\}_{T=1}^T} \left\{ \sum_{t=1}^{T} f_t(x^*) - \sum_{t=1}^{T} f_t(x_t) \right\},$$

where $x^*=\arg\max_{x \in X} \sum_{t=1}^{T} f_t(x)$ is the (typically unknown) best-in-hindsight cache decision that can be selected only with access to future requests. The regret measures the accumulated reward gap between the online decisions $\{x_t\}_T$ and benchmark $x^*$. An algorithm is said to achieve sublinear regret when its average performance gap $R_T / T$ vanishes as $T \to \infty$. In this context, recent works have proposed caching policies that offer $O(\sqrt{T})$ regret bound [12, 44, 45, 51, 53, 64, 65], which, in fact, is the optimal (as small as possible) achievable regret rate, see [50, Thm. 5.1], [12, Thm. 1].

Most of these regret-optimal algorithms have been designed for continuous caching, where it is assumed that each file is encoded and divided into a large number of small chunks such that storing them can be approximated by continuous variables [40]. In this case, the set of eligible caching states $X$ is convex and hence one can readily apply standard OCO algorithms such as the Online Gradient Ascent (OGA). Albeit a handy assumption, there are settings where continuous caching cannot be used for practical reasons. Namely, keeping chunk meta-data consumes non-negligible storage; the coding operation is often computationally demanding; and the number of chunks might not be big enough to render continuous caching a good approximation. Therefore, we consider here the more realistic, and more challenging to solve, discrete caching problem. Indeed, in discrete caching the set $X$ is naturally non-convex (containing binary file-caching decisions) and thus

\footnote{It is interesting to note that $x^*$ caches the most frequent requests, which coincides with the limit behavior of LFU.}
standard OCO policies cannot be employed. While first steps in the study of discrete caching, with equal-sized files, were recently made by [12, 51, 64]. In this paper, we extend their scope and design algorithms with substantially improved performance guarantees.

Namely, while regret minimization yields robust policies that learn under adversarial conditions, this framework receives the fair criticism that the policies have often suboptimal performance when the requests (cost functions, in general) are predictable, e.g., stationary. In such situations, we would like the policy to gauge the predictability of requests, and optimize aggressively the cache. For instance, requests in services like Facebook are often amenable to accurate forecasts; while in YouTube and Netflix the viewers receive recommendations which can effectively serve as predictions for their forthcoming requests [4, 25]. Unfortunately, regret-based caching policies, such as [12, 38, 51, 53, 64, 65], are pessimistically designed for the worst-case request sequence and cannot benefit from predictable requests. We tackle this shortcoming by designing a new suite of optimistic caching algorithms. An optimistic algorithm [46, 56] has access to a prediction of an unknown quality for the next-slot utility function. The ultimate goal is to achieve constant (independent of $T$) regret when the predictions are accurate, while maintaining the worst-case regret bounds when predictions fail. This best-of-two-worlds approach, we show theoretically and demonstrate numerically, brings significant performance gains to dynamic caching.

1.2 Methodology and Contributions

We study key variants of the discrete caching problem, namely the single cache with equal or unequal-sized files and the bipartite caching, and propose a suite of optimistic learning algorithms with different pros and cons. Our first result demonstrates the best achievable regret in the setup we consider, which turns out to be $R_T = \Omega(\|\sum_{t} (\theta_t - \tilde{\theta}_t)\|^{1/2})$, indicating a significant potential of obtaining a regret that scales with the predictor’s error rather than the time horizon $T$ (Sec. 3). We then proceed to propose variants of the seminal Follow-The-Regularized-Leader (FTRL) and Follow-the-Perturbed-Leader (FTPL) algorithms, which can be both viewed as smoothing techniques for stabilizing learning decisions (Sec. 2.2), whose regret match this lower bound up to constants. In detail, we expand the optimistic FTRL algorithm [44–46] that was designed for convex problems, to handle, through sampling, discrete (non-convex) decisions (Sec. 4). We prove this approach attains expected regret $O(\sqrt{T})$ for worst-case predictions and zero-regret for perfect predictions with an improved prefactor that does not depend on library size $N$. However, the OFTRL implementation can be hindered by an involved projection step that might be computationally expensive. Thus, we develop a new optimistic FTPL algorithm that applies prediction-adaptive perturbations to achieve a similar regret bound with linear ($O(N)$) computation overhead (Sec. 5). The flip side is that its regret bound contains $O(\text{poly-log}(N))$ term.

We first derive results for equal-sized files, in line with all prior learning-based works for discrete caching [12, 39, 51, 58] or continuous caching [44, 53, 64]. Subsequently, we drop this assumption and study the single cache problem with different file sizes (Sec. 6). These first-of-their-kind regret-based algorithms require a point-wise approximation scheme for solving efficiently the NP-Hard Knapsack instance at each slot, while keeping the accumulated regret bound sublinear. To that end, we use the help of a rounding subroutine, DepRound [13], to a known almost-discrete optimal solution Dantz [19]. We show that the proposed policies achieve $(1/2)$-approximate regret of $O(\sqrt{T})$ and zero-regret for adversarial and perfect predictions, respectively. We also extend the OFTRL

---

2 Optimistic learning was originally proposed for problems with slowly-varying (hence, predictable) cost functions [56]; in caching, we note the additional motivation coming from the abundance of forecasting models, e.g., by a Neural Network.

3 In some cases the projection can be optimized, but in general it is $O(N^2)$ even for the non-weighted capped simplex [71].
analysis to the widely used bipartite network caching model [54, 63] (Sec. 7), where we optimize jointly the discrete caching and routing decisions to obtain prediction-modulated performance.

In (Sec. 8), we change tack and incorporate the optimism through the celebrated Experts model. The caching system in this case is a meta-learner which receives caching advice from an optimistic expert that suggests to cache solely w.r.t. predicted requests, and from a pessimistic expert that ignores predictions. We propose a tailored OGD-based scheme that allows the meta-learner to adapt to predictions’ accuracy (performance of the optimistic expert) in a way that achieves negative regret when that expert is reliable, and, again, maintains an $O(\sqrt{T})$ regret for unreliable predictions.

In summary, we provide a comprehensive toolbox of algorithms having different computation overheads and performance, hence enabling practitioners to select the best approach to their problem. Moreover, we include technical results that are of independent interest, such as the non-convex OFTPL algorithm with improved regret bounds; the approximate non-convex OFTRL algorithm for the Knapsack problem; and an analysis of OFTRL/OFTPL with a probabilistic prediction model.

**Notation.** We denote sets with calligraphic capital letters, e.g., $N = \{1, 2, \ldots, N\}$; vectors with $x = (x_i, i \in N)$ where $x_i$ is the $i$th component; and denote $x^t_i$ the $i$th component of the time-indexed vector $x^t$. The shorthand notation $x^{1:t}$ is used for $\sum_{i=1}^t x_i$. Also, $\{x^t_i\}_{i=1}^k$ denotes the sequence of vectors $\{x_1, x_2, \ldots, x_k\}$, and we use the succinct version $\{x^t_i\}$ for $\{x_1, x_2, \ldots, x_T\}$. When clear from context, we often drop the notation of actions and denote the regret $R_T(\{x^t_i\})$ simply with $R_T$.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Caching and Learning

Research on caching optimization spans several decades and we refer the reader to survey [52] for an introduction to the recent developments in this area. A large body of works focuses on offline policies which use the anticipated request pattern to proactively populate the caches with files that maximize the expected hits [8]. At the other extreme, dynamic caching solutions studied variants of the LFU/LRU policies [1, 21, 32, 37]; tracked the request distribution [49, 69] and optimized accordingly the caching [36]; employed reinforcement learning to adapt the caching decisions to requests [61, 62, 66]; and, more recently, applied online convex optimization towards enabling the policies to handle unknown (adversarial) request patterns [12, 38, 44, 51, 53, 64, 65]. These latter works assume that the files can be fetched dynamically at each slot to optimize the cache configuration, as opposed to works such as [39, 58] which study pure eviction policies.

The interplay between predictions and caching has attracted attention from both machine learning and networking communities. The studies in [14, 22, 24] formulated the joint caching and recommendation problem, considering static models, and assuming full knowledge of requests and the users’ propensity to follow recommendations (i.e., they place assumption on the prediction accuracy). On the other hand, [39, 58] presented a mechanism agnostic to requests that uses untrusted predictions to achieve competitive-ratio guarantees. Their approach was generalized to metrical task systems by [7] and improved with nearly lower-bound matching for the competitive ratios in [59]. However, as proved in [6], algorithms that ensure constant competitive-ratios do not necessarily guarantee sub-linear regret, which is the performance criterion we employ here following the recent regret-based caching research [12, 38, 44, 51, 53, 64, 65]. We note that all the above works consider files with equal size, while we extend the framework to the general scenario of unequally-sized files. In addition, none of the above works studies discrete caching with predictions. Finally, it is worth stressing that employing predictions for improving the performance of communication/computing systems is not a new idea: predictions have been incorporated in stochastic optimization [34, 72] which assume the requests and system perturbations are stationary;
and in online learning [16, 73] which do not adapt to predictions’ accuracy (considered known). Here, we make no assumptions on the predictions’ quality, which can be even adversarial.

### 2.2 Adaptive Smoothing

In contrast to the above studies, our optimistic learning approach is based on adaptive smoothing. Abernethy et al. [2] introduced a unified view of FTRL and FTPL as techniques to add smoothing, through regularization or perturbation, to a non-smooth potential function. This perspective is useful to our work since we leverage both ideas. Namely, let us define: $Φ(θ) = \max_{x ∈ X}(x, θ)$, and consider the potential function $Φ(Θ_t)$, where $Θ_t = θ_{1:t}$ is the vector of aggregated gradients (file requests). An intuitive strategy is to choose the action that maximizes the rewards seen so far:

$$x_t = \arg\max_{x ∈ X}(Θ_{t-1}, x) = \nabla Φ(Θ_{t-1}),$$

which is known as Follow The Leader (FTL) and is optimal when the utility functions are samples from a stationary statistical distribution. In contrast, FTL has linear regret in the adversarial setting [20, 60], since successive gradients of non-smooth functions can be arbitrarily far from each other, thus leading to unstable actions. [2] proposed to stabilize the learner actions by smoothing the potential function, and selecting actions based on the smoothed potential $\tilde{Φ}(θ)$. In FTRL, the smoothing is achieved by adding a strongly convex function to the potential, i.e., $x_t = \arg\max_{x ∈ X}(x, Θ_t) - r_{1:t}(x)$ where $r_{1:t}(x)$ is a $\sigma_t$-strongly convex regularizer. This framework generalizes the Online Gradient Ascent (OGA) and the Exponentiated Weights (EG) algorithms, which were employed for the caching problem in [53] and [64] respectively. As for FTPL, the smoothing is done by adding perturbation to the accumulated cost parameter of the potential. And the actions are decided by $x_t = \arg\max_{x ∈ X}(x, Θ_t + \eta_t γ)$, where $γ ∼ N(0, 1)$ and $η_t$ is a scaling factor that controls the smoothing. FTPL was shown to provide optimal regret guarantees for the discrete caching problem in [12]. Computationally efficiency is also a notable feature for FTPL updates as it requires an ordering operation instead of projection.

We propose to modulate the regularization $\sigma_t$ and perturbation $γ_t$ parameters with the predictions quality. Intuitively, accurate predictions should lead to less regularization/perturbation (less smoothing), enabling the learner to align its decisions more with the predictions. On the other hand, inaccurate predictions induce more smoothing, which alleviates their effects on the decisions. We show that careful tuning of these smoothing parameters leads to regret bounds that interpolate between $R_T ≤ 0$, and $R_T ≤ O(\sqrt{T})$. Nonetheless, these two algorithms have considerable differences in terms of computational complexity and constants in the bounds, which are discussed in detail.

### 2.3 Optimistic Learning

For regret minimization with predictions, [48] used predictions $θ_t$ for the gradient $\tilde{θ}_t = \nabla f_t(x_t)$ with guaranteed correlation $⟨\tilde{θ}_t, 0⟩ ≥ c∥θ_t∥^2$ to improve the regret. In [10], this assumption was relaxed to allow predictions to fail the correlation condition at some steps, obtaining bounds that interpolate between $O(\log(T))$ and $O(\sqrt{T})$; while this idea was extended to multiple predictors in [11]. A different line of works [56], [46] use adaptive regularizers and define the $t$-slot prediction errors $∥θ_t - \tilde{θ}_t∥_2$ to obtain $O(∥Σ_t∥∥θ_t - \tilde{θ}_t∥_2^{1/2})$ regret bounds. Specifically, OFTRL versions have been proposed in [46] and recently used in [5] for problems with budget constraints, while

---

4While the maximization requires that $x_t$ to be in the convex hull of $X$, feasibility can be recovered via appropriate rounding.

5We note that these papers present their algorithms as instances of a similar framework to FTRL called Online Mirror Descent (OMD). Nonetheless, there exist equivalence results between these two frameworks (see [45, Sec. 6.1]) for specific choices of the mirror-map (in OMD), or equivalently the regularizer (in FTRL).

6In this case the gradient of the smoothed potential is in fact the expectation $\nabla Φ(Θ_t + η_t γ) = E_f[x_t]$
[44, 45] tailored these ideas to continuous caching. The problem of discrete caching is fundamentally different. Through a careful analysis, we manage to reuse these results after relaxing the cache integrality constraints, and then employing a randomized rounding technique that recovers the same prediction-modulated regret in expectation. The regret bounds have the desirable property of being dimension-free. Nonetheless, we proceed to remark that OFTRL can have a computational bottleneck due to involving a projection step, which can be avoided in FTPL.

Optimistic versions of FTPL were recently investigated in [68] and [67]. In [68], the regret bound grows polynomially w.r.t. the decision set dimension. In the caching problem, this would imply a highly-problematic polynomial growth of the regret w.r.t. the typically huge library size $N$. The dependence of the regret on the dimension was improved in [67], but it still remains linear. On the contrary, our proposed OFTPL exploits the structure of the decision set and utilizes adaptive perturbation to obtain a regret bound that depends on dimension only by $O(\log(N)^{1/4})$, is order-optimal (based on the achievable lower bound), returns zero-regret for perfect predictions, does not require knowing the time-horizon $T$, nor the prediction errors. None of these desirable features is available in these prior works. We kindly refer the reader to the table in Appendix A.1 for an overview of the presented algorithms in the context of the most related literature.

3 ACHIEVABLE REGRET FOR CACHING WITH A PREDICTOR

We first introduce a lower bound for the regret of any online caching policy $\pi$, working with a cache of capacity $C$, and has access to an untrusted and potentially adversarial prediction oracle. In general, the predictions refer to the next function $\tilde{f}_t(\cdot)$. However, since most OCO algorithms learn based on the observed gradients, it suffices to have predictions $\tilde{\theta}_t = \nabla \tilde{f}_t(x_t)$. And for caching, this coincides with a prediction for the next request $\tilde{f}_t(x_t)$. Now, unlike all prior works in optimistic learning [10, 46, 56], we adopt here the more general probabilistic prediction model where $\tilde{\theta}_t$ is not necessarily a one-hot vector (as the actual $\theta_t$), but a probability distribution over the library. Thus, each $\tilde{\theta}_t$ is drawn from the N-dimensional probability simplex $\Delta_N$. This more general approach is rather intuitive as the forecasting models (e.g., a Neural Network) typically yield probabilistic inferences. It also enhances the performance of our optimistic algorithms and allows efficient training of the forecaster using a convex loss function (please see Appendix A.7.1 for examples and justification). It does require, however, a more elaborate technical analysis, especially for the case of OFTPL. In this setup, we have the following lower bound:

**Theorem 1.** For any online caching policy $\pi$, there exist a sequence of requests $\{\theta_t\}_T$ and predictions $\{\tilde{\theta}_t\}_T$ for which the regret $R_T$ satisfies

$$\mathbb{E}[R_T] \geq \sqrt{\frac{C}{2\pi}} \sqrt{\sum_{t=1}^{T} ||\theta_t - \tilde{\theta}_t||^2_2} - \Theta\left(\frac{1}{\sqrt{T}}\right).$$

**Proof.** To prove the lower bound, we show the existence of a request and prediction sequence under which the regret is guaranteed to be larger than the stated bound regardless of the online policy $\pi$. For that, we use the standard probabilistic method [3] with an appropriately constructed random file request and prediction sequence as detailed below.

Denote by $\xi_t$ and $\tilde{\xi}_t$ the random variables representing the requested file ($\theta_t$) and its prediction ($\tilde{\theta}_t$) at time $t$, respectively. Denote by $\{X^t_t\}_{t \geq 1}$ the random variables representing the action of any policy $\pi$. We use a setup where $N \geq 2C$ and consider an ensemble of caching problems (i.e., request and prediction sequences) where at each slot $t$, the requested file $\xi_t$ is chosen independently and
uniformly at random from the library \( \mathcal{N} \). The predictions \( \tilde{\xi}_t \) are also chosen independently and uniformly at random from the probability simplex \( \Delta_N \). Specifically, we let

\[
\{ \tilde{\xi}_t \}_{t \geq 1} \overset{\text{i.i.d.}}{\sim} \text{Dirichlet}(\lambda_1, \ldots, \lambda_n, \ldots, \lambda_N) \quad \text{with} \quad \lambda_n = 1, \forall n \in \mathcal{N}.
\]

Hence, the expected reward obtained by any caching policy \( \pi \) on any slot \( t \), conditional on the information available to the policy can be bounded as

\[
\mathbb{E} \left[ \langle \tilde{\xi}_t, X^\pi_t \rangle \mid \{ \tilde{\xi}_r \}_{r=1}^{t-1}, \{ \tilde{\xi}_r \}_{r=1}^t \right] \quad (a) \quad \mathbb{E} \mathbb{E} \left[ \langle \tilde{\xi}_t, X^\pi_t \rangle \right] \leq \frac{1}{2C} \mathbb{E} \left[ 1, X^\pi_t \right] \quad (b) \quad \frac{1}{2C} \leq \frac{1}{2},
\]

where (a) follows from the tower property of expectations, (b) from the fact \( \tilde{\xi}_t \perp \{ \tilde{\xi}_r \}_{r=1}^{t-1}, \{ \tilde{\xi}_r \}_{r=1}^t, X^\pi_t \) and hence \( \mathbb{E}(\tilde{\xi}_t \mid \{ \tilde{\xi}_r \}_{r=1}^{t-1}, \{ \tilde{\xi}_r \}_{r=1}^t, X^\pi_t) = \mathbb{E}(\tilde{\xi}_t) = \frac{1}{2C} 1_{N \times 1} \). Finally (c) since \( \langle 1, X^\pi_t \rangle \leq C \), which holds because of the cache capacity constraint. Taking expectation of the above bound, we have \( \mathbb{E}[\langle \tilde{\xi}_t, X^\pi_t \rangle] \leq \frac{1}{2} \). Hence, using the linearity of expectations, the expected value of the cumulative hits up to slot \( T \) under any policy \( \pi \) is upper bounded as \( \mathbb{E} \left[ \sum_{t=1}^{T} \langle \tilde{\xi}_t, X^\pi_t \rangle \right] \leq \frac{T}{2} \).

Now we compute a lower bound to the expected number of cumulative hits achieved by the best-in-hindsight fixed cache configuration \( X^*_t \). Similar to [12], we identify the problem with the classic setup of balls (requests) into bins (files). In this framework, it follows that the offline benchmark achieves cumulative hits which are equal to the total number of balls into the most-loaded \( C \) bins when \( T \) balls are thrown uniformly at random into \( N = 2C \) bins. Hence, from [12, Lemma 1]:

\[
\mathbb{E} \left[ \sum_{t=1}^{T} \langle \tilde{\xi}_t, X^*_t \rangle \right] \geq \frac{T}{2} + \sqrt{\frac{CT}{2\pi}} - \Theta \left( \frac{1}{\sqrt{T}} \right).
\]

Hence, the expected regret achieved by any policy in the optimistic set up is lower bounded as

\[
\mathbb{E} \left[ R_T \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \langle \tilde{\xi}_t, X^*_t \rangle - \sum_{t=1}^{T} \langle \tilde{\xi}_t, X^\pi_t \rangle \right] \geq \sqrt{\frac{CT}{2\pi}} - \Theta \left( \frac{1}{\sqrt{T}} \right). \tag{2}
\]

Finally, we evaluate the expected value of the quantity \( M_T = \sum_{t=1}^{T} || \tilde{\xi}_t - \xi_t \|_2^2 \) as follows.

\[
\mathbb{E} [M_T] = \mathbb{E} \left[ \sum_{t=1}^{T} || \tilde{\xi}_t - \xi_t \|_2^2 \right] \overset{(a)}{=} T \mathbb{E} \left[ || \tilde{\xi}_1 - \xi_1 \|_2^2 \right] \overset{(b)}{=} TN \mathbb{E} \left[ (\tilde{\xi}_1^t - \xi_1^t)^2 \right] = TN \mathbb{E} \left[ (\tilde{\xi}_1^t)^2 - 2\tilde{\xi}_1^t \xi_1^t + (\xi_1^t)^2 \right] \overset{(c)}{=} TN \left[ \text{Var}(\tilde{\xi}_1) + (\mathbb{E}(\tilde{\xi}_1))^2 - 2\mathbb{E}(\tilde{\xi}_1)\mathbb{E}(\xi_1) + \mathbb{E}(\xi_1) \right] \overset{(d)}{=} TN \left[ \frac{(N-1)}{N^2(N+1)} + \frac{1}{N^2} - \frac{2}{N^2} + \frac{1}{N} \right] = T(1 - \frac{2}{N(N+1)}) \leq T,
\]

where (a) follows from the i.i.d. assumption of the random vectors at each \( t \), (b) from the i.i.d assumption of each component of vectors \( \tilde{\xi}_1 \) and \( \xi_1 \), (c) from \( \tilde{\xi}_1 \perp \xi_1 \), and (d) from standard results on Dirichlet distribution. Combining the above bound with (2), we have by Jensen’s inequality

\[
\mathbb{E} \left[ R_T \right] \geq \sqrt{\frac{CM_T}{2\pi}} - \Theta \left( \frac{1}{\sqrt{T}} \right) \geq \mathbb{E} \left[ \sqrt{\frac{CM_T}{2\pi}} - \Theta \left( \frac{1}{\sqrt{T}} \right) \right] \quad \text{i.e.,}
\]

\[
\mathbb{E} \left[ R_T - \sqrt{\frac{C \sum_{t=1}^{T} || \tilde{\xi}_t - \xi_t \|_2^2}{2\pi}} \right] \geq -\Theta \left( \frac{1}{\sqrt{T}} \right)
\]
Our analysis can be readily extended to a bounded number of requests per slot.

We will see that the proposed optimistic algorithms in Sections 4 and 5 attain this bound within an absolute and a poly-logarithmic factor, respectively.

4 CACHING THROUGH OPTIMISTIC REGULARIZATION (OFTRL-CACHE)

The first algorithm we propose is based on OFTRL. Prediction adaptive regularization was explored before in [56] and later improved via proximal regularizers in [46], all for convex sets. The gist of our approach is that we use OFTRL to obtain \(\hat{x}_t\) in expectation in expectation the same regret bounds as OFTRL for continuous caching [45].

Let us define the prediction error at slot \(t\) as \(\delta_t \triangleq \|\theta_t - \hat{\theta}_t\|_2^2\), and introduce the proximal \(\sigma_t\)-strongly convex regularizer w.r.t. the Euclidean \(\ell_2\) norm:

\[
r_t(x) = \frac{\sigma_t}{2} \|x - x_t\|_2^2.
\]

Following [44], we define parameters \(\{\sigma_t\}_t\) using the accumulated prediction errors, namely:

\[
\sigma_1 = \sigma \sqrt{\delta_1}, \quad \sigma_t = \sigma \left(\sqrt{\delta_{t-1}} - \sqrt{\delta_{t-1}}\right) \quad \forall t \geq 2, \quad \text{with} \quad \sigma = 1/\sqrt{C}.
\]

The basic OFTRL update stems from using these regularizers in the FTRL update formula. Namely, at each slot \(t\) we update the cache to maximize the aggregated utility. This maximization is regularized through a term (the above-defined regularizers) that depends on the predictor’s accuracy. The detailed steps are summarized in Algorithm 1. In the first iteration we draw randomly a feasible caching vector \(x_1\) and observe the prediction error \(\delta_1 = \|\theta_1 - \hat{\theta}_1\|_2^2\). In each iteration we need to solve a strongly convex program (line 5) which returns the continuous caching vector \(\hat{x}_t\), that is transformed to a feasible discrete \(x_t\) (line 6) using Madow’s Sampling (see Appendix A.2).

The algorithm notes the new gradient vector, by simply observing the next request and updates the accumulated gradient \(\Theta_t\) (line 7). The regret guarantee of Algorithm 1 is described next.

---

**Algorithm 1: Optimistic Follow The Regularized Leader (OFTRL-Cache)**

```latex
\begin{algorithm}
\caption{Optimistic Follow The Regularized Leader (OFTRL-Cache)}
\begin{algorithmic}[1]
\State \textbf{Input}: \(\sigma = 1/\sqrt{C}, \delta_t = \|\theta_t - \hat{\theta}_t\|_2^2, \sigma_1 = \sigma \sqrt{\delta_1}, x_1 = \arg \min_{x \in X} \langle x, \theta_1 \rangle\)
\State \textbf{Output}: \(\{x_t \in X\}_T\) // Feasible discrete caching vector at each slot
\For {\(t = 2, 3 \ldots\)}
\State \(\hat{\theta}_t \leftarrow \text{prediction} \) // Obtain request prediction for slot \(t\)
\State \(\hat{x}_t \leftarrow \arg \max_{x \in \text{conv}(X)} \left\{ -r_{t-1}(x) + \langle x, \Theta_{t-1} + \hat{\theta}_t \rangle \right\} \) // Update the continuous cache vector
\State \(x_t \leftarrow \text{MadowSample}(\hat{x}_t)\) // Obtain the discrete cache vector using Algorithm 7
\State \(\Theta_t = \Theta_{t-1} + \hat{\theta}_t\) // Receive \(t\)-slot request and update total gradient
\State \(\sigma_t = \sigma \left(\sqrt{\delta_{t-1}} - \sqrt{\delta_{t-1}}\right)\) // Update the regularization parameter
\EndFor
\end{algorithmic}
\end{algorithm}
```
THEOREM 2. Algorithm 1 ensures, for any time horizon $T$ and $N \geq 2C$, the expected regret bound:

$$
\mathbb{E}[R_T] \leq 2\sqrt{C} \sqrt{\sum_{t=1}^{T} \| \theta_t - \hat{\theta}_t \|^2_2}
$$

PROOF. We define first the regret w.r.t. the continuous actions $\{\hat{x}_t\}_T$ as $\hat{R}_T \triangleq \langle \Theta_T, x^* \rangle - \sum_{t=1}^{T} \langle \theta_t, \hat{x}_t \rangle$, where $x^*$ is the optimal-in-hindsight caching vector\(^9\). We also define the scaled Euclidean $\ell_2$ norm $\| \cdot \|_{(t)} = \sqrt{\sigma_{t:t}} \| \cdot \|_2$ so that $r_{1:t}$ is $1$-strongly convex w.r.t $\| \cdot \|_{(t)}$, and note that its dual norm is $\| \cdot \|_{(t),*} = \frac{1}{\sqrt{\sigma_{t:t}}} \| \cdot \|_2$. Our starting point is [44, Lem. 1], which we restate below:

**Lemma 1.** Let $r_{1:t}$ be a $1$-strongly convex w.r.t. a norm $\| \cdot \|_{(t)}$. Then, the OFTRL iterates produced by line 5 in Algorithm 1 guarantee the bound $\hat{R}_T \leq r_{1:T}(x^*) + \frac{1}{2} \sum_{t=1}^{T} \| \theta_t - \hat{\theta}_t \|^2_{(t),*}$.

Now, we first get a deterministic regret bound on $\hat{x}_t$. Assuming that\(^{10}\) $C \in (0, N/2]$, we can bound the $\ell_2$ diameter of the caching set as $\|x^* - x_t\|_2 \leq \sqrt{2C}$, $\forall x^*, x_t \in \text{conv}(X)$. Thus, we can upper-bound the regularizers in (3), replace in the above Lemma and telescope to get:

$$
\hat{R}_T \leq \sigma_{1:T} C + \frac{1}{2} \sum_{t=1}^{T} \frac{\delta_t}{\sigma_{1:t}}.
$$

(4)

Observing that the sum $\sigma_{1:t}$ telescopes to $\sigma \sqrt{\delta_{1:T}}$, we can substitute it in (4) and use the standard identity [50, Lem. 4.13] to bound the second term via $\sum_{t=1}^{T} \delta_t / \sqrt{\delta_{1:T}} \leq 2 \sqrt{\delta_{1:T}}$. Therefore, we obtain:

$$
\hat{R}_T \leq \sigma \sqrt{\delta_{1:T}} C + \frac{1}{2} \sigma \sqrt{\delta_{1:T}}\sqrt{2C}.
$$

(5)

The last step requires Madow’s sampling (line 6). By construction, the routine selects $C$ files and hence returns a feasible integral caching vector $x_t$ (or, sampled vector). In addition, each item is included in the sampled vector with a probability based on the continuous $\hat{x}_t$. Namely, it holds $\Pr(x^*_i = 1) = \pi_i - \pi_{i-1} = \hat{x}_t^i$, where the auxiliary parameter $\pi_i$ aggregates the (interpreted as) probabilities for caching the first $i$ files, i.e., $\pi_i = \sum_{k=0}^{i} \hat{x}_k$. Since each $x^*_i$ is binary, it holds $\mathbb{E}[x_i] = \Pr(x^*_i = 1) = \hat{x}_t$. The result follows by using (5) and observing:

$$
\mathbb{E}[R_T (\{x\}_T)] = \langle \Theta_T, x^* \rangle - \mathbb{E} \left[ \sum_{t=1}^{T} \langle \theta_t, x_t \rangle \right] = \hat{R}_T.
$$

\(\square\)

**Discussion.** The bound in Theorem 2 ensures the desirable prediction-based modulation of the algorithm’s performance, as the achieved regret shrinks with the prediction quality. If all predictions are accurate, we get $R_T \leq 0$; when all predictions fail, we get $R_T \leq 2\sqrt{2CT}$. That is, in the worst scenario (e.g., when the predictions are created by an adversary) the regret bound is worse by a constant factor of $\sqrt{2}$ compared to the FTRL algorithm that does not use predictions [43, Sec. 3.4]), and ~ 5 compared to the lower bound derived in Sec. 3. Moreover, due to selecting an $\ell_2$ regularizer, the bounds are dimension-free and do not depend on the library size $N$. This is particularly important since in caching problems oftentimes the library size is an even bigger

---

\(^9\)This benchmark remains unchanged if we switch from the continuous to the discrete space.

\(^{10}\) $N$ is typically orders of magnitude higher than $C$. In the cases where this does not hold the current analysis is still valid but can be improved by using the tighter diameter $\sqrt{2}(N - C)$. 

---

Proc. ACM Meas. Anal. Comput. Syst., Vol. 6, No. 3, Article 48. Publication date: December 2022.
Algorithm 2: Optimistic Follow The Perturbed Leader (OFTPL–Cache)

1. **Input**: \( \eta_1 = 0, y_1 = \arg \min_{y \in X} \langle y, \theta_1 \rangle \)  
2. **Output**: \( \{y_t \in X\}_T \)  

3. \( y_t \overset{iid}{\sim} \mathcal{N}(0, 1_{N \times 1}) \)  

4. for \( t = 2, 3, \ldots \) do

5. \( \hat{\theta}_t \leftarrow \text{prediction} \)  

6. \( \eta_t = \frac{1}{\sqrt{2(1 - N^{-\frac{1}{4}})}} \left( \sum_{t=1}^{T} \| \theta_t - \hat{\theta}_t \|_1^2 \right)^{\frac{1}{2}} \)  

7. \( y_t = \arg \max_{y \in X} \langle y, \Theta_{t-1} + \hat{\theta}_t + \eta_t y \rangle \)  

8. \( \Theta_t = \Theta_{t-1} + \eta_t \)  

end

Concern than the time horizon. Finally, note that the algorithm does not need to know the horizon \( T \) beforehand. The drawback of this optimistic caching approach is the computational complexity of the iteration (line 5) which involves a projection operation. While \( \ell_2 \) projections have received attention [30, Sec. 7], they can hamper the scalability of the algorithm under certain conditions\(^{11}\). In the following section we show how perturbation-based smoothing can avoid the projection step.

5 CACHING THROUGH OPTIMISTIC PERTURBATIONS (OFTPL–CACHE)

We propose next a new OFTPL algorithm that significantly improves previous OFTPL proposals [67, 68], both in terms of their bounds and implementation, and as such is of independent interest with potential applications that extend beyond caching to other \( k \)-set structured problems such as those discussed in [15]. The improvement is possible by setting the perturbation parameters \( \eta_t \) in a manner that is adaptive to prediction error witnessed until \( t - 1 \).

Following the discussion in Sec. 2, we remind the reader that the FTPL actions are derived by solving in each slot \( t \) a linear program (LP) with a parameterized perturbed cumulative utility vector, \( \Theta_{t-1} + \eta_t y \), where \( \eta_t \in \mathbb{R}_+ \) is the perturbation parameter. In order to obtain the optimistic FTPL variant we introduce two twists: (i) the prediction for the next-slot utility \( \hat{\theta}_t \) is added to the cumulative utility; and (ii) the perturbation parameter \( \eta_t \) is scaled according to the accumulated prediction error. Interestingly, due to the structure of the decision set \( X \), the LP solution reduces to identifying the \( C \) files with the highest coefficients. This step can be efficiently implemented in deterministic linear (\( O(N) \)) time using, e.g., the Median-of-Medians algorithm [17]. The steps of the proposed scheme are presented in Algorithm 2, where we denote the \( t \)-slot OFTPL decisions with \( y_t \in X \). The following theorem characterizes the performance of this new OFTPL algorithm.

**Theorem 3.** Algorithm 2 ensures, for any time horizon \( T \) and \( N \geq 2C \) with \( C \geq 11 \), the expected regret bound:

\[
\mathbb{E}_T[R_T] \leq 3.68 \sqrt{C \left( \ln \frac{Ne}{C} \right)^{1/4} \sum_{t=1}^{T} \| \theta_t - \hat{\theta}_t \|_1^2}.
\]

\(^{11}\)For instance, this can be a bottleneck if the library size is extremely large, while the slot duration is very short and the available computation power is limited.
We now bound the first term in the RHS of inequality (9):
\[
\Phi_t(\theta) \triangleq \mathbb{E}_{y \sim \mathcal{N}(0, I)} \left[ \max_{y \in \mathcal{X}} \langle y, \theta + \eta_t y \rangle \right] = \mathbb{E}_{y} \left[ \Phi(\theta + \eta_t y) \right]
\]
Clearly, \( \Phi_t(\theta) \) is convex in \( \theta \). Recall that the cumulative file request vector is defined as \( \Theta_t = \Theta_{t-1} + \hat{\theta}_t \). A Taylor expansion of \( \Phi_t(\cdot) \) around the point \( \Theta_{t-1} + \hat{\theta}_t \), evaluated at \( \Theta_t \), with a second order remainder is:
\[
\Phi_t(\Theta_t) = \Phi_t(\Theta_{t-1} + \hat{\theta}_t) + \langle \nabla \Phi_t(\Theta_{t-1} + \hat{\theta}_t), \theta_t - \hat{\theta}_t \rangle + \frac{1}{2} (\theta_t - \hat{\theta}_t, \nabla^2 \Phi_t(\hat{\theta}_t)(\theta_t - \hat{\theta}_t)),
\]
where \( \hat{\theta}_t \) is a point on the line segment connecting \( \Theta_t \) and \( \Theta_{t-1} + \hat{\theta}_t \). From the convexity of \( \Phi_t(\cdot) \):
\[
\Phi_t(\Theta_{t-1} + \hat{\theta}_t) \leq \Phi_t(\Theta_{t-1}) + \langle \nabla \Phi_t(\Theta_{t-1} + \hat{\theta}_t), \theta_t \rangle.
\]
From (6) and (7), we can eventually write:
\[
\Phi_t(\Theta_t) \leq \Phi_t(\Theta_{t-1}) + \langle \nabla \Phi_t(\Theta_{t-1} + \hat{\theta}_t), \theta_t \rangle + \frac{1}{2} (\theta_t - \hat{\theta}_t, \nabla^2 \Phi_t(\hat{\theta}_t)(\theta_t - \hat{\theta}_t)).
\]
Now, note that it holds \( \nabla \Phi_t(\Theta_{t-1} + \hat{\theta}_t) = \mathbb{E}_{y} \left[ \nabla \Phi_t(\Theta_{t-1} + \hat{\theta}_t) \right] = \mathbb{E}_{y} \left[ \arg \max_{y \in \mathcal{X}} \langle y, \Theta_{t-1} + \hat{\theta}_t + \eta_t y \rangle \right] = \mathbb{E}_{y} \left[ y_t \right], \)
where (a) stems from [9, Prop. 2.2]. Thus, (8) can be written as:
\[
\Phi_t(\Theta_t) \leq \Phi_t(\Theta_{t-1}) + \mathbb{E}_{y} \left[ \langle \theta_t, y_t \rangle \right] + \frac{1}{2} (\theta_t - \hat{\theta}_t, \nabla^2 \Phi_t(\hat{\theta}_t)(\theta_t - \hat{\theta}_t)).
\]
Subtracting \( \Phi_{t-1}(\Theta_{t-1}) \) from both sides and telescoping over \( T \) and setting \( \eta_0 = 0 \), we get:
\[
\Phi_T(\Theta_T) \leq \sum_{t=1}^{T} \left( \Phi_t(\Theta_{t-1}) - \Phi_{t-1}(\Theta_{t-1}) + \mathbb{E}_{y} \left[ \langle \theta_t, y_t \rangle \right] + \frac{1}{2} (\theta_t - \hat{\theta}_t, \nabla^2 \Phi_t(\hat{\theta}_t)(\theta_t - \hat{\theta}_t)) \right).
\]
Then, by Jensen’s inequality: \( \max_{y \in \mathcal{X}} \mathbb{E}_{y} \left[ \langle y, \Theta_T + \eta_T y \rangle \right] = \max_{y \in \mathcal{X}} \langle y, \Theta_T \rangle = \Phi(\Theta_T) \leq \Phi_T(\Theta_T) \), and writing the last term as the norm of the vector \( (\theta_t - \hat{\theta}_t) \) induced by the symmetric positive semidefinite matrix \( \nabla^2 \Phi_t(\hat{\theta}_t) \triangleq H_t \), we get the following upper bound of the regret:
\[
R_T \leq \Phi(\Theta_T) - \sum_{t=1}^{T} \mathbb{E}_{y} \left[ \langle \theta_t, y_t \rangle \right] \leq \sum_{t=1}^{T} \left( \Phi_t(\Theta_{t-1}) - \Phi_{t-1}(\Theta_{t-1}) + \frac{1}{2} ||\theta_t - \hat{\theta}_t||_{H_t} \right).
\]
We now bound the first term in the RHS of inequality (9):
\[
\sum_{t=1}^{T} \Phi_t(\Theta_{t-1}) - \Phi_{t-1}(\Theta_{t-1}) = \sum_{t=1}^{T} \mathbb{E}_{y} \left[ \Phi(\Theta_{t-1} + \eta_t y) - \Phi(\Theta_{t-1} + \eta_{t-1} y) \right] \overset{(a)}{=} \sum_{t=1}^{T} \mathbb{E}_{y} \left[ \Phi((\eta_t - \eta_{t-1}) y) \right] \overset{(b)}{=} \sum_{t=1}^{T} \mathbb{E}_{y} \left[ \Phi(\eta_t y) \right] \overset{(c)}{=} \eta_T \sqrt{\frac{2C \ln N}{C}} \overset{(d)}{=} \eta_T C \sqrt{2 \ln (Ne/C)},
\]
where inequalities (a) and (b) follow from the sub-linearity of the potential function; (c) from Massart’s lemma which gives an upper bound the expected sum of the top \( C \) elements in a Gaussian random vector (e.g., [15, Lem. 9]); and finally (d) is due to \( \left( \frac{N}{C} \right) \leq \left( \frac{N/e}{C} \right) \).

\(^{12}\)A function \( f \) is sub-linear if it is sub-additive (i.e., \( f(a) + f(b) \geq f(a+b) \), which implies \( f(a) - f(b) \leq f(a-b) \)), and positive homogeneous (i.e., \( f(\lambda a) = \lambda f(a), \lambda > 0 \)).
We now upper bound the second term in the RHS of (9). From [15, Eqn. (4)], the \((i,j)\)th entry of the Hessian matrix is given by 
\[
H'_{ij} = \frac{1}{\eta_t} \mathbb{E}_y \left[ \hat{y} (\hat{\theta}_t + \eta_t y)_i (\hat{\theta}_t + \eta_t y)_j \right],
\]
where \(\hat{y} (\cdot) = \text{arg max}_{y \in X} \langle y, \cdot \rangle\). Hence, we have the following bound on the absolute value of each entry:
\[
|H'_{ij}| = \frac{1}{\eta_t} \mathbb{E}_y \left[ |\hat{y} (\hat{\theta}_t + \eta_t y)_i| |\hat{y} (\hat{\theta}_t + \eta_t y)_j| \right] \leq \frac{1}{\eta_t} \mathbb{E}_y [ |y_i| |y_j| ] \leq \frac{1}{\eta_t} \sqrt{\frac{2}{\pi}}, \tag{11}
\]
where the first inequality follows from Jensen’s inequality, the second holds since \(\hat{y}_i = \{0, 1\}\); and the last one is a property of Gaussian r.v.s. Thus each of the quadratic forms on the RHS of Eqn. (9) can be bounded as follows:
\[
\|\theta_t - \hat{\theta}_t\|_{H_t} = \langle \theta_t - \hat{\theta}_t, H_t (\theta_t - \hat{\theta}_t) \rangle = \sum_{i,j} (\theta_t,i - \hat{\theta}_t,i) H'_{ij} (\theta_t,j - \hat{\theta}_t,j)
\leq \sum_{i,j} |(\theta_t,i - \hat{\theta}_t,i)||H'_{ij}| |(\theta_t,j - \hat{\theta}_t,j)| \leq \frac{1}{\eta_t} \sqrt{\frac{2}{\pi}} (\sum_i (\theta_t,i - \hat{\theta}_t,i))^2 \leq \frac{1}{\eta_t} \sqrt{\frac{2}{\pi}} \|\theta_t - \hat{\theta}_t\|^2_2, \tag{12}
\]
where (a) follows from the triangle inequality and (b) from the bound (11).

Another way to bound \(\|\theta_t - \hat{\theta}_t\|_{H_t}\), which will be useful later\(^{13}\), starts from (6) to get:
\[
\frac{1}{2} \|\theta_t - \hat{\theta}_t\|_{H_t} = \Phi_t (\Theta_t) - \Phi_t (\Theta_{t-1} + \hat{\theta}_t) - \langle \nabla \Phi_t (\Theta_{t-1} + \hat{\theta}_t), \theta_t - \hat{\theta}_t \rangle
\leq \mathbb{E}_y \left[ \Phi (\Theta_t + \eta_t y) - \Phi (\Theta_{t-1} + \hat{\theta}_t + \eta_t y) \right] + \langle \nabla \Phi_t (\Theta_{t-1} + \hat{\theta}_t), \theta_t - \hat{\theta}_t \rangle \leq \Phi (\theta_t - \hat{\theta}_t) + \langle \nabla \Phi_t (\Theta_{t-1} + \hat{\theta}_t), \theta_t - \hat{\theta}_t \rangle = \max_{y \in X} \langle y, \theta_t - \hat{\theta}_t \rangle + \langle \nabla \Phi_t (\Theta_{t-1} + \hat{\theta}_t), \theta_t - \hat{\theta}_t \rangle \leq 2 \|\theta_t - \hat{\theta}_t\|_1, \tag{13}
\]
where (a) follows from the sub-additivity of \(\Phi (\cdot)\), and in (b) we use that \(y_i \in \{0, 1\}, \forall i\) and bounded both terms using triangle inequality. Hence, combining the bounds (12) and (13), we get:
\[
\frac{1}{2} \|\theta_t - \hat{\theta}_t\|_{H_t} \leq \min \left( \frac{1}{\sqrt{2\pi}} \frac{\|\theta_t - \hat{\theta}_t\|^2_2}{\eta_t}, 2 \|\theta_t - \hat{\theta}_t\|_1 \right).
\]

Now we choose the learning rate \(\eta_t = \beta \sqrt{\sum_{r=1}^{t-1} \|\theta_r - \hat{\theta}_r\|^2_1}, \ t \geq 1\) for some constant \(0 < \beta \leq \frac{1}{\sqrt{2\pi}}\) that will be specified later. Hence, we have:
\[
\frac{1}{2} \|\theta_t - \hat{\theta}_t\|_{H_t} \leq \min \left( \frac{\|\theta_t - \hat{\theta}_t\|^2_1}{\sqrt{2\pi} \beta \sqrt{\sum_{r=1}^{t-1} \|\theta_r - \hat{\theta}_r\|^2_1}}, 2 \frac{\|\theta_t - \hat{\theta}_t\|^2_1}{\sqrt{2\pi} \beta \sqrt{\|\theta_{t-1} - \hat{\theta}_{t-1}\|^2_1}} \right) \leq \frac{3}{\sqrt{2\pi} \beta \sqrt{\sum_{t=1}^{T} \|\theta_t - \hat{\theta}_t\|^2_1}} \tag{14}
\]
where in (a), we used the fact that \(\text{min}(a_1/a_2, b_1/b_2) \leq \frac{a_1 + a_2}{b_1 + b_2}\) for any two positive fractions and \(\sqrt{x + y} \leq \sqrt{x} + \sqrt{y}\), for any non-negative \(x\) and \(y\)’s. Now that we have a bound for the smoothing-overhead in (10), and the per-step regret bounds (14), we can substitute them in (9) to get:
\[
\mathbb{E}_y [R_T] \leq \eta_T C \sqrt{2 \log(N e / C)} + \frac{3}{\sqrt{2\pi} \beta \sqrt{\sum_{t=1}^{T} \|\theta_t - \hat{\theta}_t\|^2_1}} \tag{15}
\]

---

\(^{13}\)This second bound on the norm enables us to set \(\eta_t\) parameters based solely on the prediction error witnessed so far \(\sum_{r=1}^{t-1} \|\theta_r - \hat{\theta}_r\|\). Consequently, the regret will depend solely on a scaled prediction error (without additive constants).

---

Proc. ACM Meas. Anal. Comput. Syst., Vol. 6, No. 3, Article 48. Publication date: December 2022.
The second term above can be upper-bounded as:

\[
\sum_{t=1}^{T} \frac{||\theta_t - \tilde{\theta}_t||_1^2}{\sum_{r=1}^{t} ||\theta_r - \tilde{\theta}_r||_1^2} \leq \sum_{t=1}^{T} \frac{\int_{0}^{\sqrt{2}} \frac{\int_{0}^{\sqrt{2}} \frac{\beta^2}{2} \ln(Ne/C)}{2 \pi} \, dx}{\frac{\beta^2}{2} \ln(Ne/C)} = \int_{0}^{\sqrt{2}} \frac{\int_{0}^{\sqrt{2}} \frac{\beta^2}{2} \ln(Ne/C)}{2 \pi} \, dx = 2 \sum_{t=1}^{T} ||\theta_t - \tilde{\theta}_t||_1^2.
\]

Substituting the above bound into (15) and using the definition of $\eta_T$ we get the regret upper bound:

\[
\mathbb{E}_\tau [R_T] \leq \sum_{t=1}^{T} ||\theta_t - \tilde{\theta}_t||_1^2 \left( C \sqrt{2 \ln(Ne/C)} \beta + \frac{6}{\beta \sqrt{2\pi}} \right)
\]

Optimizing over the constant $\beta$, we get that $\beta = \sqrt{\frac{3}{C}}(\frac{1}{\pi \ln(Ne/C)})^{1/4}, C \geq 11$ (Recall we that $0 < \beta \leq 1/\sqrt{2\pi}$). Substituting this value for $\beta$ back in (16) we arrive at the result.

**Discussion.** Similarly to Theorem 2, the regret bound here is modulated with the quality of predictions: it collapses to zero when predictions are perfect, and maintains that do not fix the cost sequence in advance but react to the choices of the algorithm. The same property of depending on prediction mistakes rather than $T$ holds for Theorem 2 but the regret scales as the square root of double the number of mistakes, due to the use of $\ell_2$ norm.

### 6 CACHING FILES WITH ARBITRARY SIZES

While the caching problem with equal-sized files has been studied using regret analysis and competitive analysis, to the best of the authors’ knowledge, there are no results for the more challenging case of files with different sizes. This section fills this gap by extending the above tools accordingly. In particular, we consider the setting where each file $i \in \mathcal{N}$ has a size of $s_i$ units, $s_i \leq C$.

---

14 The same property of depending on prediction mistakes rather than $T$ holds for Theorem 2 but the regret scales as the square root of double the number of mistakes, due to the use of $\ell_2$ norm.
Hence, the set of feasible caching vectors needs to be redefined as:

\[ X_s = \left\{ x \in \{0, 1\}^N \mid \sum_{i=1}^N s_i x_i \leq C \right\}, \]

where the caching decisions are calibrated with the respective file sizes in the capacity constraint. And similarly, the benchmark (designed-in-hindsight) policy is redefined as \( x^* = y^* \pm \arg\max_{x \in X_s} (x, \Theta_T) \). We present two solution approaches for this problem, using both OFTRL and OFTPL. These results are of independent interest with applications beyond caching.

6.1 Approximate OFTPL

Similarly to Algorithm 2, the OFTPL algorithm in this case determines the next cache configuration \( y_t \) by solving the following integer programming problem at each round \( t \):

\[
\mathbb{P}_1 : \max_{y \in X_s} \left\langle \Theta_{t-1} + \tilde{\Theta}_t + \eta_t y, y \right\rangle,
\]

which is a Knapsack instance with profit vector \( p = \Theta_{t-1} + \tilde{\Theta}_t + \eta_t y \); size vector \( s = (s_i, \forall i \in N) \); and capacity \( C \). Since the Knapsack problem is NP-Hard [42], we cannot solve \( \mathbb{P}_1 \) efficiently (fast and accurately) at each slot, and hence it is not practical (or, even possible) to use the approach of Sec. 5. Instead, we resort here to an approximation scheme for solving \( \mathbb{P}_1 \) and, importantly, do so in a way that these approximately-solved instances do not accumulate an unbounded regret w.r.t. \( y^* \). This requires a tailored approximation analysis and to define a new regret metric.

In detail, we leverage Dantzig’s approach for tackling packing problems [19], to obtain an \( \alpha \)-integral solution from the respective integrality-relaxed problem; and then recover, via a point-wise randomized rounding, a fully-integral solution which, as we prove, keeps the long-term \( \frac{1}{2} \)-approximate regret bounded. First, recall that the \( \alpha \)-approximate regret is defined as [35]:

\[
R_T^\alpha \triangleq \alpha \langle \Theta_T, x^* \rangle - \sum_{t=1}^T \langle \theta_t, x_t \rangle,
\]

for a positive constant \( \alpha \). This generalized regret metric allows to use a parameterized benchmark, in line with prior works, e.g., see [51] and references therein. Now, it is important to see that while the Knapsack problem admits an FPTAS [70], due to the online nature of our caching problem, not all \( \alpha \)-approximation schemes for the offline OFTPL problem provide an \( \alpha \)-approximate regret guarantee. In light of this, we employ the stronger notion of point-wise \( \alpha \)-approximation scheme, which yields an \( \alpha \)-regret guarantee for the online learning problem [35]. We restate the definition:

**Definition 1** (\( \alpha \)-point-wise approximation). A randomized \( \alpha \)-point-wise approximation algorithm \( \mathcal{A} \) for a fractional solution \( \hat{y} = (\hat{y}_i, i \in N) \) of a maximizing LP with non-negative coefficients, is one that returns an integral solution \( y = (y_i, i \in N) \) such that \( \mathbb{E} [y_i] \geq \alpha \hat{y}_i, \forall i \in N \) and some \( \alpha > 0 \); where the expectation is taken over possible random choices made by algorithm \( \mathcal{A} \).

In our case, we set \( \alpha = 1/2 \) and propose an \( (1/2) \)-point-wise approximation algorithm for \( \mathbb{P}_1 \). Our starting point is Dantzig’s approach which operates on the integrality-relaxed version of \( \mathbb{P}_1 \). In particular, the integrality-relaxed LP for the Knapsack problem with profit vector \( p \), weight vector \( s \), and capacity \( C \), through the following steps:

Dantz(C, p, s):

1. Index files in decreasing profit-to-size ratios, i.e., \( (p_1/s_1) \geq (p_2/s_2) \geq \ldots \geq (p_N/s_N) \).
2. Set \( k = \min \{ j \mid \sum_{i=1}^j s_i > C \} \) and \( \bar{C} = C - \sum_{i=1}^{k-1} s_i \).
To streamline presentation, we denote with \(\tilde{\theta}_t\) the prediction at slot \(t\). Then we calculate the new profits \(p_t = \Theta_{t-1} + \tilde{\theta}_t + \eta_t y\), \(i \in \mathcal{N}\), and solve the relaxed Knapsack by invoking Dantz(C, p, s) to obtain the almost-integral \(\hat{y}_t\) and parameter \(k\) (line 7). This vector has \(k-1\) components equal to 1, one additional non-negative component, and \(N-k\) components equal to 0. This solution is then rounded through the randomization scheme:\n
\[
\text{Rand}(\hat{y}_t, k):
\]

1. Set \(S = 1, 2, \ldots, k-1 \oplus [k-1]\) with probability \(1/2\); Set \(S = \{k\}\) with probability \(1/2\).
2. Set \(y'_i = 0, \forall i \in \mathcal{N}\); and update to \(y'_i = 1\) for each \(i \in S\).

The Rand operation is invoked and creates integral caching vector \(y_t\) which satisfies the capacity constraint (line 9). Finally, we observe the new gradient, update the aggregate gradient vector and repeat the process (line 10). The following theorem, proved in Appendix. A.4, characterizes the guarantees of Algorithm 3.

**Theorem 4.** Algorithm 3 ensures, for any time horizon \(T\), the expected regret bound:

\[
\mathbb{E} \left[ R_T^{(1/2)} \right] \leq 1.84 \sqrt{C} \left( \ln \frac{N e}{C} \right)^{1/4} \sqrt{\sum_{t=1}^{T} \|\theta_t - \tilde{\theta}_t\|^2_1}
\]

**Discussion.** The bounds of Theorem 3 possess the desirable property of being modulated with the prediction errors, and in fact are improved by a factor of half compared to the equal-sizes bound. However, we remind the reader that the regret metric in this section is defined w.r.t. a weaker benchmark, i.e., a benchmark that achieves \(1/2\) the utility of the best-in-hindsight utility \(\langle \Theta_T, y^* \rangle\).
We introduce next an OFTRL algorithm that can handle arbitrary file sizes. To that end, we use a subroutine from [13], which is known to achieve the useful property re-stated below. The diligent reader will observe that essentially we merge steps from Algorithm 1 and Algorithms 3. The detailed steps of the subroutine are presented in the Appendix.

Algorithm 4: OFTRL-UneqCache

1. Input: $\sigma = 1/\sqrt{C}$, $\delta_t = \|\bar{\theta}_t - \hat{\theta}_t\|_2^2$, $\sigma_1 = \sigma \sqrt{\delta_t}$, $x_1 = \arg\min_{x \in \mathcal{X}_t}(x, \theta_1)$
2. Output: $\{y_t \in \mathcal{X}_t\}_{t=1}^T$ // Feasible discrete caching vector at each slot
3. for $t = 2, 3, \ldots$ do
   4.     $\hat{\theta}_t \leftarrow$ prediction // Obtain gradient prediction for slot $t$
   5.     $\bar{x}_t = \arg\max_{x \in \text{conv}(\mathcal{X}_t)} \left\{-r_{1:t-1}(x) + (x, \Theta_{t-1} + \hat{\theta}_t)\right\}$ // Compute the "almost integral" cache vector
   6.     $\bar{y}_t \leftarrow \text{DepRound}(\bar{x}_t)$ // Perform Randomized Rounding
   7.     $x_t \leftarrow \text{Rand}(\bar{y}_t, k)$ // Receive t-slot request and update total grad
   8.     $\Theta_t = \Theta_{t-1} + \hat{\theta}_t$ // Update the regularization parameter
   9.     $\sigma_t = \sigma \left(\sqrt{\delta_{1:T}} - \sqrt{\delta_{1:t-1}}\right)$
end

Theorem 5. Algorithm 4 ensures, for any time horizon $T$, the expected regret bound:

$$\mathbb{E}\left[ R_{(1/2)}^T \right] \leq \sqrt{C} \sum_{t=1}^{T} \|\theta_t - \hat{\theta}_t\|_2^2$$

Discussion. Compared to the approach we used to extend OFTPL to unequal-sized files, an additional rounding technique (DepRound) was necessary to extend OFTRL. The complexity of this sub-routine is linear in the library size. Thus, despite preserving the order-level complexity of Algorithm 4, handling such files increases the overhead to get the almost-integral caching vectors.
where we abuse notation and redefine $x_t = (k_t, u_t) \in X_k \times \Theta_k$. We can now introduce the new caching variables $k_t$ and routing variables $u_t$. Namely, $k_{nt} \in \{0, 1\}$ decides whether file $n \in N$ is stored at cache $j \in J$ at the beginning of slot $t$, and the $t$-slot caching vector $k_t = (k_{nt} : n \in N, t \in T)$ belongs to $K = \{ k \in [0, 1]^{|N|} : \sum_{n \in N} k_{nt} \leq C_j, j \in J \}$, where $C_j$ is the capacity of cache $j \in J$. We use the routing variable $u_{ntj} \in \{0, 1\}$ to decide the delivery of file $n$ to user $i$ from cache $j$, and define the $t$-slot routing vector $u_t = (u_{ntj} : n \in N, i \in I, j \in J)$ that is selected from the set: $U = \{ u \in [0, 1]^{|N| \times |J|} : \sum_{j \in J} u_{ntj} \leq 1, n \in N, i \in I \}$. Note also that unserved requests, i.e., when the summation is strictly smaller than 1, are satisfied by the root cache. This option, however, yields zero benefit for the users (no cache-hit gains); see also [63]. The request vector $\theta_t$ is redefined to reflect a request’s source and destination: $\theta_t = (\theta_{nt} \in \{0, 1\} : n \in N, i \in I)$, and is drawn from the set: $Q = \{ \theta \in [0, 1]^{|N| \times |I|} : \sum_{n \in N} \sum_{i \in I} \theta_{nti} = 1 \}$.

We can now introduce the $t$-slot utility function:

$$f_t(x_t) = \sum_{n \in N} \sum_{i \in I} \sum_{j \in J} \theta_{ntj} k_{ntj}.$$  

where we abuse notation and redefine $x_t = (k_t, u_t)$. Therefore, the utility-maximizing caching-routing policy at each slot $t$ is found by solving the following problem:

$$P_2 : \max_x \sum_{t=1}^T f_t(x) \quad \text{s.t.} \quad u \in U, \quad k \in K, \quad u_{ntj} \leq k_{ntj}d_{ij}, \quad i \in I, \quad j \in J, \quad n \in N,$$

The request, however, yields zero benefit for the users (no cache-hit gains); see also [63]. The request vector $\theta_t$ is redefined to reflect a request’s source and destination: $\theta_t = (\theta_{nt} \in \{0, 1\} : n \in N, i \in I)$, and is drawn from the set: $Q = \{ \theta \in [0, 1]^{|N| \times |I|} : \sum_{n \in N} \sum_{i \in I} \theta_{nti} = 1 \}$.

We can now introduce the $t$-slot utility function:

$$f_t(x_t) = \sum_{n \in N} \sum_{i \in I} \sum_{j \in J} \theta_{ntj} k_{ntj}.$$  

where we abuse notation and redefine $x_t = (k_t, u_t)$. Therefore, the utility-maximizing caching-routing policy at each slot $t$ is found by solving the following problem:

$$P_2 : \max_x \sum_{t=1}^T f_t(x) \quad \text{s.t.} \quad u \in U, \quad k \in K, \quad u_{ntj} \leq k_{ntj}d_{ij}, \quad i \in I, \quad j \in J, \quad n \in N,$$

The request, however, yields zero benefit for the users (no cache-hit gains); see also [63]. The request vector $\theta_t$ is redefined to reflect a request’s source and destination: $\theta_t = (\theta_{nt} \in \{0, 1\} : n \in N, i \in I)$, and is drawn from the set: $Q = \{ \theta \in [0, 1]^{|N| \times |I|} : \sum_{n \in N} \sum_{i \in I} \theta_{nti} = 1 \}$.

We can now introduce the $t$-slot utility function:

$$f_t(x_t) = \sum_{n \in N} \sum_{i \in I} \sum_{j \in J} \theta_{ntj} k_{ntj}.$$  

where we abuse notation and redefine $x_t = (k_t, u_t)$. Therefore, the utility-maximizing caching-routing policy at each slot $t$ is found by solving the following problem:

$$P_2 : \max_x \sum_{t=1}^T f_t(x) \quad \text{s.t.} \quad u \in U, \quad k \in K, \quad u_{ntj} \leq k_{ntj}d_{ij}, \quad i \in I, \quad j \in J, \quad n \in N,$$
and we define the feasible caching/routing set as $X_h \triangleq \{K \times U \cap \{u_{ni} \leq k_n(d_i)\}\}$. $P_2$ is known to be NP-Hard via a reduction to the set cover problem [63, Sec. 3], [51, Sec. 4.1]. Hence, we will be using below also its integrality-relaxed version $P'_2$ with continuous variables $\hat{x}_t \triangleq (\hat{k}_t, \hat{u}_t)$.

Our strategy for tackling $P_2$ is to use OFTRL on the convex hull of $X_h$ (essentially learning w.r.t. $P'_2$) to optimize $\hat{x}_t$, and then obtain discrete caching vectors with Madow’s sampling applied to each cache separately. As last step we select a proper routing solution for the received request $\hat{\theta}_t$. Namely, upon receiving a request for file $n$ from user $i$, the corresponding routing variable is set to $1$ if any cache connected to $i$ stores file $n$. Thus, we define the auxiliary set $\mathcal{J}^{ni} = \{j \in \mathcal{J} | y_{nj} d_{ij} = 1\}$, and assign $u'_{ni} = 1$ for the $(n, i)$ pair and some $j^* \in \mathcal{J}^{ni}$. It is important to stress that in such uncapacitated models, the routing plan is directly determined once a caching vector is fixed\(^\text{15}\). The detailed steps of the proposed OFTRL schemed are presented in Algorithm 5, where we reuse the regularization scheme from Sec. 4 with the difference that it operates now on the newly defined variables and request vectors. The performance of the algorithm is characterized with the next theorem, the proof of which can be found in Appendix A.6

**Theorem 6.** Algorithm 5 ensures, for any horizon $T$, the expected $(1 - 1/e)$-Regret bound:

$$
\mathbb{E}\left[ R_T^{(1 - 1/e)} \right] \leq 1.3 \sqrt{1 + JC} \sqrt{\sum_{t=1}^{T} \|\theta_t - \tilde{\theta}_t\|_2^2}
$$

**Discussion.** Similar to the single cache, Theorem 6 improves the regret by removing the effect of library size $N$, as opposed to the recently-proposed bipartite OFTL algorithm for equal-sized files in [51]\(^\text{16}\). This improvement comes at the expense of a projection operation in the OFTRL step. Additionally, Algorithm 5 manages to reduce further the constant terms when it has access to high quality predictions for the next-slot requests.

### 8 EXPERT-BASED OPTIMISTIC CACHING

Changing tack in this section, we explore a different approach for optimism that is based on the classical experts framework. Specifically, we consider a model with two experts: a pessimistic (or robust) learner and an optimistic learner. The pessimist expert proposes caching decisions based on the OGA policy [53] that does not use predictions, and provides adversarial regret guarantees. The optimistic expert proposes a caching policy that is optimized solely w.r.t. the predicted request, i.e., as if the predictions are fully reliable. Finally, a meta-learner receives the proposals from the two experts and gradually discerns which of them should be trusted. The expert-based approach to optimistic learning has been previously proposed for continuous caching in [44, 45]. We expand it here to handle discrete decisions and demonstrate it using the single cache scenario.

Formally, the pessimistic expert ($p$) proposes caching $\{z_t(p)\}_t$, according to adaptive OGA:

$$
\tilde{z}_t^{(p)} = \mathcal{P}_{\text{conv}(X)} \left( z_{t-1}^{(p)} + \frac{1}{\sqrt{t}} \theta_t \right),
$$

where $\mathcal{P}_{\text{conv}(X)}$ is the Euclidean projection onto the convex hull of $X$. We denote the regret of this expert by $R_T^{(p)} = \langle \Theta_T, z^* \rangle - \sum_{t=1}^{T} \langle \theta_t, z_t^{(p)} \rangle$, where $z^* = \text{argmax}_{z \in \text{conv}(X)} \langle \Theta_T, z \rangle$. On the other hand, the optimistic expert ($o$) solves the following LP $z^{(o)} = \text{arg max}_{z \in X} \langle \tilde{\theta}_t, z \rangle$, and we denote its regret with $R_T^{(o)} = \langle \Theta_T, z^* \rangle - \sum_{t=1}^{T} \langle \theta_t, z_t^{(o)} \rangle$.

\(^{15}\)In other words, the routing variables are auxiliary in the Femtocaching model, and indeed in [63] these variables where omitted, while they appear with different name in subsequent works, e.g., as virtual caching variables in [51].

\(^{16}\)We note that the bound in [51] contains additionally the number of users since they consider a different request model of one request per user per time slot. Our work can be readily extended in that direction, as explained above.
Algorithm 6: Experts-Cache

1. **Input:** $z_t \in \mathcal{X}$
2. **Output:** $[z_t \in \mathcal{X}]_T$  
   // Feasible caching vector at each slot
3. **for** $t = 2, 3, \ldots$ **do**
   
   4. $z_t^{(p)} = \mathcal{P}_{\text{conv}(\mathcal{X})} \left( z_{t-1}^{(p)} + \frac{1}{\sqrt{t}} \Theta_t \right)$  
      // Pessimistic expert makes proposal
   
   5. $z_t^{(o)} = \arg \max_{z \in \mathcal{X}} \langle \hat{\theta}_t^*, z \rangle$  
      // Optimistic expert makes proposal based on the oracle’s prediction
   
   6. $\hat{z}_t = w_t^{(p)} z_t^{(p)} + w_t^{(o)} z_t^{(o)}$  
      // Meta-learner combines proposals
   
   7. $z_t \leftarrow \text{MadowSample}(\hat{z}_t)$  
      // Obtain the discrete cache vector using Algorithm 7
   
   8. $\Theta_t = \Theta_{t-1} + \hat{\theta}_t$  
      // Receive the request for slot $t$ and update total grad
   
   9. $\tilde{w}_t = \mathcal{P}_{\text{conv}(\Delta_2)} \left( w_{t-1} + \frac{1}{\sqrt{t}} l_t \right)$  
      // Meta-learner observes losses $l_t$ & updates weights

Unlike the previous sections where predictions were used to modify the perturbation and regularization parameters, here they are treated independently through the optimistic expert. The challenge is then to learn which of the two experts’ proposals to follow. To that end, a meta-learner combines the proposals through a set of learned weights. Namely, the meta-learner’s decision variable is $w \triangleq (w_t^{(p)}, w_t^{(o)}) \in \Delta_2$, where $\Delta_2 = \{ w \in [0, 1]^2 ||w||_1 = 1 \}$, and is used to create a convex combination of the provided caching proposals, i.e., $\hat{z}_t = w_t^{(p)} z_t^{(p)} + w_t^{(o)} z_t^{(o)}$. Clearly, by its definition, it holds that $\hat{z}_t \in \text{conv}(\mathcal{X})$. The weights are updated with adaptive OGA:

$$\tilde{w}_t = \mathcal{P}_{\text{conv}(\Delta_2)} \left( w_{t-1} + \frac{1}{\sqrt{t}} l_t \right),$$  

(19)

where $l_t \triangleq \left( \langle \hat{\theta}_t^*, \hat{z}_t^{(p)} \rangle, \langle \hat{\theta}_t^*, \hat{z}_t^{(o)} \rangle \right)$ is the experts’ utility vector at slot $t$. We then have the following result for the regret of the actual mixed action [45, Thm. 3]:

$$\hat{R}_T \{ \hat{z} \}_T = \langle \Theta_T, \hat{z}^* \rangle - \sum_{t=1}^{T} \langle \hat{\theta}_t, \hat{z}_t \rangle \leq R_T^{(w)} + \min \left\{ R_T^{(p)} + R_T^{(o)} \right\} \leq 2\sqrt{2T} + \min \left\{ R_T^{(p)} + R_T^{(o)} \right\}$$  

(20)

Finally, similar to what we have shown in the OFTRL section, it is possible to use Madow’s sampling to recover integral cache states $z_t \in \mathcal{X}$ with the associated bound

$$\mathbb{E} \left[ R_T \{ z_t \} \right] \leq R_T^{(w)} + \min \left\{ R_T^{(p)} + R_T^{(o)} \right\}.$$  

(21)

The steps of this scheme are summarized in Algorithm 6.

**Discussion.** The performance advantage of the bound in (21) is that it can be strictly negative, depending on the optimistic expert’s regret. For example, in case of perfect predictions and non-fixed cost functions, the min term evaluates to $-\epsilon T$ for some $\epsilon > 0$, making the meta-regret negative for large enough $T$. In all cases, the meta-regret is upper bounded by $O(\sqrt{T})$ due to the existence of the robust expert’s regret in the min term, hence we maintain the order-optimal regret for worst-case scenarios with this approach as well. From a computational load perspective, the most challenging step is the projection involved in the calculation of the OGD-based policy (pessimistic expert). However, one can leverage the tailored fast projection proposed in [53] for that operation. It is also important to stress that this framework allows to combine more than one expert, in order to either to e.g., include more than one predictor, see discussion also in [44].

We note that since experts-based optimism is a meta-algorithm whose regret is characterized by that of the experts (i.e., learning algorithms), it can be applied to the other setups of unequal sizes and bipartite caching. The (possibly $\alpha$) meta regret will then be related to that of the (possibly $\alpha$)
regret of the optimistic and pessimistic experts. Finally, it is worth noting that the idea of using the experts model for combining multiple caching policies has been previously proposed in [27], and evaluated in several cases, e.g., see [57] and reference therein, which however do not consider predictors nor provide any theoretical analysis (or, bounds) for the performance of this approach.

9 EXPERIMENTS

We compare the performance of our algorithms with carefully-selected competitors: the FTRL policy which generalizes the OGD from [53], and the FTPL method from [12]. We note that these competitors already showed superior performance to the classical methods of LRU and LFU in their experiments. The request traces are created using the MovieLens dataset [29] which contains time-stamped movie ratings. We assume a request is initiated to a CDN in the same chronological order as their ratings’ timestamps. We consider movies with at least 8 ratings, leading to a library of $N = 10379$ and we set capacity $C = 150$. Each prediction is assumed correct with probability $\rho$. Specifically, we generate a one-hot $\tilde{\Theta}_t$ that has 1 at the file to be requested with probability $\rho$, or at any other random file with probability $1 - \rho$. We also experiment with probabilistic predictions where the vector components represent the probabilities of files being requested (details in Appendix A.7.1). For the experiments with unequal-sized files, we generate the sizes uniformly $s_i \sim U[1, 10]$ and set $C = 500$. For the bipartite network, we use the 100k variation of the MovieLens dataset and consider files with at least 10 ratings, leading to $N = 1152$. The network consists of 3 caches ($C = 150$) and 4 user locations, the first two connected to caches 1 & 2, and the rest to caches 2 & 3.

Fig. 2 shows the average regret (hit-rate gap to the optimal) growth with time for FTPL [12], FTRL [53], and their proposed optimistic counterparts. We experiment with $\rho = 0$, and $\rho = 0.75$. If, e.g., the request predictions were based on recommendations, these reflect the cases where the users do not follow the recommendations ($\rho = 0$), or actually request the recommended movie/file with probability 75%, ($\rho = 0.75$). In addition, we experiment with a sinusoidal $\rho$, which varies between $\rho = 0.5$ and $\rho = 0.9$, with a period of $10^5$ slots. We observe that optimism accelerates and
improves learning the best files to cache, reaching an average improvement of 104\% (for OFTRL) and 37.1\% (for OFTPL) when $\rho = 0.75$, compared to their "vanilla" counterparts (no predictions). Moreover, the performance degradation due to inaccurate predictions is almost negligible: $\leq 8.3\%$ for OFTRL and $\leq 6.6\%$ for OFTPL. We also plot the 0.95-confidence interval of $R_T$ in Figures 2c and 2d, where we note the more condensed distribution for OFTRL: 44.3\% and 26.1\% tighter at $t = 10k$ when $\rho = 0.75, \rho = 0$, respectively. This is because the distribution $\{\hat{x}_t\}_T$ iterates in OFTRL becomes more concentrated with time; an argument that is not directly applicable to OFTPL, where the randomness is due to solving a perturbed linear program. In Fig. 3 we evaluate the algorithms for the unequal sizes case and plot the 1/2-regret. We observe the same pattern of negligible performance degradation when $\rho = 0$, while $\rho = 0.75$ enables an improvement of 35\% (for OFTRL) and 18.8\% (for OFTPL). We kindly refer the reader to the appendix for additional experiments for the experts-caching algorithm, the bipartite caching problem, and with probabilistic predictions of varying qualities.

10 CONCLUSIONS AND FUTURE WORK

In this paper, we presented several provably optimal algorithms that exploit predictions of unknown quality to improve the regret bounds for important variants of the discrete caching problem, while maintaining worst-case guarantees. The tackled problems are general (e.g., the Knapsack problem) and extend beyond caching; and hence the corresponding proposed optimistic algorithms can be applied to other similar problems. Our approach was based on the unified view of FTRL and FTPL algorithms as smoothing operations, where we proposed to make such smoothing adaptive to the predictions’ accuracy. This allowed us to obtain a regret that interpolates between 0 and $O(\sqrt{T})$.

This work also paves the road for several promising extensions. Given that eviction-only policies such as LFU or LRU have provably linear worst-case regret [12], we studied policies that can dynamically prefetch files. Thus, balancing the cache hits with prefetching costs remains to be tackled. Moreover, we note that static regret algorithms, like ours, can be used as a subroutine in algorithms with stronger benchmarks, such as the $\Phi$-regret [26] and the minimum regret over all Finite-State-Predictors [33], and extending the study towards such more-refined benchmarks is certainly interesting. Finally, considering unequal routing utility (e.g., link-capacitated model [55]) and unequal-sized files for the bipartite network model remains an open question [47].

ACKNOWLEDGMENTS

This publication has emanated from research conducted with the financial support of the European Commission through Grant No. 101017109 (DAEMON). Abhishek Sinha is supported in part by a US-India NSF-DST collaborative research grant coordinated by IDEAS-Technology Innovation Hub (TIH) at the Indian Statistical Institute, Kolkata.

REFERENCES

[1] A. Giovanidis, and A. Avranas. 2016. Spatial Multi-LRU: Distributed Caching for Wireless Networks with Coverage Overlaps. arXiv:1612.04363 (2016).
[2] Jacob Abernethy, Chansoo Lee, Abhinav Sinha, and Ambuj Tewari. 2014. Online Linear Optimization via Smoothing. In Proc. of COLT.
[3] Noga Alon and Joel H Spencer. 2016. The probabilistic method. John Wiley & Sons.
[4] Xavier Amatriain. 2012. Building Industrial-Scale Real-World Recommender Systems. In Proc. of RecSys.
[5] Daron Anderson, George Iossifidis, and Douglas Leith. 2022. Lazy Lagrangians with Predictions for Online Learning. arXiv preprint arXiv:2201.02890 (2022).
[6] 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. In Proc. of COLT.
[37] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 1999. On the Existence of a Spectrum of Policies That Subsumes the Least Recently Used (LRU) and Least Frequent Used (LFU) Policies. SIGMETRICS Perform. Eval. Rev. 27, 1 (1999), 134–143.

[38] Yuanyuan Li, Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2021. Online Caching Networks with Adversarial Guarantees. Proc. ACM Meas. Anal. Comput. Syst. 5, 3 (2021), 39 pages.

[39] Thodoris Lykouris and Sergei Vassilvitskii. 2018. Competitive Caching with Machine Learned Advice. In Proc. of ICML.

[40] M. A. Maddah-Ali, and U. Niesen. 2014. Fundamental Limits of Caching. IEEE Trans. Inf. Theory 60, 5 (2014), 2856–2867.

[41] William G. Madow. 1949. On the Theory of Systematic Sampling. The Annals of Mathematical Statistics 20, 3 (1949), 333–354.

[42] Silvano Martello and Paolo Toth. 1990. Knapsack Problems: Algorithms and Computer Implementations. J. Willey & Sons.

[43] William G. Madow. 1949. On the Theory of Systematic Sampling. The Annals of Mathematical Statistics 20, 3 (1949), 333–354.

[44] Naram Mhaisen, George Iosifidis, and Douglas Leith. 2022. Online Caching with no Regret: Optimistic Learning via Recommendations. https://arxiv.org/abs/2204.09345

[45] Naram Mhaisen, George Iosifidis, and Douglas Leith. 2022. Online Caching with Optimistic Learning. In Proc. of IFIP Networking.

[46] Mehryar Mohri and Scott Yang. 2016. Accelerating Online Convex Optimization via Adaptive Prediction. In Proc. of AISTATS.

[47] Silvano Martello and Paolo Toth. 1990. Knapsack Problems: Algorithms and Computer Implementations. J. Willey & Sons.

[48] et al. O. Dekel. 2017. Online Learning with a Hint. In Proc. of NeurIPS.

[49] Felipe Olmos, Bruno Kauffmann, Alain Simonian, and Yannick Carlinet. 2014. Catalog dynamics: Impact of content publishing and perishing on the performance of a LRU cache. In Proc. of ITC.

[50] Francesco Orabona. 2019. A Modern Introduction to Online Learning. https://arxiv.org/abs/1912.13213

[51] Debjit Paria and Abhishek Sinha. 2022. k-experts-Online Policies and Fundamental Limits. In International Conference on Artificial Intelligence and Statistics. PMLR, 342–365.

[52] Minh Tran, Karl H. Bhansali, and Leandros Tassiulas. 2014. Convex Optimization: Improved Regret Bounds via Smoothness. https://arxiv.org/abs/2202.03519

[53] Konstantinos Poularakis, George Iosifidis, and Leandros Tassiulas. 2014. Approximation Algorithms for Mobile Data Caching in Small Cell Networks. IEEE Trans. Commun. 62, 10 (2014), 3665–3677.

[54] Francesco Orabona. 2019. A Modern Introduction to Online Learning. https://arxiv.org/abs/1912.13213

[55] Felipe Olmos, Bruno Kauffmann, Alain Simonian, and Yannick Carlinet. 2014. Catalog dynamics: Impact of content publishing and perishing on the performance of a LRU cache. In Proc. of ITC.

[56] Alexander Rakhlin and Karthik Sridharan. 2013. Optimization, Learning, and Games with Predictable Sequences. In Proc. of NeurIPS.

[57] Dhruv Rohatgi. 2020. Near-Optimal Bounds for Online Caching with Machine Learned Advice. In Proc. of ACM-SIAM SODA.

[58] Daan Rutten, Nico Christianson, Debankur Mukherjee, and Adam Wierman. 2022. Online Optimization with Untrusted Predictions. https://arxiv.org/abs/2202.03519

[59] Sarah Sachs, Nemo Christianson, Debankur Mukherjee, and Adam Wierman. 2022. Online Optimization with Untrusted Predictions. https://arxiv.org/abs/2202.07554

[60] Alireza Sadeghi, Fatemeh Sheikholeslami, and Georgios B. Giannakis. 2018. Optimal and Scalable Caching for 5G Using Reinforcement Learning of Space-Time Popularities. IEEE J. Select. Areas Commun. 12, 1 (2018), 180–190.

[61] Alireza Sadeghi, Fatemeh Sheikholeslami, Antonio G. Marques, and Georgios B. Giannakis. 2019. Reinforcement Learning for Adaptive Caching With Dynamic Storage Pricing. IEEE J. Select. Areas Commun 37, 10 (2019), 2267–2281.

[62] Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G Dimakis, Andreas F Molisch, and Giuseppe Caire. 2013. Femtocaching: Wireless Content Delivery Through Distributed Caching Helpers. IEEE Trans. Inform. Theory 59, 12 (2013), 8402–8413.

[63] T. Si Salem, G. Neglia, and S. Ioannidis. 2021. No-Regret Caching via Online Mirror Descent. https://arxiv.org/abs/2101.12588

[64] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2021. No-Regret Caching via Online Mirror Descent. In Proc. of ICC.

[65] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. 2021. No-Regret Caching via Online Mirror Descent. In Proc. of ICC.

[66] Samuel O. Somuyiwa, András György, and Deniz Gündüz. 2018. A Reinforcement-Learning Approach to Proactive Caching in Wireless Networks. IEEE J. Select. Areas Commun. 36, 6 (2018), 1331–1344.
A APPENDIX

A.1 Comparative Summary with Related Work

Table 1 shows the performance and complexity trade-offs for the presented algorithms, and compares them to recent studies of discrete no-regret caching in the literature. The best case refers to the situation where the request predictions are perfect \( \tilde{\theta}_t = \theta_t, \forall t \). The worst case refers to the situation where predictions are furthest from the truth \( \tilde{\theta}_t = \arg \max_\theta \| \theta - \theta_t \|, \forall t \). The previous studies have the best and worst case columns merged as they do not utilize predictions. Furthermore, the works of [12] and [64] assume and utilize knowledge of the time horizon \( T \) ([38] uses the standard doubling trick) and use the Lipschitz constant for the gradient (i.e., request) vector. Thus, they are not classified as performing Adaptive Learning (Adap. Learn.) as defined by [43], which argues about the advantages of adaptive algorithms of the sort presented here. While the authors in [12] discuss the bipartite model, their simpler linear elastic model of utility is different than the one considered here (see [12, Sec. 3.2]). Hence, we compare to their single cache result. Finally, for algorithm 5, we make explicit the dependence on weights regret \( R^{(w)}_T \), although it is still \( R^{(w)}_T \leq O(\sqrt{T}) \) to clarify the cause of inferior performance of the experts-based optimism in the worst case compared to adaptive smoothing, which even appears in the simulations.

A.2 Madow’s Sampling Algorithm

Algorithm 7 describes how we obtain an integral caching vector from the continuous one. We start by sampling a uniform scalar and then loop for \( C \) iterations, including in our gradually-built set exactly one item per iteration. Hence we ensure the resulting set satisfies the capacity constraint. During an iteration, we include an item if its probability (continuous variable) falls in a carefully designed range: \([\pi_{j-1}, \pi_j]\). Hence, each item is included with probability \( \pi_{j-1} - \pi_j = \hat{x}_j \). We refer the reader to [41] for further details.

A.3 Dependent Rounding Algorithm (DepRound)

The dependent rounding algorithm operates sequentially. At each iteration, it picks two continuous variables and transfers at least one of them into an integer (through the if statements in lines 4 to 8), while adjusting the other one (lines 9 to 12). Hence, we ensure that when the algorithm terminates, only one item is still fractional. The properties of the resulting vector listed in Lemma 2 are proved in [13, Lem. 2.1].
Table 1. Online discrete caching policies with adversarial no regret guarantees: a summary of the contributions and comparison with literature. For the constant $\alpha$, recall that $\alpha = 1$ indicates the regular regret. Otherwise, we have $\alpha$-approximate regret (see equation (17)).

| Alg. | Model and Conditions | Guarantees ($R_T^{(\alpha)} \leq$) | Comput. Complex. | Approx. Const. $\alpha$ | Adap. Learn. |
|------|---------------------|----------------------------------|-----------------|--------------------------|--------------|
|      |                     | Best case | Worst case |                        |              |
| 1    | Single cache • Predictions | 0 | $O(\sqrt{T})$ | $O(N^2)$ | 1 | ✓ |
| 2    | Single cache • Predictions | 0 | $O(\polylog(N) \sqrt{T})$ | $O(N)$ | 1 | ✓ |
| 3    | Single cache • Predictions • Unequal sizes | 0 | $O(\sqrt{T})$ | $O(N^2)$ | $1/2$ | ✓ |
| 4    | Single cache • Predictions • Unequal sizes | 0 | $O(\polylog(N) \sqrt{T})$ | $O(N)$ | $1/2$ | ✓ |
| 5    | Bipartite Network • Predictions | 0 | $O(\sqrt{T})$ | $O(N^2)$ | $1 - 1/e$ | ✓ |
| 6    | Single Cache • Predictions | $b < 0$ | $R_T^{(\alpha)} + O(\sqrt{T})$ | $O(N)$ | 1 | ✓ |
| [12] | Single cache | | $O(\polylog(N) \sqrt{T})$ | $O(N)$ | 1 | − |
| [64] | Single cache | | $O(\sqrt{T})$ | $O(N)$ | 1 | − |
| [51] | Bipartite network | | $O(\polylog(N) \sqrt{T})$ | $O(N)$ | $1 - 1/e$ | ✓ |
| [38] | General network | | $O(\polylog(N) \sqrt{T})$ | $O(N)$ | $1 - 1/e$ | − |

Algorithm 7: Madow’s Sampling (MadowSample)

1. **Input:** $\hat{x} \in [0, 1]^N$, $\sum x_i \leq C$.
2. **Output:** Random set $S$, s.t $|S| = C$ and $Pr(i \in S) = x_i$.
3. Sample a uniformly random scalar $U \in [0, 1]$.
4. Define the cumulative probabilities $\pi_0 = 0$, $\pi_i = \pi_{i-1} + \hat{x}_i$, $\forall 1 \leq i \leq N$.
5. **for** $i = 0, 1, \ldots, C$ **do**
6.   | $S \leftarrow S \cup \{j : \pi_{j-1} \leq U + i < \pi_j\}$
7. **end**
8. **return** $S$.

A.4 Proof of Theorem 4

**Proof.** Since $y_{ti} \in \{0, 1\}$ $\forall t, i$, and the sampling in line 8 is uniform, we get $\mathbb{E}[y_{ti}] = \frac{1}{2}$. Hence

$$\mathbb{E}[y_{ti}] \geq \frac{1}{2} \hat{y}_{ti}. \quad (22)$$

where we have used that $\hat{y}_t \in [0, 1]^N$. Now, from the definition of $1/2$-Regret we have:

$$R_T^{(1/2)} = \frac{1}{2} \sum_{t=1}^T \langle \theta_t, y^* \rangle - \mathbb{E} \left[ \sum_{t=1}^T \langle \theta_t, y_t \rangle \right] \overset{(a)}{\leq} \frac{1}{2} \sum_{t=1}^T \langle \theta_t, y^* \rangle - \sum_{t=1}^T \langle \theta_t, \hat{y}_t \rangle \overset{(b)}{=} 1.84 \sqrt{C \ln \frac{N e}{C}} \left( \frac{\sqrt{T}}{N} \right)^{1/4} \sqrt{\sum_{t=1}^T ||\theta_t - \hat{\theta}_t||_1^2}.$$ 

where inequality (a) follows from the $1/2$-approximation property of the randomized rounding algorithm (22); and (b) follows from the result of Theorem$^{17}$ 3.

$^{17}$Theorem 3 operates on integral decisions $y_t$. Nonetheless, even if we allow $y^*, y_t \in \text{conv}(X)$, they are still integral due to the linear program in line 6 of Algorithm 2 ($\{0, 1\}$ decision variables with non-negative coefficients).
We start from the result of [44, Thm. 1] (or its earlier version from [45]), which provides guarantees where inequality (11) if

Then, by the same argument about uniform sampling in the proof of Theorem 4, we have that

1

By the definition of DepRound (23); and

Algorithm 8: Dependent Rounding (DepRound)

1. Input: \(a \in [0,1]^N, s \in \mathbb{R}_+^N\).
2. Output: \(b\) satisfying points in lemma-2.
3. while \(a \) contains two or more fractional elements do
4. Denote the two left most fractional elements \(a_i\) and \(a_j\).
5. if \(0 \leq s_i a_i + s_j a_j \leq \min\{a_i,a_j\}\) then
6. Set \(b_i = 0\) with probability \(s_j a_j / (s_i a_i + s_j a_j)\). With the remaining probability set \(b_j = 0\)
7. if \(a_i \leq s_i a_i + s_j a_j \leq a_j\) then
8. Set \(a_i = 1\) with probability \(a_i\). With the remaining probability set \(a_i = 0\)
9. if \(a_j \leq s_i a_i + s_j a_j \leq a_i\) then
10. Set \(a_j = 1\) with probability \(a_j\). With the remaining probability set \(a_j = 0\)
11. if \(\max\{a_i,a_j\} \leq s_i a_i + s_j a_j \leq a_i + a_j\) then
12. Set \(b_i = 1\) with probability \((s_j (1-a_j) / (s_i (1-a_i) + s_j (1-a_j)))\). With the remaining set \(b_j = 1\)
13. if \(b_i = 0\) set \(b_j = s_j / s_i a_i + a_j\)
14. if \(b_i = 1\) set \(b_j = a_j - s_j / s_i (1 - a_i)\)
15. if \(b_j = 0\) set \(b_i = a_i + s_j / s_i a_j\)
16. if \(b_j = 1\) set \(b_i = a_i - s_j / s_i (1 - a_j)\)
end
return \(b\)

\(\square\)

A.5 Proof of Theorem 5

First, we show that the 1/2-point-wise approximation holds for \(\{x_t\}_t\). Then, we re-use the result of Theorem 2. In detail, by Lemma 2, the DepRound subroutine returns \(\hat{x}_t\) such that \(\mathbb{E}[\hat{x}_t] = \hat{x}_t\). Then, by the same argument about uniform sampling in the proof of Theorem 4, we have that \(\mathbb{E}[x_t] \geq \frac{1}{2} \hat{x}_t\), where \(x_t \in \mathcal{X}_v\). We recover our 1/2-approximation guarantee for OFTRL iterates \(\{x_t\}_t\):

\[
\mathbb{E}[x_t] \geq \frac{1}{2} \hat{x}_t. \tag{23}
\]

By the definition of 1/2-regret guarantee, we have

\[
R_T^{(1/2)} = \frac{1}{2} \sum_{t=1}^{T} \langle \theta_t, \hat{x}_t^* \rangle - \mathbb{E} \left[ \sum_{t=1}^{T} \langle \theta_t, x_t \rangle \right] \leq \frac{1}{2} \left( \sum_{t=1}^{T} \langle \theta_t, \hat{x}_t^* \rangle - \sum_{t=1}^{T} \langle \theta_t, \hat{x}_t \rangle \right) \overset{\text{a}}{=} \sqrt{C} \sum_{t=1}^{T} ||\theta_t - \hat{\theta}_t||_2^2.
\]

where inequality \(a\) follows from the 1/2-approximation property of the randomized rounding algorithm (23); and \(b\) follows from the result of Theorem 2. Although that theorem states the bound for the regret of the integral decisions \(\{x_t\}_t\), we have seen in its proof that the bound is essentially the same for the continuous actions.

A.6 Proof of Theorem 6

We start from the result of [44, Thm. 1] (or its earlier version from [45]), which provides guarantees for the regret of the continuous variables \(\hat{x}\). For the moment, assume that we have the following point-wise approximation for the decision variables \(\mathbb{E}[\hat{k}] = k, \mathbb{E}[\hat{u}] \geq (1 - 1/e)u\), and that the capacity constraints are respected. Then, due to the linearity of the objective function \(f_t(\cdot)\), we get
that our $\alpha$-regret (with $\alpha = 1 - 1/e$) is:

$$R^{(1-1/e)}_T = \left(1 - \frac{1}{e}\right) \frac{T}{\epsilon} \sum_{t=1}^T \langle \theta_t, \hat{x}^* \rangle - \mathbb{E} \sum_{t=1}^T \langle \theta_t, x_t \rangle \leq 1 - \frac{1}{e} \left( \sum_{t=1}^T \langle \theta_t, x^* \rangle - \sum_{t=1}^T \langle \theta_t, \hat{x}_t \rangle \right)$$

$$\equiv \left(2 - \frac{2}{e}\right) \sqrt{1 + JC} \sqrt{\sum_{t=1}^T \|\theta_t - \hat{\theta}_t\|_2^2},$$

where inequality (a) follows from the $(1-1/e)$-approximation property of the randomized rounding algorithm; and (b) follows from the result of Theorem [44, Thm. 1].

Now, to show that $\mathbb{E}[\hat{k}] = k$, we follow the same argument in the proof of Theorem 2. Namely, due to Madow’s sampling, each file is included with probability $\hat{k}$, and at most $C_j$ files are included at each cache. Hence, $\mathbb{E}[\hat{k} n_j] = k n_j, \forall n, j$. Regarding the point-wise approximation for $u$, we define the set $\mathcal{J}^i$ of caches connected to user $i$:

$$\mathcal{J}^i = \{ j \in \mathcal{J} | d_{ij} = 1 \}.$$

Then, we have

$$\mathbb{E} [u_{nij}] \quad (a) \quad \Pr[u_{nij} = 1] \quad (b) \quad \Pr[\forall j \in \mathcal{J} \hat{k}_{nij} = 1] \quad (c) \quad 1 - \prod_{j \in \mathcal{J}^i} (1 - \hat{k}_{nij}) \quad (d) \quad 1 - e^{-\sum_{j \in \mathcal{J}^i} \hat{k}_{nij}} \quad (e) \quad 1 - e^{-u_{nij}} \quad (f) \quad \left(1 - \frac{1}{e}\right) \hat{u}_{nij}.$$

Where in the above chain of inequalities, (a) follows from $u_{nij}$ being binary variable; (b) by the construction of the algorithm (step 8); and (c) from the independent rounding for each cache. Also, inequality (d) follows from from $e^x \geq 1 + x, \forall x \in \mathbb{R}$; (e) from the relaxed version of the caching/routing constraint of $\mathbb{P}_2$, and finally (f) from the concavity of $1 - e^{-x}$ and the domain of $\hat{u}_{nij}$ being restricted to $[0, 1]$.

### A.7 Additional Simulations

#### A.7.1 Probabilistic Predictions

Let us first demonstrate, with a simple example, that using probabilistic predictions is beneficial for the performance of optimistic algorithms. The regret bound of the proposed algorithms depend on the terms $\|\theta_t - \hat{\theta}_t\|, \forall t$. Now, consider a prediction $\hat{\theta}_t$ that places $\epsilon$ probability mass on the correct file, and the remaining uniformly over the rest of the files in the library. Then, we get:

$$\|\theta_t - \hat{\theta}_t\|_2 \approx 1 - \epsilon, \quad \text{since} \quad \frac{(1 - \epsilon)^2}{(N - 1)} \approx 0,$$

compared to a mis-prediction (or, mistaken) one-hot $\hat{\theta}_t$ which will have $\|\theta_t - \hat{\theta}_t\|_2 = \sqrt{2}$. Using the $\ell_1$ norm, a one-hot mistake costs 2 compared to $2 - 2\epsilon$ for the probabilistic one. We stress again that all the results presented in this work hold both for probabilistic and for deterministic predictions. The former can be taken directly from the output of a forecasting model, while one can create the latter by simply using the highest-probability request.

We continue by presenting experimental results for a probabilistic prediction model with varying accuracy. In detail, in Fig. 4 we measure the regret of the proposed OFTRL and OFTPL policies after 5k time steps (i.e., file request) using the well-known YouTube request trace [74] with $N = 10^4$ and $C = 150$. $R_{5k}$ is measured using prediction vectors with varying density that is placed on the file to
be requested. Namely, if at step $t$, the requested file is $n$, we feed the optimistic algorithms with a prediction vector:

$$\tilde{\theta}_n = \xi \quad \text{and} \quad \tilde{\theta}_{n'} = (1 - \xi)/(N - 1), \forall n' \neq n.$$ 

That is, the prediction vector has $\xi$ probability placed on the file to be requested, and the remaining $(1 - \xi)$ uniformly distributed across the remaining files. We note that when the prediction vector is almost uniform (i.e., $\tilde{\theta}$ contains no useful information), the optimistic versions nearly match the non-optimistic ones.

We can see that at $\xi = 0.1$, both OFTRL and OFTPL already start outperforming their non-optimistic counterparts, by 9.8% and 1.4%, respectively. Also, they outperform the best-in-hindsight benchmark $x^*$ when the accuracy becomes reasonably high ($\xi \geq 0.8$). Lastly, we see that OFTRL has a performance advantage of up to 59.6% compared to OFTPL, when fed the same predictions, at the expense of its additional computation complexity.

A.7.2 Algorithms 5 and 6. Fig. 5 plots the regret for the expert-based Algorithm 6. Note that the $R_T$ can reach negative values, i.e., outperform better the benchmark, when $\rho = 0.5$. This is aligned with the bound in (21) and hints to the fact that stronger benchmarks can be used for this algorithm. However, it performs worse than the regularization-based optimism in the case where $\rho = 0$, achieving regret $R_T = 0.113$ at time $T = 10k$ compared to $R_T = 0.075$ (OFTRL). Lastly, the bipartite utility is shown in Fig. 6, the hit-ratio of OFTRL is approximately 0.49 when $\rho = 0.5$. Expectantly, the performance drops when $\rho = 0$, but steadily increases from 0.30 at $T = 500$, to 0.44 at $T = 10k$. 

Fig. 4. Regret with varying probability mass placed on the correct file in the prediction vector.

Fig. 5. Average regret of experts-based optimism.

Fig. 6. Average OFTRL hit-rate for cache network.

Received August 2022; revised October 2022; accepted November 2022