Online Caching in Tree Networks: Algorithms, Regret, and Complexity

Ativ Joshi ATJOSHI@UMASS.EDU
Rajarshi Bhattacharjee RBHATTACHARJ@UMASS.EDU
University of Massachusetts Amherst, USA

Rajat De RDE@CS.STONYBROOK.EDU
Stony Brook University, USA

Abhishek Sinha ABHISHEK.SINHA@TIFR.RES.IN
Tata Institute of Fundamental Research, Mumbai, India

Cameron Musco CMUSCO@UMASS.EDU
Mohammad Hajiesmaili HAJIESMAILI@CS.UMASS.EDU
University of Massachusetts Amherst, USA

Editors: G. Sukhatme, L. Lindemann, S. Tu, A. Wierman, N. Atanasov

Abstract

We study the problem of online caching when the caches are arranged in a tree network – the root is the source, the clients are at the leaves, and the intermediate nodes are the caches. Each client's request for a file is forwarded up the tree until a cache hit occurs, or the request reaches the root node, with hits at lower levels of the tree yielding higher rewards. The goal is to maximize the total reward over a sequence of requests. The tree-caching problem models caching in many content delivery networks (CDNs) and generalizes work on caching in more restricted network models, such as those with a single client connected to a single cache or a single client connected to a multi-level cache. We show that for general tree networks, finding the optimal static offline caching configuration for a given request sequence is NP-hard. However, in the natural setting where the tree has bounded depth and large enough cache capacities, we present an algorithm that computes a near-optimal configuration with high probability in polynomial time. We then leverage this result to give an online algorithm that achieves $(1+\epsilon)-$approximate sublinear regret for adversarial request sequences when the cache capacity C satisfies $C = \tilde{\Omega}(\frac{1}{\epsilon^2})$. Numerical simulations show that our algorithm outperforms several natural baseline strategies.

Keywords: Regret Minimization, Online Algorithms, Network Caching, Probabilistic Graphical Models

1. Introduction

Efficient content delivery is paramount in modern networked systems like content delivery networks, with caching playing a crucial role in reducing latency, alleviating server load, and minimizing network congestion (Borst et al., 2010; Rosensweig et al., 2013; Rossi and Rossini, 2011; Aggarwal et al., 2002; Chakareski, 2017). Multi-level caching architectures, where caches are deployed at various points in the network hierarchy (e.g., edge servers, regional data centers), offer a powerful paradigm to bring content closer to end-users (Nygren et al., 2010; Li et al., 2018). In these systems, a request from a client traverses up the hierarchy until the requested item is found in a cache or at the origin server (Li et al., 2005). Hits at caches closer to the client are generally more beneficial, leading to lower latency and reduced upstream traffic.

© 2026 A. Joshi, R. Bhattacharjee, R. De, A. Sinha, C. Musco & M. Hajiesmaili.

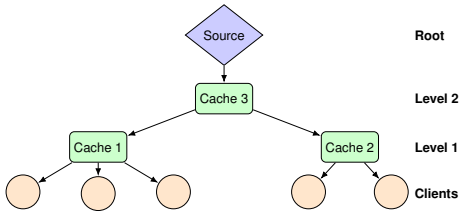


Figure 1: A Tree Network of Caches with Depth 3.

The majority of prior work on online algorithms for caching in the adversarial setting focuses on either the single cache setting (Bhattacharjee et al., 2020; Paschos et al., 2019; Mukhopadhyay and Sinha, 2021) or the bipartite network setting where the caches are on one side and clients are on the other side of the network (Shanmugam et al., 2013; Paria and Sinha, 2021). The problem becomes significantly more difficult when we transition to a multi-client setting with more complicated network topologies (Ioannidis and Yeh, 2016; Li et al., 2021).

In this paper, we consider the setting in which caches, clients, and the source server are connected according to a *tree topology*. The root is the source server, the leaves are the clients, and intermediate nodes are the caches. See Figure 1. A request from a client traverses up the tree through a sequence of caches, from the leaf and finally to the source if a miss occurs at all cache levels. A hit at a lower level yields higher rewards. This hierarchical caching topology models many real-world scenarios in web caching and content delivery networks (Li et al., 2005; Dai et al., 2012; Li et al., 2008; Paschos et al., 2018). We also consider a special case of the tree setting where we have a single client connected to a source server through a series of caches arranged in a path. This special case is important as the path topology models the multi-level cache hierarchy common in many storage configurations and large-scale distributed systems (Yadgar et al., 2007; Gill, 2008; Wu et al., 2010; Meng et al., 2019).

We work in the adversarial setting (Paschos et al., 2019; Bhattacharjee et al., 2020; Paria and Sinha, 2021), where file request sequences can be worst-case. The online caching algorithms we design in this setting focus on minimizing the *regret*, which is a popular metric in the online learning literature (Hazan et al., 2016; Shalev-Shwartz et al., 2012). Regret measures the difference between the caching algorithm's performance (in terms of hits) and that of the single best *static* offline solution – i.e., the optimal fixed cache configuration if the entire file request sequence were known in advance. The goal is to design online algorithms whose regret grows *sublinearly* with time, i.e., the hit rate of the online algorithm approaches the hit rate of the best static offline caching configuration. For additional relevant background, see Appendix A.

Our Contributions We first show that computing the offline optimal caching allocation for the general tree caching is NP-hard, using a reduction from vertex cover. Next, we focus on the more realistic setting of caching trees with bounded depth. In this setting, we propose a randomized algorithm that computes a near-optimal offline caching configuration in polynomial time and ensures that the total number of items placed in each cache is less than the capacity of the cache with good probability as long as the cache capacities are large enough. To achieve this, we consider a fractional version of the problem, where a fractional amount of each file can be stored in each cache. We can formulate this fractional version as a linear program (LP) with exponentially many variables corresponding to the exponentially many caching configurations. However, standard algorithms like

simplex will take exponential time to solve this LP. Hence, we design a separating hyperplane oracle for this LP that works in polynomial time by leveraging novel connections to probabilistic graphical models. Specifically, we formulate the problem of checking if a given solution to the LP is feasible as a MAP (maximum a posteriori) inference query on a Markov network. This problem can be solved efficiently as long as the underlying graph encoding the dependencies in the Markov network has bounded treewidth. We show that this is indeed the case for our tree-structured network and this enables us to solve the LP in polynomial time by leveraging novel connections to probabilistic graphical models. The fractional solution to the LP is then rounded to an integral solution using a randomized rounding technique.

Subsequently, we propose an online algorithm with sublinear $(1+\epsilon)$-approximate regret (assuming the cache capacity $C = \tilde{\Omega}(\frac{1}{\epsilon^2})$) in this setting by utilizing the algorithm for finding the offline integral solution to derive an online algorithm and leveraging the Follow-The-Perturbed-Leader (FTPL) framework (Cesa-Bianchi and Lugosi, 2006).

For the special case when we have a single client connected to multi-level caches (i.e. caches are arranged in a path), we first observe that the optimal cache configuration must maintain orthogonality among the caches, i.e., multiple caches never store the same file at the same time. This enables us to design a simple greedy algorithm that is optimal in the offline setting. We also design a simple FTPL-based online algorithm with sublinear regret.

In Section 2, we describe the notation and the underlying system model used throughout the paper. In Sections 3 and 4, we analyze the offline and online algorithms for general multi-level tree topology. The special case of a single client connected to multi-level caches is analyzed in Appendix H due to a lack of space. Finally, in Appendix I, we show the results of numerical simulations.

2. System Model

We consider a multi-level caching system where multiple clients are connected to a source server through a tree network. The root of the tree is the source server, the leaves are the clients, and the internal nodes are the caches (see Figure 1). We denote the number of clients by n and the number of caches by ℓ. Each cache $j \in [\ell]$ (we use $[\ell] := \{1, \ldots, \ell\}$) has an associated reward r_j and can store up to C_j files. There is a fixed library of N files, and we assume for simplicity that all files have equal size. The system operates over a time horizon of T steps.

At each time step $t \in [T]$, the online caching policy first chooses a cache configuration and then the adversary reveals the requests. Formally, at time t, the policy outputs a configuration $Y(t) = \big(\boldsymbol{y}_1(t), \ldots, \boldsymbol{y}_\ell(t)\big)$, where $\boldsymbol{y}_j(t) \in \{0,1\}^N$ is the indicator vector of files stored at cache j at time t, and $\|\boldsymbol{y}_j(t)\|_1 = C_j$. We denote by $\mathcal{Y}$ the set of all feasible configurations $Y = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_\ell)$ that satisfy the capacity constraints. After the configuration is chosen, each client issues a (possibly adversarial) file request. We assume an *oblivious* adversary, i.e., the entire request sequence is fixed in advance and does not depend on the algorithm's randomization.

For client $i \in [n]$, we denote its request at time t by the one-hot vector $\boldsymbol{z}_i(t) \in \{0,1\}^N$. Overloading the notation, we may write $\boldsymbol{z}_i(t) = f$ to mean that client i requested file f at time t. Let $Z(t) = (\boldsymbol{z}_1(t), \ldots, \boldsymbol{z}_n(t))$ be the collection of all client requests at time t.

Each client i is attached to the root via a unique path of caches. We denote this ordered sequence of caches by $\mathrm{Path}(i) = \big(c_{i,1}, c_{i,2}, \ldots, c_{i,m}\big)$, where $c_{i,1}$ is the cache closest to client i and $c_{i,m}$ is the cache closest to the root. The root (source) is at depth 0 and the clients are at depth $m+1$. For

simplicity, we assume that (i) every cache has the same capacity $C_j = C$, and (ii) all clients are at the same depth $m + 1$ (i.e., the tree is balanced). However, we note that the algorithms and bounds we present in the paper can be straightforwardly extended to a general case when each cache has a different capacity and clients are situated at different depths from the root. We assume throughout that caches are constrained only by storage capacity, not by per-round service rate. Hence a single cached copy may serve multiple requests in the same round, and if several clients have the same earliest hit cache, each request receives that cache's reward.

A request from client i for file f at time t is forwarded upward along $\mathrm{Path}(i)$ until it reaches a cache that stores f. If the file is found at cache j on this path, the algorithm receives reward r_j (we denote the maximum reward by $r_{\max} = \max_{j\in[\ell]} r_j$). If none of the caches on the path store f, the request is served at the source and the reward is 0. Since we assume rewards are non-increasing along any client-to-root path, i.e., $r_{c_{i,1}} \geq r_{c_{i,2}} \geq \cdots \geq r_{c_{i,m}}$, it is natural to say "the reward is that of the *first* cache on the path that contains the file." For later use, we express the same rule as

$$R\big(\boldsymbol{z}_i(t), Y(t)\big) = \max_{j\in\mathrm{Path}(i)} r_j \,\langle \boldsymbol{z}_i(t), \boldsymbol{y}_j(t)\rangle, \tag{1}$$

where $\langle\cdot,\cdot\rangle$ denotes the standard inner product. Let ρ be a (possibly randomized) online policy that, at each time t, chooses the cache configuration $Y(t) \in \mathcal{Y}$ based on the past requests and its own past randomization. The cumulative reward of an online policy ρ over the horizon T is then given by $\sum_{t=1}^{T}\sum_{i=1}^{n} R\big(\boldsymbol{z}_i(t), Y(t)\big)$. We evaluate an online policy ρ using the α-approximate regret defined as

$$\mathrm{Reg}_T^{\rho}(\alpha) = \max_{Y^\star\in\mathcal{Y}} \sum_{t=1}^{T}\sum_{i=1}^{n} R\big(\boldsymbol{z}_i(t), Y^\star\big) \;-\; \alpha\cdot\mathbb{E}\Big[\sum_{t=1}^{T}\sum_{i=1}^{n} R\big(\boldsymbol{z}_i(t), Y(t)\big)\Big], \tag{2}$$

where the expectation is over the randomness of ρ. The factor $\alpha \geq 1$ accounts for computational intractability of the exact offline benchmark (see Appendix B). This is a common approach in online learning literature when the exact solution to the offline problem is intractable (Paria and Sinha, 2021; Sinha et al., 2023). We do not charge an explicit switching cost for changing cache contents across rounds. This is standard in regret-based adversarial caching and isolates the problem of competing with the best static allocation.

3. An Efficient Offline Algorithm for Bounded Depth Trees

Finding the offline optimal caching configuration can be formulated as an integer linear program. We show that computing the exact offline optimum is NP-hard via a reduction from the vertex cover problem (see Appendix B for the formal statement). Accordingly, in this section we develop a polynomial-time $(1+\varepsilon)$-approximate offline oracle for trees with bounded-depth and large enough cache capacities as follows: (i) formulate an *LP relaxation* of the underlying *integral* allocation problem (Section 3.1). This LP is formulated by considering a joint distribution over all possible caching configurations for each file and then maximizing the total reward over the marginal hit probability of each file. (ii) reduce solving the fractional[1] program to finding a *membership* oracle which can efficiently check if a given solution to the LP satisfies the joint and marginal probabilities for each file. (Section 3.2), (iii) implement this oracle by solving the dual via an exact *MAP-based separation oracle* (Section 3.3), (iv) construct an integral distribution with *small-support* for each

1. By "fractional" we mean that instead of integral allocations, we allow randomization over allocations and optimize over the marginal probabilities of each file being stored in each cache.

file that matches the given marginals and sample an integral allocation (Section 3.4), and (v) put everything together to obtain a near-optimal offline caching algorithm for tree networks of bounded depth (Section 3.5).

3.1. Fractional Tree Cache

Let us first define the decision variables of our LP. For some file f, we denote its caching configuration across all caches by $\mathbf{s} = (s_1, \ldots, s_\ell)$ where $s_j = 1$ if file f is present in cache j, and $s_j = 0$ otherwise. Let $\pi_f(\mathbf{s}) \in [0,1]$ be the probability that file $f \in [N]$ is in a system-wide caching configuration $\mathbf{s} \in \{0,1\}^\ell$. We define the vector $\Pi_f = (\pi_f(\boldsymbol{s}))_{\boldsymbol{s}\in\{0,1\}^\ell}$ to be the joint distribution over all possible caching configurations for file f. Let $\Pi = \{\Pi_f\}_{f\in[N]}$ be the set of joint distributions for all files. With a slight abuse of notation, the state allocation vector for file f is denoted by $\boldsymbol{s}_f$.

To simplify the formulation of the objective function and constraints, we define two types of marginal probabilities, which act as auxiliary variables.

- x_{jf}: The marginal inclusion probability that file $f \in [N]$ is stored in cache $j \in [\ell]$. We denote the set of marginal probabilities by $\mathcal{X} = \{x_{jf}\}_{j\in[\ell],f\in[N]}$. We denote the vector of the marginal probabilities by $\boldsymbol{x} \in \mathbb{R}^{\ell N}$.
- h_{ifk}: The marginal hit probability that a request for file $f \in [N]$ by client $i \in [n]$ is "first hit" at cache $c_{i,k}$ (the k-th cache on $\mathrm{Path}(i)$). This implies that file f is stored in cache $c_{i,k}$, but is not stored in any cache $c_{i,k'}$ closer to client i (i.e., for $k' < k$). We denote the set of hit probabilities by $\mathcal{H} = \{h_{ifk}\}_{i\in[n],f\in[N],k\in[m]}$ and the vector of hit probabilities by $\mathbf{h} \in \mathbb{R}^{mnN}$.

Note that since the auxiliary variables are just the marginal and hit probabilities derived from the joint distribution Π, they can be expressed as linear functions of the primary variables $\pi_f(\mathbf{s})$. Let X_{if} denote the total number of requests for file $f \in [N]$ by client $i \in [n]$ over the time horizon. The linear program is then formulated as

$$\textbf{FLP}: \max_{\boldsymbol{x}\in\mathbb{R}^{\ell N},\mathbf{h}\in\mathbb{R}^{mnN},\{\Pi_f\in\mathbb{R}^{2^\ell}\}_{f\in[N]}} \quad \sum_{i\in[n]}\sum_{f\in[N]}\sum_{k=1}^{m} X_{if}\cdot r_{c_{i,k}}\cdot h_{ifk} \tag{3a}$$

$$\text{s.t.,} \sum_{f\in[N]} x_{jf} \le C, \quad \forall j \in [\ell], \tag{3b}$$

$$\sum_{\mathbf{s}\in\{0,1\}^\ell} \pi_f(\mathbf{s}) = 1, \quad \forall f \in [N], \tag{3c}$$

$$x_{jf} = \sum_{\mathbf{s}\in\{0,1\}^\ell : s_j = 1} \pi_f(\mathbf{s}), \quad \forall j \in [\ell], f \in [N], \tag{3d}$$

$$h_{ifk} = \sum_{\mathbf{s}\in\{0,1\}^\ell} I_{ifk}(\boldsymbol{s})\cdot\pi_f(\mathbf{s}), \forall i \in [n], f \in [N], k \in [m], \tag{3e}$$

$$\pi_f(\mathbf{s}) \ge 0, \quad \forall f \in [N], \mathbf{s} \in \{0,1\}^\ell, \tag{3f}$$

$$\text{where,} \quad I_{ifk}(\boldsymbol{s}) = \begin{cases} 1 & \text{if } \boldsymbol{s}_{c_{i,k}} = 1 \text{ and } \boldsymbol{s}_{c_{i,k'}} = 0, \forall k' < k \\ 0 & \text{otherwise.} \end{cases} \tag{3g}$$

The indicator function $I_{ifk}(\boldsymbol{s})$ is 1 if file f is stored in cache $c_{i,k}$ and not stored in any cache $c_{i,k'}$ on the path closer to client i (i.e., for $k' < k$) in the caching configuration $\boldsymbol{s}$, and 0 otherwise. The constraint (3b) ensures that the total expected number of files stored in any cache j does not exceed

its capacity C. (3c) ensures that for each file f, the sum of probabilities over all possible system-wide caching states must be 1. (3d) and (3e) ensure that the marginal probabilities x_{jf} and the path hit probabilities h_{ifk} are consistent with the joint distribution Π_f for each file f. (3f) ensures non-negativity of the probabilities.

While **FLP** exactly models the fractional caching problem, it is computationally intractable due to the exponential number ($N \cdot 2^\ell$) of variables Π_f. In particular, the "consistency constraints" in Eqs. (3c) to (3f) that ensure that there exists a joint distribution Π_f that is consistent with the marginal and hit probabilities are the main bottleneck. Thus, it is not possible to directly solve this LP in polynomial time. We now introduce a more concise reformulation of **FLP**. For each file $f \in [N]$, let $\mathrm{F}_f^{\text{marg}} \subseteq \mathbb{R}^{(mn+\ell)}$ be the set of all feasible marginal inclusion and hit probabilities for f described by the constraints in equations (3c)-(3f). We denote the union of these sets by $\mathrm{F}^{\text{marg}} = \bigcup_{f\in[N]} \mathrm{F}_f^{\text{marg}}$. We now rewrite **FLP** in terms of just the marginal and inclusion probabilities, which are only polynomially many in number, as follows:[2]

$$\textbf{FLP2}: \max_{\boldsymbol{x},\mathbf{h}} \quad \sum_{i\in[n]} \sum_{f\in[N]} \sum_{k=1}^{m} X_{if} \cdot r_{c_{i,k}} \cdot h_{ifk} \tag{4a}$$

$$\text{s.t.,} \sum_{f\in[N]} x_{jf} \leq C, \quad \forall j \in [\ell], \tag{4b}$$

$$[\boldsymbol{x}_f; \mathbf{h}_f] \in \mathrm{F}_f^{\text{marg}}, \forall f \in [N]. \tag{4c}$$

Clearly, the LP in (4) has the same optimal value as the LP in (3). Formally,

Observation 1 *The optimal value of the LP in* (3) *is equal to the optimal value of the LP in* (4)*.*

Note that we can solve an LP in polynomial time using the Ellipsoid method if we have access to a polynomial-time separation oracle for the feasible set. For **FLP2**, the constraints in (4b) can be checked in polynomial time. But, describing the set F^{marg} explicitly still takes $N \cdot 2^\ell$ variables, and hence the main challenge is to efficiently handle the constraints in (4c). Lee et al. (2018) provide a general framework to construct such a separation oracle using a polynomial-time membership oracle that determines if a point belongs to the feasible set. Thus, we focus on constructing an efficient membership oracle that, given any assignment of marginal probabilities and hit probabilities, can check whether it satisfies the constraints in (4b) and (4c), i.e., whether there exists a joint distribution over cache states that is consistent with these marginals, without explicitly constructing F^{marg}.

Having access to such an oracle, **FLP2** can be solved by a polynomial-time convex optimization algorithm using $\tilde{O}((nmN)^2)$ calls to the oracle. This is formalized in Theorem 1 below. For a complete description, see (Lee et al., 2018, Sections 3 and 4).

Theorem 1 (Theorem 14, Lee et al. (2018)) *Given a membership oracle for the feasible set of* **FLP2***, there exists a polynomial-time algorithm that computes an optimal solution using a polynomial number of membership oracle queries.*

We henceforth use the membership-based routine of Lee et al. (2018) as a black box, and focus on implementing an efficient membership oracle for the constraint set of **FLP2**.

2. We use the notation $[\boldsymbol{x}; \mathbf{h}]$ to denote the concatenation of the vectors $\boldsymbol{x}$ and $\mathbf{h}$. By $\boldsymbol{x}_f$ and $\mathbf{h}_f$, we denote the sub-vectors of $\boldsymbol{x}$ and $\mathbf{h}$ corresponding to the file f. Analogous notation is used for other vectors as well.

3.2. Construction of Membership Oracle

Given an *assignment* $\boldsymbol{\mu}^x$ and $\boldsymbol{\mu}^h$ of marginal probabilities $\boldsymbol{x}$ and hit probabilities $\boldsymbol{h}$, the membership oracle needs to verify whether these marginals belong to a valid joint distribution Π (i.e., whether there exists a joint distribution Π that is consistent with these marginals) and also satisfy the capacity constraint. In other words, the oracle needs to verify whether $\boldsymbol{\mu}^x$ satisfies (4b) and $[\boldsymbol{\mu}^x_f; \boldsymbol{\mu}^h_f]$ satisfy (4c). Verifying the capacity constraint in (4b) is straightforward and can be done explicitly. The main challenge is to determine the membership $[\boldsymbol{\mu}^x_f; \boldsymbol{\mu}^h_f] \in \mathrm{F}_f^{\text{marg}}$ for each file $f \in [N]$. To that end, for each file $f \in [N]$, consider the following linear program

$$\textbf{MLP}_f : \min_{\Pi_f} \quad 0 \tag{5a}$$

$$\text{s.t.,} \sum_{\mathbf{s}\in\{0,1\}^\ell} \pi_f(\mathbf{s}) = 1, \tag{5b}$$

$$\mu^x_{jf} = \sum_{\mathbf{s}\in\{0,1\}^\ell : s_j = 1} \pi_f(\mathbf{s}), \quad \forall j \in [\ell], \tag{5c}$$

$$\mu^h_{ifk} = \sum_{\mathbf{s}\in\{0,1\}^\ell} I_{ifk}(\boldsymbol{s})\pi_f(\mathbf{s}), \quad \forall i \in [n], k \in [m], \tag{5d}$$

$$\pi_f(\mathbf{s}) \geq 0, \quad \forall \mathbf{s} \in \{0,1\}^\ell. \tag{5e}$$

The LP in (5) is an explicit description of the set $\mathrm{F}_f^{\text{marg}}$. It is feasible if and only if there exists a joint distribution $\pi_f(\mathbf{s})$ that is consistent with the given marginals $\boldsymbol{\mu}^x_f$ and $\boldsymbol{\mu}^h_f$, i.e., if and only if $[\boldsymbol{\mu}^x_f; \boldsymbol{\mu}^h_f] \in \mathrm{F}_f^{\text{marg}}$. Formally:

Observation 2 *The LP in* (5) *is feasible if and only if* $[\boldsymbol{\mu}^x_f; \boldsymbol{\mu}^h_f] \in \mathrm{F}_f^{\text{marg}}$.

The LP in (5) is an instance of the *Consistency* problem as described by Roughgarden and Kearns (2013), which is to check whether a given set of marginals belongs to a valid joint distribution. Roughgarden and Kearns (2013) show that the consistency problem can be solved efficiently if we can solve the corresponding *MAP Inference* problem efficiently for the underlying *data graph*. We describe these concepts in the subsequent sections, and show that our problem admits a structure required to solve the MAP inference problem efficiently.

Towards that end, we first write down the dual of the LP in (5) using the dual variables α, $\boldsymbol{\beta} = \{\beta_j\}_{j\in[\ell]}$ and $\boldsymbol{\gamma} = \{\gamma_{ik}\}_{i\in[n],k\in[m]}$, as follows (we drop the subscript f for clarity):

$$\textbf{D-MLP}_f : \max_{\alpha\in\mathbb{R},\boldsymbol{\beta}\in\mathbb{R}^\ell,\boldsymbol{\gamma}\in\mathbb{R}^{nm}} \quad \alpha + \sum_{j\in[\ell]} \beta_j \mu^x_j + \sum_{i\in[n]}\sum_{k\in[m]} \gamma_{ik}\mu^h_{ik}, \quad \text{s.t.} \tag{6a}$$

$$\alpha + \sum_{j\in[\ell]} s_j\beta_j + \sum_{i\in[n]}\sum_{k\in[m]} I_{ik}(\boldsymbol{s})\gamma_{ik} \leq 0, \quad \forall \boldsymbol{s} \in \{0,1\}^\ell, \tag{6b}$$

where I_{ifk} is as defined in Equation (3g). The pseudo-code for the membership oracle is given in Algorithm 1 (see Appendix C). The algorithm checks whether the dual is infeasible or unbounded (which implies that the primal is infeasible). We explain how to solve the dual problem $\textbf{D-MLP}_f$ efficiently in Section 3.3.

3.3. MAP Inference Oracle for Constraints

It is convenient to isolate the configuration-dependent part of the dual feasibility constraints (6b) and write

$$\Phi(\boldsymbol{s}) \;:=\; \sum_{j=1}^{\ell} s_j \beta_j \;+\; \sum_{i=1}^{n}\sum_{k=1}^{m} I_{ik}(\boldsymbol{s})\,\gamma_{ik}. \tag{7}$$

A separation oracle for (6) needs to answer the following question: *Is* $\alpha + \Phi(\boldsymbol{s}) \le 0$ *for every* $\boldsymbol{s} \in \{0,1\}^{\ell}$*?* If the answer is *no*, it should produce a concrete s that violates (6b). Let $\boldsymbol{s}^{\star} \in \arg\max_{\boldsymbol{s}} \Phi(\boldsymbol{s})$. If $\alpha + \Phi(\boldsymbol{s}^{\star}) > 0$, then $\boldsymbol{s}^{\star}$ indexes a violated constraint of (6b). If instead $\alpha + \Phi(\boldsymbol{s}^{\star}) \le 0$, then *every* constraint is satisfied, because $\alpha + \Phi(\boldsymbol{s}) \le \alpha + \Phi(\boldsymbol{s}^{\star}) \le 0$ for all $\boldsymbol{s} \in \{0,1\}^{\ell}$. Thus, the problem reduces to finding the maximum of $\Phi(\boldsymbol{s})$ over $\boldsymbol{s} \in \{0,1\}^{\ell}$. Formally:

Observation 3 *Let* $\Phi(\boldsymbol{s})$ *be defined as in* (7)*. Given* $(\alpha, \boldsymbol{\beta}, \boldsymbol{\gamma})$*, let* $\boldsymbol{s}^{*} \in \arg\max_{\boldsymbol{s}\in\{0,1\}^{\ell}} \Phi(\boldsymbol{s})$*. If* $\alpha + \Phi(\boldsymbol{s}^{*}) > 0$*, then* $\boldsymbol{s}^{*}$ *certifies a violated constraint of Eq. (6b), otherwise all constraints are satisfied.*

We cannot optimize $\Phi(\boldsymbol{s})$ by optimizing each s_j individually as there are complicated dependencies among the s_j's due to the indicator variables $I_{ik}(\boldsymbol{s})$. We find $\boldsymbol{s}^{*}$ using the following strategy:

- We first define a *data graph* G encoding the dependencies among the $\boldsymbol{s}_j$'s induced by the indicator variables $I_{ik}(.)$.
- We then define a *Markov network* (a type of probabilistic graphical model) over this data graph by assigning a log-potential function to each component of the graph.
- Finally, we find $\boldsymbol{s}^{*}$ by running a MAP Inference query over this Markov network. There are efficient (polynomial time) algorithms for performing MAP inference over a Markov network as long as the underlying data graph has a bounded treewidth (Roughgarden and Kearns, 2013). Hence, in Lemma 2 we bound the treewidth of the underlying data graph G.

The Markov network construction and the MAP Inference query on it follows standard techniques and is described in Appendix D.1. We now explain the details of the data graph construction and then bound its treewidth:

Data Graph. The data graph is a transitive closure of the directed[3] cache tree and can be created as follows: create one vertex for each cache variable s_j, and connect s_j and $s_{j'}$ whenever the corresponding caches appear together along some client-to-root path (equivalently, when they jointly participate in a single indicator $I_{ik}(\cdot)$). Intuitively, along a client path $\mathrm{Path}(i) = (c_{i,1}, \ldots, c_{i,m})$ the indicators $I_{ik}(\boldsymbol{s})$ depend on a prefix of the variables $\{\boldsymbol{s}_{c_{i,1}}, \ldots, \boldsymbol{s}_{c_{i,k}}\}$, so these variables should be adjacent within G. Because the physical network is a tree of depth m, each such path has length at most m, and hence every dependency induced by $I_{ik}(\cdot)$ lives on a small subset of variables of size at most m. We leverage this structure to bound the treewidth of G in the next lemma (see Appendix D.2 for proof). A visual illustration of the data graph construction is given in Figure 3 in the Appendix.

3. All edges are directed from root to the leaves

Lemma 2 (Treewidth of G) *If the cache tree has depth m, then* $\mathrm{tw}(G) \le m - 1$.

On graphs of treewidth w, exact MAP inference with binary variables runs in time $O(\mathrm{poly}(\ell) \cdot 2^{O(w)})$ via junction-tree methods (see Murphy (2012)). The combination of Observation 3 and Lemma 2 yields:

Theorem 3 *For depth-m trees, the separation oracle for Eq. (6b) can be implemented by a MAP query on G in time $2^{O(m)}\mathrm{poly}(\ell)$ per file. In particular, for constant m, the oracle (and hence the feasibility test for the per-file consistency LP) is polynomial time.*

Also note that in the natural setting when each cache has at least two child caches, $m = O(\log \ell)$. The pseudo-code for the MAP Inference oracle is given in Algorithm 2 (see Appendix D.3). In the next section, we show how to convert the optimal fractional solution of Eq. (4) into an integral allocation by sampling from a distribution with a small support size for each file.

3.4. Sampling from the Optimal Distribution

So far, we have described how to efficiently solve **FLP2** in Eq. (4) (and by extension, **FLP** in Eq. (3)) to obtain the optimal fractional marginals $\boldsymbol{\mu}_f^x$ and $\boldsymbol{\mu}_f^h$, which specify the marginal inclusion probabilities of f in each cache and the marginal probabilities of f being served to each client by each cache along its path to the root. Now, we want to efficiently sample an integral cache allocation using the optimal fractional solution. To do this, first, for each file $f \in [N]$, we construct a *sparse* joint distribution Π_f that is consistent with the marginals $\boldsymbol{\mu}_f^x$ and $\boldsymbol{\mu}_f^h$ obtained from the optimal fractional solution. Then we sample a caching state vector $\boldsymbol{s}_f \in \{0,1\}^\ell$ according to the sparse joint distribution. The sampled caching state vectors for all files will give us an integral cache allocation that satisfies the capacity constraints in expectation.

The key result we use to construct a sparse Π_f is Theorem 3.5 from Roughgarden and Kearns (2013), which guarantees that for any file $f \in [N]$ and its corresponding marginals $\boldsymbol{\mu}_f^h$ and $\boldsymbol{\mu}_f^x$, a valid joint distribution Π_f with small support can be computed efficiently. We state the sampling procedure (Algorithm 3) and its performance guarantees (Theorem 9) in Appendix E.

3.5. Near-Optimal Offline Integral Allocation (High-Probability Capacity)

In the previous section, we described how to obtain an integral cache allocation by drawing, independently for each file f, a state $s_f \in \{0,1\}^\ell$ from a distribution π_f that matches the optimal objective, marginals and capacity constraints of **FLP** in Eq. (3) in expectation. To obtain a high-probability guarantee, we first solve **FLP** with capacities scaled to $\widehat{C} \coloneqq C/(1+\varepsilon)$ for some $\varepsilon > 0$, and then sample from the resulting per-file distributions. This ensures $\mathbb{E}[S_j] = \widehat{C}$ for all j, and a Chernoff bound plus a union bound implies no cache exceeds its original capacity C with high probability as long as $C = \tilde{\Omega}(\frac{1}{\epsilon^2})$. The complete procedure is formalized in Algorithm 4, along with a formal analysis in Theorem 4, in Appendix F. We state the final guarantee for the caching configuration returned by Algorithm 4 below.

Theorem 4 *For any $\varepsilon \in (0,1)$ and $C = \tilde{\Omega}(\frac{\log(\ell/\delta)}{\epsilon^2})$, Algorithm 4 runs in $2^{O(m)}\mathrm{poly}(\ell, n, N)$ time and returns an integral cache allocation that satisfies all cache capacity constraints with probability at least $1-\delta$ and achieves an expected total reward within a $1/(1+\varepsilon)$-factor of the optimal fractional reward.*

4. Online Algorithm

Finally, we combine the offline optimization oracle developed in the previous section with the *Follow-the-Perturbed-Leader* (FTPL) algorithmic framework to obtain an online caching policy with sublinear approximate regret. On each round t, the algorithm observes a request, adds it to the cumulative request counts, adds a perturbation to the cumulative request count, and then calls Algorithm 4 from the previous section on the perturbed cumulative reward which returns a near-optimal (integral) allocation for the perturbed problem that we play on round t. The perturbation ensures that the algorithm achieves a sublinear regret compared to the optimal offline solution in an adversarial setting. The procedure is summarized in Algorithm 5 (see Appendix G.1) with guarantees given in Theorem 5 below (see Appendix G.2 for the proof).

Theorem 5 *Let ρ be the caching policy given by Algorithm 5 with appropriate learning rates $\{\eta_t\}_{t\in[T]}$ and $r_{\max}$ be the maximum attainable reward. Then, for any ϵ such that $C \geq \Omega(\frac{\log(\ell T/\delta)}{\epsilon^2})$, the $(1+\epsilon)$-expected regret of the policy is given by:*

$$\mathrm{Reg}_T(1+\epsilon) \leq O\left(r_{\max} n^{5/4}\sqrt{mC}(\ln Nn)^{1/4}\sqrt{T}\right).$$

Moreover, the online policy ensures that the total number of items placed in each cache at every time step is less than the cache capacity C with probability at least $1-\delta$.

In other words, the average hit of our online algorithm approaches that of the optimal offline policy up to a factor of $1+\varepsilon$ as the number of requests T grows large.

Single User connected to Multiple Caches: For the special case when there is a single client connected to multi-level caches (i.e. a path network), the problem becomes significantly simpler. The key insight is that the optimal offline caching configuration will never store the same file in two different caches. This results in a simple, greedy optimal offline algorithm which stores the top C most requested files in the first cache, the next C most requested file in the second cache and so on. We leverage this to suggest a simple FTPL based online algorithm. See Appendix H for details.

Numerical simulations: We perform experiments comparing our algorithm against three natural benchmark algorithms on a simple network with one root cache and two child caches using both synthetic as well as real-world file request sequences. We find that our proposed algorithm either outperforms the other algorithms or is at least as good as them on certain sequences. Due to space limitations, we include the details of experiments in Appendix I.

5. Conclusion

In this work, we studied the problem of designing efficient caching algorithms for cache networks with tree topologies. We showed that the offline problem is NP-hard. We then proposed both offline and online algorithms that achieve near-optimal performance while satisfying cache capacity constraints with high probability (as long as the cache capacities are large enough). Future work could explore incorporating reconfiguration/switching costs into the tree-caching model, as well as extending our algorithm to a distributed setting where each cache makes decisions based on local information and limited (sublinear in T) communication with other caches.

Acknowledgments

This research is supported by National Science Foundation grants 2045641, 2325956, 2512128, and 2533814. The work of Abhishek Sinha was supported by the Department of Atomic Energy, Government of India, under project no. RTI4014 and by a Google India faculty research award.

References

Jacob Abernethy, Chansoo Lee, and Ambuj Tewari. Perturbation techniques in online learning and optimization. *Perturbations, Optimization, and Statistics*, 233:17, 2016.

Charu Aggarwal, Joel L Wolf, and Philip S. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and data Engineering*, 11(1):94–107, 2002.

Alfred V Aho, Peter J Denning, and Jeffrey D Ullman. Principles of optimal page replacement. *Journal of the ACM (JACM)*, 18(1):80–93, 1971.

Sara Alouf, Nicaise Choungmo Fofack, and Nedko Nedkov. Performance models for hierarchy of caches: Application to modern dns caches. *Performance Evaluation*, 97:57–82, 2016.

Daniel S Berger. CDN Traces. https://github.com/dasebe/optimalwebcaching, 2018.

Daniel S Berger, Philipp Gland, Sahil Singla, and Florin Ciucu. Exact analysis of ttl cache networks: The case of caching policies driven by stopping times. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):595–596, 2014.

Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. Fundamental limits on the regret of online network-caching. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(2):1–31, 2020.

Marcin Bienkowski, Jan Marcinkowski, Maciej Pacut, Stefan Schmid, and Aleksandra Spyra. Online tree caching. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 329–338, 2017.

Sem Borst, Varun Gupta, and Anwar Walid. Distributed caching algorithms for content distribution networks. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

Jacob Chakareski. Vr/ar immersive communication: Caching, edge computing, and transmission trade-offs. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 36–41, 2017.

Marek Chrobak, H Karloof, Tom Payne, and Sundar Vishwnathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

Weibo Chu, Mostafa Dehghan, John CS Lui, Don Towsley, and Zhi-Li Zhang. Joint cache resource allocation and request routing for in-network caching services. *Computer Networks*, 131:1–14, 2018.

Alon Cohen and Tamir Hazan. Following the perturbed leader for online structured learning. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1034–1042, Lille, France, 07–09 Jul 2015. PMLR.

Jie Dai, Zhan Hu, Bo Li, Jiangchuan Liu, and Baochun Li. Collaborative hierarchical caching with dynamic request routing for massive content distribution. In *2012 Proceedings IEEE INFOCOM*, pages 2444–2452. IEEE, 2012.

Julien Dallot, Amirmehdi Jafari Fesharaki, Maciej Pacut, and Stefan Schmid. Dependency-aware online caching. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, pages 871–880. IEEE, 2024.

Asit Dan and Don Towsley. An approximate analysis of the lru and fifo buffer replacement schemes. In *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 143–152, 1990.

Nicaise Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. Performance evaluation of hierarchical ttl-based cache networks. *Computer Networks*, 65:212–231, 2014.

Binny S Gill. On multi-level exclusive caching: Offline optimality and why promotions are better than demotions. In *FAST*, volume 8, pages 1–17, 2008.

Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

Stratis Ioannidis and Edmund Yeh. Adaptive caching networks with optimality guarantees. *ACM SIGMETRICS Performance Evaluation Review*, 44(1):113–124, 2016.

Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 134–143, 1999.

Yin Tat Lee, Aaron Sidford, and Santosh S Vempala. Efficient convex optimization with membership oracles. In *Conference On Learning Theory*, pages 1292–1294. PMLR, 2018.

Keqiu Li, Hong Shen, Francis YL Chin, and Si Qing Zheng. Optimal methods for coordinated enroute web caching for tree networks. *ACM Transactions on Internet Technology (TOIT)*, 5(3): 480–507, 2005.

Wenzhong Li, Edward Chan, Yilin Wang, Daoxu Chen, and Sanglu Lu. Cache placement optimization in hierarchical networks: Analysis and performance evaluation. In *International Conference on Research in Networking*, pages 385–396. Springer, 2008.

Xiuhua Li, Xiaofei Wang, Peng-Jun Wan, Zhu Han, and Victor CM Leung. Hierarchical edge caching in device-to-device aided mobile networks: Modeling, optimization, and design. *IEEE Journal on Selected Areas in Communications*, 36(8):1768–1785, 2018.

Yuanyuan Li, Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. Online caching networks with adversarial guarantees. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–39, 2021.

Xiaodong Meng, Chentao Wu, Minyi Guo, Long Zheng, and Jingyu Zhang. Pam: an efficient power-aware multilevel cache policy to reduce energy consumption of storage systems. *Frontiers of Computer Science*, 13:850–863, 2019.

Samrat Mukhopadhyay and Abhishek Sinha. Online caching with optimal switching regret. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 1546–1551. IEEE, 2021.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

Debjit Paria and Abhishek Sinha. Leadcache: Regret-optimal caching in networks. *Advances in Neural Information Processing Systems*, 34:4435–4447, 2021.

Georgios S Paschos, George Iosifidis, Meixia Tao, Don Towsley, and Giuseppe Caire. The role of caching in future communication systems and networks. *IEEE Journal on Selected Areas in Communications*, 36(6):1111–1125, 2018.

Georgios S Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. Learning to cache with no regrets. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 235–243. IEEE, 2019.

Konstantinos Poularakis, George Iosifidis, Vasilis Sourlas, and Leandros Tassiulas. Exploiting caching and multicast for 5g wireless networks. *IEEE Transactions on Wireless Communications*, 15(4):2995–3007, 2016.

Elisha J Rosensweig, Daniel S Menasche, and Jim Kurose. On the steady-state of cache networks. In *2013 Proceedings IEEE INFOCOM*, pages 863–871. IEEE, 2013.

Dario Rossi and Giuseppe Rossini. Caching performance of content centric networks under multi-path routing (and more). *Relatório técnico, Telecom ParisTech*, 2011:1–6, 2011.

Tim Roughgarden and Michael Kearns. Marginals-to-models reducibility. *Advances in Neural Information Processing Systems*, 26, 2013.

Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G Dimakis, Andreas F Molisch, and Giuseppe Caire. Femtocaching: Wireless content delivery through distributed caching helpers. *IEEE transactions on information theory*, 59(12):8402–8413, 2013.

Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. No-regret caching via online mirror descent. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 8(4):1–32, 2023.

Abhishek Sinha, Ativ Joshi, Rajarshi Bhattacharjee, Cameron Musco, and Mohammad Hajiesmaili. No-regret algorithms for fair resource allocation. *Advances in Neural Information Processing Systems*, 36:48083–48109, 2023.

Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

Martin J Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge university press, 2019.

Chentao Wu, Xubin He, Qiang Cao, and Changsheng Xie. Hint-k: An efficient multi-level cache using k-step hints. In *2010 39th International Conference on Parallel Processing*, pages 624–633. IEEE, 2010.

Gala Yadgar, Michael Factor, and Assaf Schuster. Karma: Know-it-all replacement for a multilevel cache. In *Fast*, volume 7, pages 25–25, 2007.

Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

Neal Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pages 928–936, 2003.

Appendix A. Related Work

Caching is a very well-studied problem. Most existing caching algorithms have performance guarantees in terms of one of two different metrics – *competitive ratio* or *regret*. The majority of early work on caching algorithms focused on optimizing the competitive ratio which is the ratio between the performance of the caching policy (usually expressed in terms of number of cache misses of the requested files) and that of the optimal caching policy that knows the whole sequence of file requests in advance. The policies studied under this model are mostly cache *eviction* policies which must admit the requested file in the cache if it is not already present. The offline optimal policy among all cache eviction policies is known to be Belady's MIN algorithm (Aho et al., 1971). The seminal work in Sleator and Tarjan (1985) was among the first to study the competitive ratio of caching algorithm with adversarial file requests. Popular caching algorithms studied under this model include the Least Recently Used (LRU), Least Frequently Used (LFU) (Lee et al., 1999), and First-In-First-Out (FIFO) (Dan and Towsley, 1990) policies. LRU and FIFO have a competitive ratio equal to the size of the cache while LFU is known to have an unbounded competitive ratio in the worst case. Variants of the caching problem taking into account different fetching costs for files (Chrobak et al., 1991; Young, 1994) different file sizes (Young, 2002) and dependencies between the files being cached (Bienkowski et al., 2017; Dallot et al., 2024) have also been studied.

In contrast, our work focuses on the *regret* of the proposed online caching algorithms. The regret is a metric which is popular in the online learning literature (Hazan et al., 2016; Shalev-Shwartz et al., 2012). It measures the difference between the cost/reward of the proposed algorithm and the

cost/reward of the optimal *static* offline solution which has access to the entire request sequence in advance. The goal here is to design algorithms which have regret which grows sublinearly with the time horizon. Paschos et al. (2019) proposed an online gradient descent (Zinkevich, 2003) based fractional caching policy (i.e. which allows caching of file fractions) which has sublinear regret under an adversarial file request sequence. They also considered two different scenarios in the paper – a single client connected to a single cache and a bipartite graph network model where a set of clients are connected to a set of caches (known as the femtocaching problem from Shanmugam et al. (2013)). Bhattacharjee et al. (2020) showed that the online gradient based policies for fractional caching proposed in (Paschos et al., 2019) are optimal in the single client setting and nearly optimal for the femtocaching setting. They also proposed a Follow-The-Perturbed-Leader (FTPL) (Cesa-Bianchi and Lugosi, 2006) based online integral caching algorithms which is nearly regret optimal for the single client case. Subsequently, Paria and Sinha (2021) proposed an FTPL based integral caching algorithm with sublinear regret for the femtocaching setting. More general online caching algorithms where files are requested in batches have also been studied (Si Salem et al., 2023)

Our paper studies the case when caches are arranged according to a tree network. Most of the existing work on networks of caches either considers the offline setting (Poularakis et al., 2016; Borst et al., 2010; Shanmugam et al., 2013; Li et al., 2005; Gill, 2008) or considers the request sequence to be stochastic (Fofack et al., 2014; Ioannidis and Yeh, 2016; Chu et al., 2018; Berger et al., 2014; Alouf et al., 2016). There are some results which show that finding the offline optimal becomes hard in caching networks. For example, Shanmugam et al. (2013) showed that finding the offline optimal version of the fractional femtocaching problem (i.e. when clients and caches are connected as a bipartite graph) is NP-hard. Li et al. (2021) analyzed a setting which is very close to the one considered in this paper. They study online algorithms for a network of caches arranged in an arbitrary topology with adversarial file request sequences. In their setting, file requests can originate at any node in the network and the requests are routed along some pre-specified paths over the network. They show that finding the offline optimal in this setting is NP-hard. Our setting can be considered to be a special case of theirs since we consider a tree topology where file requests always originate at the leaves and travels upwards towards the root. However, note that the hardness from Li et al. (2021) doesn't directly translate to our special case of tree networks.

Appendix B. Problem Complexity and Hardness

In this section, we show that the offline tree caching problem is NP-hard via a reduction from the wel-known NP-hard problem of Vertex Cover. Let us first define a decision version of the tree caching problem and also recall the definition of the vertex cover problem, from which we will derive our hardness result

Definition 6 (Tree Cache) *Given a tree $\mathcal{T}$ with ℓ caches, where each cache j has capacity C_j and reward 1, a set $\mathcal{F}$ of files of equal size, horizon T and a sequence of file requests $\mathcal{Z} = \{\boldsymbol{z}(t)\}_{t=1}^{T}$ and an integer threshold r, the tree cache problem* $\text{TC}(\mathcal{T}, \mathcal{Z}, \mathcal{F}, T, r)$ *asks whether there exists a fixed cache configuration $\boldsymbol{y}^*$ such that the total reward $\sum_{t=1}^{T}\sum_{i=1}^{n} R(\boldsymbol{z}_i(t), \boldsymbol{y}_i^*)$ (as defined in Eq. (1)) is at least r.*

Definition 7 (Vertex Cover) *Given a graph $G(\mathcal{V}, \mathcal{E})$ and an integer k, the vertex cover problem* $\text{VC}(G(\mathcal{V}, \mathcal{E}), k)$ *is to check if there exists a subset of vertices $\mathcal{V}' \subseteq \mathcal{V}$ such that $|\mathcal{V}'| \leq k$ and every edge in $\mathcal{E}$ is incident to at least one vertex in $\mathcal{V}'$.*

The Vertex Cover problem is a well-known NP-Hard problem. The following theorem shows that the TC problem is NP-hard via reduction from Vertex Cover.

Theorem 8 $\text{TC}(\mathcal{T}, \mathcal{Z}, \mathcal{F}, T, r)$ *is NP-hard.*

Proof [Proof of Theorem 8] Given an instance $\text{VC}(G(\mathcal{V}, \mathcal{E}), k)$ of vertex cover, we construct an instance of $\text{TC}(\mathcal{T}, \mathcal{Z}, \mathcal{F}, T, r)$ with a cache tree $\mathcal{T}$ of depth 3 as shown in Figure 2. At level 3, there are $n = |\mathcal{E}|$ clients as the leaves, each corresponding to an edge in $\mathcal{E}$. Each client $i \in [n]$ is connected to its own unique cache c_i, located at level 2. Each of these caches has a capacity $C_j = 1, \forall j \in [n]$. Each cache c_i is connected to the cache c_{n+1} of capacity $C_{n+1} = k$ at level 1, which is connected to the source server at level 0. The total number of files is $N = |\mathcal{V}|$, each corresponding to a vertex in $\mathcal{V}$. We take a time horizon of $T = 2$. For each edge $(u_i, v_i) \in \mathcal{E}$, the corresponding client i requests two files $\{u_i, v_i\}$ i.e. $\boldsymbol{z}_i(1) = u_i$ and $\boldsymbol{z}_i(2) = v_i$. All the caches have the same reward of 1, and we set the target reward to $r = 2n$.

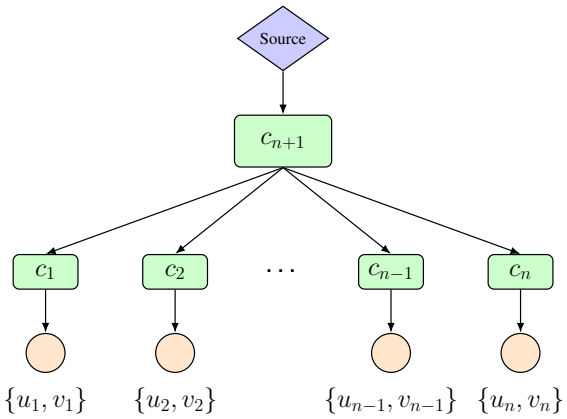


Figure 2: Tree Network for Lower Bound

Clearly, the total number of file requests is $2n$. If there exists a caching configuration that gives a reward of $2n$ (i.e. all requests are satisfied), then the size k vertex cover of G corresponds to the set of files stored in the cache c_{n+1}. So, given an algorithm for $\text{TC}(\mathcal{T}, \mathcal{Z}, \mathcal{F}, T, r)$, we can solve $\text{VC}(G(\mathcal{V}, \mathcal{E}), k)$. ■

Appendix C. Membership Oracle for Marginal Consistency

Algorithm 1 Membership Oracle for Marginal Consistency

Input : Candidate marginals $(\boldsymbol{\mu}^h, \boldsymbol{\mu}^x)$
Output : `true` if $(\boldsymbol{\mu}^h, \boldsymbol{\mu}^x)$ is feasible, `false` otherwise
Oracle Access: MAP Inference Oracle `MAPInferenceOracle` (Algorithm 2)

// Check capacity constraints
foreach *cache* $j \in [\ell]$ **do**
  if $\sum_{f \in [N]} \mu^x_{jf} > C$ **then**
    return `false`
  end
end
// Check consistency for each file
foreach *file* $f \in [N]$ **do**
  Solve **D-MLP**$_f$ via ellipsoid using `MAPInferenceOracle` as a separation oracle (Algorithm 2).
  // dual is infeasible
  if ***D-MLP***$_f$ *is not feasible* **then**
    return `false`
  end
  //dual is unbounded
  if *Optimal value of* ***D-MLP***$_f \neq 0$ **then**
    return `false`
  end
end
return `true`

Appendix D. MAP Inference and Markov Networks

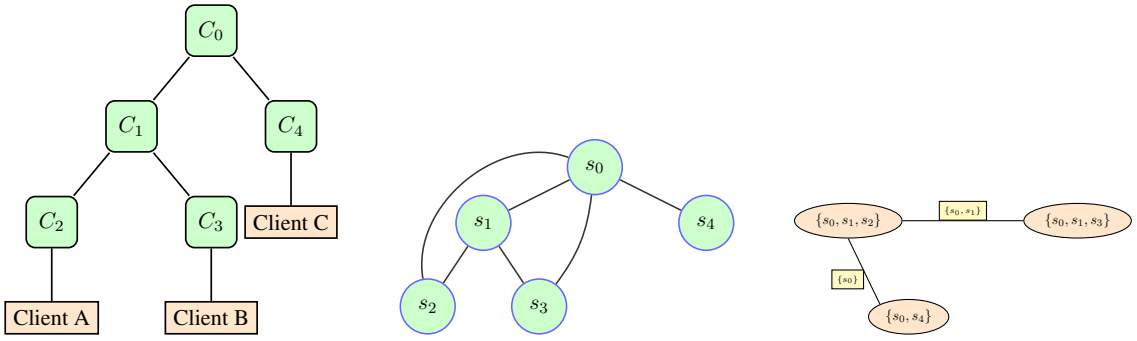
D.1. Background

Markov network for Φ. We now construct a *Markov network* over the vertices of G_f by assigning:

- a unary clique at each cache j with log-potential $\log \psi_j(\boldsymbol{s}_j) = \beta_j \boldsymbol{s}_j$, and
- for each pair (i, k), a clique supported on the prefix $\{\boldsymbol{s}_{c_{i,1}}, \ldots, \boldsymbol{s}_{c_{i,k}}\}$ with log-potential $\log \psi_{i,k}(\boldsymbol{s}) = \gamma_{ik} I_{ik}(\boldsymbol{s})$.

The total log-score of a configuration s in this network is precisely $\sum_j \log \psi_j(\boldsymbol{s}_j) + \sum_{i,k} \log \psi_{i,k}(\boldsymbol{s}) = \Phi(\boldsymbol{s})$ (up to the additive constant α, which does not affect maximization). Next, we show how to maximize the potential function $\Phi(\boldsymbol{s})$ efficiently by running MAP inference query on this Markov network.

MAP inference. Given an undirected graphical model (Markov network) over binary variables $\boldsymbol{s} \in \{0,1\}^\ell$ with clique potentials $\{\psi_C\}$, the (unnormalized) score of a configuration s is $\prod_C \psi_C(\boldsymbol{s}_C)$,



(*a*) Cache Tree (*b*) Data Graph for MAP Inference (*c*) Junction Tree

Figure 3: Cache Tree and the Corresponding Data Graph and Junction Tree for MAP Inference. The source node is omitted for clarity.

or equivalently the log-score $\sum_C \log \psi_C(\boldsymbol{s}_C)$. A *MAP (maximum a posteriori) inference query* returns a configuration

$$\boldsymbol{s}^* \in \arg \max_{s \in \{0,1\}^\ell} \sum_C \log \psi_C(\boldsymbol{s}_C),$$

and optionally the optimum value $\max_{\boldsymbol{s}} \sum_C \log \psi_C(\boldsymbol{s}_C)$. Importantly, MAP does *not* require computing the normalizing constant (partition function) or any marginal probabilities; it only finds the highest-scoring assignment under the specified potentials. In our construction, the clique log-potentials are chosen so that the log-score equals $\Phi(\boldsymbol{s})$ up to an additive constant, hence a MAP query directly produces $\boldsymbol{s}^*$ maximizing $\Phi(\boldsymbol{s})$. We then check the sign of $\alpha + \Phi(\boldsymbol{s}^*)$ to either return a violated constraint of Eq. (6b) or certify feasibility (see Lemma 3).

D.2. Proof of Lemma 2

Proof Lemma 2 We define a tree decomposition (also called a *Junction tree*) of G_f. For each client i with path $\mathrm{Path}(i) = (c_{i,1}, \ldots, c_{i,m})$, define bags $B_{i,k} = \{\boldsymbol{s}_{c_{i,1}}, \ldots, \boldsymbol{s}_{c_{i,k}}\}$ for $k = 1, \ldots, m$ and connect adjacent bags $(B_{i,j}, B_{i,j+1})$ for $j = 1, \ldots, m-1$. An example of a cache tree, the corresponding data graph (i.e. G_f), and the tree decomposition (junction tree) of the data graph is shown in Figure 3. Note that every edge of G_f lies along a pair of vertices on some $\mathrm{Path}(i)$, hence it must be present in some $B_{i,k}$. Each variable s_j appears in a contiguous set of bags (those where its cache is within the first k positions on some $\mathrm{Path}(i)$), so the running intersection property holds. Bag size $\leq m$ gives treewidth $\leq m - 1$. ■

D.3. Algorithm

Algorithm 2 `MAPInferenceOracle`: MAP Inference Separation Oracle

Input : Dual variables $(\alpha_f, \boldsymbol{\beta}_f, \boldsymbol{\gamma}_f)$ for a file f. Network tree $\mathcal{T}$ of depth m.
Output: A tuple (`status`, $\mathbf{s}^*$), where `status` is `VIOLATED` or `SATISFIED`.

Construct a graphical model for file f with nodes representing the binary variables $\{s_j\}_{j\in[\ell]}$.
Solve the MAP inference problem using the junction tree algorithm to find the optimal assignment: $\mathbf{s}^* \leftarrow \arg\max_{\mathbf{s}\in\{0,1\}^\ell} \Phi(\mathbf{s})$.
Compute the maximum potential $\Phi_{\max} = \Phi(\mathbf{s}^*)$.
if $\Phi_{\max} > -\alpha_f$ **then**
  return (`VIOLATED`, $\mathbf{s}^*$)
end
else
  return (`SATISFIED`, $\emptyset$)
end

Appendix E. Sampling from the Optimal Distribution

Algorithm 3 Sampling for Integral Allocation

Input: Feasible (μ^H, μ^X)
Output: Integral cache configuration $Y \in \{0,1\}^{\ell\times N}$
Oracle Access: Access to membership oracle `MAPInferenceOracle` (Algorithm 2)
for $f = 1$ **to** N **do**
  Run ellipsoid on D-MLP$_f$ with `MAPInferenceOracle` and collect the violated constraints to form the reduced primal (Theorem 9);
  Solve the (polynomial-sized) reduced primal to obtain π_f with small support via Theorem 9;
  Sample $s_f \sim \pi_f$ and set $Y_{j,f} \leftarrow (s_f)_j$ for all $j \in [\ell]$;
end
return Y

Theorem 9 (Per-file small-support construction) *Fix a tree of depth m and a file f. Given feasible marginals $(\boldsymbol{\mu}_f^X, \boldsymbol{\mu}_f^H)$, there is a polynomial-time algorithm that computes a distribution π_f over $\{0,1\}^\ell$ which is consistent with $(\boldsymbol{\mu}_f^X, \boldsymbol{\mu}_f^H)$ and has support size at most $m_f + 1$, where $m_f = |\boldsymbol{\mu}_f^X| + |\boldsymbol{\mu}_f^H|$ is the total number of marginal equalities for file f. Moreover, the algorithm runs in time* $\mathrm{poly}(m_f) \cdot 2^{O(m)}\mathrm{poly}(\ell)$.

Proof We briefly outline the main steps of the procedure; see Roughgarden and Kearns (2013) for full details. Fix a file f and its feasible marginals $(\boldsymbol{\mu}_f^X, \boldsymbol{\mu}_f^H)$ obtained by solving **FLP2** in Eq. (4). Let $m_f = |\boldsymbol{\mu}_f^X| + |\boldsymbol{\mu}_f^H|$ be the total number of marginal equalities for this file. The idea is to construct a "reduced primal" LP with only polynomially many variables that still captures the feasibility of the original per-file consistency **MLP**$_f$ in Eq. (5). The steps are as follows:

First, we will construct the "reduced dual" from the dual LP **D-MLP**$_f$ in Eq. (6) by only retaining those constraints in Eq. (6b) that are necessary to certify feasibility. Recall that we solve

D-MLP$_f$ using the ellipsoid method. Its separation oracle, given a candidate dual point, searches for a state $s \in \{0,1\}^\ell$ that maximizes the potential $\Phi(\boldsymbol{s})$ on the data graph G_f; a strict maximizer identifies a violated dual constraint. The ellipsoid runs for only $\mathrm{poly}(m_f)$ iterations, so we collect at most $K \le \mathrm{poly}(m_f)$ violated constraints in total. Keeping only these constraints yields the *reduced dual*, a smaller dual system containing exactly the constraints needed to certify feasibility and optimality for file f (cf. Figure 2 in Roughgarden and Kearns (2013)). Note that unlike the original dual, the reduced dual has only polynomially many constraints.

By strong duality, the reduced dual constructed above is feasible. Therefore, the "reduced primal" that is *restricted* to only the K variables corresponding to the K constraints in the reduced dual is also feasible. Interpret this reduced primal as choosing a distribution π_f over a finite set $S_f \subseteq \{0,1\}^\ell$ (the K states we collected), subject to: (i) probability mass sums to one, (ii) π_f matches the m_f marginal equalities for $(\boldsymbol{\mu}_f^X, \boldsymbol{\mu}_f^H)$, and (iii) nonnegativity for all probabilities. Thus, the reduced primal has a polynomial number of variables, and there is an optimal solution π_f supported on at most $m_f + 1$ states in S_f.

The ellipsoid method performs only polynomially many iterations in m_f. Each iteration makes one MAP call, which costs $2^{O(m)}\mathrm{poly}(\ell)$. Solving the final reduced primal has only $K \le \mathrm{poly}(m_f)$ variables, so it is polynomial in m_f. Thus, the total time is $\mathrm{poly}(m_f) \cdot 2^{O(m)}\mathrm{poly}(\ell)$.

■

Let R^*_{FRAC} and R^*_{INT} be the rewards of the optimal fractional and integral allocations respectively. Clearly, $R^*_{\mathrm{FRAC}} \ge R^*_{\mathrm{INT}}$ since the fractional program is a relaxation of the integral one. Let $R_{\mathrm{INT}}(Y)$ be the reward of the integral allocation Y constructed by Algorithm 3. Hence, the expected reward of Algorithm 3 is denoted by $\mathbb{E}[R_{\mathrm{INT}}(Y)]$, where the expectation is taken over the randomness in sampling. The next corollary formalizes the guarantees of Algorithm 3.

Corollary 10 (Sampling guarantees in expectation) *Construct π_f with small support consistent with the optimal **FLP** marginals (μ^X, μ^H) and sample $s_f \sim \pi_f$ independently for each $f \in [N]$; set $Y_{j,f} \leftarrow (s_f)_j$. Then:*

1. ***Expected objective equals the FLP optimum.***

$$\mathbb{E}[R_{\mathrm{INT}}(Y)] \;=\; \mathbb{E}\Big[\sum_{i,f}\sum_{k=1}^{m} X_{if}\, r_{c_{i,k}}\, I_{ifk}(Y_f)\Big] \;=\; R^*_{\mathrm{FRAC}} \quad \textit{(by Eq. (3a)).}$$

2. ***Marginals are matched in expectation.*** *For all caches j, files f, clients i, and depths k,*

$$\mathbb{E}[Y_{j,f}] \;=\; \mu^X_{jf} \quad \textit{(by Eq. (3d))}, \qquad \mathbb{E}[I_{ifk}(Y)] \;=\; \mu^H_{ifk} \quad \textit{(by Eq. (3e)).}$$

3. ***Capacity holds in expectation.*** *For every cache j,*

$$\mathbb{E}\Big[\sum_{f} Y_{j,f}\Big] \;=\; \sum_{f} \mu^X_{jf} \;\le\; C \quad \textit{(by the **FLP** capacity constraint, Eq. (3b)).}$$

Proof By construction, π_f realizes the **FLP** marginals (μ^X, μ^H) for each file (Eqs. (3d) and (3e)), which directly gives the second statement. The first statement follows by linearity of expectation

applied to the **FLP** objective (Eq. (3a)). The third statement follows from $\mathbb{E}[Y_{j,f}] = \mu^X_{jf}$ and Eq. (3b). ■

Note that the allocation Y may violate capacity on a particular sample, since Corollary 10 only guarantees that capacity holds in expectation. The next section upgrades this to a high-probability statement without degrading the expected objective value too much.

Appendix F. Near-Optimal Offline Integral Allocation (High-Probability Capacity)

Algorithm 4 Near-Optimal Offline Integral Allocation

Input: Tree T, requests $\{X_{if}\}$, rewards $\{r_j\}$, capacities $\{C\}$, parameter $\varepsilon > 0$
Output: Integral configuration $Y \in \{0,1\}^{\ell \times N}$

Solve the **FLP** with capacities set to $\widehat{C} = C/(1+\varepsilon)$ to obtain optimal marginals (μ^h, μ^x);
For each file $f \in [N]$, construct a sparse-support joint π_f consistent with (μ^h_f, μ^x_f) and sample $s_f \sim \pi_f$; set $Y_{j,f} \leftarrow (s_f)_j$ for all $j \in [\ell]$;
return Y

Let $R^*_{\mathrm{FRAC}}(C)$ be the optimal value of **FLP** as a function of the cache capacity. Fix $\varepsilon \in (0,1)$ and set $\widehat{C} := C/(1+\varepsilon)$. Let $(\widehat{\boldsymbol{\mu}}^x, \widehat{\boldsymbol{\mu}}^h)$ be an optimal solution of **FLP** with capacities $\widehat{C}$.

Theorem 4 (restated) *Run Algorithm 4 with parameter $\varepsilon \in (0,1)$. Then:*

1. ***Expected reward.***

$$\mathbb{E}\big[R_{\mathrm{INT}}(\widehat{Y})\big] \;=\; R^*_{\mathrm{FRAC}}(\widehat{C}) \;\in\; \big[\, R^*_{\mathrm{FRAC}}(C)/(1+\varepsilon)\;,\; R^*_{\mathrm{FRAC}}(C)\,\big].$$

2. ***Capacity in expectation and with high probability.*** *For every cache j,*

$$\mathbb{E}\Big[\sum_f Y_{j,f}\Big] \;=\; \sum_f \mu^x_{jf} \;\le\; \widehat{C} \;=\; \frac{C}{1+\varepsilon}.$$

Moreover, letting $S_j := \sum_f Y_{j,f}$, we have the tail bound

$$\Pr\Big(\exists j \in [\ell] :\; S_j > C\Big) \;\le\; \ell \cdot \exp\Big(-\frac{\varepsilon^2\, C}{3(1+\varepsilon)}\Big),$$

so the original capacity C is satisfied at all caches simultaneously with high probability.

Proof For the first claim, linearity of expectation applied to Eq. (3a) gives $\mathbb{E}\big[R_{\mathrm{INT}}(\widehat{Y})\big] = \widehat{R}^*_{\mathrm{FRAC}}$. Concavity of $R^*_{\mathrm{FRAC}}(C)$ with $R^*_{\mathrm{FRAC}}(0) = 0$ implies $R^*_{\mathrm{FRAC}}(\widehat{C}) = R^*_{\mathrm{FRAC}}(\frac{C}{1+\varepsilon}) \ge R^*_{\mathrm{FRAC}}/(1+\varepsilon)$ and trivially $R^*_{\mathrm{FRAC}}(\widehat{C}) \le R^*_{\mathrm{FRAC}}(C)$.

For the second claim, for fixed j, the variables $\{Y_{j,f}\}_{f=1}^N$ are independent Bernoulli, with means μ^x_{jf} summing to at most $\widehat{C}$. Applying the multiplicative Chernoff bound to S_j and a union bound over $j \in [\ell]$ yields the stated probability. ■

Runtime. Solving **FLP** with capacity $\widehat{C}$ uses the same membership/separation framework as Sections 3.1 to 3.3, requiring $\widetilde{O}((nmN)^2)$ membership calls; each call and the per-file small-support construction invoke the MAP oracle in time $2^{O(m)}\mathrm{poly}(\ell)$. Thus, our final algorithm runs in $\mathrm{poly}(n, \ell, N) \cdot 2^{O(m)}$ time.

Appendix G. Online Algorithm and Regret Analysis

G.1. Online Algorithm

Algorithm 5 Online Integral Cache Allocation via FTPL

Input: Tree network $\mathcal{T}$, client requests $\{Z(t)\}_1^T$, rewards $\{r_j\}$, capacity C, learning rates $\{\eta_t\}$
Output: Online cache allocation $\mathbf{Y}_t \in \{0,1\}^{\ell \times N}$ at each time step t

Sample $\boldsymbol{\gamma} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{N\times n})$; $\boldsymbol{X}_1 \leftarrow \mathbf{0}$
for $t = 1$ **to** T **do**
  $\boldsymbol{\Theta}(t) \leftarrow \boldsymbol{X}_t + \eta_t \boldsymbol{\gamma}$
  Return the integral cache allocation for the perturbed counts $\boldsymbol{\Theta}(t)$ using Algorithm 4.
  $\boldsymbol{X}_{t+1} \leftarrow \boldsymbol{X}_t + Z(t)$
end

G.2. Regret Analysis: Proof of Theorem 5

Proof We will reuse the notation introduced in the previous section. Let R^*_{INT} and R^*_{FRAC} be the reward of the optimal integral cache allocation and the optimal fractional cache allocation given by solving **FLP**, respectively. Note that R^*_{INT} is the reward of the optimal static offline caching policy which our online algorithm competes against. Then, recall that:

$$R^*_{\text{INT}} \leq R^*_{\text{FRAC}}(C) \leq (1+\epsilon) R^*_{\text{FRAC}}(\widehat{C}). \tag{8}$$

where $\widehat{C} = C/(1+\epsilon)$.

Let R_{ALG} be the sum of rewards over time horizon T of the online algorithm given in Algorithm 5. Then, the $(1+\epsilon)$ regret is given by:

$$\mathrm{Reg}_T(1+\epsilon) = R^*_{\text{INT}} - (1+\epsilon)\mathbb{E}_{\gamma,S}[R_{\text{ALG}}]. \tag{9}$$

where the expectation is taken over the randomness from sampling (denoted by S) as well as from the noise γ in Algorithm 5. Note that, at every time step, the caching configuration returned by Algorithm 5 is found by running Algorithm 4 with the perturbed count vector (and reduced cache capacity $C/(1+\epsilon)$) as input. Thus, by Theorem 4, the cache capacity constraints of all caches are satisfied at every time step with probability at least $1-\delta$ by taking a union bound over the time steps. Also note that the solution returned by Algorithm 4 is obtained by sampling an integral solution from a fractional solution. For the purpose of this proof, we now consider the fractional version of the online algorithm where we are allowed to output fractional caching configurations. More specifically, suppose that, at every time step, the online algorithm outputs the fractional caching configuration instead of the integral caching configuration given by the sampling step. Let $R^{\text{FRAC}}_{\text{ALG}}$

be the total reward over time horizon T obtained by these fractional solutions. Then, as described in Theorem 4, we will have $\mathbb{E}_{\gamma,S}[R_{\text{ALG}}] = \mathbb{E}_{\gamma}[R^{\text{FRAC}}_{\text{ALG}}]$. Thus, from (8) and (9), we have:

$$\text{Reg}_T(1+\epsilon) \leq (1+\epsilon)[R^*_{\text{FRAC}}(\widehat{C}) - \mathbb{E}_{\gamma}[R^{\text{FRAC}}_{\text{ALG}}]]. \tag{10}$$

Thus, it is enough to bound the regret of the online algorithm with fractional solutions given by $\mathbb{E}[\text{Reg}^{\text{FRAC}}_T] = R^*_{\text{FRAC}}(\widehat{C}) - \mathbb{E}_{\gamma}[R^{\text{FRAC}}_{\text{ALG}}]$. To bound this quantity, we will follow the proof for the regret analysis of FTPL policy given in Paria and Sinha (2021).

First, recall that the request vector for client i at time t is given by $\boldsymbol{z}_i(t) \in \{0,1\}^N$ and $\boldsymbol{z}(t)$ is a length Nn obtained by concatenating all $\boldsymbol{z}_i(t)$'s together. Let $\boldsymbol{w}_i(t)$ be length N reward vector corresponding to client i at time step t due to the online policy of Algorithm 5 such that $(\boldsymbol{w}_i(t))_j \in [0, r_{\max}]$ is the reward if file $j \in [N]$ was requested by client i at time t. Note that $\sum_{i=1}^{n} \langle \boldsymbol{w}_i(t), \boldsymbol{z}_i(t) \rangle$ is the reward at time step t of the policy in Algorithm 5. Also note that $\|\boldsymbol{w}_i(t)\|_1 \leq \frac{r_{\max} mC}{1+\epsilon}$ since each cache is of capacity $\frac{C}{1+\epsilon}$ in the modified LP in Algorithm 4 and there are m caches in the path from client i to the root in a tree of depth m. Also, let $\boldsymbol{w}(t)$ be the length Nn reward vector corresponding to all clients obtained by concatenating all the $\boldsymbol{w}_i(t)$ vectors together. Then,

$$\|\boldsymbol{w}(t)\|_1 \leq \frac{r_{\max} nmC}{1+\epsilon}. \tag{11}$$

Let $\mathcal{W}$ be the set of feasible reward vectors obtained by solving the (fractional) LP with cache capacity $C/(1+\epsilon)$. Then, we have:

$$\boldsymbol{w}(t) \in \underset{\boldsymbol{w} \in \mathcal{W}}{\text{argmax}} \langle \boldsymbol{w}, \boldsymbol{X}_t + \eta_t . \boldsymbol{\gamma} \rangle \tag{12}$$

Now, we follow the proof of Theorem 1 of Paria and Sinha (2021) to bound the regret $\mathbb{E}[\text{Reg}^{\text{FRAC}}_T] = R^*_{\text{FRAC}}(\widehat{C}) - \mathbb{E}_{\boldsymbol{\gamma}}[R^{\text{FRAC}}_{\text{ALG}}]$. Let the potential function $\Phi_{\eta_t}(\boldsymbol{z})$ be defined as $\Phi_{\eta_t}(\boldsymbol{z}) = \mathbb{E}_{\boldsymbol{\gamma}}[\max_{\boldsymbol{w} \in \mathcal{W}} \langle \boldsymbol{w}, \boldsymbol{z} + \eta_t \cdot \boldsymbol{\gamma} \rangle]$. Then, following the proof of Theorem 1, we have (see Eq. (29) of Paria and Sinha (2021)):

$$\mathbb{E}[\text{Reg}^{\text{FRAC}}_T] \leq \eta_{T+1} \mathbb{E}[\max_{\boldsymbol{w} \in \mathcal{W}} \langle \boldsymbol{w}, \boldsymbol{\gamma} \rangle] + \frac{1}{2} \sum_{t=1}^{T} (\boldsymbol{z}(t))^T \nabla \Phi^2_{\eta_t}(\tilde{\mathbf{X}}_t) \boldsymbol{z}(t) \tag{13}$$

where $\tilde{\mathbf{X}}_t$ lies on the line segment joining $\mathbf{X}_t$ and $\mathbf{X}_{t+1}$ for all $t \in [T]$. We now bound the two terms in (13) separately. We can bound $\mathbb{E}[\max_{\boldsymbol{w} \in \mathcal{W}} \langle \boldsymbol{w}, \boldsymbol{\gamma} \rangle]$ following the proof of Proposition 1 of Paria and Sinha (2021) as follows:

$$\begin{aligned} \mathbb{E}[\max_{\boldsymbol{w} \in \mathcal{W}} \langle \boldsymbol{w}, \boldsymbol{\gamma} \rangle] &\overset{\text{Hölder's ineq.}}{\leq} \mathbb{E}[\max_{\boldsymbol{w} \in \mathcal{W}} \|\boldsymbol{w}\|_1 \|\boldsymbol{\gamma}\|_\infty] \\ &\leq \frac{r_{\max} \eta_{T+1} nmC}{1+\epsilon} \sqrt{4 \ln(Nn)} \end{aligned} \tag{14}$$

where for the second inequality, we bounded $\|\boldsymbol{w}\|_1$ using (11) and $\|\boldsymbol{\gamma}\|_\infty$ was bounded using a standard upper bound on the expectation of the maximum of the absolute value of a set of i.i.d. standard Gaussian random variables Wainwright (2019). Now, for bounding the second term in (13), first note that we can write the (i,j)'th term of the matrix as (using Lemma 1.5 of Abernethy et al. (2016)):

$$(\nabla \Phi^2_{\eta_t}(\tilde{\mathbf{X}}_t))i, j = \frac{1}{\eta_t} \mathbb{E}[\hat{w}_i \boldsymbol{\gamma}_j]$$

where $\hat{w}_i \in \operatorname{argmax}_{\boldsymbol{w}\in\mathcal{W}}\langle \boldsymbol{w}, \tilde{\mathbf{X}}_t + \eta_t\boldsymbol{\gamma}\rangle$. Thus, we get:

$$\begin{aligned}(\boldsymbol{z}(t))^T\nabla\Phi^2_{\eta_t}(\tilde{\mathbf{X}}_t)\boldsymbol{z}(t) &= \frac{1}{\eta_t}\sum_{i=1}^{Nn}\sum_{j=1}^{Nn}(\boldsymbol{z}(t))_i(\boldsymbol{z}(t))_j\mathbb{E}[\hat{z}_i\boldsymbol{\gamma}_j] \\ &= \frac{1}{\eta_t}\mathbb{E}[(\sum_{i=1}^{Nn}(\boldsymbol{z}(t))_i\hat{w}_i)(\sum_{j=1}^{Nn}(\boldsymbol{z}(t))_j\boldsymbol{\gamma}_j)] \\ &\le \frac{1}{\eta_t}\sqrt{\mathbb{E}[(\sum_{i=1}^{Nn}(\boldsymbol{z}(t))_i\hat{w}_i)^2]} \\ &\times \sqrt{\mathbb{E}[(\sum_{j=1}^{Nn}(\boldsymbol{z}(t))_j\boldsymbol{\gamma}_j)^2]} \end{aligned} \tag{15}$$

where the final step follows from the Cauchy-Schwarz inequality. Now, each request $\boldsymbol{z}(t)$ has a 1 at n indices corresponding to the file requests of the n clients and 0 everywhere else. Also, every $\hat{w}_i$ for $i \in [Nn]$ is bounded by $r_{\max}$. Thus, $(\sum_{i=1}^{Nn}(\boldsymbol{z}(t))_i\hat{w}_i)^2 \le n^2 r_{\max}^2$. Also, $\sum_{j=1}^{Nn}(\boldsymbol{z}(t))_j\boldsymbol{\gamma}_j$ is a Gaussian random variable with mean 0 and variance $\|\boldsymbol{z}(t)\|_2^2 = n$. Thus, $\mathbb{E}[(\sum_{j=1}^{Nn}(\boldsymbol{z}(t))_j\boldsymbol{\gamma}_j)^2] = n$. Substituting this in (15), we get:

$$(\boldsymbol{z}(t))^T\nabla\Phi^2_{\eta_t}(\tilde{\mathbf{X}}_t)\boldsymbol{z}(t) \le \frac{1}{\eta_t}n^{3/2}r_{\max}$$

Using the above result and (14) in (13), we get:

$$\mathbb{E}[\text{Reg}_T^{\text{FRAC}}] \le \frac{r_{\max}\eta_{T+1}nmC}{1+\epsilon}\sqrt{4\ln(Nn)} + \frac{n^{3/2}r_{\max}}{2}\sum_{t=1}^{T}\frac{1}{\eta_t}$$

Choosing $\eta_t = \beta\sqrt{T}$ and setting β appropriately to balance the two terms on the right hand side above, we get:

$$\mathbb{E}[\text{Reg}_T^{\text{FRAC}}] \le r_{\max}n^{5/4}\sqrt{\frac{mC}{1+\epsilon}}(\ln Nn)^{1/4}\sqrt{T}.$$

Substituting the above bound in (9), we get the final regret bound:

$$\text{Reg}_T(1+\epsilon) \le r_{\max}n^{5/4}\sqrt{(1+\epsilon)mC}(\ln Nn)^{1/4}\sqrt{T}$$

■

Appendix H. Special Case: Path Network

Having established the hardness of the general tree caching problem along with presenting a sophisticated algorithm to solve it, we now turn to a key special case: a single client connected to the source server via a path of caches. This linear topology (Figure 4) is a special case of a tree with a single leaf and no branching nodes. Note that the treewidth of the underlying graph is 1. We will

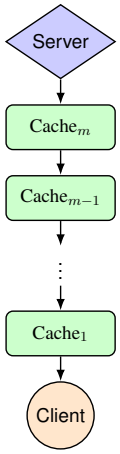


Figure 4: Single Client: Path Network

Algorithm 6 Caching for Single Client

Input: Learning rates $\{\eta_t\}_{t\in[T]}$

Sample $\boldsymbol{\gamma} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{N\times 1})$
$\boldsymbol{X}(1) \leftarrow \mathbf{0}$
for $t = 1$ **to** T **do**
  $\boldsymbol{\Theta}(t) \leftarrow \boldsymbol{X}(t) + \eta_t \boldsymbol{\gamma}$
  Sort $\boldsymbol{\Theta}(t)$ in descending order
  for $k = 1$ **to** m **do**
    Load Cache$_k$ with files at indices $[(k-1)\cdot C + 1, \ldots, k \cdot C]$
  end
  $\boldsymbol{X}(t+1) \leftarrow \boldsymbol{X}(t) + \boldsymbol{x}(t)$
end

show that in this case, there is a simple, optimal greedy algorithm for the offline (and hence, online) problem. Note that in this case, we will have $\ell = m$ since the number of caches is equal to the depth of the tree.

The online caching policy is given in Algorithm 6. Note that this is a simple Follow-The-Perturbed-Leader based caching policy which maintains a vector $\boldsymbol{X}(t)$ containing the total number of requests of each file up to time t and then perturbs the vector $\boldsymbol{X}(t)$ by a small Gaussian noise vector (scaled by η_t at time t). Also note that the policy maintains orthogonality of the cache configurations across different levels, i.e. for caches i and j, we will always have $\boldsymbol{y}_i^T(t)\boldsymbol{y}_j(t) = 0$. We now bound the regret of this simple policy.

Theorem 11 *Let π be the caching policy given by Algorithm 6 with learning rates $\{\eta_t\}_{t\in[T]}$. Then, the expected regret of the policy is bounded as follows:*

$$\mathbb{E}\left[\text{Reg}_\pi(T)\right] \leq O\left(\sqrt{T}\right). \tag{16}$$

Proof Let $\boldsymbol{z}(t) = \sum_{i=1}^m r_i \boldsymbol{y}_i(t)$ and $\boldsymbol{z}^* = \sum_{i=1}^m r_i \boldsymbol{y}_i^*$. Let $\boldsymbol{x}(t) \in \mathbb{R}^N$ be the one-hot vector that represents the file requested at time step t. Then, the set of feasible reward vectors are given by $\boldsymbol{z} \in \mathcal{Z} \subseteq \{0, r_m, \ldots, r_1\}^N$ such that $\sum_{i=1}^N \mathbb{I}(\boldsymbol{z}_i = r_j) \leq C$ for $j \in [m]$. The reward vector at time step t given by the algorithm is

$$\boldsymbol{z}(t) \in \operatorname*{argmax}_{\boldsymbol{z}\in\mathcal{Z}} \langle \boldsymbol{z}, \boldsymbol{X}_t + \eta_t \cdot \boldsymbol{\gamma}\rangle$$

where $\boldsymbol{\gamma}_t \sim [N](\boldsymbol{0}, \mathbf{I}_{N\times 1})$ and $\boldsymbol{X} = \sum_{i=1}^{t-1} \boldsymbol{x}(i)$. For simplicity, assume that $\sum_{i=1}^m r_i^2 = 1$, i.e. the rewards are normalized.

Define a potential function

$$\phi_\eta(\boldsymbol{x}) = \mathbb{E}_\gamma[\max_{\boldsymbol{z}\in\mathcal{Z}} \langle \boldsymbol{z}, \boldsymbol{x} + \eta_t \cdot \boldsymbol{\gamma}\rangle]. \tag{17}$$

Then, following the analysis in (Bhattacharjee et al., 2020, Theorem 2.4), or equivalently from Cohen and Hazan (2015), we can bound the regret after T steps as:

$$\mathbb{E}[R_T] \leq \Phi_\eta(\boldsymbol{X}_1) + \frac{1}{2}\sum_{t=1}^T \langle \boldsymbol{x}(t), \nabla^2(\phi(\tilde{\boldsymbol{x}}(t)))\boldsymbol{x}(t)\rangle \tag{18}$$

for some $\tilde{\boldsymbol{x}}(t)$ on the line segment connecting $\boldsymbol{X}_t$ and $\boldsymbol{X}_{t+1}$. The second term can be bounded using (Bhattacharjee et al., 2020, Eq. 27) since $\boldsymbol{x}(t)$ has a 1 at only a single index at every t. So, we get

$$\frac{1}{2}\sum_{t=1}^T \langle \boldsymbol{x}(t), \nabla^2(\phi(\tilde{\boldsymbol{x}}(t)))\boldsymbol{x}(t)\rangle \leq \frac{T}{\eta\sqrt{2\pi}}.$$

. Next, note that we have $\Phi_\eta(\Theta_1) = \eta\mathbb{E}[\max_{\boldsymbol{z}\in\mathcal{Z}}\langle \boldsymbol{z}, \boldsymbol{\gamma}\rangle]$. Following Cohen and Hazan (2015), since $\langle \boldsymbol{z}, \boldsymbol{\gamma}\rangle$ is a normal random variable with mean 0 and variance $\|\boldsymbol{z}\|_2^2 \leq C$ we get

$$\mathbb{E}[\max_{\boldsymbol{z}\in\mathcal{Z}}\langle \boldsymbol{z}, \boldsymbol{\gamma}\rangle] \leq \sqrt{2C\log|\mathcal{Z}|}.$$

Now, note that $|\mathcal{Z}| < N^{mC}$. Thus, we get $\Phi_\eta(\Theta_1) = C\eta\sqrt{2m\log N}$. So, we get:

$$\mathbb{E}[R_T] \leq C\eta\sqrt{2m\log N} + \frac{T}{\eta\sqrt{2\pi}}$$

Hence, we get

$$\mathbb{E}[R_T] \leq O\left((m\log N)^{1/4}\sqrt{CT}\right)$$

for

$$\eta = \frac{1}{(4\pi m\log N)^{1/4}} \cdot \sqrt{\frac{T}{C}}.$$

.

■

Appendix I. Experiments

We compare the performance of our Tree Cache algorithm with the commonly used network caching policies such as LRU and LFU (run independently on each cache), and also with the TBGRD algorithm proposed in Li et al. (2021).

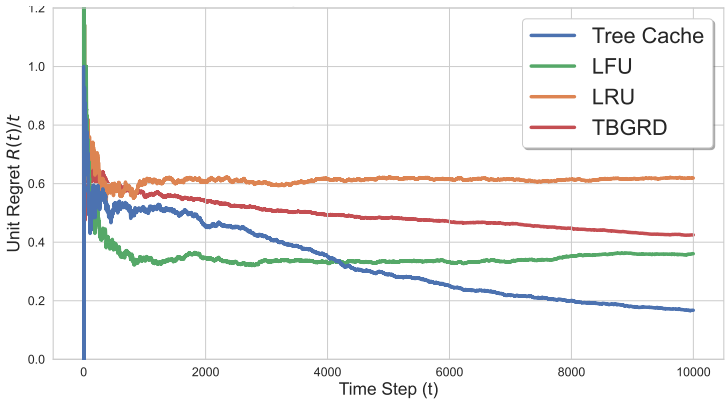
The tree network used for the simulations has one root cache and two leaf caches. Each leaf has a client attached to it, that requests files from the cache at each time step. All the caches have a capacity of 5 files. A cache hit at the leaves awards a reward of 2, while a cache hit at the root cache awards a reward of 1. The library consists of 21 files. The time horizon for the experiments is set to 10,000 time steps.

We evaluate the algorithms on three types of request sequences: an adversarial sequence, a synthetic Zipf sequence, and a CDN trace from Berger (2018).

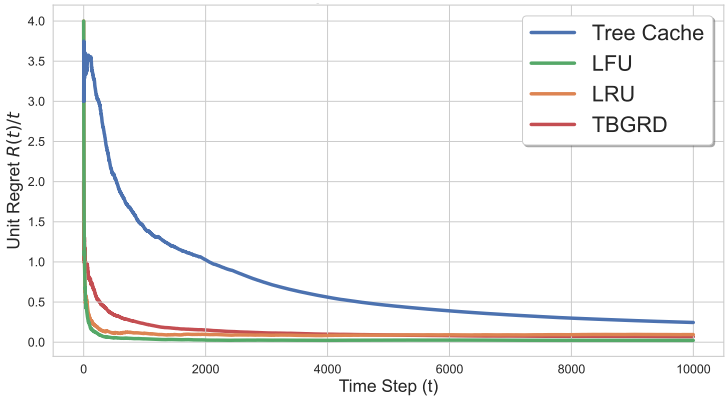
For the adversarial sequence, the library is divided into three sets A, B, and C. At each time step, client 1 chooses set A with probability 0.55 and set B with probability 0.45, while client 2 selects set A with probability 0.55 and set C with probability 0.45. Each client then requests a file uniformly at random from the chosen set.

In the synthetic Zipf sequence, both clients independently request files from the library according to an identical Zipf distribution. The CDN trace is a real-world trace from a Content Delivery Network (CDN) available at Berger (2018). For preprocessing, we sort the requests by time and discard the requests of file indices greater than 21, since that is the library size. Then, we take a continuous sequence of 20,000 requests from the trace, and split it into two equal parts, one for each client. The first half of the requests is assigned to client 1 and the second half to client 2.

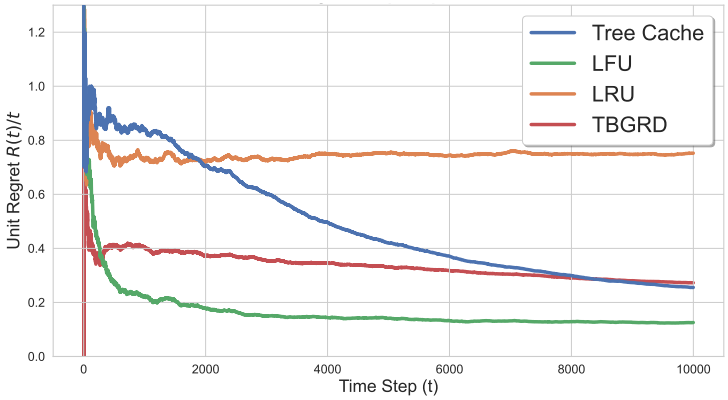
We find that our proposed algorithm outperforms other algorithms on the adversarial sequence while performing comparably on the Zipf sequence and the CDN trace.



(*a*) Adversarial sequence



(*b*) CDN trace



(*c*) Uniform Zipf

Figure 5: Regret per unit time for different request sequences. The Tree Cache algorithm outperforms other algorithms on the adversarial sequence while performing comparably on the Zipf sequence and the CDN trace.