Online learning for network resource allocation

Tareq Si Salem

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Apprentissage Séquentiel pour l'Allocation de Ressources dans les Réseaux

Tareq SI SALEM

Inria Sophia Antipolis

Présentée en vue de l'obtention du grade de docteur en Informatique **de l'**Université Côte d'Azur

Dirigée par : Giovanni NEGLIA
Soutenue le : 17 Octobre 2022

Devant le jury, composé de :
Walid DABBOUS, Directeur de Recherche, Inria France
Douglas LEITH, Professeur, Trinity College Dublin, Irlande
Leandros TASSIULAS, Professeur, Université Yale, États-Unis
Edmund YEH, Professeur, Université Northeastern, États-Unis
György DÁN, Professeur, École Royale Polytechnique, Suède

Inria

Apprentissage Séquentiel pour l'Allocation de Ressources dans les Réseaux

---

Online Learning for Network Resource Allocation

Tareq Si Salem

⋈

Jury :

Président du jury
Walid Dabbous, Directeur de Recherche, Inria France

Rapporteurs
Douglas Leith, Professeur, Trinity College Dublin, Irlande
Leandros Tassiulas, Professeur, Université Yale, États-Unis

Examinateurs
Edmund Yeh, Professeur, Université Northeastern, États-Unis
György Dán, Professeur, École Royale Polytechnique, Suède

Directeur de thèse
Giovanni Neglia, Chargé de Recherche, Inria, France

Université Côte d'Azur

Tareq Si Salem

Apprentissage Séquentiel pour l'Allocation de Ressources dans les Réseaux

xv+267 p.

I dedicate my thesis to my parents for their encouragement throughout my pursuit for education, 𐤕𐤍𐤌𐤓𐤈 *(thanemirth).*

Apprentissage Séquentiel pour l'Allocation de Ressources dans les Réseaux

Résumé

L'allocation de ressources dans les réseaux est un problème complexe et fondamental en informatique. Il s'agit d'un processus dans lequel les composants d'un système de réseau visent à fournir un service plus rapide aux demandes ou à réduire la charge de calcul ou de communication sur le système. Les principaux facteurs qui contribuent à la complexité de ce problème sont que les demandes arrivent au système de manière imprévisible et séquentielle et peuvent entrer en concurrence pour les différentes ressources du réseau. L'ubiquité des problèmes d'allocation de ressources dans les réseaux a motivé des recherches approfondies pour concevoir de nouveaux algorithmes avec des garanties prouvables. Cette thèse étudie plusieurs instances du problème d'allocation de ressources dans les réseaux et propose des algorithmes adaptatifs avec de fortes garanties de performances s'appuyant sur le cadre d'apprentissage séquentiel.

Premièrement, nous étudions le problème de mise en cache séquentiel, dans lequel les demandes de fichiers peuvent être servies par un cache local pour éviter les coûts de récupération à partir d'un serveur distant. Nous étudions des algorithmes avec des garanties de performance basés sur des stratégies de descente miroir (DM). Nous montrons que la stratégie DM optimale dépend de la diversité présente dans un lot de demandes. Nous prouvons également que, lorsque le cache doit stocker le fichier entier, plutôt qu'une fraction, les stratégies DM peuvent être couplées à un schéma d'arrondi aléatoire qui préserve garanties de performance. Nous présentons de plus une extension aux réseaux de caches, et nous proposons un algorithme adaptatif distribué.

Deuxièmement, nous étudions les caches de similarité qui peuvent répondre à une demande d'un objet avec des objets similaires stockés localement. Nous proposons un nouvel algorithme de mise en cache de similarité séquentiel qui utilise la descente de gradient pour naviguer dans l'espace de représentation continue des objets et trouver les objets appropriés à stocker dans le cache. Nous montrons que l'algorithme proposé réduit les coûts de service encourus par le système pour les systèmes de diffusion vidéo à 360° et les systèmes de recommandation. Par la suite, nous montrons que le problème de mise en cache de similarité peut être formulé dans le cadre d'apprentissage séquentiel en utilisant un algorithme MD associée à un arrondi aléatoire.

Troisièmement, nous présentons les réseaux de distribution d'inférence (RDI) émergents, des réseaux de nœuds informatiques qui se coordonnent pour satisfaire les demandes d'inférence d'apprentissage automatique (AA) en obtenant le meilleur compromis entre latence et précision. Nous proposons un algorithme adaptatif distribué pour l'allocation de modèles d'AA dans un RDI : chaque nœud met à jour dynamiquement son ensemble local de modèles d'inférence en fonction des demandes observées au cours du passé récent et d'un échange d'informations limité avec ses nœuds voisins.

Finalement, nous étudions l'équité du problème d'allocation des ressources réseau sous le critère d'α-fairness. Nous reconnaissons deux objectifs d'équité différents qui surgissent naturellement dans ce problème : l'objectif d'équité de tranche bien compris qui vise à assurer l'équité à chaque tranche de temps, et l'objectif d'équité d'horizon moins exploré qui vise à assurer l'équité entre les utilités accumulées sur un horizon temporel. Nous étudions l'équité de l'horizon avec le regret comme métrique de performance et montrons que la disparition du regret ne peut être atteinte en présence d'un adversaire sans restriction. Nous proposons des restrictions sur les capacités de l'adversaire correspondant à des scénarios réalistes et un algorithme adaptatif qui garantit en effet la disparition du regret sous ces restrictions.

Mots-clés : Allocation des ressources, Apprentissage Séquentiel, Réseau informatique.

Online Learning for Network Resource Allocation

Abstract

Network resource allocation is a complex and fundamental problem in computer science. It is a process in which components of a networked system aim to provide a faster service to demands, or to reduce the computation or communication load on the system. The main factors that contribute to the complexity of this problem are that the demands arrive to the system in an unpredictable and sequential fashion and may compete for the different network resources. The ubiquity of network resource allocation problems has motivated extensive research to design new policies with provable guarantees. This thesis investigates several instances of the network resource allocation problem and proposes online policies with strong performance guarantees leveraging the online learning framework.

First, we study the online caching problem in which demands for files can be served by a local cache to avoid retrieval costs from a remote server. We study no-regret algorithms based on online mirror descent (OMD) strategies. We show that the optimal OMD strategy depends on the request diversity present in a batch of demands. We also prove that, when the cache must store the entire file, rather than a fraction, OMD strategies can be coupled with a randomized rounding scheme that preserves regret guarantees. We also present an extension to cache networks, and we propose a no-regret distributed online policy.

Second, we investigate similarity caches that can reply to a demand for an object with similar objects stored locally. We propose a new online similarity caching policy that employs gradient descent to navigate the continuous representation space of objects and find appropriate objects to store in the cache. We provide theoretical convergence guarantees under stationary demands and show the proposed policy reduces service costs incurred by the system for 360°-video delivery systems and recommendation systems. Subsequently, we show that the similarity caching problem can be formulated in the online learning framework by utilizing an OMD policy paired with randomized rounding to achieve a no-regret guarantee.

Third, we present the novel idea of inference delivery networks (IDNs), networks of computing nodes that coordinate to satisfy machine learning (ML) inference demands achieving the best trade-off between latency and accuracy. IDNs bridge the dichotomy between device and cloud execution by integrating inference delivery at the various tiers of the infrastructure continuum (access, edge, regional data center, cloud). We propose a no-regret distributed dynamic policy for ML model allocation in an IDN: each node dynamically updates its local set of inference models based on demands observed during the recent past plus limited information exchange with its neighboring nodes.

Finally, we study the fairness of network resource allocation problem under the α-fairness criterion. We recognize two different fairness objectives that naturally arise in this problem: the well-understood slot-fairness objective that aims to ensure fairness at every timeslot, and the less explored horizon-fairness objective that aims to ensure fairness across utilities accumulated over a time horizon. We argue that horizon-fairness comes at a lower price in terms of social welfare. We study horizon-fairness with the regret as a performance metric and show that vanishing regret cannot be achieved in presence of an unrestricted adversary. We propose restrictions on the adversary's capabilities corresponding to realistic scenarios and an online policy that indeed guarantees vanishing regret under these restrictions. We demonstrate the applicability of the proposed fairness framework to a representative resource management problem considering a virtualized caching system where different caches cooperate to serve content requests.

Keywords: Resource allocation, Online Learning, Computer Networks.

Acknowledgment

First and foremost, I would like to express my sincere gratitude to my thesis advisor Giovanni Neglia for his patience, support, help, and guidance throughout this journey. I am grateful for our numerous discussions that helped to shape this thesis, and in many instances, he exceeded the expectation of a Ph.D. advisor. I want to thank Douglas Leith, Edmund Yeh, György Dán, Leandros Tassiulas, and Walid Dabbous for accepting to be part of my Ph.D. defense jury.

I am privileged to work with excellent collaborators. Stratis Ioannidis, whom I learned from tremendously; his positive outlook on our research contributions is contagious, and it is a quality that propelled other contributions during this thesis. George Iosifidis, whom I had the chance to work with during my visit to TU Delft. I am grateful for hosting me at TU Delft, and for our fruitful discussions that spawn the last chapter of this thesis. I am also thankful to work with the talented researchers in academia, Damiano Carra, Andrea Araldo, Michele Garetto, and Emilio Leonardi, as well as researchers in industry, Gabriele Castellano and Fabio Pianese. This thesis benefited tremendously from our exchanges. I also enjoyed working on several research projects with the talented Ph.D. students Anirudh Sabnis and Yuanyuan Li.

I am exceedingly grateful to have conducted my research at the rich and supportive learning environments of Inria Sophia Antipolis research laboratory and Côte d'Azur University. I am thankful to work alongside the superb NEO team members. In particular, the Moroccan duo, Othmane Marfoq, and Younes Ben Mazziane, whom I enjoyed stirring occasional Algeria vs. Morocco feuds and several discussions that only a North African would understand. I also thank Angelo Rodio, Maximilien Dreveton, Andrei Bobu, Kishor Patil, Chuan Xu, Caelin Kaplan, Mikhail Kamalov, Olga Chuchuk, Ke Sun, Sadaf Ul Zuhra, Vinay Kumar B. R., Jake Clarkson, Francescomaria Faticanti, Sara Alouf, Samir M. Perlaza, Konstantin Avrachenkov, Eitan Altman, and Alain Jean-Marie for enjoyable lunch discussions, coffee pauses, and social activities. I also thank the team members of the ENS group at TU Delft, Naram Mhaisen, and Michail Kalntis, for sharing drinks and barbecues and making my stay just as enjoyable and informative.

I am thankful to my previous advisors who lead me to the path of starting the Ph.D. program. Arnaud Legout, for supervising my Master thesis at Ubinet master (Ubiquitous Networking) in the DIANA team. As well as, my Master's thesis advisors Dalila Cherifi and Rachid Deriche during my studies in Algeria, who sparked my curiosity to pursue a research path. I also thank them for providing me the opportunity to attend MedDays (Mediterranean Students Days) in Sophia Antipolis, where I was first exposed to the Ubinet master program that paved the way for this thesis.

The pandemic was a difficult period for everyone, but I am lucky to have by my side Margaux Dubois, who supported and tolerated my sporadic work schedule. Special thanks to Yanis Boussad for helping me settle during my first period in France, and watching his Ph.D. journey helped me in many ways.

Finally, I would like to express my gratitude to my family: my father Youcef, my mother Fetima, my brothers Ferhat, and Aghiles, and my sister Razika. They have always supported and been there for me. This manuscript is dedicated to them.

Table of Contents

Appendix

Chapter 1

Introduction

1.1 Motivation

Connectivity and ubiquity of computing devices enabled a wide spectrum of network applications such as content delivery, interpersonal communication, and intervehicular communication. New use cases (e.g., autonomous driving [1], augmented reality [2], and tactile internet [3]) require satisfying user-generated and machine-generated demand with stringent low-latency and high-bandwidth guarantees.

Network resource allocation is utilized to provide a faster service to demands and to reduce the computation or communication load on a networked system. This is achieved through optimization and appropriate placement of resources at different locations in a network. However, this remains a challenging task in many practical scenarios where different network parameters, such as latency and operating costs may vary over time and the demands arrive to the system in an unpredictable and sequential fashion. For instance, in small-cell mobile networks the user churn is typically very high and unpredictable, hindering appropriate allocation of spectrum to cells [4]. Similarly, placing content files at edge caches to balance the latency gains across the served areas is non-trivial due to the non-stationary and fast-changing patterns of requests [5]. At the same time, the increasing virtualization of these systems introduces cost and performance volatility, as extensive measurement studies have revealed [6–8]. This uncertainty is exacerbated for services that process user-generated data (e.g., streaming data applications) where the performance (e.g., inference accuracy) depends also on a priori unknown input data and dynamically selected machine learning libraries [9–11].

1.2 Objective

This thesis investigates several instances of the network resource allocation problem and proposes online policies with strong performance guarantees. We leverage the online learning framework under minimal assumptions on the external environment's behavior. Performance guarantees in the online learning framework are established assuming the existence of an adversary; therefore, a policy designed for such adversarial setting is also robust under unpredictable environments exhibiting no statistical regularity.

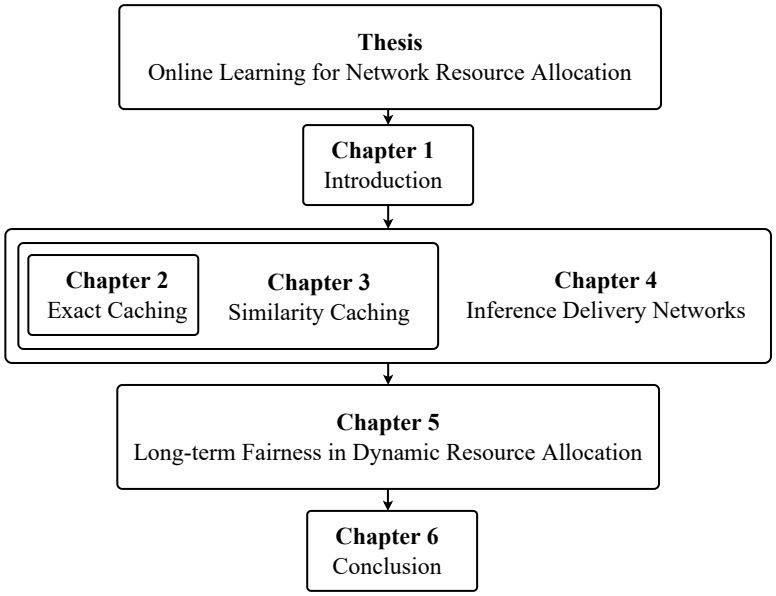


Figure 1.1: Structure of the thesis

1.3 Outline of the Thesis

This thesis is split into six chapters. The first chapter provides a brief introduction to the thesis. The second chapter revisits the problem of exact caching, in which demands for objects can be served by a local cache or a network of caches to avoid retrieval costs from a remote server. The third chapter deals with the similarity caching problem, in which demands for objects can be approximately served with similar objects; it considers both infinite (continuous) and finite catalogs. The fourth chapter introduces inference delivery networks, networks of computing nodes that coordinate to satisfy ML inference requests. The fifth chapter studies the fairness dimension of the network resource allocation problem. The final and sixth chapter provides brief concluding remarks and presents directions for possible future research. The structure of the thesis is provided in Figure 1.1.[1]

1.4 Publications

1.4.1 Published

- T. Si Salem, G. Iosifidis, G. Neglia. Enabling Long-term Fairness in Dynamic Resource Allocation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (ACM SIGMETRICS), 2023.*
- T. Si Salem and G. Neglia and D. Carra. Ascent Similarity Caching with Approximate Indexes. *IEEE/ACM Transactions on Networking (ToN), 2022.*

[1]Inclusion of a chapter box represents a generalization of its system model.

- A. Sabnis, T. Si Salem, G. Neglia, M. Garetto, E. Leonardi, and R. K. Sitaraman. GRADES: Gradient Descent for Similarity Caching. *IEEE/ACM Transactions on Networking (ToN), 2022.*
- Y. Li, T. Si Salem, G. Neglia, and S. Ioannidis. Online Caching Networks with Adversarial Guarantees. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (ACM SIGMETRICS), 2022.*
- T. Si Salem and G. Neglia and D. Carra. AÇAI: Ascent Similarity Caching with Approximate Indexes. *Proceedings of the 33rd International Teletraffic Congress (ITC 33), Aug. 31st - Sep. 3rd 2021.*
- T. Si Salem, G. Castellano, G. Neglia, F. Pianese, and A. Araldo. Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees. Proceedings of the 19th Mediterranean Communication and Computer Networking Conference (MedComNet 2021), June 15-17, 2021.
- T. Si Salem, G. Neglia, and S. Ioannidis. No-Regret Caching via Online Mirror Descent. *Proceedings of the IEEE International Conference on Communications (ICC 2021), June 14-23, 2021.*
- A. Sabnis, T. Si Salem, G. Neglia, M. Garetto, E. Leonardi, and R. K. Sitaraman. GRADES: Gradient Descent for Similarity Caching. *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM 2021), May 10-13, 2021.*

1.4.2 Submitted

- T. Si Salem, G. Castellano, G. Neglia, F. Pianese, and A. Araldo. Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees. A preprint is available [12].
- T. Si Salem, G. Neglia, and S. Ioannidis. No-Regret Caching via Online Mirror Descent.. A preprint is available [13].

CHAPTER 2

Exact Caching

2.1 Introduction

Caches are deployed at many different levels in computer systems: from CPU hardware caches to operating system memory caches, from application caches at clients to CDN caches deployed as physical servers in the network or as cloud services like Amazon's ElastiCache [14]. They aim to provide a faster service to the user and/or to reduce the computation/communication load on other system elements, like hard disks, file servers, etc.

The ubiquity of caches has motivated extensive research on the performance of existing caching policies, as well as on the design of new policies with provable guarantees. To that end, most prior work has assumed that caches serve requests generated according to a stochastic process, ranging from the simple, memory-less independent reference model [15] to more complex models trying to capture temporal locality effects and time-varying popularities (e.g., the shot-noise model [6]). An alternative modeling approach is to consider an *adversarial* setting. Assuming that the sequence of requests is generated by an adversary, an online caching policy can be compared to the optimal offline policy that views the sequence of requests in advance. Caching was indeed one of the first problems studied by Sleator and Tarjan in the context of the competitive analysis of online algorithms [16]. In competitive analysis, the metric of interest is the *competitive ratio*, i.e., the worst-case ratio between the costs incurred by the online algorithm and the optimal offline *dynamic* algorithm. This line of work led to the study of metrical task systems [17], a popular research area in the algorithms community [18]. Recently, Paschos et al. [19] proposed studying caching as an online convex optimization (OCO) problem [20]. OCO considers again an adversarial setting, but the metric of interest is the *regret*, i.e., the difference between the costs incurred over a time horizon T by the algorithm and by the optimal offline *static* solution. Online algorithms whose regret grows sublinearly with T are called *no-regret* algorithms, as their time-average regret becomes negligible for large T.

2.1.1 Contributions

Single Cache. We make the following contributions in the context of the *single cache* problem:

- We show that caching policies based on OMD enjoy $O\left(\sqrt{T}\right)$ regret in the fractional setting. Most importantly, we show that bounds for the regret crucially depend on the diversity of the request process. In particular, the regret depends on the *diversity ratio* R/h, where R is the size of the batch, and h is the maximum multiplicity of a request in a given batch.

- We characterize the optimality of OMD caching policies w.r.t. regret under different diversity regimes. We observe that, for a large region of possible values of the diversity ratio, the optimum is either OGD or OMD with a neg-entropy mirror map (OMD_{NE}). In particular, OGD is optimal in the *low diversity* regime, while OMD_{NE} is optimal in the *high diversity* regime.
- OMD algorithms include a gradient update followed by a projection to guarantee that the new solution is in the feasible set (e.g., it does not violate the cache capacity constraints). The projection is often the most computationally expensive step of the algorithm. We show that efficient polynomial algorithms exist both for OGD (slightly improving the algorithm in [19]) and for OMD_{NE}.
- OMD algorithms work in a continuous space, and are therefore well-suited for the fractional setting originally studied by Paschos et al. Still, we show that, if coupled with opportune rounding techniques, they can also be used when the cache can only store a file in its entirety, while preserving their regret guarantees.

Caching Network. We make the following contributions in the context of the *caching network* problem:

- We revisit the general cache network setting of Ioannidis and Yeh [21] from an adversarial point of view.
- We propose DISTRIBUTEDTGONLINE, a distributed, online algorithm that attains $O(\sqrt{T})$ regret with respect to an offline solution that is within a $(1 - 1/e)$-approximation from the optimal, when cache update costs are not taken into account.
- We also extend our algorithm to account for update costs. We show that an $O(\sqrt{T})$ regret is still attainable in this setting, replacing however independent caching decisions across rounds with coupled ones; we determine the latter by solving an optimal transport problem.
- Finally, we extensively evaluate the performance of our proposed algorithm against several competitors, using (both synthetic and trace-driven) experiments involving non-stationary demands.

2.1.2 Organization

This chapter is organized as follows. Section 2.2 studies the single cache problem, and Section 2.3 studies the caching network problem. In detail:

Single Cache. In Section 2.2.1, we review related work to the single cache problem, then we introduce our model assumptions in Section 2.2.2 and provide technical background on gradient algorithms in Section 2.2.3. Section 2.2.3.3 presents our main results on the regret of OMD caching policies and their computational complexity. A discussion about extending the model to include cache update costs, in Section 2.2.4, is required to introduce the integral setting in Section 2.2.5. Numerical results are presented in Section 2.2.6.

Caching Network. In Section 2.3.1, we review related work to the caching network problem. Our model and distributed online algorithm are presented in Sections 2.3.2 and 2.3.3, respectively. We present our analysis of the regret under update costs in Section 2.3.4 and extend our results in Section 2.3.5. Numerical results are presented in Section 2.3.6.

2.2 Single Cache

2.2.1 Related Work

The caching problem has been extensively studied in the literature under different assumptions on the request process. When the requests occur according to a given stochastic process, the analysis usually leads to complex formulas even in simple settings. For example, even the hit ratio of a single cache managed by the LRU eviction policy under the independent reference model is hard to precisely characterize [22, 23]. The characteristic time approximation (often referred to as Che's approximation) significantly simplifies this analysis by assuming that a file, in absence of additional requests for it, stays in the cache for a random time sampled independently from requests for other files. Proposed by Fagin [24] and rediscovered and popularized by Che et al. [25], the approximation has been justified formally by several works [26–28] and has allowed the study of a large number of existing [29] and new [30, 31] caching policies. It also applies to networked settings [32–35] and to more general utilities beyond the hit ratio [36, 37], all under stochastic requests.

Online caching policies based on gradient methods have also been studied in the stochastic request setting, leading to Robbins-Monro/stochastic approximation algorithms (see, e.g., [21, 38]). Though related to OCO, guarantees are very different than the regret metric we study here. Many works have also explored the offline, network-wide static allocation of files, presuming demand is known [39–41]. We differ from the work above, as we consider adversarial requests.

Caching under adversarial requests has been studied since Sleator and Tarjan's seminal paper [16] through the competitive ratio metric. An algorithm is said to be α-competitive when its competitive ratio is bounded by α over all possible input sequences. The problem has been generalized by Manasse et al. [42] under the name *k-server problem*, and further generalized by Borodin et al. under the name *metrical task systems (MTS)* [17]. The literature on both the k-server and MTS problems is vast. A recent trend is to apply continuous optimization techniques to solve these combinatorial problems. Bansal et al. [43] study the k-server problem on a weighted star metric space. In the same spirit, Bubeck et al. [44] use the framework of continuous online mirror descent to provide an $o(k)$-competitive algorithm for the k-server problem on hierarchically separated trees. In this chapter, we focus on regret rather than competitive ratio as the main performance metric. Andrew et al. [45] give a formal comparison between competitive ratio and regret and prove that there is an intrinsic incompatibility between the two: no algorithm can have both sub-linear regret and a constant competitive ratio. At the same time, they propose an algorithm with sub-linear regret and slowly increasing competitive ratio.

Online convex optimization (OCO) was first proposed by Zinkevich [46], who showed that projected gradient descent attains sublinear regret bounds in the online setting. OCO generalizes previous online problems like the experts problem [47], and has become widely influential in the learning

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

Table 2.1: Notation Summary for Section 2.2

community [20,48]. To the best of our knowledge, Paschos et al. [19] were the first to apply the OCO framework to caching. Beside proposing OGD for the single cache, they extended it to a simple networked scenario, where users have access to a set of parallel caches that store pseudo-random linear combinations of the files. They proposed no-regret algorithms in both settings. Bhattacharjee et al. [49] extended this work proving tighter lower bounds for the regret and proposing new caching policies for the networked setting that do not require file coding; Mukhopadhyay and Sinha [50] accounted for switching costs due to file retrievals. We depart from these works in considering OMD algorithms, a more general request process, and allowing for integral cache states obtained through randomized rounding.

2.2.2 System Description

The notation used across this section is provided in Table 2.1.

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

A local cache of finite capacity is placed in between the source of requests and the remote server(s). The local cache's role is to reduce the costs incurred by satisfying requests locally. We denote by $k \in \{1, 2, \dots, N\}$ the capacity of the cache. The cache is allowed to store fractions of files (this assumption will be removed in Section 2.2.5). We assume that time is slotted, and denote by $x_{t,i} \in [0, 1]$ the fraction of file $i \in \mathcal{N}$ stored in the cache at timeslot $t \in \{1, 2, \dots, T\}$. The cache state is then given by vector $\boldsymbol{x}_t = [x_{t,i}]_{i \in \mathcal{N}} \in \mathcal{X}$, where $\mathcal{X}$ is the capped simplex determined by the

capacity constraint, i.e.,

$$\mathcal{X} = \left\{ \boldsymbol{x} \in [0,1]^N : \sum_{i=1}^{N} x_i = k \right\}. \tag{2.1}$$

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

$$\mathcal{R}_{R,h} = \left\{ \boldsymbol{r} \in \{0, \ldots, h\}^N : \sum_{i=1}^{N} r_i = R \right\}. \tag{2.2}$$

Intuitively, the ratio $\frac{R}{h}$ defines the diversity of request batches in a timeslot. For example, when $\frac{R}{h} = 1$, all R requests are concentrated on a single file. When $\frac{R}{h} = N$, requests are spread evenly across the catalog $\mathcal{N}$. In general, $\frac{R}{h}$ is a lower bound for the number of distinct files requested in the batch. For that reason, we refer to $\frac{R}{h}$ as the *diversity ratio*.[1] We note that our request model generalizes the setting by Paschos et al. [19], which can be seen as the case $R = h = 1$, i.e., the batch contains only one request per timeslot. We make no additional assumptions on the request arrival process; put differently, we operate in the adversarial online setting, where a potential adversary may select an arbitrary request sequence $\{\boldsymbol{r}_t\}_{t=1}^{T}$ in $\mathcal{R}_{R,h}$ to increase system costs.

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

$$f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) = \sum_{i=1}^{N} w_i r_{t,i} (1 - x_{t,i}). \tag{2.3}$$

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

The cost objective (2.3) captures several possible real-life settings. First, it can be interpreted as a QoS cost paid by each user for the additional delay to retrieve part of the file from the server. Second, assuming that the R requests arrive and are served individually (e.g., because they are spread-out within a timeslot), Eq. (2.3) can represent the load on the servers or on the network to provide the missing part of the requested files. Our model also applies when all requests for the same file are aggregated and served simultaneously by a single fetch operation. In this case, $r_{t,i}$ in Eq. (2.3) should be the interpreted as the indicator variable denoting if file i was requested; correspondingly, R then indicates the total number of *distinct* files requested, and $h = 1$.

[1] This definition of diversity is consistent with other notions of diversity, such as, e.g., the entropy; indeed the diversity ratio provides a lower bound on the entropy of the normalized batch vector $\frac{\boldsymbol{r}_t}{R}$, as $E\left(\frac{\boldsymbol{r}_t}{R}\right) \geq \log\left(\frac{R}{h}\right)$ [51, Lemma 3], where $E(\boldsymbol{p}) = -\sum_i p_i \log(p_i)$ is the entropy function.

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

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

$$\text{Regret}_T(\mathcal{A}) = \sup_{\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_t\} \in \mathcal{R}_{R,h}^T} \left\{ \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) - \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x}_*) \right\}, \tag{2.4}$$

where $\boldsymbol{x}_* = \arg\min_{\boldsymbol{x} \in \mathcal{X}} \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{x})$ is the optimal static cache state (in hindsight). Note that, by taking the supremum in Eq. (2.4), we indeed measure regret in the adversarial setting, i.e., against an adversary that potentially picks requests in $\mathcal{R}_{R,h}$ trying to jeopardize cache performance.

Update Costs. An online algorithm $\mathcal{A}$ updating the cache state at timeslot t may require moving a portion of a file from a remote server to the cache to implement this update. The update cost of the online algorithm is not explicitly modeled in our cost and regret (Eqs. (2.3) and (2.4), respectively). We postpone the discussion of such cost in Section 2.2.4. For the moment we observe that updates come "for free" for files requested in the current timeslot. For example, increasing the fraction $x_{t,i}$ for some file i such that $r_{t,i} > 0$, can be performed by recovering the additional part out of the $(1 - x_{t,i})$ missing fraction that needs to be retrieved to serve the file; updates can thus "free-ride" on regular traffic, at no additional cost. We prove in Proposition 2.2.10 that the main algorithms studied in this chapter (OGD and OMD_{NE}) are in this regime, as *they only increase current state coordinates corresponding to files requested in the previous timeslot*; as such, their update costs can be considered to be zero.

2.2.3 Fractional Caching and Gradient-based Algorithms

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

Proposition 2.2.1. *The aggregate cost minimization policy is a policy $\mathcal{A}$ that selects for every timeslot $t \in [T-1]$ the state*

$$\boldsymbol{x}_{t+1} = \arg\min_{\boldsymbol{x} \in \mathcal{X}} \sum_{t'=1}^{t} f_{\boldsymbol{r}_{t'}}(\boldsymbol{x}). \tag{2.5}$$

This policy has linear (worst-case) regret, i.e., $\text{Regret}(\mathcal{A}) = \Omega(T)$.

We provide a proof in Appendix 1.1. A more conservative approach, that indeed leads to sub-linear regret, is to take gradual steps, moving in the direction of a better decision according to the latest cost; we present algorithms of this nature in this section.

Algorithm 2.1 Online mirror descent

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

1: **for** $t \leftarrow 1, 2, \ldots, T$ **do** ▷ Incur a cost $f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$, and receive a gradient $\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x})$
2: $\quad \hat{\boldsymbol{x}}_t \leftarrow \nabla\Phi(\boldsymbol{x}_t)$ ▷ Map primal point to dual point
3: $\quad \hat{\boldsymbol{y}}_{t+1} \leftarrow \hat{\boldsymbol{x}}_t - \eta\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$ ▷ Take gradient step in the dual space
4: $\quad \boldsymbol{y}_{t+1} \leftarrow (\nabla\Phi)^{-1}(\hat{\boldsymbol{y}}_{t+1})$ ▷ Map dual point to a primal point
5: $\quad \boldsymbol{x}_{t+1} \leftarrow \Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\boldsymbol{y}_{t+1})$ ▷ Project new point onto feasible region $\mathcal{X}$
6: **end for**

2.2.3.1 Online Gradient Descent (OGD)

In OGD, introduced by Paschos et al. [19] for online caching, the cache is initialized with a feasible state $\boldsymbol{x}_1 \in \mathcal{X}$ and updated as follows. Upon receiving a request batch $\boldsymbol{r}_t$, the cost $f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$ is incurred and the next state becomes:

$$\boldsymbol{x}_{t+1} = \Pi_{\mathcal{X}}\left(\boldsymbol{x}_t - \eta\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)\right), \quad \text{for all } t \in [T-1], \tag{2.6}$$

where $\Pi_{\mathcal{X}}(\cdot)$ is the Euclidean projection onto $\mathcal{X}$, that ensures feasibility, and $\eta \in \mathbb{R}_+$ is called the learning rate. Note that the state $\boldsymbol{x}_{t+1}$ obtained according Eq. (2.6) is indeed a function of $\{(\boldsymbol{r}_t, \boldsymbol{x}_t)\} \subset \{(\boldsymbol{r}_s, \boldsymbol{x}_s)\}_{s=1}^{t}$ for every $t \geq 1$; hence, OGD is indeed an online caching policy as defined in Section 2.2.2. Paschos et al. [19] show that OGD attains sub-linear regret when $R = h = 1$; more specifically:

Theorem 2.2.2. *([19, Theorem 2]) When $R = h = 1$, the regret of OGD is bounded as follows:*

$$\text{Regret}_T(\text{OGD}) \leq \|\boldsymbol{w}\|_\infty \sqrt{\min(2k, 2(N-k))T}. \tag{2.7}$$

In other words, OGD attains an $O\left(\sqrt{T}\right)$ regret when $R = h = 1$. In this chapter, we study a broader class of gradient descent algorithms that include OGD as a special case. As we will see below (see Theorem 2.2.7), the regret attained by OGD is not necessarily the tightest possible when $R \neq 1 \neq h$; broadening the class of algorithms we consider allows us to improve upon this bound.

2.2.3.2 Online Mirror Descent (OMD)

OMD [20, Section 5.3] is the online version of the mirror descent (MD) algorithm [52] for convex optimization of a fixed, known function. The main premise behind mirror descent is that variables and gradients live in two distinct spaces: the *primal space*, for variables, and the *dual space*, for gradients. The two are linked via a function known as a *mirror map*. Contrary to standard gradient descent, updates using the gradient occur on the dual space; the mirror map is used to invert this update to a change on the primal variables. For several constrained optimization problems of interest, mirror descent leads to faster convergence compared to gradient descent [53, Section 4.3]. OMD arises by observing that MD is agnostic to whether the gradients are obtained from a *fixed* function, or a sequence revealed adversarially.

OMD for Caching. Applied to our caching problem, OMD takes the form summarized in Algorithm 2.1. In our case, both the primal and dual spaces are $\mathbb{R}^N$. To disambiguate between the two,

we denote primal points by $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$ and dual points by $\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}} \in \mathbb{R}^N$, respectively. Formally, OMD is parameterized by (1) a fixed learning rate $\eta \in \mathbb{R}_+$, and (2) a differentiable map $\Phi : \mathcal{D} \to \mathbb{R}$, strictly convex over $\mathcal{D}$ and ρ-strongly convex over $\mathcal{X} \cap \mathcal{D}$, where $\mathcal{X}$ is included in the closure of $\mathcal{D}$; that is

$$\mathcal{X} \subseteq \text{closure}(\mathcal{D}). \tag{2.8}$$

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

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

$$\hat{\boldsymbol{x}}_t = \nabla\Phi(\boldsymbol{x}_t). \tag{2.9}$$

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

$$\hat{\boldsymbol{y}}_{t+1} = \hat{\boldsymbol{x}}_t - \eta \nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t). \tag{2.10}$$

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

$$\boldsymbol{y}_{t+1} = (\nabla\Phi)^{-1}(\hat{\boldsymbol{y}}_{t+1}). \tag{2.11}$$

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

$$\boldsymbol{x}_{t+1} = \Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\boldsymbol{y}_{t+1}), \tag{2.12}$$

where $\Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\,\cdot\,)$ is the Bregman projection, which we define formally below, in Definition 2.2.1.

Together, steps (2.9)–(2.12) define OMD. Note that, as it was the case for OGD, $\boldsymbol{x}_{t+1}$ is a function of $\{(\boldsymbol{r}_t, \boldsymbol{x}_t)\} \subset \{(\boldsymbol{r}_s, \boldsymbol{x}_s)\}_{s=1}^{t}$, hence OMD is indeed an online algorithm. Two additional technical assumptions on Φ and $\mathcal{D}$ must hold for steps (2.11) and (2.12) to be well-defined.[2] First, the gradient of Φ must diverge at the boundary of $\mathcal{D}$; this, along with strict convexity, ensures the existence and uniqueness of the Bregman projection in (2.12). Second, the image of $\mathcal{D}$ under the gradient of Φ should take all possible values, that is $\nabla\Phi(\mathcal{D}) = \mathbb{R}^N$; this, along again with strict convexity, ensures that $\nabla\Phi$ is one-to-one and onto, so its inverse exists and Eq. (2.11) is well-defined.

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

To conclude our description of OMD, we define the Bregman projection [54].

Definition 2.2.1. The Bregman projection denoted by $\Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}} : \mathbb{R}^N \to \mathcal{X} \cap \mathcal{D}$, is defined as

$$\Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\boldsymbol{y}) = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathcal{X}\cap\mathcal{D}} D_\Phi(\boldsymbol{x}, \boldsymbol{y}), \qquad \text{where} \qquad D_\Phi(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x}) - \Phi(\boldsymbol{y}) - \nabla\Phi(\boldsymbol{y})^T(\boldsymbol{x} - \boldsymbol{y}) \tag{2.13}$$

is the Bregman divergence associated with the mirror map Φ.

[2] All hold for the algorithms we consider in Section 2.2.3.3.

2.2.3.3 Analysis of Online Mirror Descent Algorithms

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

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

$$\Phi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_q^2, \quad \text{where } q \in (1,2], \text{ and } \quad \mathcal{D} = \mathbb{R}^N. \tag{2.14}$$

It is easy to verify that Φ and $\mathcal{D}$, defined as above, satisfy all technical requirements set in Section 2.2.3.2 on a mirror map and its domain. We define $\text{OMD}_{q\text{-norm}}$ to be the OMD Algorithm 2.1 with Φ and q given by Eq. (2.14). Note that this map generalizes OGD, which corresponds to the special case $q = 2$. In what follows, we denote by $\|\cdot\|_p$ the dual norm of $\|\cdot\|_q$. Then, $p \in [2,\infty)$ is such that $\frac{1}{p} + \frac{1}{q} = 1$. Note that sometimes $\text{OMD}_{q\text{-norm}}$ is referred to as a *p-norm algorithm* [48].

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

Theorem 2.2.3. *For* $\eta = \sqrt{\frac{(q-1)k^2\left(k^{-\frac{2}{p}} - N^{-\frac{2}{p}}\right)}{\|\boldsymbol{w}\|_\infty^2 h^2 \left(\frac{R}{h}\right)^{\frac{2}{p}} T}}$*, the regret of* $\text{OMD}_{q\text{-norm}}$ *over* $\mathcal{X}$ *satisfies:*

$$\text{Regret}_T(\text{OMD}_{q\text{-norm}}) \leq \|\boldsymbol{w}\|_\infty hk \left(\frac{R}{h}\right)^{\frac{1}{p}} \sqrt{\frac{1}{q-1}\left(k^{-\frac{2}{p}} - N^{-\frac{2}{p}}\right) T}. \tag{2.15}$$

The proof can be found in Appendix 1.3. We use an upper bound on the regret of general OMD from [53, Theorem 4.2] and relate it to our setting; in doing so, we bound the diameter of $\mathcal{X}$ w.r.t. Bregman divergence under Φ as well as the dual-norm $\|\cdot\|_p$ of the gradients $\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$.

Comparing Theorem 2.2.3 to Theorem 2.2.2, we see that both attain an $O\left(\sqrt{T}\right)$ regret. A natural question to ask when comparing the two bounds is whether there are cases where $\text{OMD}_{q\text{-norm}}$ with $q \neq 2$ outperforms OGD (i.e., $\text{OMD}_{2\text{-norm}}$). The constants in the r.h.s. of Eq. (2.15) depend on the diversity ratio $\frac{R}{h}$; this, in turn, affects which is the optimal q, i.e., the one that minimizes the bound in Eq. (2.15). Let $q_* = \arg\inf_{q\in(1,2]} \mathsf{ub}(q)$ be the optimal q, where $\mathsf{ub} : (1,2] \to \mathbb{R}_+$ is the upper bound in Eq. (2.15). Note that $q_* \in [1,2]$. Figure 2.1 shows q_* as a function of the diversity ratio, for different values of cache capacity k. We observe that OGD ($q = 2$) is optimal for lower diversity regimes and larger caches; when diversity $\frac{R}{h}$ increases or cache capacity k decreases, values $q < 2$ become optimal. The transition from $q_* = 2$ to $q_* = 1$ is sharp, and becomes sharper as k increases.

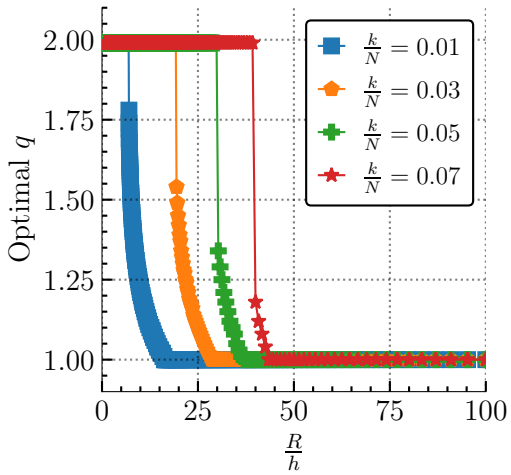


Figure 2.1: Numerical characterization of $q_* \in [1, 2]$ as a function of the diversity ratio R/h, for different cache capacities k expressed as fractions of the catalog size ($N = 100$). Given R/h, the optimal q_* is determined as the value in $[1, 2]$ that minimizes the upper-bound in Eq. (2.15). Higher values of R/h represent more diverse requests. Under small diversity, OGD is optimal; as diversity increases, mirror maps for which $q < 2$ attain a more favorable upper bound than OGD.

Optimality Regimes. Motivated by these observations, we turn our attention to formally characterizing the two regimes under which optimality transitions from $q_* = 2$ to $q_* = 1$. We first determine the upper bound on the regret for these two regimes. Indeed, by setting $q = 2$ in Theorem 2.2.3, we obtain the following bound, generalizing Theorem 2.2.2 to the case $R/h > 1$:

Corollary 2.2.4. *For* $\eta = \sqrt{\frac{k\left(1-\frac{k}{N}\right)}{\|\boldsymbol{w}\|_\infty^2 hRT}}$ *the regret of OGD, satisfies:*

$$\text{Regret}_T(\text{OGD}) \leq \|\boldsymbol{w}\|_\infty \sqrt{hRk\left(1 - \frac{k}{N}\right)T}. \tag{2.16}$$

This a direct consequence of Theorem 2.2.3 by replacing $q = 2$ in Eq. (2.15). We note that, in this result, we tighten the bound of Paschos et al. [19]: for $R = h = 1$, the bound in Eq. (2.16) is smaller than the one in Theorem 2.2.2 by at least a $\sqrt{2}$ factor.

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

Corollary 2.2.5. *As q converges to 1, the upper bound on* $\text{OMD}_{q\text{-norm}}$ *regret given by Eq.* (2.15) *converges to:*

$$\|\boldsymbol{w}\|_\infty hk\sqrt{2\log\left(\frac{N}{k}\right)T}. \tag{2.17}$$

The proof can be found in Appendix 1.4. This limit is precisely the bound on the regret attained under the neg-entropy mirror map (see Theorem 2.2.8 below). Armed with Corollaries 2.2.4 and 2.2.5, we can formally characterize the regimes in which either of the two strategies become dominant:

Theorem 2.2.6. *The request bound for* $\text{OMD}_{q\text{-norm}}$ *in Eq.* (2.15) *is minimized for* $q = 2$*, when* $\frac{R}{h} \leq k$.

In other words, when the diversity ratio is smaller than the cache size, it is preferable to update the cache via OGD. The proof, in Appendix 1.5, establishes that the upper bound in Eq. (2.15) is monotonically decreasing w.r.t q in the specified interval $\frac{R}{h} \leq k$. Our next result characterizes then the neg-entropy (q converges to 1) mirror map outperforms OGD:

Theorem 2.2.7. *The limit, as* q *converges to* 1*, of the* $\text{OMD}_{q\text{-norm}}$ *regret bound in Eq.* (2.17) *is smaller than the corresponding bound for OGD (*$\text{OMD}_{q\text{-norm}}$ *with* $q = 2$*) when* $\frac{R}{h} > 2\sqrt{Nk}$.

The proof is provided in Appendix 1.6. We stress that Theorem 2.2.7 implies the sub-optimality of OGD in the regime $\frac{R}{h} > 2\sqrt{Nk}$. The experiments in Figure 2.1 suggest the bound in Theorem 2.2.7 is quite tight: for example for $k = 7$ the bounds suggest $q = 1$ should be optimal when R/h exceeds $2\sqrt{100 \times 7} \approx 52.9$, while experiments show that it is optimal when R/h exceeds 45. On the contrary, we observe that the bound in Theorem 2.2.6 seems to be loose and the transitions we observe in Figure 2.1 are sharper than what one would predict from the bounds.

Dual-Primal Update and Bregman Projection. Having characterized the regret of $\text{OMD}_{q\text{-norm}}$ algorithms, we turn our attention to implementation issues. The map to the dual space and back in Eq. (2.9) and Eq. (2.11) (Lines 2 and 4 in Algorithm 2.1), have the following expression [55], respectively:

$$\hat{x}_{t,i} = \left(\nabla\Phi(\boldsymbol{x}_t)\right)_i = \text{sign}(x_{t,i})\frac{|x_{t,i}|^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}}, \quad \text{for all } \; i \in \mathcal{N}, \tag{2.18}$$

$$y_{t+1,i} = \left(\left(\nabla\Phi\right)^{-1}\left(\hat{\boldsymbol{y}}_{t+1}\right)\right)_i = \text{sign}(\hat{y}_{t+1,i})\frac{|\hat{y}_{t+1,i}|^{p-1}}{\left\|\hat{\boldsymbol{y}}_{t+1}\right\|_p^{p-2}}, \quad \text{for all } \; i \in \mathcal{N}. \tag{2.19}$$

Finally, for all $q \in (1, 2]$ the Bregman projection in Eq. (2.12) (Line 5 in Algorithm 2.1) involves solving a convex optimization problem, in general. For the OGD Algorithm however ($q = 2$) the projection is the usual Euclidean projection, and can be performed in $O\left(N^2\right)$ steps, using the projection algorithm by Wang and Lu [56]. Specifically when $\frac{R}{h} = 1$, only a single coefficient is updated through the gradient step (Lines 2–4 in Algorithm 2.1) per iteration, and Paschos et al. [19] provide an algorithm that performs the projection in $O\left(N\right)$ time.[3]

2.2.3.4 Neg-Entropy Mirror Map

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

[3]To be precise, the projection algorithm as presented in [19] requires at each iteration a preliminary step with complexity $O\left(N\log(N)\right)$ to sort a vector of size N, followed by $O\left(N\right)$ steps. However, it is possible to replace sorting by $O\left(\log(N)\right)$ binary search and insertion operations reducing the complexity to $O\left(N\right)$ per iteration.

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

$$\Phi(\boldsymbol{x}) = \sum_{i=1}^{N} x_i \log(x_i), \text{ and } \mathcal{D} = \mathbb{R}^N_{>0}. \tag{2.20}$$

Note that, as per the requirements in Section 2.2.3.2, $\mathcal{X} \subseteq \text{closure}(\mathcal{D})$. Also, $\nabla\Phi$ indeed diverges at the boundary of $\mathcal{D}$, and $\nabla\Phi(\mathcal{D}) = \mathbb{R}^N$, as

$$\frac{\partial\Phi(\boldsymbol{x})}{\partial x_i} = 1 + \log x_i, \quad \text{for all } i \in \mathcal{N}. \tag{2.21}$$

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

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

Theorem 2.2.8. *For* $\eta = \sqrt{\frac{2\log(N/k)}{\|\boldsymbol{w}\|_\infty^2 h^2 T}}$, *the regret of* OMD_{NE} *satisfies:*

$$\text{Regret}_T(\text{OMD}_{\text{NE}}) \leq \|\boldsymbol{w}\|_\infty hk\sqrt{2\log(N/k)}. \tag{2.22}$$

The proof, in Appendix 1.8, is similar to the proof of Theorem 2.2.3. Using again the general bound of the regret of OMD algorithms in Bubeck [53, Theorem 4.2], we bound the diameter of $\mathcal{X}$ w.r.t. to the Bregman divergence as well as the dual norm $\|\cdot\|_\infty$ of gradients $\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$. Crucially, we observe that OMD_{NE} indeed attains the same regret bound as the one in Corollary 2.2.5, namely, the bound on $\text{OMD}_{q\text{-norm}}$ when q converges to 1. This immediately implies the advantage of OMD_{NE} over OGD in high diversity ratio regimes, as described in Section 2.2.3.3 and Theorem 2.2.7.

Dual-Primal Update and Bregman Projection. As $\nabla\Phi(\boldsymbol{x})$ is given by Eq. (2.21), the inverse mapping is given by $\left((\nabla\Phi)^{-1}(\hat{\boldsymbol{y}}_{t+1})\right)_i = \exp(\hat{y}_{t,i} - 1)$. Hence, the map to the dual space and back in Eq. (2.9)–Eq. (2.11) (Lines 2–4 in Algorithm 2.1) can be concisely written for all $i \in \mathcal{N}$ as:

$$y_{t+1,i} = \exp\left(\hat{x}_{t,i} - \eta\frac{\partial f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)}{\partial x_i} - 1\right) = \exp\left(\log(x_{t,i}) - \eta\frac{\partial f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)}{\partial x_i}\right) = x_{t,i}\, e^{-\eta\frac{\partial f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)}{\partial x_i}}. \tag{2.23}$$

In other words, OMD under the neg-entropy mirror map adapts the cache state via *a multiplicative rule* (namely, the one implied by the above equation), as opposed to the additive rule of OGD (see Eq. (2.6)). In Theorem 1.2 we prove that $\text{OMD}_{q\text{-norm}}$ when q converges to 1 also adapts the cache state via a multiplicative update rule; moreover, it is equivalent to OMD_{NE} over the simplex. This justifies why the regret bounds for the two algorithms in Eq. (2.17) and Eq. (2.22) are identical.

Finally, the projection algorithm onto the capped simplex can be implemented in $O(N + k\log(k))$ time for arbitrary R and h values using a waterfilling-like algorithm. The full procedure is presented in Algorithm 2.2. The algorithm receives as input the top-k elements of $\boldsymbol{y}$, sorted in a descending order. It then identifies via a linear search which elements exceed an appropriate threshold and set them to one. The other elements are scaled by a constant factor to satisfy the capacity constraint. The following theorem holds:

Algorithm 2.2 Neg-Entropy Bregman projection onto the capped simplex

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

Theorem 2.2.9. *Algorithm 2.2 returns the projection* $\Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\boldsymbol{y})$ *onto the capped simplex* $\mathcal{X}$ *under the neg-entropy* Φ. *It requires* $O(N + k\log(k))$ *operations per* OMD_{NE} *iteration, for general values of* R *and* h, *and only* $O(k)$ *operations, when* $\frac{R}{h} = 1$.

The proof is given in Appendix 1.9. To prove this theorem, we characterize the KKT conditions of the minimization problem. Then we show that these conditions can be checked in $O(k)$ time. Finally, we show how maintaining $\boldsymbol{y}$ in a partially sorted list across iterations leads to the reported complexity results. Theorem 2.2.9 implies that OMD_{NE} *has significant computational savings when compared to OGD* (cf. Section 2.2.3.3), both when $\frac{R}{h} = 1$ and for general values of R and h.

2.2.4 Update Costs

The model presented in Section 2.2.2 can be extended by adding the cost to update the cache state after the batch of R requests has been served. This cost may quantify the additional load on the server or on the network. This update cost is often called *movement cost* [53] or *switching cost* [45]. As the state changes from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$, the cache evicts part of the file i if $x_{t+1,i} < x_{t,i}$ and stores additional bytes of it if $x_{t+1,i} > x_{t,i}$. We make the following assumptions:

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

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

$$\text{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}) = \sum_{i \notin \text{supp}(\boldsymbol{r}_t)} w'_i \max\left\{0, x_{t+1,i} - x_{t,i}\right\}, \tag{2.24}$$

where $\text{supp}(\boldsymbol{r}_t) = \{i \in \mathcal{N} : r_{t,i} \neq 0\}$ denotes the support of $\boldsymbol{r}_t$, i.e., the set of files that have been requested during the t-th timeslot, and $w'_i \in \mathbb{R}_+$ is the cost to retrieve the whole file i, and can in general be different from the cost w_i appearing in (2.3).

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

$$\text{E-Regret}_T(\mathcal{A}) = \sup_{\{\boldsymbol{r}_1,\boldsymbol{r}_2,\ldots,\boldsymbol{r}_t\}\in\mathcal{R}_{R,h}^T} \left\{ \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) + \text{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t,\boldsymbol{x}_{t+1}) - \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_*) \right\} \tag{2.25}$$

$$\leq \sup_{\{\boldsymbol{r}_1,\boldsymbol{r}_2,\ldots,\boldsymbol{r}_t\}\in\mathcal{R}_{R,h}^T} \left\{ \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) - \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_*) \right\} + \sup_{\{\boldsymbol{r}_1,\boldsymbol{r}_2,\ldots,\boldsymbol{r}_t\}\in\mathcal{R}_{R,h}^T} \left\{ \sum_{t=1}^T \text{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t,\boldsymbol{x}_{t+1}) \right\}. \tag{2.26}$$

Equation (2.26) shows that the regret of an arbitrary online algorithm can be bounded by considering the regret we have derived so far (Eq. (2.4)), ignoring update costs, and subsequently accounting for an additional term corresponding to the update. Note that the optimal static allocation does not incur any update cost. Equation (2.26) implies that any policy with $O\left(\sqrt{T}\right)$ regret and $O\left(\sqrt{T}\right)$ update cost in expectation has also $O\left(\sqrt{T}\right)$ extended regret.

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

Proposition 2.2.10. *For any request batch $\boldsymbol{r}_t$ received at time slot $t \in [T]$, the update of fractional cache state from $\boldsymbol{x}_t \in \mathcal{X}$ to $\boldsymbol{x}_{t+1} \in \mathcal{X}$ obtained by* OMD_{NE} *or* OGD *has no cost, i.e.,* $\text{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t,\boldsymbol{x}_{t+1}) = 0$.

The proof is provided in Appendix 1.10. In fact, the gradient step increases the fraction $x_{t,i}$ only for files i that have been requested, and the projection step reduces the fraction for all other files in order to satisfy the capacity constraint. It follows that $x_{t+1,i} - x_{t,i} > 0$ if and only if $i \in \text{supp}(\boldsymbol{r}_t)$, and thus $\text{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t,\boldsymbol{x}_{t+1}) = 0$. Hence, the $O\left(\sqrt{T}\right)$ regret guarantees we proved in the previous sections for OGD and OMD_{NE} extend to the more general definition in (2.25). In the next section, we show that update costs *cannot be neglected* when caches are forced to store files in their entirety.

2.2.5 Integral Caching

In the previous sections, we assumed that the cache can store arbitrarily scriptsize chunks of a file, and this allowed us to design no-regret policies that employ fractional caching. However, this assumption can be too strong in some applications. For example, when the catalog is composed of scriptsize-sized files, the discreteness of chunks sizes cannot be neglected; moreover, the metadata needed for each chunk can cause memory and computational overheads. These observations motivate us to study the case when the cache can only store the entire file. We refer to this setting as the *integral* caching. Formally, we restrict the cache states to belong to the set $\mathcal{Z} = \left\{\boldsymbol{\zeta} \in \{0,1\}^N : \sum_{i\in\mathcal{N}} \zeta_i = k\right\}$. Note that the set $\mathcal{Z}$ is a restriction of the set of fractional caching states $\mathcal{X}$ to its corners, i.e., $\mathcal{Z} = \mathcal{X} \cap \{0,1\}^N$; thus, we maintain the same definition of the requests and the service cost objective in Section 2.2.2. In this setting, we allow policies to be randomized.

This extension turns out to be necessary in order to have a sublinear regret policy; formally, we have:

Proposition 2.2.11. *Any deterministic policy restricted to select integral cache states in $\mathcal{Z}$ has the following lower bound on its regret:*

$$\text{Regret}_T(\mathcal{A}) \geq k\,(1 - k/N)\,T. \tag{2.27}$$

To prove the proposition, we show that an adversary can exploit the deterministic nature of the policy by continuously requesting the files that are not stored in the cache. We provide the proof in Appendix 2.1.

We thus turn our attention to randomized policies. In particular, we focus on a special class of randomized policies, constructed by (1) a fractional online caching policy $\mathcal{A}$, i.e., of the type we have studied so far (see Section 2.2.2), combined with (2) a randomized rounding scheme Ξ, that maps fractional caching states to integral ones. In particular, for every $t \geq 1$ the randomized rounding scheme $\Xi : \mathcal{X}^t \times \mathcal{Z}^{t-1} \times [0,1] \to \mathcal{Z}$ maps the previous fractional cache states $\{\boldsymbol{x}_s\}_{s=1}^{t-1} \in \mathcal{X}^{t-1}$, the current fractional cache state $\boldsymbol{x}_t \in \mathcal{X}$, the previous random cache states $\{\boldsymbol{z}_s\}_{s=1}^{t-1} \in \mathcal{Z}^{t-1}$, and a source of randomness $\xi_t \in [0,1]$ to a new random cache state $\boldsymbol{z}_t \in \mathcal{Z}$ where

$$\mathbb{E}[\boldsymbol{z}_t] = \boldsymbol{x}_t. \tag{2.28}$$

Note that the rounding function takes into account not only the current fractional state $\boldsymbol{x}_t$, which determines its expectation, but also the past fractional and integral states $(\{(\boldsymbol{z}_s, \boldsymbol{x}_s)\}_{s=1}^{t-1})$; this is in fact instrumental in attaining a sublinear extended regret (see Theorems 2.2.13 and 2.1 below).

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

$$\text{Regret}_T(\mathcal{A}, \Xi) = \sup_{\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_t\} \in \mathcal{R}_{R,h}^T} \left\{ \mathbb{E}\left[\sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{z}_t) \right] - \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{z}_*) \right\}, \tag{2.29}$$

and

$$\text{E-Regret}_T(\mathcal{A}, \Xi) = \sup_{\{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_t\} \in \mathcal{R}_{R,h}^T} \left\{ \mathbb{E}\left[\sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{z}_t) + \text{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}) \right] - \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{z}_*) \right\}, \tag{2.30}$$

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

$$\boldsymbol{z}_* = \arg\min_{\boldsymbol{z} \in \mathcal{Z}} \sum_{t=1}^{T} f_{\boldsymbol{r}_t}(\boldsymbol{z}) \tag{2.31}$$

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

Proposition 2.2.12. *Any randomized caching policy constructed by an online policy $\mathcal{A}$ combined with a randomized rounding scheme Ξ has the same regret as $\mathcal{A}$, i.e.,* $\text{Regret}_T(\mathcal{A}, \Xi) = \text{Regret}_T(\mathcal{A})$, *given by* (2.29) *and* (2.4), *respectively.*

The result follows from the linearity of the cost functions and the expectation operator; moreover, the static optimum can always be selected to be integral from the integrality of the capacity constraint and linearity of the objective function. The proof is provided in Appendix. 2.2.

Proposition 2.2.12 thus implies that regret guarantees for a fractional policy $\mathcal{A}$ readily transfer to the integral regime, when coupled with rounding Ξ. Unfortunately, when considering the extended regret (Eq. (2.30)) instead, naïve rounding policies can arbitrary evict and fetch objects to the cache causing large update costs (see Theorem 2.2.13). Thus, unless rounding is carefully designed, we may fail to have sublinear regret guarantees when accounting for update costs. In the next section, we show how a randomized rounding scheme Ξ can be selected to avoid incurring large update costs.

2.2.5.1 Rounding Schemes and Extended Regret

Online Independent Rounding. If we consider a fractional caching state $\boldsymbol{x}_t \in \mathcal{X}$, then a random integral caching state $\boldsymbol{z}_t \in \mathcal{Z}$ with the marginal $\mathbb{E}[\boldsymbol{z}_t] = \boldsymbol{x}_t$ exists and can be sampled in polynomial time (see, e.g., [21, 57]). Thus, a rounding scheme Ξ can be constructed with such strategy that takes as input the current fractional cache state $\boldsymbol{x}_t$ ignoring previous fractional cache states $\{\boldsymbol{x}_s\}_{s=1}^{t-1} \in \mathcal{X}^{t-1}$, and previous random cache states $\{\boldsymbol{z}_s\}_{s=1}^{t-1} \in \mathcal{Z}^{t-1}$. linearWe provide pseudocode for this procedure in Algorithm 2.3.[4] Because at any time t the random cache states are sampled independently from previous random cache states, we refer to this rounding as *online independent rounding*. Unfortunately, when considering the extended regret (2.30), any caching policy coupled with this rounding scheme loses its $O\left(\sqrt{T}\right)$ regret guarantee. Formally, we have the following:

Theorem 2.2.13. *Any randomized caching policy constructed by an online policy $\mathcal{A}$ combined with online independent rounding as a randomized rounding scheme Ξ has linear (worst-case) extended regret, i.e.,* $\text{E-Regret}_T(\mathcal{A}, \Xi) = \Omega(T)$.

The proof is provided in Appendix 2.3. Online independent rounding causes frequent cache updates, as it samples a new state from $\boldsymbol{z}_t$ ignoring the previous state $\boldsymbol{\zeta}_{t-1}$ sampled from $\boldsymbol{z}_{t-1}$. Intuitively, imposing dependence (coupling) between the two consecutive random states may significantly reduce the expected update cost.

Online Coupled Rounding. To address this issue, our proposed *online coupled rounding* scheme is described also in Algorithm 2.3, using however the same randomization source across all timeslots. In particular, the coupling across states comes from the use of the same uniform random variable ξ. A consequence of this coupling is that the next integral state can be computed efficiently and leads to scriptsize movement costs. Note that Algorithm 2.3 does not necessarily find an optimal coupling, still it yields a sublinear update cost, and thus preserves the sublinearity of the regret. This is formally expressed in the following Theorem:

[4] Algorithm 2.3 provides a linear-time variant of the algorithms proposed in [21, 57, 58]. The algorithm samples an integral caching state without constructing a distribution and its support.

Algorithm 2.3 Online Rounding

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

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

Theorem 2.2.14. *Consider a randomized caching policy constructed by an OMD policy $\mathcal{A}$ with sublinear regret (i.e., configured with learning rate $\eta = \Theta\left(1/\sqrt{T}\right)$) combined with online coupled rounding Ξ in Algorithm 2.3 (fixed $\xi_t = \xi$ for $t \in [T]$). The expected movement cost of the random integral cache states is $\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] = O\left(\sqrt{T}\right)$. Moreover, the extended regret is sublinear $\text{E-Regret}_T(\mathcal{A}, \Xi) = O\left(\sqrt{T}\right)$.*

We provide the proof in Appendix 2.5. In summary, any OMD policy combined with online coupled rounding yields $O\left(\sqrt{T}\right)$ extended regret in the integral caching setting. The computational complexity of online coupled rounding is $O(N)$ (see also Figure 2.2).

Online Optimally-Coupled Rounding. It is possible in general to reduce the update cost of online coupled rounding. In particular, minimizing the expected update cost over all joint distributions of the random variables $\boldsymbol{z}_t$ and $\boldsymbol{z}_{t+1}$ leads to an optimal transport problem [59]. For completeness, we describe this rounding scheme here, though (1) it does not reduce the extended regret guarantee attained by online coupled rounding (up to multiplicative constants), and (2) it has an increased computational cost.

Formally, at each time t the random variables $\boldsymbol{z}_t$ with marginal $\boldsymbol{x}_t$ can be constructed by sampling from a distribution $\boldsymbol{p}_t$ with $O(N)$ support $\left\{\boldsymbol{\zeta}_t^1, \boldsymbol{\zeta}_t^2, \ldots, \boldsymbol{\zeta}_t^{|\boldsymbol{p}_t|}\right\}$, where $p_{t,i} = \mathbb{P}(\boldsymbol{z}_t = \boldsymbol{\zeta}_t^i)$ for $i \in [|\boldsymbol{p}_t|]$. The decomposition can be performed in $O(kN\log(N))$ steps [21]. We denote the joint probability $\mathbb{P}\left(\boldsymbol{z}_{t+1} = \boldsymbol{\zeta}_{t+1}^j, \boldsymbol{z}_t = \boldsymbol{\zeta}_t^i\right)$ by the flow $f_{i,j}$ for all $(i,j) \in [|\boldsymbol{p}_t|] \times [|\boldsymbol{p}_{t+1}|]$. The optimal transport problem can be described by the following linear program:

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

$$\text{s.t.} \quad \sum_{j=1}^{|\boldsymbol{p}_{t+1}|} f_{i,j} = p_{t,i}, \quad \sum_{i=1}^{|\boldsymbol{p}_t|} f_{i,j} = p_{t+1,j}, \quad f_{i,j} \in [0,1], \forall (i,j) \in [|\boldsymbol{p}_t|] \times [|\boldsymbol{p}_{t+1}|].$$

We solve the above linear program to obtain a minimum-cost flow $\boldsymbol{f}$. If the random state at time t is $\boldsymbol{\zeta}_t^i$, then we select the new random state to be $\boldsymbol{\zeta}_{t+1}^j$ with (conditional) probability

$\mathbb{P}\left(\boldsymbol{z}_{t+1} = \boldsymbol{\zeta}^j_{t+1} \mid \boldsymbol{z}_t = \boldsymbol{\zeta}^i_t\right) = \frac{f^*_{i,j}}{p^t_i}$. Such coupling ensures that the expected update cost is minimized. When we combine this rounding scheme with a no-regret fractional policy we obtain sublinear extended regret (2.30):

Corollary 2.2.15. *Consider an OMD policy $\mathcal{A}$ configured with learning rate $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$ combined with online optimally-coupled rounding Ξ. The obtained randomized integral caching policy has sublinear extended regret, i.e.,* E-Regret$(\mathcal{A}, \Xi) = O\left(\sqrt{T}\right)$.

The corollary follows from Theorem 2.2.14, because online coupled rounding constructs a feasible transportation flow (see Figure 2.3 in Appendix 2.4 for an illustration) that gives sublinear update costs, and the optimal flow can only have lower update costs. The naïve implementation of the optimal transport problem has $O\left(N^3\right)$ time complexity, but several efficient approximations exist in the literature [59] at the expense of losing the established guarantee.

2.2.6 Experiments

2.2.6.1 Experimental setup

Datasets. Throughout all experiments, we assume equal costs per file, i.e., $w_i = w'_i = 1, \forall i \in \mathcal{N}$. The learning rate η^* denotes the learning rate value specified in Corollary 2.2.4 and in Theorem 2.2.8 for OGD and OMD_{NE}, respectively. The learning rate is selected to be η^* unless otherwise mentioned. In what follows, we distinguish the number of batches in the trace (B) and the time horizon (T).

We generate the following synthetic datasets, summarized in Table 2.2.

Fixed Popularity. Requests are i.i.d. and sampled from a catalog of $N = 200$ files according to a *Zipf* distribution with exponent $\alpha = 0.8$. Each batch counts a single request ($R = 1$). We set set the time horizon as $T = 10^5$. The cache capacity is $k = 100$. The total number of requests is the product of the requests in each batch (R) and the number of batches (B), both values are reported in Table 2.2.

Batched Fixed Popularity. Request are generated as above from a Zipf distribution with exponent α, but are now grouped in batches of $R = 5{\times}10^3$ requests. We take different exponents $\alpha \in \{0.1, 0.2, 0.7\}$ for traces *Batched Fixed Popularity* (1), (2), and (3), respectively, in Table 2.2. The parameter α controls the diversity of the files in the request batches. If $\alpha = 0$, then each file is requested with equal probability, corresponding to $\frac{R}{h} \to N$ (high diversity). As we increase α, the requests become more concentrated; this corresponds to $\frac{R}{h} \to 1$ (low diversity). Table 2.2 shows the value of h observed in each trace. In all cases, we select catalog size $N = 10^4$, cache size $k \in \{25, 125, 250\}$, and time horizon $T = 10^4$.

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

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

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

Table 2.2: Trace Summary

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

Table 2.3: Performance Metrics. All are better if lower.

Correspondingly, the time horizon is also set to $T = 10^3$ and, at the end of each time horizon, the cache redistributes uniformly the cache space currently allocated by those files. The cache size is $k = 50$.

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

Akamai Trace. We consider also a real file request trace collected from Akamai Content Delivery Network (CDN) [60]. The trace spans 1 week, and we extract from it about 8.5×10^7 requests for the $N = 10^4$ most popular files. We group requests in batches of size $R = 5 \times 10^3$, and we consider a time horizon $T = 100$ time slots corresponding roughly to 1 hour. The cache size is $k = 25$.

Online Algorithms. Starting with the gradient based algorithms, we implemented $\mathrm{OMD}_{\mathrm{NE}}$ with the projection defined in Algorithm 2.2. We implemented two different projection algorithms for OGD: the one by Paschos et al. [19] for the setting $\frac{R}{h} = 1$, and the one by Wang and Lu [56] for the general setting $\frac{R}{h} > 1$.

In addition, we implemented four caching eviction policies: LRU, LFU, W-LFU, and FTPL. LRU and LFU evict the least recently used and least frequently used file, respectively. While LFU estimates file popularities considering all requests seen in the past, W-LFU [61] is an LFU variant that only considers requests during a recent time window W, which we set equal to $T \times R$ in our experiments. The policies LRU, LFU, and W-LFU are allowed to process individual requests. FTPL is a no-regret

policy proposed by Mukhopadhyay and Sinha [50], which, roughly speaking, behaves as a LFU policy whose request counters are perturbed by some Gaussian noise. Finally, we define *Best Static* to be the optimal static allocation $\boldsymbol{x}_*$, i.e., the configuration storing the k most popular files as we consider $w_i = 1, \forall i \in \mathcal{N}$. We also define *Best Dynamic* to be the caching policy that stores the k most popular files at any time for the synthetic traces (for which the instantaneous popularity is well defined). The optimality of such policy is formally studied in [62].

Online Rounding. We also implemented the three rounding schemes described in Section 2.2.5: (a) the online independent rounding in Algorithm 2.3, (b) the online coupled rounding in Algorithm 2.3, and (c) the online optimally-coupled rounding. The rounding schemes are combined with OGD configured with learning rate $\eta = 0.01$ under the *Downscaled Global Popularity Change* trace.

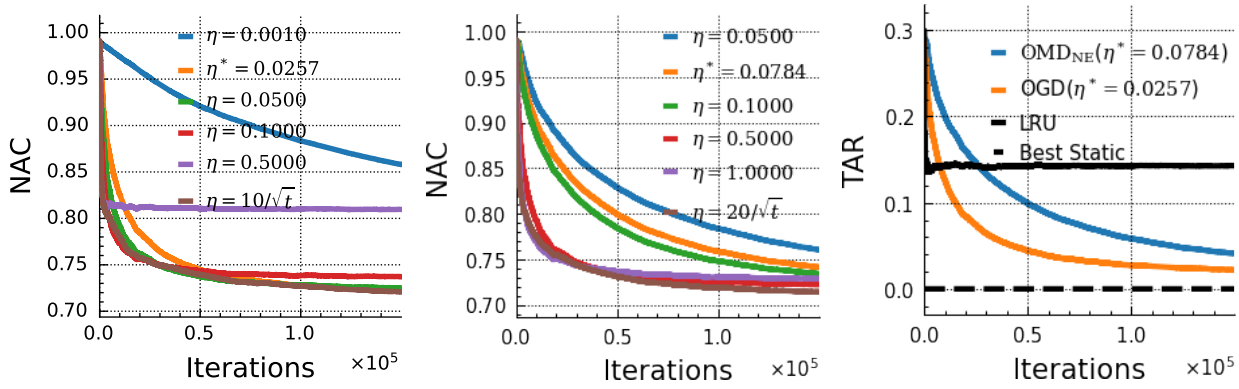
Performance Metrics. We measure performance w.r.t. four metrics defined in Table 2.3. The Normalized Average Cost $\text{NAC}(\mathcal{A}) \in [0, 1]$ corresponds to the time-average cost over the first t time slots, normalized by the batch size R. The Normalized Moving Average Cost $\text{NMAC}(\mathcal{A}) \in [0, 1]$ is computed similarly, using a moving average instead over a time window $\tau > 0$; we use $\tau = 500$ in our experiments. We also consider the Time Average Regret $\text{TAR}(\mathcal{A}) \in [0, R]$, which is precisely the time average regret over the first t time slots. Finally, when studying rounding algorithms, we also measure and report the Cumulative Update Cost $\text{CUC}(\mathcal{A}) \in [0, \infty)$.

2.2.6.2 Results

Stationary Requests. Figures 2.2 (a) and 2.2 (b) show the performance w.r.t. NAC of OGD and OMD_{NE}, respectively, under different learning rates η on the *Fixed Popularity* trace. We observe that both algorithms converge slower under small learning rates, but reach a final lower cost, while larger learning rates lead to faster convergence, albeit to higher final cost. This may motivate the adoption of a diminishing learning rate, that combines the best of the two options, starting large to enable fast convergence, and enabling eventual fine-tuning. We show curves corresponding to a diminishing learning rate both for OGD and OMD_{NE}, and indeed they achieve the smallest costs. The learning rate η^* gives the tightest worst-case regrets for OGD and OMD_{NE}, as stated in Theorems 2.2.4 and 2.2.8. While this learning rate is selected to protect against any (adversarial) request sequence, it is not too pessimistic: Figures 2.2 (a) and 2.2 (b) show it performs well when compared to other learning rates.

Figure 2.2 (c) shows the time-average regret TAR of OGD and OMD_{NE} over the *Fixed Popularity* trace. As both algorithms have sub-linear regret, their time average regret goes to 0 for $T \to \infty$. Note how instead LRU exhibits a constant time average regret.

Effect of Diversity. Figure 2.3 shows the NAC performance of OMD_{NE} and OGD on the traces *Batched Fixed Popularity* (1), (2), and (3) under different cache capacities k and exponent values α. We observe that OMD_{NE} outperforms OGD in the more diverse regimes ($\alpha \in \{0.1, 0.2\}$). This is more apparent for smaller cache sizes k. In contrast, OGD outperforms OMD_{NE} when requests are less diverse ($\alpha = 0.7$); again, this is more apparent for larger cache size k. These observations agree



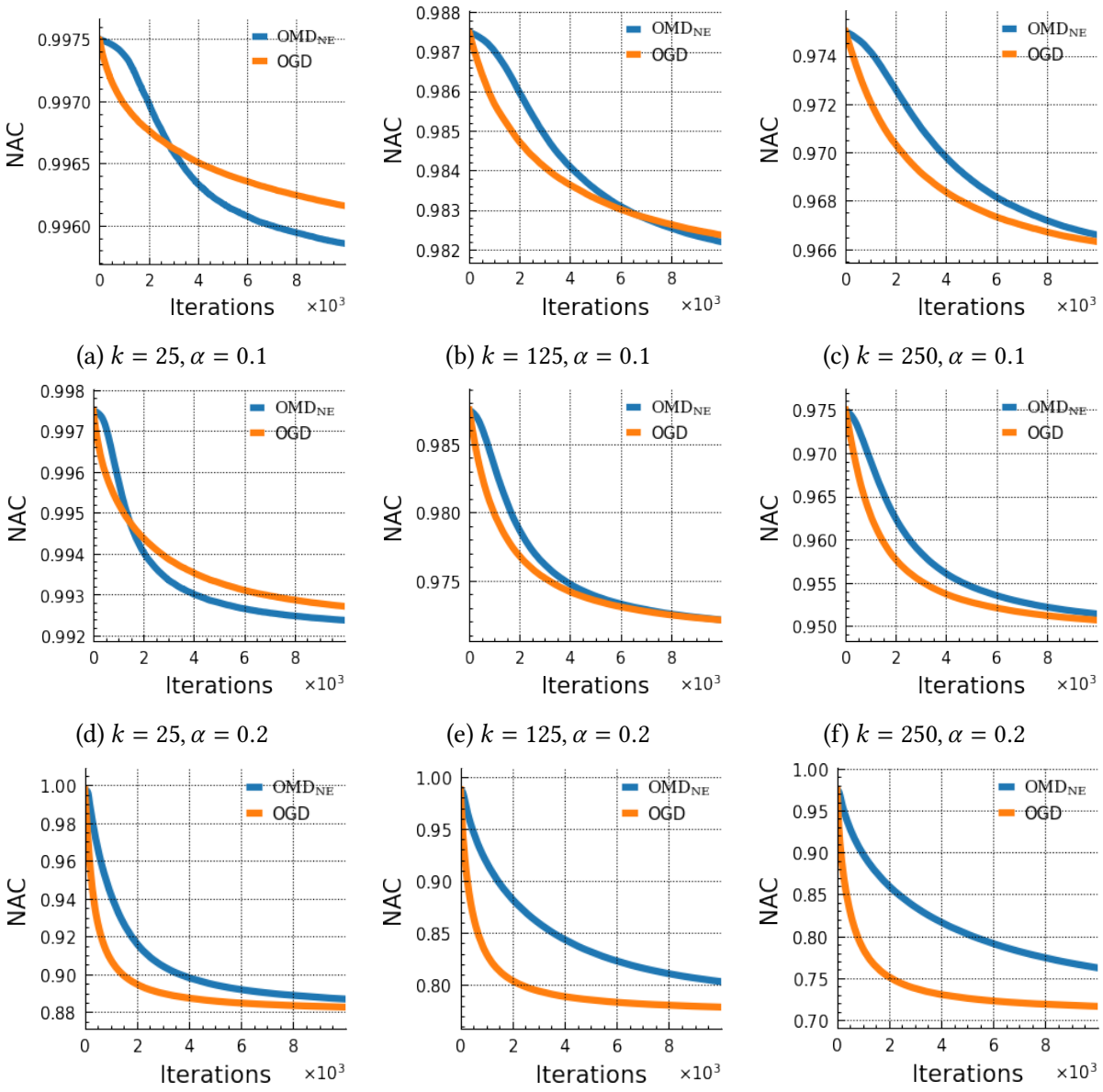
(a) NAC of OGD (b) NAC of OMD_{NE} (c) Time-Average Regret

Figure 2.2: NAC of the different caching policies over the *Fixed Popularity* trace. Subfigures (a) and (b) show the performance of OGD and OMD_{NE} respectively under different learning rates. For small learning rates the algorithms both converge slower but more precisely, while for larger learning rates they converge faster, but to a higher cost. Subfigure (c) shows the time average regret of the two gradient algorithms. When the regret is sub-linear, the time average regret converges to 0 as $T \to \infty$.

with Theorems 2.2.7 and 2.2.6 in Section 2.2.3.3: high diversity and small cache sizes indeed favor OMD_{NE}.

Robustness to Transient Requests. Figure 2.4 shows the normalized average cost of OMD_{NE} and OGD over the *Partial Popularity Change* traces, evaluated under different diversity regimes. Dashed lines indicate the projected performance in the stationary setting (if request popularities stay fixed). Across the different diversity regimes, we find the OMD_{NE} is more robust to popularity changes. In (a), (b) and (c) OMD_{NE} outperforms OGD in the non-stationary popularity setting: we observe a wider performance gap as compared to the stationary setting.

Figure 2.4 (d) and (e) show the normalized average cost over the *Global Popularity Change* trace for the policies OGD and OMD_{NE}, respectively. We observe in Figure 2.4 (b) the NAC of OMD_{NE} performance degrades after each popularity change. This is a limitation due to the multiplicative nature of OMD_{NE}. When the algorithm learns that a file, say it i, is not important, it can set $x_{t,i}$ arbitrarily close to 0. If, suddenly, this content becomes popular, then OMD_{NE} adapts slowly, due to its multiplicative nature—remember Eq. (2.23). This is shown in Figure 2.4 (e). We can overcome this limitation by requiring all state variables to be larger than some small $\delta > 0$; OMD_{NE} is then limited to $\mathcal{X}_\delta$, the δ interior of the capped simplex $\mathcal{X}$. More precisely, the δ interior of the capped simplex is defined as $\mathcal{X}_\delta \triangleq \mathcal{X} \cap [\delta, 1]^N$. In Figure 2.4 (f), we use $\delta = 10^{-4}$. This parameter indeed prevents the algorithm from driving the fractional allocations arbitrary close to 0, improving its adaptability. In Figure 2.4, we show the performance of FTPL [49]; we observe that this policy fails to adapt to popularity changes. Both our mirror descent algorithms outperform competitors (Figure 2.4 (h)).



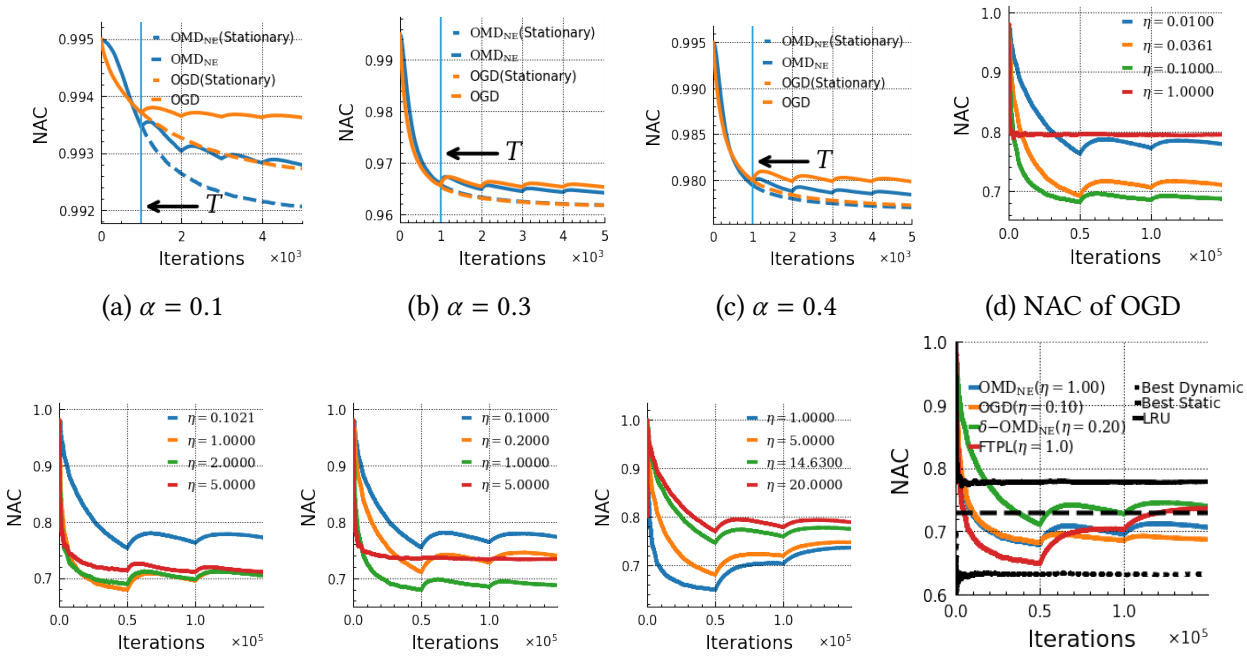
(a) $k = 25, \alpha = 0.1$ (b) $k = 125, \alpha = 0.1$ (c) $k = 250, \alpha = 0.1$

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

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

Figure 2.3: NAC of OMD_{NE} and OGD evaluated under different cache sizes and diversity regimes. We use traces *Batched Fixed Popularity* (1), (2), and (3) corresponding to different diversity regimes. Figures from left to right correspond to different cache sizes $k \in \{25, 125, 250\}$, while figures from top to bottom correspond to different exponent values $\alpha \in \{0.1, 0.2, 0.7\}$. OMD_{NE} outperforms OGD in the more diverse regimes and for small cache sizes, while OGD outperforms for large cache sizes and concentrated requests.

Akamai Trace. Figure 2.5 shows that the two gradient algorithms, OMD_{NE} and OGD, perform similarly over the Akamai Trace w.r.t. NMAC; OGD is slightly better in parts of the trace. Overall,



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

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

Figure 2.4: Subfigures (a)–(c) show NAC of OGD and $\mathrm{OMD_{NE}}$ evaluated under different diversity regimes when 10% of the files change popularity over time. We use traces *Partial Popularity Change* (1), (2), and (3) corresponding to the different diversity regimes. The diversity regimes are selected, such that, in the stationary setting (dashed line): (a) $\mathrm{OMD_{NE}}$ outperforms OGD, (b) $\mathrm{OMD_{NE}}$ has similar performance to OGD and (c) $\mathrm{OMD_{NE}}$ performs slightly worse than OGD. $\mathrm{OMD_{NE}}$ is consistently more robust to partial popularity changes than OGD. Subfigures (d)–(h) show the NAC of the different caching policies evaluated on the *Global Popularity Change* trace, where file popularity changes every 5×10^4 iterations. While OGD adapts seamlessly to popularity changes (d), multiplicative updates can make $\mathrm{OMD_{NE}}$ less reactive (e), unless $\mathrm{OMD_{NE}}$ is forced to operate over the δ-interior of $\mathcal{X}$ (f) ($\delta = 10^{-4}$). Finally, our mirror descent policies outperform competitors (h).

these algorithms consistently outperform LFU, W-LFU, LRU, and FTPL. Note that these caching policies process requests individually, while $\mathrm{OMD_{NE}}$ and OGD adapt slower, *freezing* their state for the entire batch size ($R = 5000$). Nevertheless, $\mathrm{OMD_{NE}}$ and OGD still perform better.

Randomized Rounding. Figure 2.6 shows the cumulative update cost for the online independent rounding, the online coupled rounding, and the online optimally-coupled rounding algorithms over the *Downscaled Global Popularity Change* trace. All the rounding algorithms exhibit the same service cost in expectation. The update cost of online coupled rounding and the online optimally-coupled rounding is small, in the order of the learning rate η; moreover, we observe that online optimally-coupled rounding yields lower update costs than the online coupled rounding. In contrast, online independent rounding incurs a significantly larger update cost.

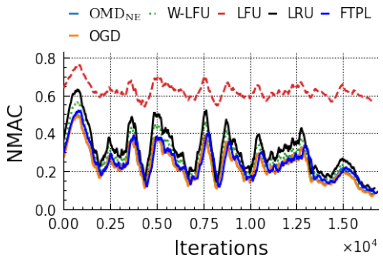
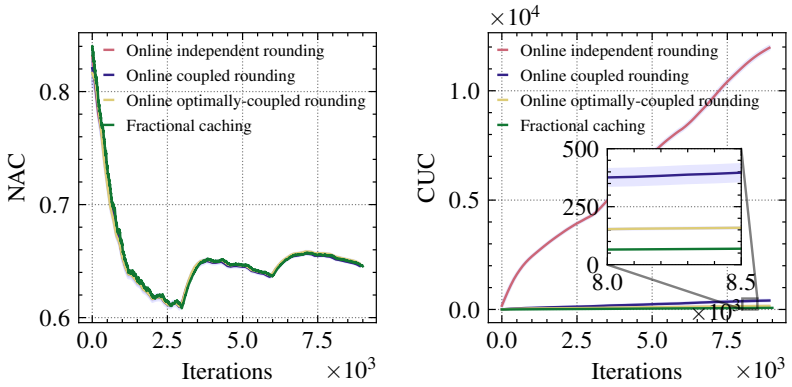


Figure 2.5: NMAC of the different caching policies evaluated on the *Akamai Trace*. OMD$_{NE}$ and OGD provide consistently the best performance compared to W-LFU, LFU, LRU, and FTPL. OGD performs slightly better than OMD in some parts of the trace.

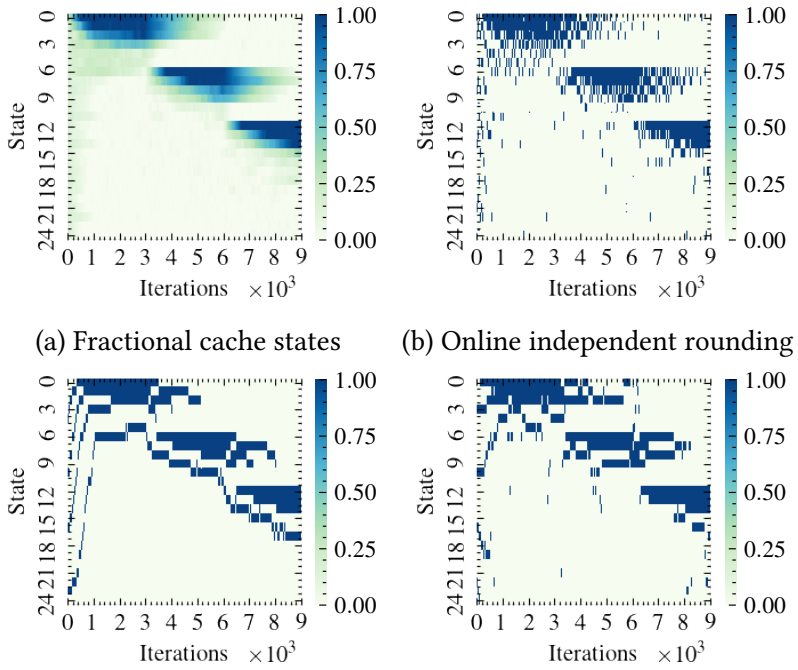


(a) Normalized average cost (b) Cumulative update cost

Figure 2.6: Costs associated with the rounded integral caching over the *Downscaled Global Popularity Change* trace. The normalized average costs shown in (a) are the same for the online independent rounding, the online coupled rounding and the online optimally-coupled rounding. The cumulative update cost of the online coupled rounding and online optimally-coupled rounding in (b) is of the same order as in the fractional setting, while the online independent rounding in (b) gives a much higher update cost. The reported values are averaged over 20 experiments, and the blue shaded area represents 10% scaling of the standard deviation.

Figure 2.7 shows the fractional and (rounded) integral cache states under *Downsampled Global Popularity Change* trace. Online independent rounding indeed leads to more frequent updates than online coupled rounding, while the latter maintains a more stable cache configuration by coupling the consecutive states and avoiding unnecessary updates.

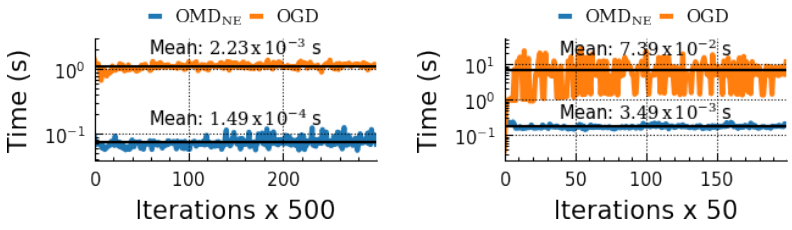
Computational Cost. Figure 2.8 shows the time taken by both policies OMD and OMD$_{NE}$ to perform 500 iterations over the *Fixed Popularity* trace (Figure 2.8 (a)), and the time taken to perform 50 iterations over the *Batched Fixed Popularity (2)* trace (Figure 2.8 (b)). We observe that OMD$_{NE}$ is at least 15 times faster in computing cache states on average.



(a) Fractional cache states (b) Online independent rounding

(c) Online coupled rounding (d) Online optimally-coupled rounding

Figure 2.7: Online rounding of fractional caching states. Visually, we see that the online independent rounding has more frequent updates than the online coupled rounding. This leads to large update costs. The online coupled rounding and the online optimally-coupled rounding prevents the cache to perform unnecessary updates.



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

Figure 2.8: Runtime per iteration of OMD_{NE} and OGD. Subfigure (a) shows the runtime per 500 iterations over the *Fixed Popularity* trace. Subfigure (b) shows the runtime per 50 iterations over the *Batched Fixed Popularity (2)* trace.

2.2.7 Conclusion

We study no-regret caching algorithms based on OMD with q-norm and neg-entropy mirror maps. We find that batch diversity impacts regret performance; a key finding is that OGD is optimal in low-diversity regimes, while OMD_{NE} is optimal under high diversity. With an appropriately designed

rounding scheme, our $O\left(\sqrt{T}\right)$ bound on the regret for general OMD algorithms extends to integral caches as well, despite the need to account for update costs in this setting.

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

2.3 Caching Networks

2.3.1 Related Work

Content allocation in networks of caches has been explored in the offline setting, presuming demand is known [39, 41, 63]. In particular, Shanmugam et al. [63] were the first to observe that caching can be formulated as a submodular maximization problem under matroid constraints and prove its NP-hardness. Dynamic caching policies have been mostly investigated under a stochastic request process. In particular, Ioannidis and Yeh [64] present (1) a projected gradient ascent (PGA) policy that attains $(1 - 1/e)$-approximation guarantee in expectation when requests are stationary and (2) a practical greedy path replication heuristic (GRD) that performs well in many cases, but comes without guarantee. Our work inherits all modeling assumptions on network operation and costs from [64], but differs from it (and all papers mentioned above) by considering requests that arrive adversarially. In our experiments, we compare our caching policy with PGA and GRD, that have no guarantees in the adversarial setting. We also prove that GRD in particular has linear regret (see Lemma 2.3.4).

Content placement at caches can be formulated as a submodular optimization problem [21, 63]. The offline problem is already NP-hard, but the greedy algorithm achieves $1/2$ approximation ratio [65]. Calinescu et al. [66] develop a $(1 - 1/e)$-approximation through the so called continuous greedy algorithm. The algorithm finds a maximum of the multilinear extension of the submodular objective using a Frank-Wolfe like gradient method. The solution is fractional and needs then to be rounded via pipage [67] or swap rounding [68]. Filmus and Ward [69] obtain the same approximation ratio without the need of a rounding procedure, by performing a non-oblivious local search starting from the solution of the usual greedy algorithm. These algorithms are suited for deterministic objective functions. Hassani et al. [70] study the problem of stochastic continuous submodular maximization and use stochastic gradient methods to reach a solution within a factor $1/2$ from the optimum. Mokhtari et al. [71] propose then the stochastic continuous greedy algorithm, which reduces the noise of gradient approximation by leveraging averaging technique. This algorithm closes the gap between stochastic and deterministic submodular problems achieving a $(1 - 1/e)$-approximation ratio.

There are two kinds of online submodular optimization problems. In the first one, a.k.a. competitive online setting, the elements in the ground set arrive one after the other, a setting considerably different from ours. The algorithm needs to decide whether to include revealed elements in the solution without knowing future arrivals. Gupta et al. [72] consider the case when this decision is irrevocable. They give a $O(\log r)$-competitive algorithm where r is the rank of matroid. Instead, Hubert Chan et al. [73] allow the algorithm also to remove elements from the current tentative solution. They propose a randomized 0.3178-competitive algorithm for partition matroids. In the second kind of online submodular optimization problems, objective functions are initially unknown and are progressively revealed over T rounds. This setting indeed corresponds to our problem, as our caching policy needs to decide the content allocation before seeing the requests. Streeter et al. [74] present an online greedy algorithm, combining the greedy algorithm with no-regret selection algorithm such as the hedge selector, operating under cardinality (rather than general matroid) constraints. Radlinski et al. [75] also propose an online algorithm by simulating the offline greedy algorithm,

using a separate instance of the multi-armed bandit algorithm for each step of the greedy algorithm, also for cardinality constraints. Chen et al. [76] convert the offline Frank-Wolfe variant/continuous greedy to a no-regret online algorithm, obtaining a sublinear $(1 - 1/e)$-regret. Chen et al. [77] use Stochastic Continuous Greedy [71] to achieve sublinear $(1 - 1/e)$-regret bound without projection and exact gradient. These algorithms however operate in a continuous domain, producing fractional solutions that require subsequent rounding steps; these rounding steps do not readily generalize to our distributed setting. Moreover, rounding issues are further exacerbated when needing to handle update costs in the regret.

Our work is based on the (centralized) TGOnline algorithm by Streeter et al. [78], which solves the so-called *online assignment problem*. In this problem, a fixed number of K slots is used to store items from distinct sets: that is, slot $k = 1, 2, \ldots, K$ can store items from a set P_k. The motivation comes, from, e.g., placing advertisements in K distinct positions on a website. Submodular reward functions arrive in an online fashion, and the goal of the online assignment problem is to produce assignments of items to slots that attain low regret. TGOnline, the algorithm proposed by Streeter et al., achieves sublinear $1 - 1/e$-regret in this setting. We depart from Streeter et al. by considering both objectives as well as constraints arising from the cache network design problem. We show that (1) when applied to this problem, TGOnline admits a distributed implementation, but also (2) we incorporate update costs, which are not considered by Streeter et al. A direct, naïve implementation of TGOnline to our setting would require communication between *all* caches at *every* request; in contrast, DistributedTGOnline restricts communication only among nodes on the request path. As an additional technical aside, we exploit the fact that an adaptation step within the TGOnline algorithm, namely, color shuffling, can in fact happen at a reduced frequency. The latter is imperative for bounding regret when cost updates are considered: without this adjustment, the TGOnline algorithm of Streeter et al. attains a linear regret when incorporating update costs.

2.3.2 System Description

Following the caching network model of Ioannidis and Yeh [21], we consider a network of caches that store items from a fixed catalog. Nodes in the network generate requests for these items, routed towards designated servers. However, intermediate nodes can cache items and, thereby, serve such requests early. We depart from Ioannidis and Yeh in assuming that request arrivals are adversarial, rather than stochastic. Notation used across the paper is summarized in Table 2.4.

Caching Network. We model a caching network as a directed graph $G(V, E)$ of n nodes. For convenience, we set $V = [n]$. Each edge e in the graph is represented by $e = (u, v) \in E \subseteq V \times V$. We assume G is symmetric, i.e., if $(u, v) \in E$, then $(v, u) \in E$. A fixed set of nodes, called designated servers, permanently store items of equal size. Formally, each item $i \in C$, where set C is the item catalog, is stored in designated servers $\mathcal{D}_i \subseteq V$.

Beyond designated servers, all other nodes in V are also capable of storing items. For each $v \in V$, let $c_v \in \mathbb{N}$ denote its storage capacity, i.e., the maximum number of items it can store. Let also $\mathcal{S}_v = \{(v, j)\}_{j=1}^{c_v}$ be v's set of storage slots; then, $s = (v, j) \in V \times [c_v]$ is the j-th storage slot on node v. We denote the set of storage slots in the whole network by $\mathcal{S}$, where $\mathcal{S} = \bigcup_{v \in V} \mathcal{S}_v$, and

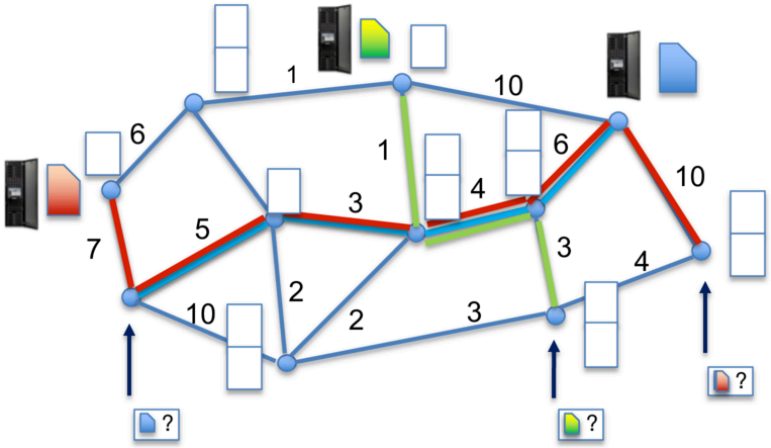


Figure 2.9: A general cache network, as proposed by Ioannidis and Yeh [21]. Designated servers store items in a catalog permanently. Requests arrive at arbitrary nodes in the network and follow predetermined paths towards these servers; responses incur costs indicated by weights on the edges. Intermediate nodes can serve as caches; the objective is to determine what items to store in each cache, to minimize the overall routing cost or, equivalently, maximize the caching gain.

	Notational Conventions
$[n]$	Set $\{1, \ldots, n\}$
$A + a$	Union $A \cup \{a\}$
	Cache Networks
$G(V, E)$	Network graph, with nodes V and edges E
C	The item catalog
c_v	Cache capacity at node $c \in V$
(v, j)	j-th storage slot on node v ($s = (v, j)$)
$\mathcal{S}$	The set of storage slots
$\mathcal{S}_v$	The set of storage slots in node v
$\mathcal{S}_p$	The set of storage slots in path p
$\mathcal{S}_{i,p}$	The set of storage slots in path p storing item i
A	The set of item allocations
$\mathcal{D}$	The set of feasible allocations
$\mathcal{R}_t$	The set of requests arriving at round t
$\bar{R}$	The upper bound of $\lvert\mathcal{R}_t\rvert$
$\overline{\lvert\mathcal{R}_t\rvert}$	Average number of requests per round
w_{uv}	The routing cost along edge (u, v)
$\bar{L}$	The upper bound of possible routing cost
f^t	Caching gain at round t
	Online Optimization
T	The rounds horizon
R_T	α-regret
$\mathcal{E}$	Hedge selector
$\boldsymbol{W}$	The weight vector maintained by hedge selector
$\boldsymbol{\ell}$	The reward vector fed to hedge selector
m_s	The active color of slot s
M	Number of colors
$\mathcal{I}$	Information collected by control message
w_v^p	The cumulative cost of edges upstream of v on path p
UC	Update costs
$\tilde{R}_T$	The extended α-regret considering update costs

Table 2.4: Notation Summary for Section 2.3

$|\mathcal{S}| = \sum_{v \in V} c_v$. We assume the slots in $\mathcal{S}$ are ordered lexicographically, i.e.,

$$(v, j) \prec (v', j') \text{ if and only if } v < v' \text{ or } v = v' \text{ and } j < j'. \tag{2.32}$$

We can describe content allocation as a set $A \subset \mathcal{S} \times C$, where $a = (s, i) \in A$ indicates that item $i \in C$ is stored in slot $s \in \mathcal{S}$. The set of feasible allocations is

$$\mathcal{D} = \{A \subseteq \mathcal{S} \times C : |A \cap (\{s\} \times C)| \leq 1, \forall s \in \mathcal{S}\}. \tag{2.33}$$

This ensures that each slot is occupied by at most one item; note that the cache capacity constraint at each node $v \in V$ is captured by the definition of $\mathcal{S}_v$.

Requests and Responses. A request $r = (i, p)$ is determined by (a) the item $i \in C$ requested, (b) the path p along which the request is forwarded. A path p is a sequence $\{p_k\}_{k=1}^{|p|}$ of adjacent nodes $p_k \in V$. As in Ioannidis and Yeh [21], we assume that paths are simple, i.e., they do not contain repeated nodes, and well-routed, i.e., they terminate at a node in $\mathcal{D}_i$. A request (i, p) is generated at node p_1 and follows the path p; when the request reaches a node storing item i, a response is generated. This response carries item i to query node p_1 following the reverse path. We assume that time is slotted, and requests arrive over a total of $T \in \mathbb{N}$ rounds. We denote by $\mathcal{R}$ the set of all possible requests in the system. At each round $t \in [!t]$, a set of requests $\mathcal{R}^t \subseteq \mathcal{R}$ arrive in the system. Requests in $\mathcal{R}^t$ can arrive in any order, and at any point in time within a round.[5] However, we assume that the total number of requests at each round is bounded by $\bar{R}$, i.e., $|\mathcal{R}^t| \leq \bar{R}$. Note that, when $\bar{R} = 1$, at most one request arrives per round.

Routing Costs. We assume request routing does not incur any cost, but response routing does. In particular let $w_{uv} \in \mathbb{R}_+$ denote the cost of routing the response along the edge $(u, v) \in E$.

Then, given an allocation $A \in \mathcal{S} \times C$, the cost of serving a request $(i, p) \in \mathcal{R}$ is:

$$C_{(i,p)}(A) = \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(A \cap \left\{ \bigcup_{k' \in [k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} = \emptyset \right). \tag{2.34}$$

Intuitively, Eq. (2.34) states that $C_{(i,p)}(A)$ includes $w_{p_{k+1}p_k}$, the cost of edge (p_{k+1}, p_k), only if no cache preceding p_{k+1} in path p stores item i. We denote by

$$\bar{L} = \max_{(i,p) \in \mathcal{R}} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \tag{2.35}$$

the maximum possible routing cost; note that this upper-bounds $C_{(i,p)}(A)$, for all $(i, p) \in \mathcal{R}$, $A \in \mathcal{S} \times C$.

The *caching gain* [21] of a request (i, p) due to caching at intermediate nodes is:

$$f_{(i,p)}(A) = C_{(i,p)}(\emptyset) - C_{(i,p)}(A) = \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(A \cap \left\{ \bigcup_{k' \in [k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} \neq \emptyset \right). \tag{2.36}$$

where $C_{(i,p)}(\emptyset)$ is the routing cost in the network when all caches are empty. The caching gain captures the cost reduction in the network due to caching allocation A.

Offline Problem. In the offline version of the cache gain maximization problem [21, 63], the request sequence $\{\mathcal{R}^t\}_{t=1}^{T}$ is assumed to be known in advance; the goal is then to determine a feasible allocation $A \in \mathcal{D}$ that maximizes the total caching gain. Formally, given an allocation $A \in \mathcal{S} \times C$, let

$$f^t(A) = \sum_{r \in \mathcal{R}^t} f_r(A), \tag{2.37}$$

[5] Our analysis readily extends to a multiset $\mathcal{R}^t$, whereby the same request is submitted multiple times within the same round. We restrict the exposition to sets for notational simplicity.

be the caching gain at round t. Then, the offline caching gain maximization problem amounts to:

$$\underset{A}{\text{maximize}} \quad f(A) = \sum_{t=1}^{T} f^t(A) = \sum_{t=1}^{T} \sum_{r \in \mathcal{R}^t} f_r(A), \tag{2.38a}$$

$$\text{subject to} \quad A \in \mathcal{D}. \tag{2.38b}$$

The following lemma implies this problem is approximable in polynomial time:

Lemma 2.3.1 ([21, 63]). *Function $f : \mathcal{S} \times C \to \mathbb{R}_+$ is non-decreasing and submodular. Moreover, the feasible region $\mathcal{D}$ is a partition matroid.*

Hence, Problem (2.38) is a submodular maximization problem under matroid constraints. It is known to be NP-hard [21, 63], and several approximation algorithms with polynomial time complexity exist. The classic greedy algorithm [79] produces a solution within $\frac{1}{2}$-approximation from the optimal. The so-called continuous greedy algorithm [66] further improves this ratio to $1 - \frac{1}{e}$. A different algorithm based on a convex relaxation of Problem (2.38) is presented in [21, 63]. The Tabular Greedy algorithm [78] also constructs a $1 - \frac{1}{e}$ approximate solution in poly-time. We describe it in details in Appendix 3, as both TGOnline [78] and our DistributedTGOnline build on it.

Online Problem. In the online setting, requests are not known in advance, and we seek algorithms that make caching decisions in an online fashion. In particular, at the beginning of round t, an online algorithm selects the current allocation $A^t \in \mathcal{D}$. Requests $\mathcal{R}^t \subset \mathcal{R}$ subsequently arrive, and the cache gain $f^t(A^t)$ is rewarded, where $f^t : \mathcal{S} \times C \to \mathbb{R}_+$ is given by Eq. (2.37).

As in standard online learning literature [19, 20], while choosing A^t, the network has no knowledge of the upcoming requests $\mathcal{R}^t$, but can rely on past history. Formally, we seek an online algorithm $\mathcal{A}$ that maps the history of past requests $\mathcal{H}^t = \{\mathcal{R}^1, .., \mathcal{R}^{t-1}\}$ to a new allocation, i.e., $A^t = \mathcal{A}(\mathcal{H}^t)$. In particular, we aim for an algorithm $\mathcal{A}$ with sublinear α-regret, given by

$$R_T = \mathbb{E}\left[\alpha \sum_{t=1}^{T} f^t(A^*) - \sum_{t=1}^{T} f^t(A^t)\right] = \alpha \sum_{t=1}^{T} f^t(A^*) - \mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t)\right], \tag{2.39}$$

where α is an approximation factor, and A^* is the optimal solution to (the offline) Problem (2.38). Note that the expectation is over the (possibly) randomized choices of the algorithm $\mathcal{A}$; we make no probabilistic assumptions on request arrivals $\{\mathcal{R}^t\}_{t=1}^T$, and wish to minimize regret in the adversarial setting, i.e., w.r.t. to the worst-case sequence $\{\mathcal{R}^t\}_{t=1}^T$. Put differently, our regret bounds will be against an arbitrarily powerful adversary, that can pick *any* sequence $\{\mathcal{R}^t\}_{t=1}^T$, as long as the total number of requests at each round is bounded by $\bar{R}$, i.e., $|\mathcal{R}^t| \leq \bar{R}$ for all $t = 1, \ldots, T$.

Several remarks are in order regarding Eq. (2.39). First, the definition of the regret in Eq. (2.39), which compares to a static offline solution, is classic. Several bandit settings, e.g., simple multi-armed bandits [75, 80, 81], contextual bandits [82–84], submodular bandits [74, 76, 85] and, of course, their applications to caching problems [13, 19, 49, 50], adopt this definition. In all these cases, the dynamic, adaptive algorithm is compared to a static policy that has full hindsight of the entire trace of actions. Nevertheless, as is customary in the context of online problems in which the offline

Algorithm 2.4 Hedge Selector $\mathcal{E}$

Require: Parameter $\epsilon \in \mathbb{R}_+$, action set C, horizon $T \in \mathbb{N}$.
procedure $\mathcal{E}$.INITIALIZE( )
  Set $W_i \leftarrow 1$ for all $i \in C$
end procedure
procedure $\mathcal{E}$.ARM( )
  return $i \in C$ with probability $p_i = \frac{W_i}{\sum_{j \in C} W_j}$
end procedure
procedure $\mathcal{E}$.FEEDBACK($\boldsymbol{\ell}$)
  Set $W_i \leftarrow W_i e^{\epsilon \ell_i}$ for all $i \in C$
end procedure

problem is NP-hard [76], the regret is not w.r.t. the optimal caching gain, but the gain obtained by an offline approximation algorithm. Second, from the point of view of bandits, we operate in the full-information feedback setting [20]: upon the arrival of requests $\mathcal{R}^t$, the entire function $f^t : \mathcal{S} \times C \to \mathbb{R}_+$ is revealed,[6] as the latter is fully determined by request set $\mathcal{R}^t$. Third, Eq. (2.39) captures the cost of serving requests, but not the cost of adaptation: changing an allocation from A^t to A^{t+1} changes cache content, which in turn may require the movement of items. Neglecting adaptation costs may be realistic if, e.g., adaptation happens in off-peak hours (e.g., the end of a round occurs at the end of a day), and does not come with the same latency requirements as serving requests in $\mathcal{R}^t$. Nevertheless, we revisit this issue, incorporating update costs in the regret, in Section 2.3.4. Finally, we stress that we seek online algorithms $\mathcal{A}$ that have sublinear regret but are also distributed: each cache v should be able to determine its own contents using past history it has observed, as well as some limited information it exchanges with other nodes.

2.3.3 Distributed Online Algorithm

We describe our distributed online algorithm, DISTRIBUTEDTGONLINE, in this section. We first give an overview of the algorithm and its adversarial guarantees; we then fill out missing implementation details.

2.3.3.1 Hedge Selector

Our construction uses as a building block the classic Hedge algorithm[7] for the expert advice problem [20, 78, 86]. This online algorithm selects an action from a finite set at the beginning of a round. At the conclusion of a round, the rewards of all actions are revealed; the algorithm accrues a reward based on the action initially selected, and adjusts its decision.

In our case the set of possible actions coincides with the catalog C, i.e., the algorithm selects an item $i^t \in C$ per round $t \in \mathbb{N}$. The hedge selector maintains a weight vector $\boldsymbol{W}^t = [W_i^t]_{i \in C} \in \mathbb{R}^{|C|}$, where weight W_i^t corresponds to action $i \in C$. The hedge selector supports two operations (see Algorithm 2.4. The first, $\mathcal{E}$.arm( ), selects an action from action set C. The second, $\mathcal{E}$.feedback($\boldsymbol{\ell}^t$),

[6] In contrast to the classic bandit feedback model, where only the reward $f^t(A^t)$ is revealed in each round t.
[7] This is also known as the multiplicative weight algorithm.

Algorithm 2.5 DistributedTGOnline

for each $s \in \mathcal{S}$ **do**
  for each $m \in [M]$ **do**
    $\mathcal{E}_{s,m}$.initialize().
  end for
  set $t_s = 1$, choose m_s uniformly at random from $[M]$
  $i_s \leftarrow \mathcal{E}_{s,m_s}$.arm() ▷ Sets $A \leftarrow \{(s, i_s)\}_{s \in \mathcal{S}}$
end for
for $t = 1, 2, ..., T$ **do** ▷ During round t:
  for each $r = (i, p) \in \mathcal{R}^t$ **do**
    Send request upstream over p until a hit occurs.
    Send response carrying i downstream, and incur routing costs.
    Send control message upstream over p to construct $\mathcal{I}$ and $\mathcal{W}$, given by (2.43).
    Send control message carrying $\mathcal{I}$ and $\mathcal{W}$ downstream over p, and do the following:
    for each $s \in \mathcal{S}_p$ **do**
      Use $\mathcal{I}$ and $\mathcal{W}$ to construct $\boldsymbol{\ell}(s, m_s) \in \mathbb{R}_+^{|C|}$ via (2.46).
      Call $\mathcal{E}_{s,m_s}$.feedback($\boldsymbol{\ell}(s, m_s)$)
    end for
  end for ▷ At the end of round t:
  for each $s \in \bigcup_{(i,p) \in \mathcal{R}^t} \mathcal{S}_p$ **do**
    if $t_s \bmod K = 0$ **then**
      Select m_s u.a.r. from $[M]$ ▷ Shuffle color m_s
  end if
  $i_s \leftarrow \mathcal{E}_{s,m_s}$.arm() ▷ Update allocation A at s
  $t_s \leftarrow t_s + 1$
end for
end for

ingests the reward vector $\boldsymbol{\ell}^t = [\ell_i^t]_{i \in C} \in \mathbb{R}_+^{|C|}$, where ℓ_i^t is the reward for choosing action $i \in C$ at round t, and adjusts action weights as described below. In each iteration t, the hedge selector alternates between (1) calling $i^t = \mathcal{E}$.arm( ), to produce action i^t, (2) receiving a reward vector $\boldsymbol{\ell}^t$, and using it via $\mathcal{E}$.feedback($\boldsymbol{\ell}^t$) to adjust its internal weight vector. In particular, $\mathcal{E}$.arm( ) selects action $i \in C$ with probability:

$$p_i^t = \frac{W_i^t}{\sum_{j \in C} W_j^t}, \tag{2.40}$$

i.e., proportionally to weight W_i^t. Moreover, when $\mathcal{E}$.feedback($\boldsymbol{\ell}^t$) is called, weights are updated via:

$$W_i^{t+1} = W_i^t e^{\epsilon \ell_i^t}, \quad \text{for all } i \in C, \tag{2.41}$$

where $\epsilon > 0$ is a constant. In a centralized setting where an adversary selects the vector of weights $\boldsymbol{\ell}^t$, the no-regret hedge selector attains an $O(\sqrt{T})$ regret for an appropriate choice of $\epsilon > 0$ (see Lemma 4.1 in Appendix 4). We use this as a building block in our construction below.

2.3.3.2 DistributedTGOnline Overview

To present the DistributedTGOnline algorithm, we first need to introduce the notion of "colors". The algorithm associates each storage slot $s = (v, j) \in \mathcal{S}$ with a "color" m_s from set $[M]$ of M

distinct values (the "color palette"). The online algorithm makes selections in the extended action space $\mathcal{S} \times C \times [M]$, choosing not only an item to place in a slot, but also how to color it.

This coloring is instrumental to attaining a $1 - \frac{1}{e}$-regret. In offline tabular greedy algorithm of Streeter et al. [78], which we present in Appendix 3, when $M = 1$, i.e., there is only one color, the algorithm reduces to simply the locally greedy algorithm (see Section 3.1 in [78]), achieving only a $\frac{1}{2}$ approximation ratio. When $M \to \infty$, the algorithm can intuitively be viewed as solving a continuous extension of the problem followed by a rounding (via the selection of the color instance), in the same spirit as the so-called ContinuousGreedy algorithm [66]. A finite size "color palette" represents a midpoint between these two extremes. We give more insight into this relationship between these two algorithms in Section 2.3.3.5.

In more detail, every storage slot $s \in \mathcal{S}$ maintains (1) the item $i_s \in C$ stored in it, (2) an active color $m_s \in [M]$ associated with this slot, and (3) M different no-regret hedge selectors $\{\mathcal{E}_{s,m}\}_{m=1}^{M}$, one for each color. All selectors $\{\mathcal{E}_{s,m}\}_{m=1}^{M}$ operate over action set C: that is, each such selector can have its arm "pulled" to select an item i to place in a slot. Though every slot s maintains M different selectors, one for each color, it only uses one at a time. The active colors $\{m_s\}_{s\in\mathcal{S}}$ are initially selected u.a.r. from $[M]$, and remain active continuously for a total K pull/feedback interactions, where $K \in \mathbb{N}$; at that point, m_s is refreshed, selected again u.a.r., bringing another selector into play. All in all, the algorithm proceeds as follows during round t.

1. When a request $(i, p) \in \mathcal{R}^t$ is generated, it is propagated over the path p until a hit occurs; a response is then backpropagated over the path p, carrying i, and incurring a routing cost.

2. At the same time, an additional control message is generated and propagated upstream over the entire path p. Moving upstream, it collects information from slots it traverses.

3. After reaching designated server at the end of the path, the control message is backpropagated over the path p in the reverse direction. Every time it traverses a node $v \in p$, storage slot $s \in \mathcal{S}_v$ fetches information stored in the control message and computes a reward vector $\boldsymbol{\ell}^t(s, m_s)$. This is then fed to the active hedge selector via $\mathcal{E}_{s,m_s}$.feedback($\boldsymbol{\ell}^t(s, m_s)$).

4. At the end of the round, we check if the arm of $\mathcal{E}_{s,m_s}$ has been pulled for a total of K times under active color m_s; if so, a new color m_s is selected u.a.r. from $[M]$.

5. Irrespective of whether m_s changes or not, at the end of the round, each slot updates its contents via the current active hedge selector $\mathcal{E}_{s,m_s}$, by calling operation $\mathcal{E}_{s,m_s}$.arm() to choose a new item i_s to place in s.

We define the control messages exchanged, the information they carry, and the reward vectors fed to hedge selectors in Section 2.3.3.3. Only slots in a request's path need to exchange messages, provide feedback to their selectors, and (possibly) update their contents at the end of a round. Moreover, messages exchanged are of size $O(|p|)$. We allow $K \geq T$; in this case, colors are selected u.a.r. only once, at the beginning of the execution of the algorithm, and remain constant across all rounds.[8] Finally, note that updating cache contents at the end of a round does not affect the incurred cost; we remove this assumption in Section 2.3.4.

[8] In such a case, selectors corresponding to inactive colors need not be maintained, and can be discarded.

Our first main result is that DistributedTGOnline has a $(1-1/e)$-regret that grows as $O(\sqrt{T})$:

Theorem 2.3.2. *Consider the sequence of allocations $\{A^t\}_{t=1}^T$ produced by the DistributedTGOnline algorithm using hedge selectors determined by Algorithm 2.4, with $\epsilon = \frac{1}{\bar{L}}\sqrt{\frac{\log|C|}{T}}$. Then, for all $T \geq \log|C|$ and all $K \geq 1$:*

$$\mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t)\right] \geq \beta(|\mathcal{S}|, M) \cdot \max_{A\in\mathcal{D}}\left\{\sum_{t=1}^{T} f^t(A)\right\} - 2\bar{R}\bar{L}|\mathcal{S}|M\sqrt{T\log|C|}, \tag{2.42}$$

where $\beta(|\mathcal{S}|, M) = 1-(1-\frac{1}{M})^M - \binom{|\mathcal{S}|}{2}M^{-1}$.

The main intuition behind the proof is to view DistributedTGOnline as a version of the offline TabularGreedy that, instead of greedily selecting a single item $i_{s,m} \in C$ per step, *it greedily selects an entire item vector $\boldsymbol{i}_{s,m} \in C^T$ across all rounds*, where T is the number of rounds. To cast the proof in this context, we define new objective functions f and F whose domain is over decisions across T rounds, as opposed to the original per time-slot functions (whose domain is only over one round). Due to the properties of the no-regret hedge selector, and the formal guarantees of the offline case (c.f. Theorem 3.1), these new objectives attain $1-\frac{1}{e}$ bound shown in Eq. (3.37), yielding the bound on the regret. The detailed proof of this theorem is provided in Appendix 5. Note that for M large enough (at least $\Omega(|\mathcal{S}|^2)$), quantity $\beta(|\mathcal{S}|, M)$ can be made arbitrarily close to $1-1/e$. Hence, Theorem 2.3.2 has the following immediate corollary:

Corollary 2.3.3. *For any $\delta > 0$, there exists an $M = \Theta(\frac{|\mathcal{S}|^2}{\delta})$ such that the expected $(1-\frac{1}{e}-\delta)$-regret of DistributedTGOnline is $R_T \leq \frac{2\bar{R}\bar{L}|\mathcal{S}|^3}{\delta}\sqrt{T\log|C|}$.*

DistributedTGOnline is a distributed implementation of the (centralized) online tabular greedy algorithm of Streeter et al. [78], which is itself an online implementation of the so-called tabular greedy algorithm [78], which we present in Appendix 3. We depart however from [78] in several ways. First, the analysis by Streeter et al. requires that feedback is provided at every slot $s \in \mathcal{S}$. We amend this assumption, as only nodes along a path need to update their selectors/allocations in our setting. Second, we show that feedback provided to arms can be computed in a distributed fashion, using only messages along the path p, as described below in Section 2.3.3.3. These two facts together ensure that DistributedTGOnline is indeed distributed. Finally, the analysis of Streeter et al. assumes that colors are shuffled at every round, i.e., applies only to $K = 1$. We extend this to arbitrary $K \geq 1$. As we discuss in Section 2.3.4, this is instrumental to bounding regret when accounting for update costs; the later becomes $\Theta(T)$ when $K = 1$.

2.3.3.3 Control Messages, Information Exchanged, and Rewards

The algorithm is summarized in Algorithm 2.5. We describe here the control messages, information exchanged, and reward vectors fed to hedge selectors during the generation of a request $r = (i, p) \in \mathcal{R}^t$. Let $\mathcal{S}_p = \bigcup_{v\in p}\mathcal{S}_v$ be the set of all slots in nodes in p. Let also $\mathcal{S}_{i,p} = \{s \in \mathcal{S}_p : (s, i) \in A^t\} \subseteq \mathcal{S}_p$ be the slots in path p that store the requested item $i \in C$. The control message propagated upstream

towards the designated server collects both the (a) color of every slot in $\mathcal{S}_{i,p}$ and (b) the upstream cost at each node $v \in p$. Formally, the information collected by the upstream control message is

$$\mathcal{I} = \{(s, m_s)\}_{s \in \mathcal{S}_{i,p}}, \quad \text{and} \quad \mathcal{W} = \{w_v^p\}_{v \in p}, \tag{2.43}$$

where w_v^p is the cumulative cost of edges upstream of v on path p, i.e.,

$$w_v^p = \sum_{k=k_p(v)}^{|p|-1} w_{p_{k+1}p_k}, \tag{2.44}$$

where $k_p(v) = \{1, 2, ..., |p|\}$ is position of v in p, i.e., $k_p(v) = k$ if $p_k = v$. Note that both $|\mathcal{I}|$ and $|\mathcal{W}|$ are $O(|p|)$.

Upon reaching the end of the path, a message carrying this collected information ($\mathcal{I}$,$\mathcal{W}$) is sent downstream to every node in the path. For each slot $s \in \mathcal{S}_p$, let:

$$\mathcal{S}_{\preceq s} = \{s' \in \mathcal{S}_{i,p} : m_{s'} < m_s \text{ or } m_{s'} = m_s, s' \prec s\} \tag{2.45}$$

be the slots in the path p that store i and either (a) are colored with a color smaller than m_s, or (b) are colored with m_s, and precede s in the ordering given by Eq. (2.32). Note that $\mathcal{S}_{\preceq s}$ can be computed having access to $\mathcal{I}$. Then, the reward vector $\boldsymbol{\ell}^r(s, m_s) \in \mathbb{R}_+^{|C|}$ fed to hedge selector $\mathcal{E}_{s,m_s}$ at slot $s \in \mathcal{S}_p$ comprises the following coordinates:

$$\ell_{i'}^r(s, m_s) = \begin{cases} \max_{(v',j') \in \mathcal{S}_{\preceq s} + s} w_{v'}^p, & \text{if } i' = i \\ \max_{(v',j') \in \mathcal{S}_{\preceq s}} w_{v'}^p, & o.w. \end{cases} \quad \text{for all } i' \in C, \tag{2.46}$$

which captures the marginal gain of adding an element to the allocation at time t, assuming that the latter is constructed adding one slot, item, and color selection at a time (following an ordering w.r.t. colors first and slots second). This is stated formally in Lemma 5.1 in Appendix 5. Note again that all these quantities can be computed at every $s \in \mathcal{S}_p$ having access to $\mathcal{I}$ and $\mathcal{W}$.[9]

2.3.3.4 A Negative Result

We conclude this section with a negative result, further highlighting the importance of Theorem 2.3.2. DISTRIBUTEDTGONLINE comes with adversarial guarantees; our experiments in Section 2.3.6 indicate that also works well in practice. Nevertheless, for several topologies we explore, we observe that a heuristic proposed by Ioannidis and Yeh [21], termed GREEDYPATHREPLICATION, performs as well or slightly better than DISTRIBUTEDTGONLINE in terms of time-average caching gain. As we observed this in both stationary as well as non-stationary/transient request arrivals, this motivated us to investigate further whether this algorithm also comes with any adversarial guarantees.

Our conclusion is that, despite the good empirical performance in certain settings, GREEDYPATHREPLICATION does not enjoy such guarantees. In fact, the following negative result holds:

[9] In practice, trading communication for space, the full $\mathcal{W}$ can also be just computed once, at the first time request (i, p) is generated, Each v can store their own w_v^p for paths that traverse them, and only weights of nodes storing i need to be included in $\mathcal{W}$ in subsequent requests.

Lemma 2.3.4. *Consider the online policy* GreedyPathReplication *due to Ioannidis and Yeh [21], which is parametererized by $\beta > 0$. Then, for all $\alpha \in (0, 1]$ and all $\beta > 0$,* GreedyPathReplication *has $\Omega(T)$ α-regret.*

We prove this lemma in Appendix 7. The adversarial counterexample we construct in the proof informed our design of experiments for which GreedyPathReplication (denoted by GRD in Section 2.3.6) performs poorly, *attaining a zero caching gain throughout the entire algorithm's execution* (see Figure 2.11-2.13). The same examples proved hard for several other greedy/myopic policies, even though DistributedTGOnline performed close to the offline solution in these settings. Both Lemma 2.3.4, as well as the experimental results we present in Section 2.3.6, indicate the importance of Theorem 2.3.2 and, in particular, obtaining a universal bound on the regret against any adversarially selected sequence of requests.

2.3.3.5 Color Palette

To provide further intuition behind the "color palette" induced by M and the role it plays in our algorithm, we describe here in more detail how it relates to the ContinuousGreedy algorithm, the standard algorithm for maximizing submodular functions subject to a matroid constraint [66].

Intuitively, given a submodular function $f : \Omega \to \mathbb{R}_+$ and a matroid constraint set $\mathcal{D}$, the ContinuousGreedy algorithm maximizes the *multilinear relaxation* of objective, given by

$$\tilde{f}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{x}}[f(A)] = \sum_{A \subseteq \Omega} f(A) \prod_{i \in A} x_i \prod_{i \notin A} (1 - x_i). \tag{2.47}$$

That is, the multilinear extension $\tilde{f} : [0, 1]^{|\Omega|} \to \mathbb{R}_+$ is the expected value of the objective assuming that each element in A is sampled independently, with probability given by $x_i \in [0, 1]$, $i \in \Omega$. ContinuousGreedy first obtains a fractional solution in the matroid polytope of $\mathcal{D}$. This is constructed by incrementally growing the probability distribution parameters $\boldsymbol{x}$, starting from $\boldsymbol{x} = \boldsymbol{0}$, with a variant of the so-called Frank-Wolfe algorithm. Then, ContinuousGreedy rounds the resulting fractional solution via, e.g., pipage rounding [67] or swap rounding [68], mapping it to set solutions. We refer the reader to Calinescu et al. [66] for a more detailed description of the algorithm.

In comparison, the color palette also creates a distribution across item selections, as implied by the M colors. When the number of colors M approaches infinity, the selection process of DistributedTGOnline also "grows" this probability distribution (starting, again, from an empty support) infinitesimally, by an increment inversely proportional to M (see also the offline version TabularGreedy in Appendix 3). As M goes to infinity, this recovers the $1 - 1/e$ approximation guarantee of continuous greedy [78, 87]. For any finite M, the guarantee provided by Theorem 2.3.2 lies in between this guarantee and $\frac{1}{2}$ approximation of the locally greedy algorithm ($M = 1$).

2.3.4 Update Costs

We now turn our attention to incorporating update costs in our analysis. We denote by $w'_{s,i}$ the cost of fetching item i at storage slot s at the end of a round. Then, the total update cost of changing an

Algorithm 2.6 Coupled Hedge Selector

Require: Parameter $\epsilon \in \mathbb{R}_+$, action set C, horizon $T \in \mathbb{N}$.

procedure $\mathcal{E}$.INITIALIZE( )
  Set $W_i \leftarrow 1$ and
  $p_i^{\mathsf{old}} \leftarrow \frac{1}{|C|}$ for all $i \in C$
  Pick i^{old} u.a.r. from C
end procedure
procedure $\mathcal{E}$.FEEDBACK($\boldsymbol{\ell}$)
  Set $W_i \leftarrow W_i e^{\epsilon \ell_i}$ for all $i \in C$
end procedure
procedure $\mathcal{E}$.CORRELATED_ARM( )
  Set $p_i^{\mathsf{new}} \leftarrow \frac{W_i}{\sum_{j \in C} W_j}$ for all $i \in C$
  Pick $i^{\mathsf{new}} \leftarrow$ coupled_movement ($\boldsymbol{p}^{\mathsf{old}}$,$\boldsymbol{p}^{\mathsf{new}}$,$i^{\mathsf{old}}$) ▷ Marginal is $\boldsymbol{p}^{\mathsf{new}}$
  $\boldsymbol{p}^{\mathsf{old}} \leftarrow \boldsymbol{p}^{\mathsf{new}}$
  $i^{\mathsf{old}} \leftarrow i^{\mathsf{new}}$
  return i^{new}
end procedure
procedure COUPLED_MOVEMENT($\boldsymbol{p}^{\mathsf{old}}$,$\boldsymbol{p}^{\mathsf{new}}$,$i^{\mathsf{old}}$)
  Compute $I = \{i \in C : p_i^{\mathsf{new}} - p_i^{\mathsf{old}} > 0\}$.
  Set $m_i \leftarrow |p_i^{\mathsf{new}} - p_i^{\mathsf{old}}|$, for all $i \in C$
  Compute a feasible flow $[\delta_{i,j}]_{(i,j) \in C^2}$, where $\sum_{j \in I} \delta_{i,j} = m_i$ for $i \in C \setminus I$, and $\sum_{i \in C \setminus I} \delta_{i,j} = m_j$ for $j \in I$, to transport $\sum_{i \in C \setminus I} m_i$ mass from the
  components in I to the components in $C \setminus I$.
  $\boldsymbol{p} \leftarrow \boldsymbol{p}^{\mathsf{old}}$.
  $i^{\mathsf{temp}} \leftarrow i^{\mathsf{old}}$
  for $i \in C \setminus I$ **do**
    for $j \in I$ **do**
      $\boldsymbol{p}, i^{\mathsf{temp}} \leftarrow$ elementary_δmovement($\boldsymbol{p}, i^{\mathsf{temp}}, \delta_{i,j}, i, j$)
    end for
  end for
  return i^{temp}
end procedure
procedure ELEMENTARY_δMOVEMENT($\boldsymbol{p}, i^{\mathsf{temp}}, \delta, i, j$)
  if $i^{\mathsf{temp}} = i$ **then**
    $i^{\mathsf{temp}}, \boldsymbol{p} \leftarrow \begin{cases} i, \boldsymbol{p} + \delta(\boldsymbol{e}_j - \boldsymbol{e}_i) & \text{w.p. } \frac{p_i - \delta}{p_i} \\ j, \boldsymbol{p} + \delta(\boldsymbol{e}_j - \boldsymbol{e}_i) & \text{w.p. } \frac{\delta}{p_i} \end{cases}$
  else
    $i^{\mathsf{temp}}, \boldsymbol{p} \leftarrow i^{\mathsf{temp}}, \boldsymbol{p} + \delta(\boldsymbol{e}_j - \boldsymbol{e}_i)$
  end if
  return $\boldsymbol{p}, i^{\mathsf{temp}}$
end procedure

allocation A^t to A^{t+1} at the end of round t is given by:

$$\mathrm{UC}(A^t, A^{t+1}) = \sum_{(s,i) \in A^{t+1} \setminus A^t} w'_{s,i}. \tag{2.48}$$

When such update costs exist, we need to account for them when adapting cache allocations. We thus incorporate them in the α-regret as follows:

$$\tilde{R}_T = \alpha \sum_{t=1}^{T} f^t(A^*) - \mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t) - \sum_{t=1}^{T-1} \mathrm{UC}(A^t, A^{t+1})\right] = R_T + \mathbb{E}\left[\sum_{t=1}^{T-1} \mathrm{UC}(A^t, A^{t+1})\right]. \tag{2.49}$$

Note that, in this formulation, the optimal offline policy A^* has no update cost, as it is static. We refer to $\tilde{R}_T$ as the *extended α-regret* of an algorithm. This extension corresponds to adding the expected update cost incurred by a policy $\mathcal{A}$ to its α-regret.

Unfortunately, the update costs of DistributedTGOnline (and, hence, its extended regret) grow as $\Theta(T)$ in expectation. In particular, the following lemma, proved in Appendix 6, holds:

Lemma 2.3.5. *When DistributedTGOnline is parametrized with the hedge selector in Algorithm 2.4, it incurs an expected update cost of $\Omega(T)$ for any choice of $\epsilon \in \mathbb{R}_+$.*

Nevertheless, the extended regret can be reduced to $O(\sqrt{T})$ by appropriately modifying Algorithm 2.4, the no-regret-hedge selector used in slots $\mathcal{E}_{s,m}$. In particular, the linear growth in the regret is due to the independent sampling of cache contents within each round. Coupling this selection with the presently selected content can significantly reduce the update costs. More specifically, instead of selecting items independently across rounds, the new item can be selected from a probability distribution that depends on the current allocation in the cache. This conditional distribution can be design in a way that the (marginal) probability of the new item is the same as in Algorithm 1, thereby yielding the same expected caching gain. On the other hand, coupling can bias the selection towards items already existing in the cache. This reduces update costs, especially when marginal distributions change slowly.

The *coupled hedge selector* described in Algorithm 2.6 accomplishes exactly this. As seen in line 9 of Algorithm 2.6, pulling the arm of the hedge selector is dependent on (a) the previously taken action/selected item and (b) the change in the distribution implied by weights W_i, $i \in C$. We give more intuition as to how this is accomplished in Appendix 6. In short, the coupled hedge selector solves a minimum-cost flow problem via an iterative algorithm. The minimum-cost flow problem models a so-called optimal transport or earth mover distance problem [59] from $\boldsymbol{p}^t$ to $\boldsymbol{p}^{t+1}$, the distributions over catalog C at rounds t and $t+1$, respectively, The resulting solution comprises conditional distributions for "jumps" among elements in C, which are used to determine the next item selection using the current choice: by being solutions of the minimum-cost flow problem, they incur small update cost, while ensuring that the posterior distribution after the "jump" is indeed $\boldsymbol{p}^{t+1}$.

Our second main result establishes that using this hedge selector instead of Algorithm 2.4 in DistributedTGOnline ensures that the extended regret grows as $O(\sqrt{T})$.

Theorem 2.3.6. *Consider DistributedTGOnline with hedge selectors $\mathcal{E}_{s,m}$ implemented via Algorithm 2.6 parameterized with $\epsilon = \frac{1}{L}\sqrt{\frac{\log|C|}{T}}$. Assume also that colors are updated every $K = \Omega(\sqrt{T})$ rounds. Then, the standard regret R_T is again bounded as in Corollary 2.3.3. Moreover, the update cost*

of DISTRIBUTEDTGONLINE *is such that*

$$\mathbb{E}\left[\sum_{t=1}^{T-1} \mathrm{UC}(A^t, A^{t+1})\right] = O(\sqrt{T}), \tag{2.50}$$

and, as a consequence, the extended regret is $\tilde{R}_T = O(\sqrt{T})$.

The proof can be found in Appendix 8. We note that the theorem requires that $K = \Omega(\sqrt{T})$, i.e., that colors are shuffled infrequently. Whenever a color changes, DISTRIBUTEDTGONLINE, the corresponding change in the active hedge selector can lead to sampling vastly different allocations than the current ones. Consequently, such changes can give rise to large update costs whenever a color is changed. The requirement that $K = \Omega(\sqrt{T})$ allows for some frequency in changes, but not, e.g., at a constant number of rounds (as, e.g., in Streeter et al. [74], where $K = 1$). Put differently, Theorem 2.3.6 allows for experiencing momentarily large update costs, as long as we do not surpass a budget of $O(\sqrt{T})$ color updates overall. We note that the couple hedge selector in Algorithm 2.6 has the same time complexity as the hedge selector given in Algorithm 2.4, which is $O(|C|)$ per iteration.

2.3.5 Extensions

Jointly Optimizing Caching and Routing. Our model can be extended to consider joint optimization of both cache and routing decisions, following an extended model by Ioannidis and Yeh [64]. Under appropriate variable transformations, the new objective is still submodular over the extended decision space. Moreover, the constraints over the routing decisions can similarly be cast as assignments to slots. Together, these allow us to directly apply DISTRIBUTEDTGONLINE and attain the same $1 - 1/e$ approximation guarantee. We describe this extension in detail in Appendix 10.

Anytime Regret Guarantee. Our algorithm assumes prior knowledge of the time horizon T; this is necessary to set parameter ϵ in Theorem 2.3.2. Nevertheless, we can use the well-known *doubling trick* [88] to obtain anytime regret guarantees. In short, the algorithm starts from a short horizon; at the conclusion of the horizon, the algorithm restarts, this time doubling the horizon. We show in Appendix 11 that, by using this doubling trick, DISTRIBUTEDTGONLINE indeed attains an $O(\sqrt{T})$ regret, without requiring prior knowledge of T. This remains true when update costs are also considered in the regret.

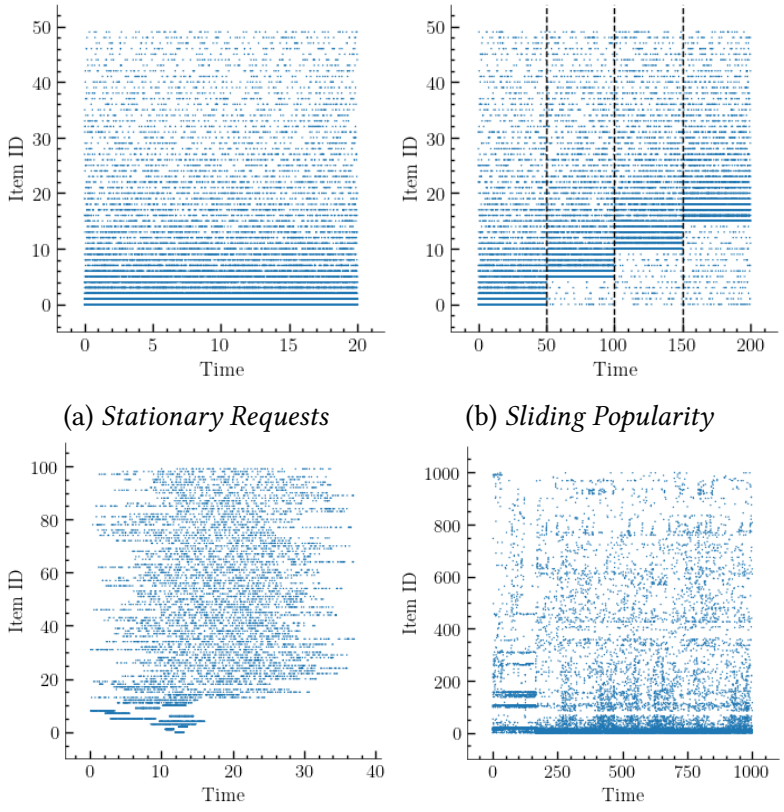
2.3.6 Experiments

2.3.6.1 Experimental Setting

Networks. We use four synthetic graphs, namely, Erdős-Rényi (`ER`), balanced tree (`BT`), hypercube (`HC`), and a path (`path`), and three backbone network topologies: [89] Deutsche Telekom (`dtelekom`), `GEANT`, `Abilene`. The parameters of different topologies are shown in Tab. 2.5. For the first five topologies (`ER`–`GEANT`), weights w for each edge is uniformly distributed between 1–100. Each item $i \in C$ is permanently stored in a designated servers $\mathcal{D}_i$ which is designated uniformly at random (u.a.r.) from V. All nodes in V are also has c_v storage space, which is u.a.r. sampled between 1 to

topologies	$\|V\|$	$\|E\|$	$\|Q\|$	c_v	w	$f_{stationary}$	$f_{sliding}$	f_{SN}	f_{CDN}
ER	100	1042	5	1-5	1-100	1569.93	1123.76	72.03	976.65
BT	341	680	5	1-5	1-100	5547.89	3211.63	220.58	3504.58
HC	128	896	5	1-5	1-100	2453.66	2045.69	152.38	1895.41
dtelekom	68	546	5	1-5	1-100	969.55	772.53	49.55	1005.24
GEANT	22	66	5	1-5	1-100	1564.02	981.60	66.50	1185.56
abilene	9	26	2	0-5	100	81.21	81.21	41.39	-
path	4	6	1	0-5	100	20.00	20.00	10.00	-

Table 2.5: Graph Topologies and Experiment Parameters. We indicate the number of nodes in each graph ($|V|$), the number of edges ($|E|$), the number of query nodes ($|Q|$), and the ranges of cache capacities c_v and edge weights w. In the last four columns we also report $f_{stationary}$, $f_{sliding}$, f_{SN}, and f_{CDN}: which are the caching gain attained by OFL with *Stationary Request*, *Sliding Popularity*, *Shot Noise*, and *CDN* trace, respectively.



(a) *Stationary Requests* (b) *Sliding Popularity*

(c) *Shot Noise* (d) *CDN* trace

Figure 2.10: Request traces for different scenarios. Each dot indicates an access to an item in the catalog; items are ordered in an overall increasing popularity from top to bottom. In *Sliding Popularity*, popularity changes at fixed time intervals, through a cyclic shift (most popular items become least popular). In *Shot Noise*, each item remains active for a limited lifetime.

5. Requests are generated from nodes Q u.a.r. selected from V. Given the source $p_1 \in Q$ and the destination $p_{|p|} \in \mathcal{D}_i$ of the request r, path p is the shortest path between them. For the remaining two topologies (abiline and path), we select parameters in a way that is described in Appendix 9.

Demand. We consider three different types of synthetic request generation processes, and one trace-driven. In the *Stationary Requests* scenario (see Figure 2.10(a), each $r = (i, p) \in \mathcal{R}$ is associated with an exogenous Poisson process with rate 1.0, and i is chosen from C via a power law distribution with exponent 1.2. In the *Sliding Popularity* scenario, requests are again Poisson with a different exponent 0.6, and popularities of items are periodically reshuffled (see Figure 2.10(b)). In the *Shot Noise* scenario, each item is assigned a lifetime, during which it is requested according to a Poisson process; upon expiration, the item is retired (see Figure 2.10(c)). In the *CDN* scenario (see Figure 2.10(d)), we generate requests using a real-life trace from a CDN provider. The trace spans 1 week, and we extract from it about 10×10^5 requests for the $N = 10^3$ most popular files.

For abiline and path, we replace the Poisson arrivals on the three synthetic traces (*Stationary Requests*, *Sliding Popularity*, *Shot Noise*) with requests generated in a round-robin manner, as described in Appendix 9.2. This is designed in an adversarial fashion, that leads to poor performance for greedy/myopic algorithms.

Algorithms. We implement the following online algorithms:

- Path replication with least recently used (LRU), least frequently used (LFU), first-in-first-out (FIFO), and random-replacement (RR) eviction policies: In all these algorithms, when responses are back-propagated over the reverse path, all nodes they encounter store requested item, evicting items according to one of the aforementioned policies.
- Projected gradient ascent (PGA): This is the distributed, adaptive algorithm oringinally proposed by Ioannidis and Yeh [21]. This attains an $(1-1/e)$-approximation guarantee in expectation when requests are stationary, but comes with no guarantee against adversarial requests. Similar to our setting, it also operated in rounds, at the end of which contents are shuffled.
- Greedy path replication (GRD): This is a heuristic, also proposed by Ioannidis and Yeh [21]. Though it performs well in many cases, we prove in Appendix 6 that its $(1 - 1/e)$-regret is $\Omega(T)$ in the worst case.
- DISTRIBUTEDTGONLINE (TBGRD): this is our proposed algorithm. We implement it with both independent hedge selector shown in Algorithm 2.4 and coupled hedge selector in Algorithm 2.6.

Unless indicated otherwise, we set $\epsilon = 0.005$, number of colors $M = 100$, $\bar{R} = 1$, and $K = T$ for TBGRD. For PGA and GRD, we explore parameters γ and β range from 0.005-5 and 0.005-1 individually, and pick the optimal values. In experiments where we do not measure update costs, we implement TBGRD with the independent hedge selector (Algorithm 2.4), as it yields the same performance as the coupled hedge selector (Algorithm 2.6) in expectation (see also Figure 2.17(a) and 2.17(b)).

Finally, we also implement the offline algorithm (OFL) by Ioannidis and Yeh [21], and use the resulting $(1 - 1/e)$-approximate solution as baseline (see metrics below).

Performance Metrics. We use normalized time-average cache gain (TACG) as the metric to measure the performance of different algorithms. More specifically, leveraging PASTA [90], we measure $f^{t_s}(A^{t_s})$ at epochs t_s generated by a Poisson process with rate 1.0 for 5000 time slots, and average

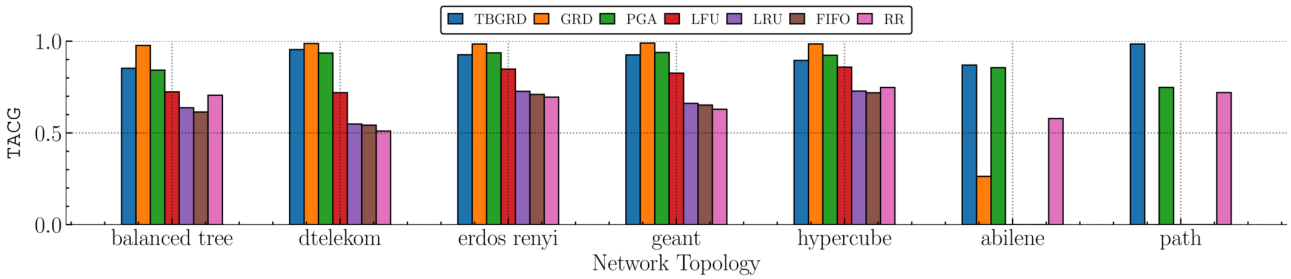


Figure 2.11: TACG of different algorithms over different topologies with *Stationary Requests*. The total simulation time is 1000 time units. TBGRD, GRD, and PGA perform well in comparison to path replication policies. However, GRD and other myopic strategies attain zero TACG over abilene and path, the round-robin scenarios. In comparison, TBGRD and PGA still perform well.

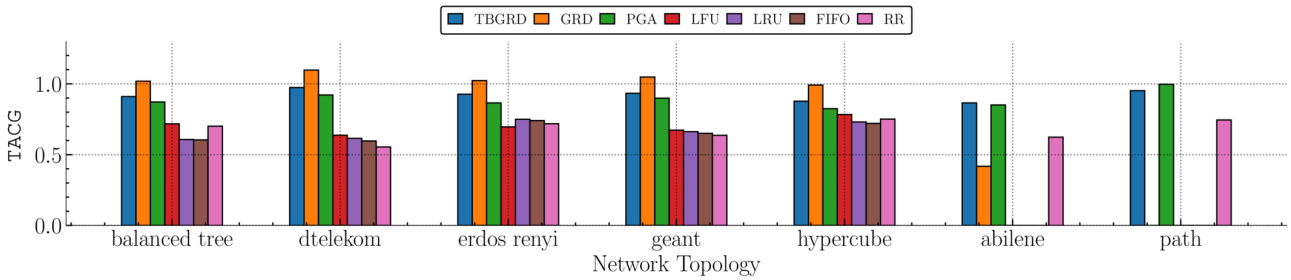


Figure 2.12: TACG of different algorithms over different topologies with *Sliding Popularity*. The total simulation time is 1000 time units. TBGRD, GRD, and PGA again outperform path replication algorithms; GRD sometimes even outperforms the (static) OFL solution, attaining a normalized TACG larger than one. However, GRD and several path replication algorithms again fail catastrophically over the abilene and path scenarios, while TBGRD and PGA again attain a normalized TACG close to one.

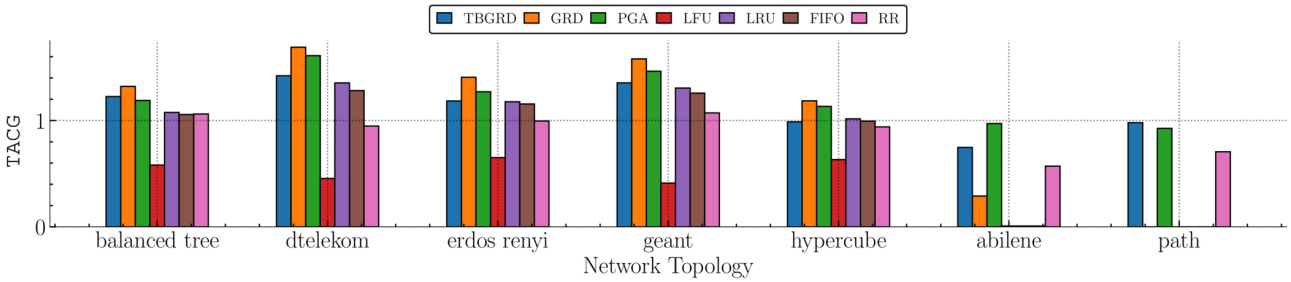


Figure 2.13: TACG of different algorithms over different topologies with *Shot Noise*. We again observe TBGRD, GRD, and PGA perform well in this non-stationary request arrival setting. Moreover, several algorithms outperform the (static) offline solution OFL in this setting. Again, GRD and other myopic path replication policies fail over abilene and path, while TBGRD and PGA still attain a non-zero TACG.

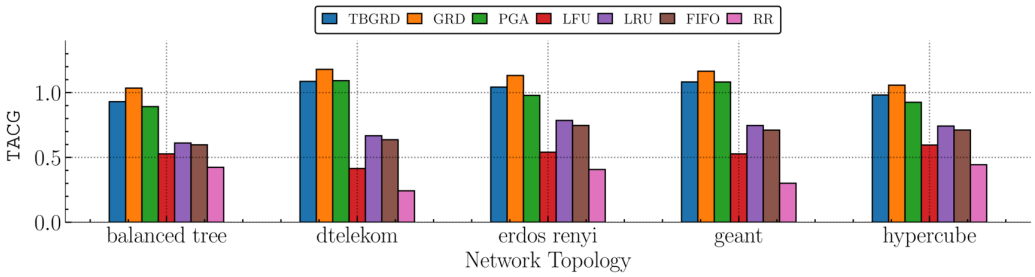


Figure 2.14: TACG of different algorithms over different topologies with *CDN* trace. The total simulation time is 2000 time units. We again observe that TBGRD, GRD, and PGA outperform path replication policies.

these measurements. To compare performance across different topologies, we normalize the average by f_{OFL}, the caching gain attained by OFL, yielding:

$$\text{TACG} = \frac{1}{T_s f_{\text{OFL}}} \sum_{t_s=1}^{T_s} f^{t_s}(A^{t_s}). \tag{2.51}$$

The corresponding f_{OFL} values are reported in Table 2.5. We also measure the cumulative update cost (CUC) of TBGRD over time under the hedge and coupled hedge selectors, i.e.,

$$\text{CUC} = \sum_{t=1}^{T-1} \text{UC}(A^t, A^{t+1}), \tag{2.52}$$

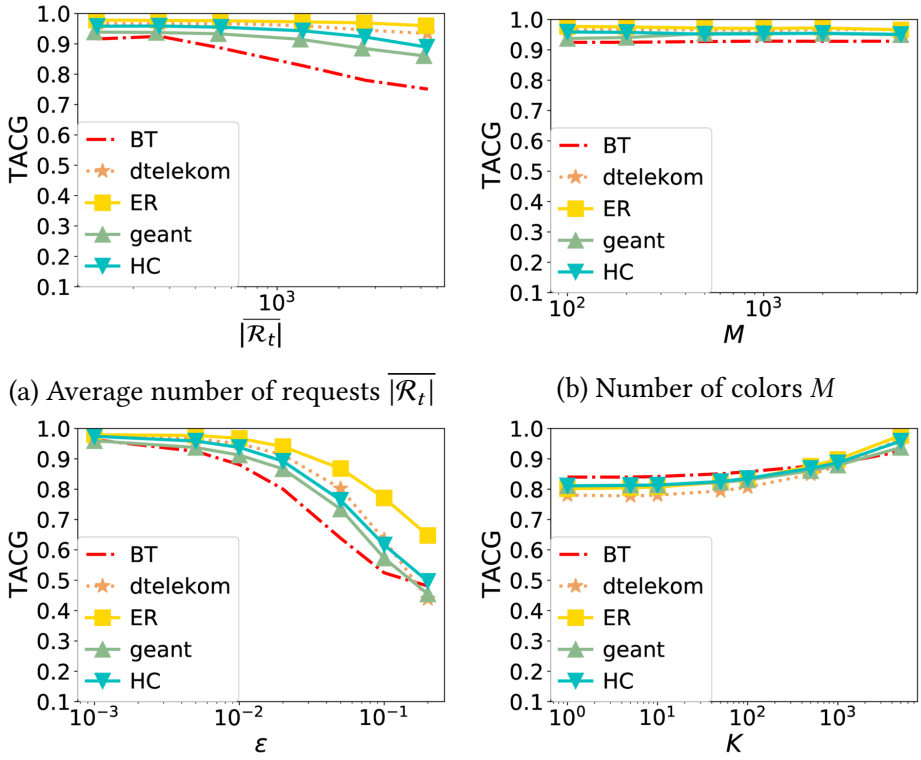
where we measure the instantaneous update cost UC using (2.48) with weights set to 1.

2.3.6.2 Results

TACG Comparison. Figures 2.11-2.13 show the performance of different algorithms w.r.t. TACG across multiple topologies, for different synthetic traces (*Stationary*, *Sliding Popularity*, and *Shot Noise*, respectively). For GRD here, we explore parameters ϵ range from 0.0001-1, and pick the optimal values. We observe that TBGRD, GRD, and PGA have similar performance across topologies on all three traces for the first five topologies, with GRD being slightly higher performing than the other two; nevertheless, on the last two topologies, that have been designed to lead to poor performance for myopic/greedy strategies, both GRD and other myopic strategies (e.g., LFU, LRU, and FIFO) are stymied, attaining a zero caching gain throughout. This also verifies the suboptimality of GRD stated in Lemma 2.3.4. In contrast, TBGRD and PGA still attain a TACG close to the offline value; not surprisingly RR also has a suboptimal, but non-zero gain in these scenarios as well.

The more a trace departs from stationarity, the more the performance of OFL degrades: As seen in Tab. 2.5, the caching gain obtained by OFL consistently across the different topologies has the highest value in the *Stationary* trace, then decreases as we change to *CDN*, *Sliding Popularity SN* traces, in that order.

We also note that in the *Shot Noise* case several algorithms attain a normalized TACG that is higher than 1. This indicates that the dynamic algorithms beat the static offline policy in this setting. The above observations largely carry over to the *CDN* trace, shown in Figure 2.14, for which however we do not consider the two round-robin demand scenarios (abilene and path), as the demand is driven by the trace.



(a) Average number of requests $|\overline{\mathcal{R}_t}|$

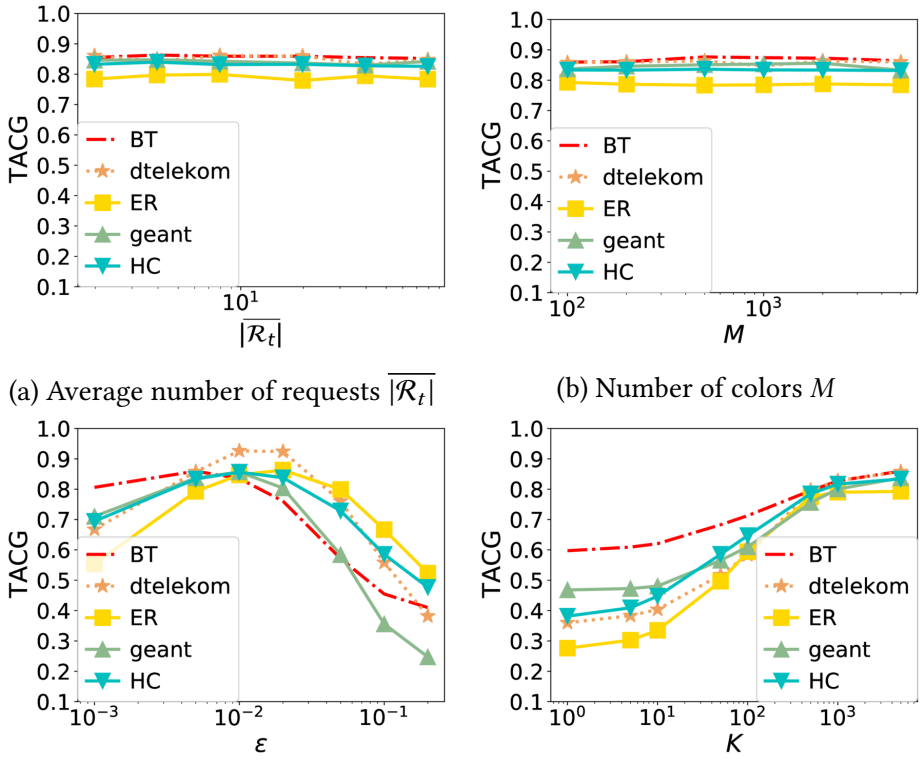
(b) Number of colors M

(c) Parameter ϵ

(d) Color update period K

Figure 2.15: TACG vs. different parameters under *Stationary Request*. As the average number of requests increases, TACG decreases. Number of colors does not affect a lot. As ϵ increases, TACG decreases. As color update period increases, TACG increases.

Impact of Different Parameters. We explore the effect of different parameters in TBGRD with both stationary and sliding popularity requests in Figures 2.15 and 2.16, respectively, for five different topologies. We plot the normalized TACG with different values of colors M, parameter ϵ, color update period K, and average number of requests per round $|\overline{\mathcal{R}_t}|$. For the latter, we select a round duration of B time units, and group all requests within a duration together in to a single request set $\mathcal{R}_t$; note that, due to stochasticity, the number of requests varies at each round t. In general, the normalized TACG for stationary requests is slightly higher than for sliding popularity. This is expected, as stationary requests are easier to learn. The number of colors does not affect the performance of algorithm a lot, shown in Figure 2.15(b) and 2.16(b). From both Figure 2.15(a), we see that smaller request set size leads to better TACG, which again makes sense: that more frequent cache updates are, the faster they adapt to current requests. Besides this, we see that $|\overline{\mathcal{R}_t}|$ has bigger impact under stationary requests, while the sliding window scenario is less affected by varying this parameter. We also observe in Figure 2.15(c) that greater ϵ values lead to worse performance in the stationary setting; however in the sliding popularity setting, shown in 2.16(c), the optimal ϵ is at about 0.01 for multiple topologies; this selection corresponds to a decay rate of the item selection probability that is most appropriate for the popularity refreshing period of these traces. Finally, even



(a) Average number of requests $\overline{|\mathcal{R}_t|}$

(b) Number of colors M

(c) Parameter ϵ

(d) Color update period K

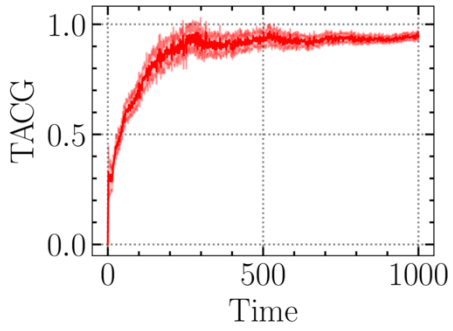
Figure 2.16: TACG v.s. different parameter with *Sliding Popularity* scenario. The average number of requests and number of colors do not affect performance significantly. The optimal ϵ is at about 0.01 for multiple topologies; this selection corresponds to a decay rate of the item selection probability that is most appropriate for the popularity refreshing period of these traces. As the color update period increases, TACG increases.

though higher K is better on both Figure 2.15(d) and 2.16(d), we see more variability/bigger impact of this selection in the sliding popularity trace.

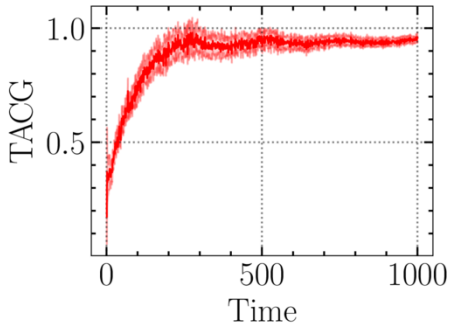
Update Costs. Recall from Theorem 2.3.6 both the (independent) hedge selector and the coupled hedge selector lead to same caching gain in expectation. This also verified experimentally by results of Figure 2.17 (a) and (b): we observe that both hedge selectors lead to almost identical TACG on the sliding popularity trace. We also observe that the cumulative update cost (CUC), shown 2.17(c) and Figure 2.17(d), is vastly different across the two selectors: within the duration of the simulation, the CUC of the hedge selector is more than 15× the CUC of the coupled hedge selector.

2.3.7 Conclusion

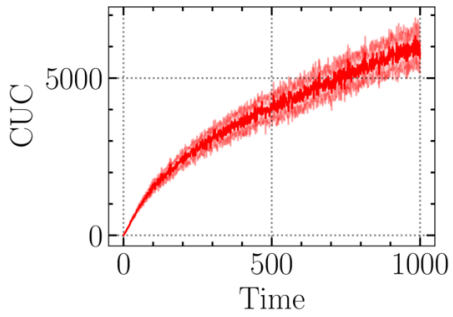
We propose a distributed, online algorithm that achieves sublinear $(1-1/e)$-regret for the adversarial caching gain maximization problem, even when accounting for update costs. An interesting future research direction is to provide regret guarantees for the class of path replication algorithms. These algorithms are appealing precisely because they do not involve updates that happen separately from



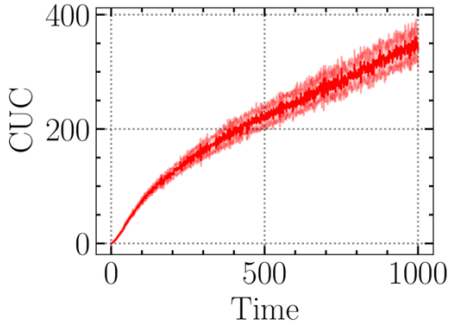
(a) TACG of independent hedge selector Algorithm 2.4



(b) TACG of coupled hedge selector Algorithm 2.6



(c) Cumulative update cost (CUC) of independent hedge selector Algorithm 2.4



(d) Cumulative update cost (CUC) of coupled hedge selector Algorithm 2.6

Figure 2.17: TACG and CUC of DistributedTGOnline over *Sliding Popularity* trace/`dtelekom`. The learning rate is $\epsilon = 5 \times 10^4$. Values reported are averaged over 30 experiments with different random seeds.

the normal response traffic: whenever a response packet carrying an item traverses a cache, the latter makes a decision of whether to cache this content or not on the spot. This restricts the type of allocations that an online algorithm can construct at any point in time, but does not incur any additional cost beyond the one generated by response traffic. This property makes path replication algorithms quite popular in practice [91–93]. Our proof that GreedyPathReplication has linear regret (see Lemma 2.3.4) is a negative result in this direction. Nevertheless, determining whether a path replication algorithm that has sublinear α-regret exists remains an interesting open problem, from both a theoretical and practical point of view. Another important future research direction is to consider dynamic regret [46], whereby the performance of a policy is compared to a dynamic optimum. Dynamic regret was studied under different settings of online convex optimization [46, 94–96], multi-armed bandits [97–100], and non-stationary reinforcement learning [101–103], and would be interesting to apply to our setting.

Chapter 3

Similarity Caching

3.1 Introduction

Similarity search [104] is a key building block for a large variety of applications including multimedia retrieval [105–107], recommendation systems [108–110], genome study [111,112], machine learning training [113–115], and serving [116–125]. Given a query for an object, the goal is to retrieve one or more similar objects from a repository. In the traditional setting, a cache is used to speed up object retrieval: once similarity search has identified the set of similar objects in the global catalog, the system checks if some of these objects are stored in the cache memory. In this setting, the cache performs a local *exact* lookup for the objects. Similarity search over the catalog can be itself a time-consuming operation, equivalent to linearly scanning the whole catalog [126]. Moreover, if users generating the queries are located far from the repository, they may experience long delays.

In order to solve these problems, the seminal papers [106, 108] proposed, almost at the same time, a different use of the cache: clients' requests are directly forwarded to the cache; then the cache performs a similarity search over the set of locally stored objects and possibly serves the requests without the need to forward the query to the (remote) repository. The cache thus reduces the overall serving time at the cost of providing objects *less similar* than those the repository would provide. This operation was named *similarity caching*, in contrast to the traditional *exact caching*. As recognized in [127], the idea of similarity caching has been rediscovered a number of times under different names: recognition caches [118, 119], approximate deduplication [121], semantic caches [122], prediction caches [117], approximate caches [123], and soft caches [110, 128].

In practice, objects and requests are mapped to vectors in $\mathbb{R}^d$ (called *embeddings*), so that the dissimilarity cost can be represented as (a function of) a selected distance between the corresponding embeddings. Examples of commonly employed distances are the p-norm, Mahalanobis or cosine similarity distances. For instance, in augmented reality applications we often require identifying similar objects: the image is coded into a query, i.e., an embedding in $\mathbb{R}^d$ (e.g., set of descriptors like SIFT [129], or ORB [130], or the set of activation values at an intermediate layer of a neural network [131, 132]), and the application logic finds similar images to be returned to the user [118–120, 122]; moreover, in recommendation systems, objects are also embedded in $\mathbb{R}^d$ with a distance metric to capture content dissimilarity. This also the case for other potential applications of similarity caching like 360° videos, where tiles at the periphery of the user's field of view could be approximated by neighboring tiles that are stored at the cache and can then be served with low latency [133]. Similarity caching may then be useful in this context, specially in the future when the number of tiles will increase and close tiles will become more similar.

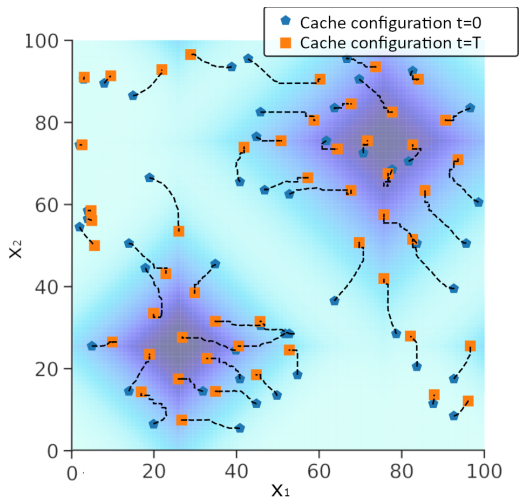


Figure 3.1: Cached objects' movements in the representation space ($\mathbb{R}^2$) during $[0, T]$ when the cache is managed by GRADES. The catalog is made by the points in a 100×100 grid, dark shaded areas correspond to more popular objects. Dissimilarity cost $C_d(x, y) = 1/10 \, \|x - y\|_1$, retrieval cost $C_r = 1$, cache size $k = 50$. See the description in Section 3.2.4.

3.1.1 Contributions

Continuous Catalog Similarity Caching. This work makes the following contributions in the context of *continuous catalog* similarity caching:

- We propose GRADES, the first similarity caching policy designed to exploit object embeddings in $\mathbb{R}^d$ with a distance that captures dissimilarity costs. While previous policies update the cache state by replacing a cached object with (in general) a distant one—corresponding to "jumps" in the representation space—GRADES incrementally updates the (embedding of) each object using a gradient descent step to progressively reduce the dissimilarity cost. Qualitatively, as shown in Figure 3.1, the objects in the cache smoothly move in the representation space to find their optimal position, i.e., where they can serve numerous requests with small dissimilarity cost.
- We prove that in a stationary setting, with an opportune choice of the gradient step sizes, GRADES converges to a cache configuration that corresponds to a critical point of the service cost (likely a local minimum).
- Our experiments based on realistic traces (made available online [134]) show that GRADES outperforms existing similarity caching policies both for 360° videos and recommendation systems applications.

Finite Catalog Similarity Caching. We provide the following contributions in the context of *finite catalog* similarity caching:

- We formulate the problem of kNN optimal caching taking into account both dissimilarity costs and system costs.

- We propose a new similarity caching policy, AÇAI (Ascent Similarity Caching with Approximate Indexes), that (1) relies on fast, approximate similarity search indexes to decide which objects to serve from the local datastore and which ones from the remote repository and (2) uses an online mirror ascent algorithm to update the cache content in order to minimize the total service cost. AÇAI offers strong theoretical guarantees without any assumption on the traffic arrival pattern.
- We compare our solution with state-of-the-art algorithms for similarity caching and show that AÇAI consistently improves over all of them under realistic traces.

3.1.2 Organization

This chapter is organized as follows. Section 3.2 studies continuous catalog similarity caching, and Section 3.3 studies finite-catalog similarity caching. In detail:

Continuous Catalog Similarity Caching. An overview of the related work is provided in Section 3.2.1. We present the formal problem definition in Section 3.2.2. We introduce GRADES and its theoretical guarantees in Section 3.2.3. Finally, experimental results are presented in Section 3.2.4.

Finite Catalog Similarity Caching. We present related work in Section 3.3.1 and other relevant background in Section 3.3.2. We provide the description of similarity caching systems and introduce AÇAI in Section 3.3.3. We present the experimental results in Section 3.3.4.

3.2 Continuous Catalog Similarity Caching

3.2.1 Related Work

Most existing policies for similarity caches generalize well-known exact caching policies, like LRU and LFU, to the new context, where besides the exact hits and misses, *approximate hits* are also possible. For example SIM-LRU [108,116] maintains objects in an ordered queue and serves an object from the cache if its distance to the requested object is less than a given threshold (an approximate hit occurs). The object is then moved to the front of the queue. When no object in the cache is close enough to the request, there is a *miss*. The object is then retrieved from the server and inserted at the front of the queue, possibly evicting objects from the back. RND-LRU [108] is a variant of SIM-LRU where the threshold is replaced by a random variable that is a function of the dissimilarity cost, i.e., the cost associated to the distance between a cached object and the request. As with SIM-LRU and RND-LRU that are adaptations of LRU, qLRU-ΔC [127] modifies qLRU [29] by introducing a refresh probability that depends on the similarity. Finally, DUEL [127] is inspired by LFU, and decides which object to evict by tracking the dissimilarity cost, i.e., the cost associated to the distance between a cached object and the request accumulated over a given time-window.

GRADES was inspired by the work from Jorge Cortés et al. on coordination algorithms for mobile agents [135–137]). In their setting, mobile agents (e.g., drones) place themselves in the space to be able to detect the largest number of events in the environment. Similarly, the objects in the cache need to position themselves to optimally serve the requests appearing over the space. Despite

similarities at a high-level, their work focuses on a two-dimensional space and needs to take into account agents' movement and communication constraints that do not hold in our context.

From another point of view, GRADES gradient update can be considered as a generalization of stochastic k-means algorithms, where the function we want to minimize is not necessarily the squared Euclidean distance (as it is the case for k-means). Our proofs rely on techniques for non-convex optimization originally proposed in [138] also to study k-means.

Online caching policies based on gradient methods have been proposed in the stochastic request setting (see, e.g., [21, 38]), and, more recently, in the adversarial setting [19, 139]. These papers follow a similar methodology to ours in Sections 2.2.3 and 2.2.5 in Chapter 2, where the gradient step updates a vector of length equal to the catalog size, whose component i (in $[0, 1]$) represents which fraction of object i should be stored in the cache or alternatively the probability to store i in the cache. Differently from this line of work, GRADES uses the gradient step to modify the objects in the cache and maintains a vector of size equal to the cache—then much smaller than the catalog size.

A costly operation in any similarity search system is to find the closest object to the request. A simple solution is to index the collection, e.g., with a tree based data structure, to find the exact closest object. Unfortunately, when the number of dimensions d of the representation space exceeds 10, such an approach has a computational cost comparable to a full scan of the collection [126]. For this reason a number of approximate search techniques have been developed, which trade accuracy for speed and provide one or more points close to the request, but not necessarily the closest. Prominent examples are the solutions based on locality sensitive hashing [140], product quantization [141,142], pivots [143], or graphs [144]. In the experiments in this chapter, we have performed an exact similarity search, but any of these approximated search techniques could be used in GRADES.

The original envisaged applications of similarity caches were content-based image retrieval (CBIR) [106] and contextual advertising [108]. In a CBIR system, given an image (used as a query), users can query the CBIR system to obtain images that are most similar to the query by comparing their visual contents. Here, a cache can respond with the most similar images that are available locally. Similarly, in the case of contextual advertising, the cache can provide ads similar to those matching the user profile [108]. Likewise, recommender systems can leverage similarity caches [110, 128]: a recommender system can save operating costs and decrease its response time through recommendation of relevant contents from a cache to user-generated queries, i.e., in case of a cache miss, an application proxy (e.g., YouTube) running close to the helper node (e.g., at a multi-access edge computing server) can recommend the most related files that are locally cached. More recently, similarity caches have been employed extensively for machine learning based inference systems to store queries and the respective inference results to serve future requests, for example, prediction serving systems [117], image recognition systems [118, 119, 121], object classification on the cloud [122], caching hidden layer outputs of a neural network to accelerate computation [123], network traffic classification tasks [145]. The cache can indeed respond with the results of a previous query that is very similar to the current one, thus reducing the computational burden and latency of running complex machine learning inference models.

	System Model		
$\mathcal{N}$	Catalog	λ_r	Arrival rate of request r
$\mathcal{R}$	Set of possible requests	$\mathcal{C}(\mathcal{S})$	Expected cost to serve a request
$\mathcal{S}$	Set of cached objects		GRADES
$\boldsymbol{x}$	Cache state	h	Cache size
c_f	Retrieval cost	VC	Virtual cache
$c_d(r,z)$	Dissimilarity cost between r and z	PC	Physical cache
c_θ	Dissimilarity threshold	$\mu(y_V)$	Matched vector to virtual vector y_V
$c(r,z)$	Combined dissimilarity and retrieval cost	$\rho(r)$	Closest object to r in the catalog
		p	Grafting probability

Table 3.1: Notation Summary for Subsection 3.2

3.2.2 System Description

We consider a similarity search system where a server answers users' queries with the most similar object from a locally stored catalog. In some applications, it is required to serve similar cached objects instead of a single object. GRADES can be augmented to provide k similar answers using the same techniques introduced in [106, 108, 146]. For example, the cache may store key-value pairs, where the key is a past query and the value is the set of k closest objects to the query. Upon a new query, the cache looks for the most similar key stored locally and returns the corresponding set of objects. However, such augmentation is of a heuristic nature; we provide a more careful treatment in Section 3.3. The notation used across this section is provided in Table 3.1.

Requests satisfied by the server incur a *retrieval cost* c_f, which quantifies the delay the user experiences to retrieve the object from the remote server, and/or the additional load for the server, and/or the additional load for the network. Alternatively, the request may be satisfied by a *similarity cache* which stores a subset of the catalog. The cache provides, in general, a less similar object than what the server could provide, but incurs a negligible retrieval cost, as, for example, it is located closer to the user, or uses a faster memory storage or can perform faster lookup operations on the smaller set of stored contents.

We assume that each request or object in the catalog can be represented as a point in the d-dimensional Euclidean space $\mathbb{R}^d$. In what follows we will refer to such representations as *embeddings* and, for the sake of simplicity, we will identify each object/request with its embedding (e.g., we will say that object r belongs to $\mathbb{R}^d$). We assume all objects have the same size and the cache can store up to h objects.

Our model of the system is similar to the one considered in previous papers on similarity caching in the continuous setting like [108, 127]. Let $\mathcal{N}$ and $\mathcal{R}$ denote the catalog and the set of possible requests, respectively. Both sets may be finite or infinite, but we require them to be compact (to be able to retrieve a closest object to a given request). The "quality" of a similarity search for r depends on how similar the response object y is to the request. We assume the *dissimilarity cost* is quantified by the function $c_d(r,y) = u(\|r-y\|)$, where $u : \mathbb{R} \to \mathbb{R}^+$ is a non-decreasing non-negative function and $\|\cdot\|$ is a norm in $\mathbb{R}^d$ (e.g., the Euclidean one). For example Faiss (Facebook AI Similarity Search) library [147] for multimedia retrieval supports all p-norms for $p \in [1,\infty]$.

The state of the cache at time t is given by the set of objects $\mathcal{S}_t$ currently stored in it, $\mathcal{S}_t = \{y_t^1, y_t^2, \ldots, y_t^k\}$, with $y_t^i \in \mathcal{R} \subset \mathbb{R}^d$. Requests arrive first at the cache. Given a request for object r_t at time t, let i_t denote the index of the most similar object to the request, i.e., $i_t \in \arg\min_{i=1,\ldots,k} c_d\left(r_t, y_t^{(1)}\right)$ (if there are many equally similar ones we arbitrarily select one). If

the cache satisfies the request r_t, it will use content $y_t^{(i_t)}$, and the user will incur the dissimilarity cost $c_d(r_t, \mathcal{S}_t) \triangleq c_d\left(r_t, y_t^{(i_t)}\right) = \min_{y \in \mathcal{S}_t} c_d(r_t, y)$, but the retrieval cost is negligible. Alternatively, the cache can forward the request to the server, where it will be satisfied by the most similar object in the catalog. The request will generate the retrieval cost c_f, and the user will experience the dissimilarity cost $c_d(r_t, \mathcal{N}) = \min_{y \in \mathcal{N}} c_d(r_t, y) \leq c_d(r_t, \mathcal{S}_t)$.

Ideally, the cache should compare the costs of serving request locally ($c_d(r_t, \mathcal{S}_t)$) and from the server ($c_d(r_t, \mathcal{N}) + c_f$) and select the most convenient action. But, in order to evaluate $c_d(r_t, \mathcal{N})$, the cache would need to store locally metadata for the whole set $\mathcal{N}$ and find the closest object in it. The memory and computation requirements could defeat the whole utility to have a cache. For this reason, we consider the cache does not know $c_d(r_t, \mathcal{N})$, but it is easy to adapt our algorithm when it is not the case and its theoretical guarantees still hold. We demonstrate how this assumption can be omitted in Section 3.3.

In the impossibility to compare $c_d(r_t, \mathcal{S}_t)$ with the request-dependent value $c_d(r_t, \mathcal{N}) + c_f$, the cache compares $c_d(r_t, \mathcal{S}_t)$ with a constant threshold value c_θ. If $c_d(r_t, \mathcal{S}_t) \leq c_\theta$, the cache serves the request locally, otherwise it forwards it to the server. We assume c_θ is set once and for all offline. Figure 3.2 illustrates how requests are served.

As $c_d(r_t, \mathcal{S}_t) = c_d\left(r_t, y_t^{(i_t)}\right)$, the final cost to serve request r (denoted by $c(r_t, \mathcal{S}_t)$) depends only on $y_t^{(i_t)}$:

$$\begin{aligned} c(r_t, \mathcal{S}_t) &= c\left(r_t, y_t^{(i_t)}\right) \\ &= \begin{cases} c_d\left(r_t, y_t^{(i_t)}\right), & \text{if } c_d\left(r_t, y_t^{(i_t)}\right) \leq c_\theta, \\ c_f + c_d\left(r_t, \mathcal{N}\right), & \text{otherwise.} \end{cases} \end{aligned} \tag{3.1}$$

After a request, the cache can update its state. As updates can themselves generate retrieval costs, we restrain to reactive policies that can only update their state by inserting the object retrieved from the server to satisfy a request.

In our setting, we assume that the two costs, c_f and c_d, can be expressed in the same unit. For instance, the two costs can quantify different aspects contributing to the overall user's quality of experience (QoE). The dissimilarity cost function $c_d(r, y) = u(||r - y||)$ can describe the QoE loss for the end user upon receiving a dissimilar object, while c_f can capture the QoE loss due to the experienced delay. As in our model, the ITU-T E-model combines additively different metrics, such as signal-to-noise-ratio, packet loss ratio, and delay, to obtain a single scalar QoE rating for voice communications [148]. More in general, it is quite common in multi-objective resource allocation problems to express the cost as a weighted sum of the different objective functions [149–151].

In our theoretical analysis in Section 3.2.2, we consider the case when requests arrive according to a Poisson process and are i.i.d. distributed. In the finite case ($|\mathcal{R}| < \infty$), we recover the classic independent reference model [15], where object r is requested with rate λ_r. In the continuous case, we need to consider a spatial density of requests and objects in a set $\mathcal{A} \subset \mathcal{R}$ are requested with rate $\int_{\mathcal{A}} \lambda_r \, \mathrm{d}r$.

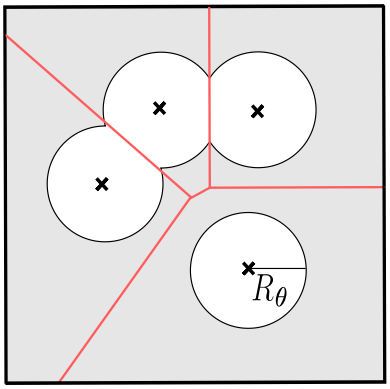


Figure 3.2: Coverage of the request space ($\subset \mathbb{R}^2$) by 4 objects in the cache with norm-2 as dissimilarity cost. Crosses represent the objects. Each object is the closest point to requests in the corresponding Voronoi cell delimited by the red lines. Consider a request r falling in the Voronoi cell of an object y_i. If r is closer to y_i than the critical radius R_θ (such that $u(R_\theta) = c_\theta$), r receives y_i as reply, otherwise (it falls in the gray shaded area) it generates a miss.

Under the above assumptions, for a given cache state $\mathcal{S} = \{y_1, y_2, \ldots, y_k\}$, we can compute the corresponding expected cost to serve a request:

$$\mathcal{C}(\mathcal{S}) \triangleq \begin{cases} \sum_{r \in \mathcal{R}} \lambda_r c(r, \mathcal{S}), & \text{finite case} \\ \int_{\mathcal{R}} \lambda_r c(r, \mathcal{S})\, \mathrm{d}r, & \text{continuous case.} \end{cases} \tag{3.2}$$

Finding an optimal set of objects $\mathcal{S}^*$ to store in the cache that minimizes the cost $\mathcal{C}(\mathcal{S})$ is NP-hard as it is a generalization of the problem considered in [127] (where $c_d(r, \mathcal{N}) = 0$ for each $r \in \mathcal{R}$). Nevertheless, for the continuous case, we propose a dynamic gradient descent based algorithm, that, under the stationary request process, can achieve a stationary point of $\mathcal{C}(\mathcal{S})$. The gradient descent based algorithm is a natural choice for the continuous setting; the algorithm takes a descent step to decrease $\mathcal{C}(\mathcal{S})$ at each time step. Further, in Section 3.2.3.3, we describe a heuristic adaptation of the proposed algorithm for the finite case.

3.2.3 A Gradient-based Algorithm

The key idea of our algorithm is to let the objects stored in the cache gradually "move" in the space $\mathbb{R}^d$ to reach a configuration where they can be used as approximate answers for a large number of requests (see Figure 3.1). Upon a request at time t for r_t, the most similar object in the cache, $y_t^{(i_t)}$, is moved in the direction opposite to the gradient of the service cost ($\nabla_y c(r_t, y_t^{(i_t)})$) proportionally to a time-varying step-size (or learning rate) η_t:

$$y_{t+1}^{(i_t)} = y_t^{(i_t)} - \eta_t \nabla_y c\left(r_t, y_t^{(i_t)}\right). \tag{3.3}$$

It is possible to prove that $c(r, y)$ is differentiable everywhere and then the gradients in (3.3) exist with probability 1 when the request process is continuous.[1] When the request process is discrete, the probability that the gradient $\nabla_y c(r_t, y_t)$ does not exist may be non-zero, but we can then perturb the request by a small random vector $\epsilon \in \mathbb{R}^d$ and consider $\nabla_y c(r_t + \epsilon, y_t)$.

Note that the algorithm is oblivious to the request rate (λ_{r_t}) of the object r_t. However, every time a request is made for object r_t, the most similar object in the cache $(y_t^{(i_t)})$ is moved in the direction of the requested object. Therefore, the algorithm dynamics are sensitive to λ_{r_t}.

An attentive reader may frown upon the simple algorithm (3.3). First, it potentially updates the cache upon every request, even when $c_d(r_t, y_t^{(1)}) \leq c_\theta$ and the cache would not need to retrieve any object. Second, even if $y_t^{(1)}$ is the embedding of an object in the catalog, $y_{t+1}^{(1)}$ may not correspond to any object in the catalog.

In the following sections we address all issues mentioned above. After having refined the update rule (3.3) (Section 3.2.3.1), we prove that this idealized algorithm indeed converges to a critical point of $\mathcal{C}(\mathcal{S})$ (Section 3.2.3.2). Then, in Section 3.2.3.3 we present a practical algorithm which (1) satisfies all our requirements, (2) keeps the state of the cache "close" to the state of the idealized algorithm, and () is more reactive and thus more suitable to non-stationary request processes.

3.2.3.1 Introducing a Projection

As requests are only for objects in the bounded set $\mathcal{X}$, there exists a norm-2 ball with radius R—$\mathcal{B}_2(R) = \{y \in \mathbb{R}^d, \|y\|_2 \leq R\}$)—such that $\mathcal{X} \subset \mathcal{B}_2(R)$ and $c(r, y) = c_f$ for each $y \notin \mathcal{B}_2(R)$ and $x \in \mathcal{X}$. There is no advantage to store in the cache objects that do not belong to $\mathcal{B}_2(R)$ as they do not contribute to approximate any request. We then modify (3.3) in order to make closer to $\mathcal{B}_2(R)$ any cached object $y_t^{(1)}$ that the gradient update may have brought out of $\mathcal{B}_2(R)$. We write

$$
\begin{aligned}
y_{t+1}^{(1)} &= y_t^{(1)} - \eta_t g_t^{(1)}, \\
g_t^{(1)} &= \begin{cases} \nabla_y c\left(r_t, y_t^{(1)}\right), & \text{if } (i = i_t) \wedge \left(y_t^{(1)} \in \mathcal{B}_2(R)\right) \\ f(\|y_t^{(1)}\|_2 - R)\frac{y_t^{(1)}}{\|y_t^{(1)}\|_2}, & \text{if } y_t^{(1)} \notin \mathcal{B}_2(R), \\ \mathbf{0}, & \text{otherwise,} \end{cases}
\end{aligned}
\tag{3.4}
$$

where $f(\cdot)$ is an increasing non-negative function (so that $-g_t^{(1)}$ points to the origin of the space). For technical reasons that will be required in the proof of Theorem 3.2.2, we want $f(0) = f'(0) = f''(0) = f'''(0) = 0$, and $f'(\infty) \in \Theta(1)$. For these reasons, we select $f(x) = \frac{\mathrm{d}}{\mathrm{d}x}\frac{x^4}{1+x^2} = 2x^3\frac{2+x^2}{(1+x^2)^2}$ for the proof.

[1] As far as all the $y_t^{(1)}$ are different, $\nabla_y c(r, y_t^{(i_t)})$ exists for all the points in $x \in \mathcal{X}$ but at most for a measure zero set: the set of points for which $\nabla_y c_d(r, y_t^{(1)})$ does not exist, the points for which $\|x - y_t^{(1)}\| = \|x - y_t^{(j)}\|$ for $i \neq j$, and finally the set of points in $\partial\{x \in \mathbb{R}^d : c_d(r, y_t^{(1)}) > c_\theta\}$, where ∂A denotes the boundary of the set A.

3.2.3.2 Theoretical Guarantees

In this section we provide convergence results for the basic algorithm described in (3.4). We assume the cache update rule generates embeddings that always correspond to objects in the catalog. Moreover, we will ignore the cost of updates made *after* the request is served. These two simplifications will be removed in the next section.

It will be useful to denote the cache state as a vector $\boldsymbol{y}_t = (y_{t,1}, \dots, y_{t,k}) \in \mathbb{R}^{k \times d}$, obtained by concatenating the embeddings of the different objects in the cache. Similarly, we define the different costs as function of $\boldsymbol{y}_t$ and then write $c_d(\boldsymbol{y}_t)$, $c(\boldsymbol{y}_t)$, and $\mathcal{C}(\boldsymbol{y}_t)$. We are now going to prove that algorithm (3.4) converges almost surely to a stationary point of $\mathcal{C}(\boldsymbol{y})$ and the trajectory of $\boldsymbol{y}_t$ is bounded almost surely.

Lemma 3.2.1. *Let the learning rate η_t be selected so that $\sum_{t=1}^{+\infty} \eta_t = +\infty$ and $\sum_{t=1}^{+\infty} \eta_t^2 < +\infty$. The sequence $(\boldsymbol{y}_t)$ is bounded almost surely.*

The proof of the lemma is in Appendix 12. The lemma is used in the proof of the following convergence result.

Theorem 3.2.2. *Let the learning rate η_t be selected so that $\sum_{t=1}^{+\infty} \eta_t = +\infty$ and $\sum_{t=1}^{+\infty} \eta_t^2 < +\infty$. If $\mathcal{C}(\cdot)$ is continuously differentiable up to the second order then*

$$\liminf_{t \to \infty} \|\nabla_{\boldsymbol{y}} \mathcal{C}(\boldsymbol{y}_t)\|_2 = 0 \text{ a.s.}$$

If $\mathcal{C}(\cdot)$ is continuously differentiable up to the third order then

$$\lim_{t \to \infty} \nabla_{\boldsymbol{y}_t} \mathcal{C}(\boldsymbol{y}_t) = 0 \text{ a.s.}$$

The proof of Theorem 3.2.2 is in Appendix 13. Our proof relies on techniques for non-convex optimization originally proposed in [138]. We think it is possible to derive similar results, under different hypotheses, using the approach based on ordinary differential equations proposed in [152].

Theorem 3.2.2 states that the sequence $(\boldsymbol{y}_t)$ converges to a critical point of $\mathcal{C}(\cdot)$, i.e., a point where the gradient is zero. This may be a saddle point, a local maximum or a local minimum of $\mathcal{C}(\cdot)$. The latter is more likely as it is the only one locally stable. The saddle points and local maxima of $\mathcal{C}(\cdot)$ are not stable, as on reaching either of these two types of points, requests that appear in the neighborhood may perturb $\boldsymbol{y}_t$ and the gradient descent algorithm moves $\boldsymbol{y}_t$ away from these points. Given the stochastic nature of the request process this is highly likely to happen.

3.2.3.3 Implementation

In this section we present our complete caching policy Grades, whose pseudo-code is in Algorithm 3.1. Theorem 3.2.2 shows that the basic gradient update (3.4) attains a critical point of the expected cost $\mathcal{C}(\cdot)$. Nevertheless, we have assumed that this update rule always generates embeddings in $\mathbb{R}^d$ that correspond to objects in the catalog. However, if the catalog has a finite number of objects, this is unlikely to happen, as the update (3.4) can potentially generate any real vector. Moreover, the update (3.4) may modify an object in the cache upon each request and then generate a high load on the server and the network to retrieve the new modified objects.

Algorithm 3.1 GRADES

Let h be the cache size and r the object requested
if $(|\mathcal{S}_{V,t}| < h) \wedge (r \notin \mathcal{S}_{V,t})$ **then** ▷ still space in cache
    Insert r in VC
    Retrieve and insert $\rho(r)$ in PC
    $\mu(r) = \rho(r)$
end if
$y_V = \arg\min_{y \in \mathcal{S}_{V,t}} c_d(r, y)$
Update $S_{V,t}$ according to (3.4)
if $c_d(r, y_V) \leq c_\theta$ **then** ▷ virtual hit
    if $\|r - y_V\| < \|\mu(y_V) - y_V\|$ **then** ▷ r approximates y_V better than $\mu(y_V)$
        Evict $\mu(y_V)$
        Retrieve and Insert $\rho(r)$ in PC
        $\mu(y_V) = \rho(r)$
    end if
    GRAFT_HIT_UPDATE$(r, S_{V,t})$
end if
$y_P = \arg\min_{y \in \mathcal{S}_{P,t}} c_d(r, y)$
$\xi \sim \text{Uniform}(0, 1)$
if $\xi < p$ **then**
    (update, ω) =GRAFT_MISS_UPDATE$(S_{V,t}, r, \rho(r))$
    if update **then**
        Evict ω and $\mu(\omega)$
        Retrieve and Insert $\rho(r)$ in VC and PC
        $\mu(\rho(r)) = \rho(r)$
        $y_P = \rho(r)$
    end if
end if
end if
if $(c_d(r, y_P) \leq c_\theta) \vee (\rho(r)$ inserted in PC$)$ **then**
    Serve y_P
else
    Retrieve and Serve $\rho(r)$
end if

In Section 3.2.3.3 we describe how our algorithm addresses these issues. We then move on in Section 3.2.3.3 to describe some additional features that provide a higher adaptivity of the algorithm to deal with highly non-stationary request processes, allowing for some random insertions with probability p.

Dealing with Finite Catalog and Reducing Server Load. We propose to maintain a virtual cache (VC) and a physical cache (PC). The VC only stores some metadata, but no actual object; its use is common to other policies like 2-LRU [29] or AdaptSize [153]. The VC is sometimes called shadow cache.

In our case the VC stores h vectors in $\mathbb{R}^d$ that are updated upon each request according to the basic algorithm in (3.4). These vectors are the embeddings of the objects we would like to store in the cache, but, as discussed above, such objects may not exist, or they may not have been retrieved yet from the server. The PC contains objects from the catalog together with their embeddings.

At a high level, the main idea behind GRADES is to maintain the PC as close as possible to the VC. We use then the current state of the VC to drive updates at the PC, i.e., object eviction and insertion. In particular each vector y_V in the VC is matched to an actual object $\mu(y_V)$ in the PC and GRADES will opportunistically update $\mu(y_V)$ to make it as close as possible to y_V.

We now describe in details Algorithm 3.1 using the following additional notation:

- $\mathcal{S}_{V,t}$ and $\mathcal{S}_{P,t}$ denote the state of the VC and the PC, respectively.
- $\rho(r)$ denotes the closest object in the catalogue to r.

The shaded lines correspond to changes to increase algorithm adaptivity and will be discussed in Section 3.2.3.3.

Upon a request for r, if there is still space in the cache, we retrieve the most similar object in the catalogue $\rho(r)$. GRADES inserts r and $\rho(r)$ in the VC and in the PC, respectively, and matches them ($\mu(r) = \rho(r)$). These operations are described in lines 2–5. The cache will finally serve $\rho(r)$.

If the cache is already full, the closest object in VC y_V will be updated according to (3.4) (lines 6–7). Upon a virtual hit, if r is closer to y_V than the currently matching object $\mu(y_V)$ in the PC, GRADES takes advantage of this request to replace $\mu(y_V)$ with $\rho(r)$ (lines 9–12). In a stationary setting, the state of VC converges to a critical point of the cost (Theorem 3.2.2) and the PC should become closer and closer to it. Finally, the most similar object in PC is served if it is close enough to r, or if in any case $\rho(r)$ has been retrieved (line 11).

Increasing Adaptivity. According to what we described above, only the closest object in VC is updated upon a request (unless some projection back to $\mathcal{B}(R)$ is needed). A potential problem is that if an object r far from any other object has been accidentally inserted in VC (and the corresponding object $\rho(r)$ in PC), it may never be updated and may uselessly occupy cache space. Moreover, if at some point the request process changes abruptly, some objects in the cache that were initially useful may find themselves too far from the new requests. Again, the gradient algorithm, by itself, would not update such objects. To overcome this problem, we can graft to GRADES a more dynamic caching policy that occasionally (with probability p) updates the VC, hopefully evicting the least useful object in the VC.

The "grafting" is described by the green-shaded lines in Algorithm 3.1 and has been designed to support general cache eviction algorithms like LRU, LFU, and their variants. The grafted caching policy internally maintains its own data structure, e.g., an ordered queue for LRU. Upon an approximate hit, the hit update rule of the grafted policy is called (line 13). For example, LRU would move the requested object (if present in the cache) to the front of the queue. Also, with probability p, GRADES invokes the miss update rule of the grafted policy, that may lead to select an element ω to be evicted. GRADES then updates accordingly the VC and the PC (lines 19–22).

Algorithm Complexity. A straightforward implementation of GRADES has a time complexity of O(hd), where h is the cache capacity and d is the embedding dimension. This is because one has to iterate through the h objects in the cache to find the most similar object to the requested object. However, one could use approximate nearest neighbor index, such as those based on hierarchical navigable small worlds (HNSW) graphs [144]. Numerically, HNSW is able to answer a 10NN query

Trace	Number of requests	Catalog size	Dimension (d)
Synthetic	2,000,000	97,969	2
360° videos	10,000,000	25,393	3
Amazon	908,179	63,891	100
CiteULike	2,411,819	153,277	100
Movielens	620,222	136,677	200

Table 3.2: Traces description

over a dataset with 1 million objects in a 128-dimensional space in less than 0.5 ms with a recall greater than 97% [154]. As for the memory footprint, a typical configuration of the HNSW index requires $O(d)$ bytes per objects, where d is the number of dimensions. For instance, in case of $d = 128$ dimensional vectors, the memory required to index 10 million objects is approximately 5 GB.

3.2.4 Experiments

In this section, we empirically validate our algorithm through simulations. First we demonstrate the benefit of the algorithm by using synthetic traces. Next, to demonstrate real world applicability of our algorithm, we use GRADES in the domain of caching for 360° videos and recommendation systems. We assume that the catalog coincides with the set of possible requests ($\mathcal{N} = \mathcal{R}$) and then set $c_\theta = c_f$. The retrieval cost c_f is always equal to 1. Table 3.2 summarizes the main characteristics of the traces. Further details about the experimental setup and the properties of request traces will be described in the corresponding subsections. To the best of our knowledge, there are no public traces for similarity caching; we made our traces available online [134].

We compare GRADES with the following algorithms.

a) GREEDY is an offline static algorithm that progressively fills the cache inserting the object that provides the largest cost saving given the set of objects already inserted. The algorithm provides a $\frac{1}{2}$ approximation in terms of cost savings [65].

b) LRU+ updates the cache as the classic LRU evicting the least recently used content when needed, but it can provide approximate objects.

c) SIM-LRU [108] maintains the content in an ordered queue as LRU. It moves objects to the front upon an approximate hit, and evicts objects from the back when needed.

d) qLRU-ΔC [127] is a variant of qLRU [29] that, upon an approximate hit, moves the object to the front with a probability which is proportional to the service cost reduction the object has guaranteed on the current request.

e) DUEL [127], upon a request for object r not in cache, r is matched with an object y in the cache in a tournament aimed at deciding if r is a better candidate to be stored in the cache as compared to y. The decision is made by comparing the cost savings r and y provide over a fixed interval of time (f). If the new object r provides a larger cost saving, then r replaces y in the cache.

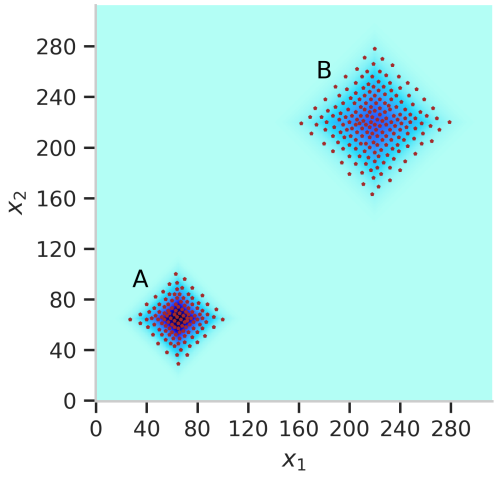


Figure 3.3: The heatmap depicts the popularity distribution of objects in the grid. The darker regions A and B contain the most popular content. The circles represent the final configuration produced by the GRADES policy ($\eta = 0.64$) under the trace *Synthetic*.

3.2.4.1 Synthetic Traces

We consider a setting similar to [127]. The catalog is made by the points of a $L \times L$ bi-dimensional grid with $L = 313$. The cache size $h = 313$.[2]

For any two objects x and y on the grid we define the approximation cost to be proportional to the norm-1 distance between the two points x and y, in particular $C_d(x, y) = \frac{1}{10}||x - y||_1$. In our experiments, we observe that GRADES converges to an expected cost that is slightly better than the approximate optimal cost as computed in [127], suggesting that GRADES converges very close to optimal.

The traffic is generated under the *Independent Reference Model* [15]. There are two popular regions A and B centered around coordinates $(65, 65)$ and $(220, 220)$, respectively; they are produced by a mix of two Gaussian distributions. In particular, an object at (norm-1) distances d_1 from the center of A and d_2 from the center of B is requested with probability

$$\Pr(d_1, d_2) \propto 0.4 \times \frac{e^{\frac{-d_1^2}{2\times 15^2}}}{\sqrt{2\pi \times 15}} + 0.6 \times \frac{e^{\frac{-d_2^2}{2\times 25^2}}}{\sqrt{2\pi \times 25}}.$$

The popularity distribution of the objects in the grid is depicted in the heat-map in Figure 3.3. Figure 3.1 corresponds to a rescaled version of the same process.

Figure 3.4 shows the performance of GRADES without any graft and with different grafts (LRU+, SIM-LRU, qLRU-ΔC) for a quite large value of the grafting parameter ($p = 10^{-2}$). GRADES/X denotes GRADES grafted with policy X. We observe that GRADES achieves the smallest cost, and by grafting

[2] As noted in [127], when object requests fall uniformly over the points of a $L \times L$ grid (with wrap-around conditions), with $h = L = 1 + 2l(l + 1)$, for some positive integer l, an optimal cache configuration can be computed. The value $h = L = 313$ results from the particular choice of $l = 12$. For a non-homogeneous request process an approximate optimal cost can be computed as well (see Appendix F in [127]).

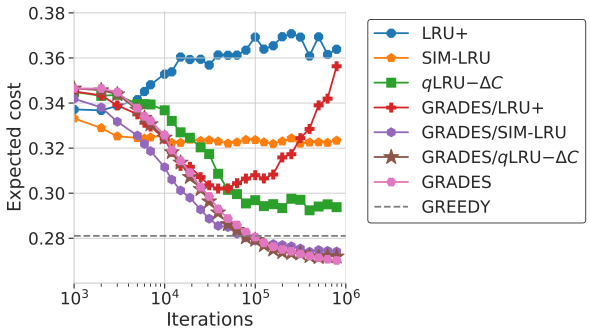
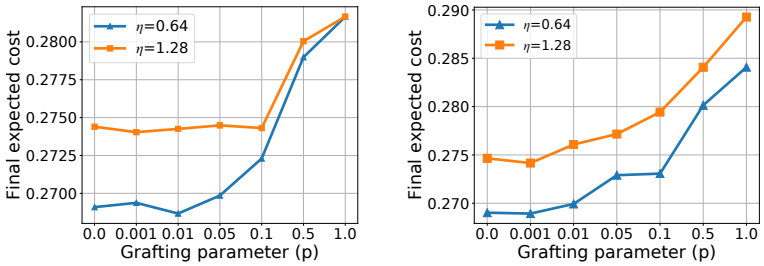


Figure 3.4: Expected cost $\mathcal{C}(\cdot)$ incurred by different policies under the trace *Synthetic*. LRU+, SIM-LRU, qLRU-ΔC ($q = 10^{-2}$), GREEDY, and GRADES ($\eta = 0.64$, plain and grafted with $p = 10^{-1}$). Cache size $h = 313$.



(a) qLRU-ΔC (b) SIM-LRU

Figure 3.5: Effect of the grafting parameter p on the final expected cost $\mathcal{C}(\cdot)$. GRADES/qLRU-ΔC ($q = 10^{-2}$) and GRADES/SIM-LRU for different learning rates under the trace *Synthetic*. Cache size $h = 313$.

GRADES with SIM-LRU we can make the initial transient faster. Figure 3.3 also shows the final cache configuration reached by GRADES: as expected, the density of the objects in the cache is higher where the request density is higher.

The effect of the grafting parameter p is shown in Figure 3.5 and depends on the specific grafted policy. We see that GRADES/qLRU-ΔC[3] is relatively insensitive to the grafting up to $p = 0.05$, but for larger values of p the cost increases, and approaches the cost of qLRU-ΔC alone (about 0.295 as it can bee seen in Figure 3.4). This happens because, for large p, more and more cache updates are due to qLRU-ΔC, which, even upon an approximate hit, may introduce the requested object with probability proportional to q. For GRADES/SIM-LRU, the cost again increases as p increases, but it is always much smaller than the cost of SIM-LRU alone (about 0.32). The explanation is that SIM-LRU never introduces new objects on approximate hits. Hence, as far as the current cache allocation is providing approximate answers, the function GRAFT_MISS_UPDATE in Algorithm 3.1 does not modify the current cache allocation.

[3]Note that GRADES grafted with $p = p'$ to a qLRU-ΔC with $q = q'$ is equivalent to GRADES grafted with $p = 1$ to a qLRU-ΔC with parameter $q = p' \times q'$.

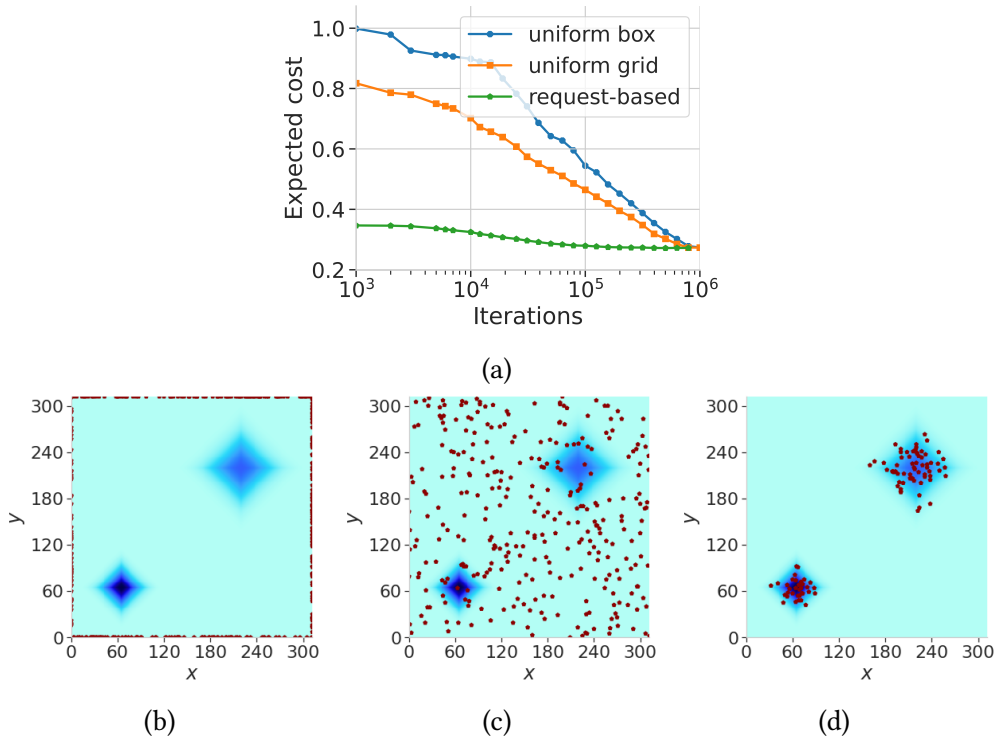


Figure 3.6: Subfigure (a) shows the expected cost $\mathcal{C}(\cdot)$ incurred by GRADES/qLRU-ΔC ($\eta = 0.64$ grafted with $p = 1$ and $q = 10^{-3}$) for different initialization (uniform box, uniform grid, and request-based depicted in (b), (c), and (d), respectively) under the trace *Synthetic*. Cache size is $h = 313$. The heatmap depicts the popularity distribution of objects in the grid.

In Figure 3.6, we study the effect of the initialization on the the performance of GRADES. We test three different initialization schemes: uniform box, uniform grid, and request-based depicted in Figure 3.6 (b), (c), (d), respectively. We sample the initial set of objects uniformly at random without replacement among those at the boundary ($x \in \{0, 312\}$ or $y \in \{0, 312\}$) and from the whole catalog (313×313 grid), respectively under the *uniform box* initialization and the *uniform grid* one. Instead, we take the first h distinct requested objects to obtain the *request-based* initialization. Note how these three schemes lead to store initially in the cache progressively more popular objects. Figure 3.6 (a) shows the three initializations provide different initial costs, with the configuration with the least (resp. most) popular objects leading to the highest (resp. lowest) initial cost. The figure shows also that, independently of the initial cache configuration, GRADES is able to move to configurations with smaller cost (similar to the one in Figure 3.3).

Until now, we have considered a stationary request scenario, where there is no evident advantage from grafting a more reactive policy to GRADES. In Figure 3.7 we consider a highly non-stationary setting. At time 0, the cache is initialized as in Figure 3.3 , i.e., the cache configuration reached by GRADES after a large number of requests (a million) made from a mix of the two gaussian distributions. Then, the request process changes abruptly and no more requests for objects in region A are generated. The cache should reach a new configuration where all cached objects are located in region B, achieving a lower cost, as now the same number of objects should cover a smaller area.

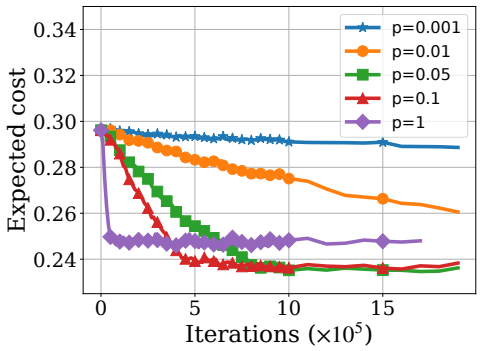


Figure 3.7: Effect of grafting parameter p on the expected cost $\mathcal{C}(\cdot)$ in a dynamic setting for GRADES/qLRU-ΔC ($\eta = 0.64$). The cache is initialized to a stationary configuration as in Figure 3.3. Now, only requests corresponding to region B from the trace *Synthetic* are made. Cache size $h = 313$.

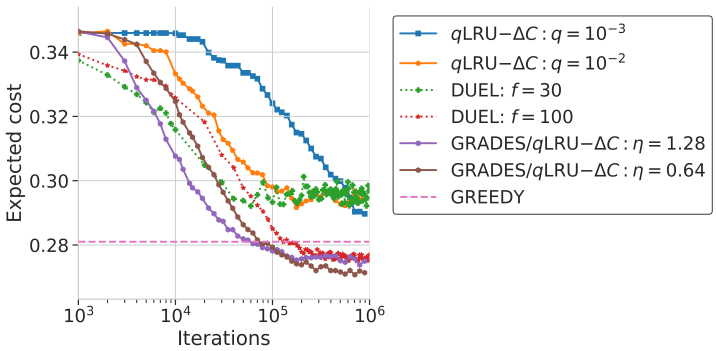


Figure 3.8: Expected cost $\mathcal{C}(\cdot)$ incurred by the different policies under the trace *Synthetic*: qLRU-ΔC ($q = 10^{-3}, 10^{-2}$), DUEL , GREEDY, and GRADES ($\eta = 1.28, 0.64$ grafted with $p = 1$ and $q = 10^{-3}$). Cache size $h = 313$.

Figure 3.7 shows that a higher value of p enables faster migration of objects from region A to region B in the cache.

Figure 3.8 shows the synergy between GRADES and the grafted policy. qLRU-ΔC, DUEL, and GRADES, all have parameters (q, η, and f) that can be tuned to find an optimal trade off between convergence speed and final cost. They can converge fast to configurations within a large neighborhood of a critical point (for high q, high η, and low f, respectively), or slowly to configurations within a smaller neighborhood.

Experiments in [127], in a setting similar to ours, show that qLRU-ΔC achieves a worse cost-vs-speed tradeoff than DUEL. Figure 3.8 confirms that this is the case, but when qLRU-ΔC is grafted on GRADES, the resulting policy improves on top of DUEL. In fact GRADES/qLRU-ΔC achieves a better trade-off as it is able to converge to a cost comparable to DUEL in a shorter time, or equivalently to a smaller cost in roughly the same time. Note also that DUEL's expected cost at steady state is noisier than GRADES/qLRU-ΔC's cost, showing the advantage of smoothly updating the state using gradients.

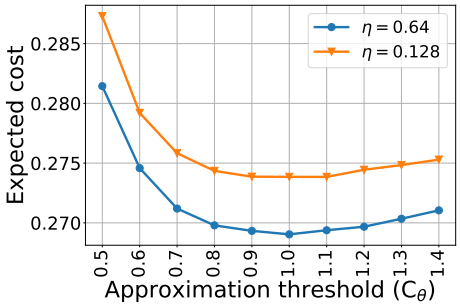


Figure 3.9: Expected cost $\mathcal{C}(\cdot)$ incurred by the GRADES/SIM-LRU under the trace *Synthetic* for different values of the approximation threshold c_θ. Cache size $h = 313$, $c_f = 1$, $\eta = 0.64$, 1.28 and $p = 0.001$.

We now study the impact of the approximability threshold (c_θ) on the expected cost using the *Synthetic* trace. We vary the value of c_θ from 0.5 to 1.4 in increments of 0.1. Figure 3.9 shows that, as c_θ increases, the expected cost first decreases and then increases. In fact, for $c_\theta \ll c_f = 1$, a larger c_θ increases the number of approximate hits and then avoids the need to pay the cost c_f to retrieve the objects from the server. On the contrary, for $c_\theta \gg c_f = 1$, a larger c_θ is not beneficial, because misses are finally less costly than approximate hits. Figure 3.9 suggests that the optimal configuration is $c_\theta \approx c_f = 1$ (in our experiments the server can always provide an exact hit). We observe that, while this choice minimizes the cost to serve a request *given the current cache configuration*, the choice of c_θ also influences how the cache state evolves. This is because newer objects are only introduced into the cache on misses. Therefore, the optimal value for c_θ could be, in principle, different.

3.2.4.2 360° Videos

We test our algorithms on 360° video traces. A 360° video is an immersive, spherical video [155,156]. The video is first projected on to a 2D plane to be encoded by classic 2D video encoders. The video is divided into time segments, and each segment is further spatially divided into tiles. The VR headset is optimized to fetch the required tiles based on the head position of the user. A system responsible for the delivery of 360° videos can store the popular tiles in nearby caches [133].

We generated a sequence of tiles' requests for 360° videos using the approach proposed in [157]. We took real traces from 8 videos watched by 48 users each, and then built a *navigation graph* for each video, i.e., a Markov Chain that represents the spatial and temporal viewing correlations for the video. The videos we considered have on average 207 segments, each with 25 tiles. From each navigation graph we can generate an arbitrary number of possible views of the video. We generated then a trace with 10,000 users as follows. At time $t = 0$, each user selects one of the videos at random and starts watching the video from a random segment of the video. The user then walks through the navigation graph to view the complete video. Once the user reaches the last segment in the video, it selects a new video uniformly at random (with replacement) and starts watching the selected video from the first segment. The process is repeated till 10 million requests are generated. We assume each tile can approximate at most 4 tiles (the adjacent ones), with a fixed dissimilarity cost $c_d = 0.1$.

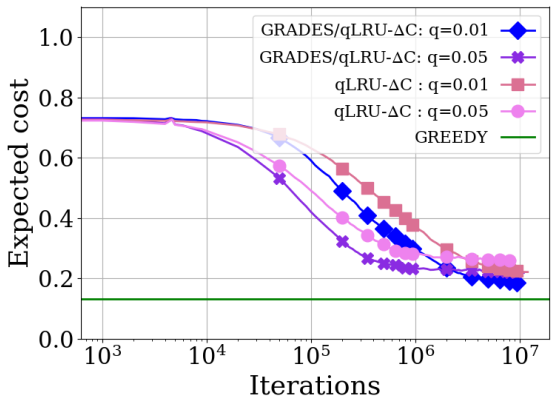


Figure 3.10: Expected cost $\mathcal{C}(\cdot)$ incurred by the different caching policies under the trace 360° *videos*. GRADES/qLRU-ΔC ($q = 0.01, 0.05$) with $\eta = 1.0$ and qLRU-ΔC ($q = 0.01, 0.05$). Cache size $h = 4000$.

Figure 3.10 compares the performance of GRADES/qLRU-ΔC and qLRU-ΔC. Note that in this setting, the representation space exhibits a very rough granularity, as the tiles of a segment cannot be used to approximate those of another segment and each segment is decomposed in a 5×5 grid of tiles. Nevertheless, GRADES/qLRU-ΔC shows significant improvement with respect to existing similarity caching policies and approaches the cost of GREEDY.

3.2.4.3 Machine Learning Traces

We study the performance of similarity caching under the following traces in high-dimensional spaces.

Amazon trace. The paper [158] proposes a technique to embed the images of Amazon products in a 100-dimensional space, where the Euclidean distance between two items captures the similarity of the sets of users who purchased or viewed both items. We have restricted ourselves to the products in the category "Baby" and we have assumed that a request for a given item was issued at time t, if a user left a review for the considered item at the same time.

CiteULike trace. The CiteULike dataset [159] contains a bipartite network of 22,715 users and 153,277 tags, where each edge represents a timestamped tag creation. The embeddings in a 100-dimensional space are obtained using the collaborative metric learning model proposed in [160]. As for the Amazon trace, the Euclidean distance within this space encodes the similarity between users and items, where the items here are the tags. We generated the trace considering that an object (tag) is requested when one user adds the corresponding tag.

Movielens trace. We have trained the RecVAE collaborative filtering model from [161] on the Movielens dataset [162] to embed users' rating histories in a $d = 200$ dimensional space. Users with similar rating histories are mapped to vectors close according to the Euclidean distance. We have

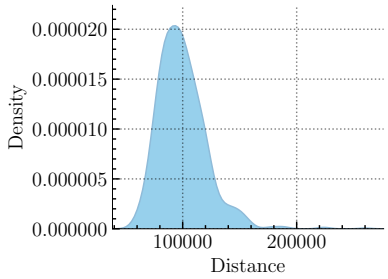


Figure 3.11: The empirical distribution of pairwise distances under the trace *Amazon*. The distribution is computed on a random subset of the catalog containing 1000 products.

generated the trace by embedding every batch of 38 ratings from the same user (38 is the median number of ratings across all users) and assigning it the timestamp of the latest rating in the batch.

In all previous traces similarity is captured by the Euclidean distance. We then assume the dissimilarity cost to be proportional to the squared Euclidean distance, i.e., $c_d(r, y) = b\|r - y\|_2^2$. As the absolute value of such distance has not a clear meaning, we select the constant b so that on average an object can approximate a given fraction α of the catalogue. We say that r can approximate y if $c_d(r, y) \leq c_f = 1$, and we call α the *approximability* value. We set the cache size to $h = 100$.

The time-average cost of different caching policies is shown in Figure 3.12 for the Amazon trace and 10% approximability. Although an object is able to approximate only 10% of the catalog on average, similarity caching policies significantly reduce the cost in comparison to an exact caching policy like LRU, with qLRU-ΔC and Grades/qLRU-ΔC achieving the lowest service cost.

The empirical distribution of pairwise distances in Amazon trace is shown in Figure 3.11. The figure shows that objects are quite scattered in this high-dimensional space with mean distance of around 85,000 between any two objects. The objects in the virtual cache are then in general far from any object in the catalog and we could expect gradient methods to perform poorly. Nevertheless, Figure 3.12 shows that Grades/qLRU-ΔC outperforms existing similarity caching policies. Similar results hold for the other two traces.

Finally, Figure 3.13 reports the costs obtained for the three traces under different values of approximability. As expected, the service cost reduces as the approximability becomes larger. In the CiteULike trace, the service cost flattens rapidly (it is almost constant after 10% approximability): a close look at the dataset shows that popular objects are clustered in a small region of space. Once the approximability value guarantees that these objects can approximate each other, the marginal improvement from further increasing approximability becomes negligible. The other two traces show instead a similar behaviour, with the service cost that is still decreasing after 20% approximability. We also observe that the relative improvement of Grades/qLRU-ΔC in comparison to qLRU-ΔC becomes larger as the approximability increases.

3.2.5 Conclusion

In this chapter, we propose Grades, a new caching policy for similarity caching systems that takes advantage from the fact that objects and requests can many times be embedded in a continuous metric space. Grades outperforms traditional caching policies in stationary scenarios, converging

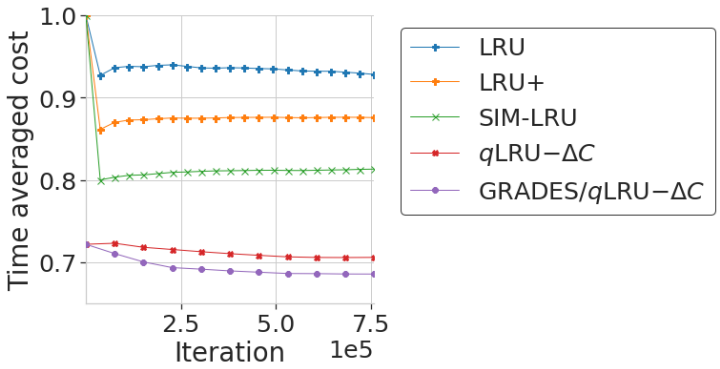
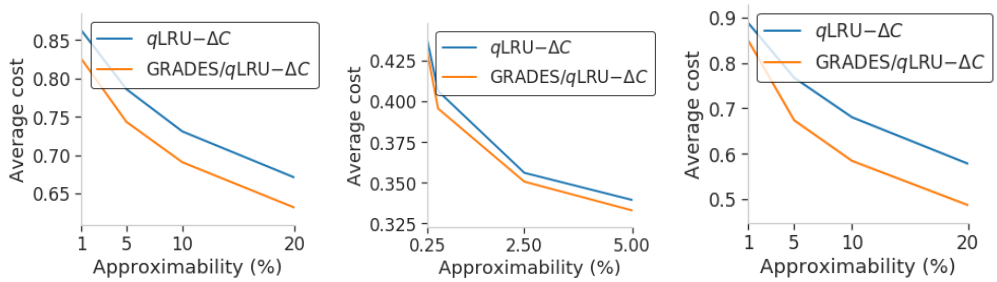


Figure 3.12: The time averaged cost incurred by different policies under the trace *Amazon*: LRU, LRU+, SIM-LRU, qLRU-ΔC ($q = 10^{-3}$) and GRADES/qLRU-ΔC ($\eta = 6.9 \times 10^2$, grafted with p = 1). The level of approximability is 10%. Cache size $h = 100$.



(a) Amazon (b) CiteULike (c) MovieLens

Figure 3.13: The average cost incurred by qLRU-ΔC ($q = 10^{-3}$) and GRADES/qLRU-ΔC (grafted with $p = 1$) under the machine learning traces. The learning rates picked for different approximability levels are: (a) ($\eta = 5.4\times10^2, 6.3\times10^2, 6.9\times10^2, 7.7\times10^2$), (b) ($\eta = 2.4\times10^{-2}, 2.6\times10^{-2}, 3.0\times10^{-2}, 3.2\times10^{-2}$) and (c) ($\eta = 1.6\times10^{-1}, 2.7\times10^{-1}, 3.2\times10^{-1}, 3.8\times10^{-1}$). Cache size $h = 100$.

to provably optimal configurations under mild assumptions. Moreover, we show that GRADES can be grafted to any traditional caching policy, obtaining flexible schemes that achieve arbitrary trade-offs between convergence speed and average costs at steady state. The performance of GRADES and its extensions are evaluated in several synthetic and realistic scenarios.

3.3 Finite Catalog Similarity Caching

3.3.1 Related Work

Consider a remote server that stores a catalog of objects $\mathcal{N} \triangleq \{1, 2, \ldots, N\}$. A similarity search request r aims at finding the k objects $o_1, o_2, \ldots, o_k \in \mathcal{N}$ that are most similar to r given an application-specific definition of *similarity*. To this purpose, similarity search systems rely on a function $c_d(r, o) \in \mathbb{R}_{\geq 0}$, that quantifies the dissimilarity of a request r and an object o. We call such function the *dissimilarity cost*; as was done in Section 3.2.2.

The server replies to each request r with the k most similar objects in the catalog $\mathcal{N}$. As the dissimilarity is captured by the distance in the specific metric space, these objects are also the k closest objects (neighbors) in the catalog to the request r ($k\text{NN}(r, \mathcal{N})$).[4] The mapping translates the similarity search problem in a kNN problem [163, 164]. We can also associate a dissimilarity cost to the reply provided by server (e.g., by summing the dissimilarity costs for all objects in $k\text{NN}(r, \mathcal{N})$). This cost depends on the catalog $\mathcal{N}$ and we do not have control on it. In addition, there is a *fetching cost* to retrieve those objects. The fetching cost captures, for instance, the extra load experienced by the server or the network to provide the objects to the user, the delay experienced by the user or a mixture of those costs.

A common assumption in the existing literature is that the cache can only store h objects and the index needed to manage them has essentially negligible size. We also maintain this assumption that is justified in practice when objects have a size of a few tens of kilobytes (see the quantitative examples in Section 3.3.2).

Caching policies. The performance of the cache depends heavily on which objects the cache stores. Several papers (e.g., [110, 165]) consider the offline object placement problem: a set of objects is selected on the basis of historical information about object popularity and prefetched in the cache. But object popularity can be difficult to estimate and can change over time, specially at the level of small geographical areas (as in the case of areas served by an edge server) [166]. Other papers [118–122, 167] present more a high-level view of the different components of the application system, without specific contributions in terms of cache management policies (e.g., they apply minor changes to exact caching policies like LRU or LFU). Some recent papers [127, 168, 169] propose online caching policies that try to minimize the total cost of the system (the sum of the dissimilarity cost and the fetching cost), also in a networked context [168, 170], but their schemes apply only to the case $k = 1$, which is of limited practical interest.

To the best of our knowledge, the only dynamic caching policies conceived to manage the retrieval of $k > 1$ similar objects are SIM-LRU, CLS-LRU, and RND-LRU proposed in [108] and QCache proposed in [107]. Next, we describe in detail these policies to highlight AÇAI's differences and novelty.

All these policies maintain an ordered list of key-value pairs where the key is a previous request and the value is the set of k' closest objects to the request in the catalog (in general $k' \geq k$). The cache, whose size is h, maintains a set of h/k' past requests. This approach allows to decompose the potentially expensive search for close objects in the cache (see Section 3.3.2) in two separate

[4] More precisely, these are the k objects whose embeddings are closer to the embedding of r. From now on we identify objects and their embeddings.

less expensive searches on smaller sets. Upon the arrival of a request r, the cache identifies the l closest requests to r among the h/k' in the cache. Then, it merges their corresponding values and looks for the k closest objects to r in this set including at most $l \times k'$ objects. If this answer is evaluated to be good enough, then an *approximate hit* occurs and the answer is provided to the user, otherwise the request r is forwarded to the server that needs to provide all k closest objects. The cache state is updated following a LRU-like approach: upon an approximate hit, all key-value pairs that contributed to the answer are moved to the front of the list; upon a miss, the new key-value pair provided by the server is stored at the front of the list and the pair at the end of the list is evicted.

This operation is common to SIM-LRU, CLS-LRU, RND-LRU, and QCACHE. They differ in the choice of the parameters k' and l and in the way to decide between an approximate hit and a miss. As they assume no knowledge about the catalog at the server, they cannot compare the quality (i.e., the dissimilarity cost) of the answer the cache can provide with the quality of the answer the server can provide. They need then to rely on heuristics. We emphasize that the parameters k' and l are only required by the LRU-like policies and do not play a role in AÇAI's workflow.

SIM-LRU (described in Section 3.2.1) considers $k' \geq k$ and $l = 1$ and has the property that no two keys in the cache have a dissimilarity cost lower than $c_\theta \in \mathbb{R}_{\geq 0}$ (a given threshold), but the corresponding hyperspheres may still intersect. CLS-LRU [108] is a variant of SIM-LRU, that can update the stored keys (the centers of the hyperspheres) and push away intersecting hyperspheres to cover the largest possible area of the request space. To this purpose, CLS-LRU maintains the history of requests served at each hypersphere and, upon an approximate hit, moves the center to the object that minimizes the distance to every object within the hypersphere's history. When two hyperspheres overlap, this mechanism drives their centers apart, which in turn reduces the overlapping region. Finally, QCACHE [107] considers $k' = k$ and $l > 1$. The policy decides if the k objects selected from the cache are an approximate hit if (1) at least two of them would have been provided also by the server—a sufficient condition can be obtained from geometric considerations—or (2) the distribution of distances of the k objects from the request looks similar to the distribution of objects around the corresponding request for other stored key-value pairs.

These policies share potential inefficiencies: (1) the sets of closest objects to previous queries are not necessarily disjoint (but CLS-LRU tries to reduce their overlap) and then the cache may store less than h distinct objects; (2) the two-level search may miss some objects in the cache that are close to r, but are indexed by requests that are not among the l closest requests to r; (3) the policy takes into account the dissimilarity costs at the caches but not at the server; (4) objects are served in bulk, all from the cache or all from the server, without the flexibility of a per-object choice. As we are going to see, AÇAI design prevents such inefficiencies by exploiting new advances in efficient approximate kNN search algorithms, which allows us to abandon the key-value pair indexing and to estimate the dissimilarity costs at the server. Also AÇAI departs from the LRU-like cache updates, considering gradient update schemes inspired by online learning algorithms [171].

3.3.2 Other Relevant Background

Indexes for approximate kNN search. Indexes are used to efficiently search objects in a large catalog. In case of kNN, one of the approaches is to use tree-based data structures. Unfortunately, in high dimensional spaces, e.g., $\mathbb{R}^d$ with $d > 10$, the computational cost of such search is comparable

to a full scan of the collection [126]. *Approximate Nearest Neighbor* search techniques trade accuracy for speed and provide k points close to the query, but not necessarily the closest, sometimes with a guaranteed bounded error. Prominent examples are the solutions based on locality sensitive hashing [140], product quantization [141, 142], and graphs [144]. Despite being approximate, these indexes are in practice very accurate, as showed over different benchmarks in [154].

As we are going to describe, AÇAI employs two approximate indexes (both stored at the edge server): one for the content stored in the cache, and one for the whole catalog $\mathcal{N}$ stored in the remote server. For the former, since cache content varies over time, we rely on a graph-based solution, such as HNSW [144], that supports dynamic (re-)indexing with no speed loss. On various benchmarks [154], HNSW results the fastest index, and it is able to answer a 100NN query over a dataset with 1 million objects in a 128-dimensional space in less than 0.5 ms with a recall greater than 97%. As for the memory footprint, a typical configuration of the HNSW index requires $O(d)$ bytes per objects, where d is the number of dimensions. For instance, in case of $d = 128$ dimensional vectors, the memory required to index 10 million objects is approximately 5 GB. As the server catalog changes less frequently (e.g., contextual advertising applications [108], and image retrieval applications [107]), AÇAI can index it using approaches with a more compact object representation like FAISS [141]. FAISS is slightly slower than HNSW and does not support fast re-indexing if the catalog changes, but it can manage a much larger set of objects. With a dataset of 1 billion objects, FAISS provides an answer in less than 0.7 ms per query, using a GPU [141]. Practically, the global catalog index can be fully reconstructed whenever a given predefined percentage of the catalog changes. This operation can be done in a parallel procedure to AÇAI without hindering its normal workflow. When a new global catalog index is reconstructed, the state of AÇAI is modified accordingly by redistributing the total mass on the contents that disappeared on the new introduced contents. Moreover, the states corresponding to the disappearing (resp., appearing) items are removed (resp., added). As for the memory footprint, for a typical configuration (IVFPQ), FAISS is able to represent an object with 30 bytes (independently from d): only 3 GB for a dataset with 100 million objects!

Summing up our numerical example, if each object has size 20 KB, an edge server with AÇAI storing locally 10 million objects from a catalog with 100 million objects, needs 200 GB for the objects and only 8 GB for the two indexes. The larger the objects, the smaller the indexes' footprint: for example, when the server has a few Terabytes of disk space to store large multimedia objects, the indexes' size can be ignored.

Gradient descent approaches. Online caching policies based on gradient methods have been studied in the stochastic request setting for exact caching, with provable performance guarantees [21,38]. More recently, the authors of [169] have proposed a gradient method to refine the allocation of objects stored by traditional similarity caching policies like SIM-LRU. Similarly, the reference [168] considers a heuristic based on the gradient descent/ascent algorithm to allocate objects in a network of similarity caches. In both papers, the system provides a single similar content ($k = 1$). A closely related recent work [125] considers the problem of allocating different inference models that can satisfy users' queries at different quality levels. The authors propose a policy based on mirror descent, and provide guarantees under a general request process. But, their policy does not scale to a large catalog size.

We deviate from these works by considering $k > 1$, large catalog size, and the more general family of online mirror ascent algorithms (of which the usual gradient ascent method is a particular

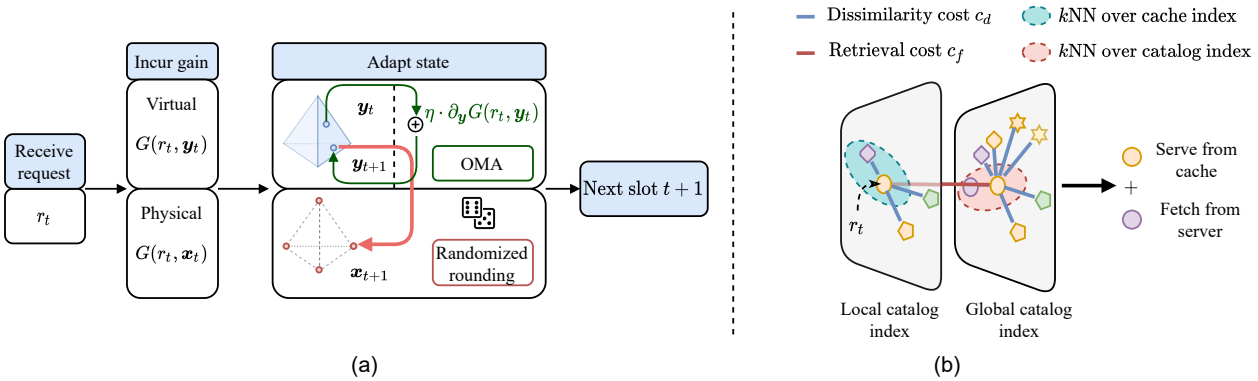


Figure 3.14: Subfigure (a) illustrates AÇAI's state adaptation. A time slot is initiated when a request r_t is received. A virtual (fictitious) gain $G(r_t, \boldsymbol{y}_t)$ and a physical gain $G(r_t, \boldsymbol{x}_t)$ are incurred. The virtual cache adapts its fractional state by calling Online Mirror Ascend to obtain a new state $\boldsymbol{y}_{t+1} \in \text{conv}(\mathcal{X})$ employing the subgradient of the virtual gain $\partial_{\boldsymbol{y}} G(r_t, \boldsymbol{y}_t)$, and the new state is randomly rounded to a valid cache state $\boldsymbol{x}_{t+1} \in \mathcal{X}$. Subfigure (b) depicts how AÇAI employs the two indexes (local catalog index and global catalog index). An approximate kNN queries are performed on each index, and the contents with the least overall costs are selected.

instance). Also our policy provides strong performance guarantees under a general request process, where requests can even be selected by an adversary. Our analysis relies on results from online convex optimization [20] and is similar in spirit to what done for exact caching using the classic gradient method in [171] and mirror descent in [139]. Two recent papers [13, 50] pursued this line of work taking into account update costs for a single exact cache.

3.3.3 System Description and AÇAI's Design

AÇAI's design is summarized in Figure 3.14, and the notation used across this section is provided in Table 3.3.

3.3.3.1 Cost Assumptions

Many of the similarity caching policies proposed in the literature (including SIM-LRU, CLS-LRU, RND-LRU, and QCACHE) have not been designed with a clear quantitative objective, but with the qualitative goal of significantly reducing the fetching cost without increasing too much the dissimilarity cost. Because of such vagueness, the corresponding papers do not make clear assumptions about the dissimilarity costs and the fetching costs. On the contrary, AÇAI has been designed to minimize the total cost of the similarity search system and we make explicit the corresponding hypotheses.

Our main assumption is that all costs are additive. The function $c_d(r, o)$ introduced in Section 3.3.1 quantifies the dissimilarity of the object o and the request r. Let $\mathcal{A}$ be the set of objects in the answer to request r: it is natural to consider as dissimilarity cost of the answer $\sum_{o \in \mathcal{A}} c_d(r, o)$.

	Notational Conventions
$\mathbb{1}(\chi)$	Indicator function set to 1 when condition χ is true
$[n]$	Set of integers $\{1, 2, \ldots, n\}$
$\text{conv}(S)$	Convex hull of a set S
	System Model
$\mathcal{N}$	Catalog of N objects
$\mathcal{U}$	Augmented catalog of $2N$ objects
x_i	Set to 1 when $i \in \mathcal{N}$ is cached
h	Cache capacity
r / $\mathcal{R}$	Request / Request set
c_f	Retrieval cost
$c_d(r, o)$	Dissimilarity cost of serving object o to request r
$c(r, o)$	Overall cost of serving object o to request r
π^r	π^r_i gives the i-th closest object to r
α^r_i	Cost difference between the $(i+1)$-th smallest cost and the i-th smallest cost of serving request r
K^r	The order of the largest possible cost when r is requested.
$k\text{NN}(r, S)$	Set of k closest objects to r in $S \subset \mathcal{N}$ according to $c(r, \cdot)$
$\boldsymbol{x}$ / $\mathcal{X}$	Cache state / Set of valid cache states
t / T	Time slot / Time horizon
$C(r, \boldsymbol{x})$	Total cost to serve request r under $\boldsymbol{x}$
$G(r, \boldsymbol{x})$	Total gain to serve request r under $\boldsymbol{x}$
$C_{\text{UC},T}$	Systems's update cost over T requests
$G_T(\boldsymbol{x})$	Time-averaged caching gain
	AÇAI
Φ	Mirror map
$\mathcal{D}$	Domain of the mirror map
$\boldsymbol{y}$	Fractional cache state
η	Learning rate
$\boldsymbol{g}_t$	Subgradient of $G(r_t, \boldsymbol{y})$ at point $\boldsymbol{y}_t$
$\prod^{\Phi}_S(\cdot)$	Bregman projection onto S
M	Freezing period
c^k_d	Upper bound on the dissimilarity cost of the k-th closest object
ψ	Static optimum discount factor

Table 3.3: Notation Summary for Subsection 3.2.3.3

In addition, if fetching a single object from the server incurs a cost $c_f \in \mathbb{R}_{>0}$, the fetching cost to retrieve m objects is $m \times c_f$. This is an obvious choice when c_f captures server or network cost. When c_f captures the delay experienced by the user, then summing the costs is equivalent to consider the round trip time negligible in comparison to the transmission time, which is justified for large multimedia objects. It is easy to modify AÇAI to consider the alternative case when the fetching cost does not depend on how many objects are retrieved. Finally, as common in other works [127, 169], we assume that both the dissimilarity cost and the fetching cost can be directly compared (e.g., they can both be converted in dollars). Under these assumptions, when, for example, the k nearest neighbors in $\mathcal{N}$ to the query r ($k\text{NN}(r, \mathcal{N})$) are retrieved from the remote server, the total cost experienced by the system is $\sum_{o \in k\text{NN}(r,\mathcal{N})} c_d(r, o) + kc_f$.

3.3.3.2 Cache Indexes

AÇAI departs from the key-value indexes of most the similarity caching policies. As discussed in Section 3.3.1, such an approach was essentially motivated by the need to simplify kNN searches by performing two searches on smaller datasets (the set of keys first, and then the union of the values for l keys), and may lead to potential inefficiencies including sub-utilization of the available caching space.

The two-level search implemented by existing similarity caching policies can be seen as a naïve way to implement an approximate kNN search on the set of objects stored locally (the *local catalog* C). Thanks to the recent advances in approximate kNN searches (Section 3.3.2), we have now better approaches to search through large catalogs with limited memory and computation requirements. We assume then that the cache maintains two indexes supporting kNN searches: one for the local catalog (the objects stored locally) and one for the remote catalog (the objects stored at the server). A discussion about which approximate index is more appropriate for each catalog is in Section 3.3.2.

The local catalog index allows AÇAI to (1) fully exploit the available space (the cache stores at any time h objects and can perform a kNN search on all of them), (2) potentially find closer objects in comparison to the non-optimized key-value search. Instead, the remote catalog index allows AÇAI

to evaluate what objects the server would provide as answer to the request, and then to correctly evaluate which objects should be served locally and which one should be served from the server, as we are going to describe next.

3.3.3.3 Request Serving

Differently from existing policies, AÇAI has the possibility to compose the answer using both local objects and remote ones. Upon a request r, AÇAI uses the two indexes to find the closest objects from the local catalog C and from the remote catalog $\mathcal{N}$. We denote the set of objects identified by these indexes as $k\text{NN}(r, C)$ and $k\text{NN}(r, \mathcal{N})$ respectively. AÇAI composes the answer $\mathcal{A}$ by combining the objects with the smallest costs in the two sets. For an object o stored locally ($o \in C$), the system only pays $c_d(r, o)$; for an object o fetched from the remote server ($o \in \mathcal{N} \setminus C$), the system pays $c_d(r, o) + c_f$. The total cost experienced is

$$C(r, \mathcal{A}) \triangleq \sum_{o \in \mathcal{A} \cap k\text{NN}(r,C)} c_d(r, o) + \sum_{o \in \mathcal{A} \setminus k\text{NN}(r,C)} \left(c_d(r, o) + c_f\right). \tag{3.5}$$

The answer $\mathcal{A}$ is determined by selecting k objects that minimize the total cost, that is

$$\mathcal{A} = \underset{\substack{\mathcal{B} \subset (k\text{NN}(r,C) \cup k\text{NN}(r,\mathcal{N})) \\ |\mathcal{B}|=k}}{\arg\min} C(r, \mathcal{B}). \tag{3.6}$$

3.3.3.4 Cache State and Service Cost/Gain

In order to succinctly present how AÇAI updates the local catalog and its theoretical guarantee, it is convenient to express the cost in (3.5) as a function of the current cache state and replace the set notation with a vectorial one.

First, we define the *augmented catalog* $\mathcal{U} \triangleq \mathcal{N} \cup \{N+1, N+2, \ldots, 2N\}$ and define the new costs

$$c(r, i) = \begin{cases} c_d(r, i), & \text{if } i \in \mathcal{N}, \\ c_d(r, i - N) + c_f, & \text{if } i \in \mathcal{U} \setminus \mathcal{N}. \end{cases} \tag{3.7}$$

Essentially, i and $i + N$ (for $i \in \{1, \ldots, N\}$) correspond to the same object, with i capturing the cost when the object is stored at the cache and $i + N$ capturing the cost when it is stored at the server. From now on, when we talk about the closest objects to a request, we are considering $c(\cdot, \cdot)$ as the distance.

Note that AÇAI can easily be modified to account for heterogeneous retrieval costs by modifying Eq. (3.7) and replacing c_f by an object dependent retrieval cost $c_{f,i}$ for every object $i \in \mathcal{N}$. Moreover, we assume that the fetching cost and dissimilarity cost are added together linearly in the objective, however our model can capture the scenario where the cost is not necessarily additive in c_d and c_f by a redefinition of the second line in Eq. (3.7). In particular, we can observe that the algorithm only requires the existence of a function $c(r, i)$ and the particular form of this function can be arbitrary. To streamline the presentation we considered this simplified model and the theoretical guarantees will also hold under such modifications.

It is also convenient to represent the state of the cache (the set of objects stored locally) as a vector $\boldsymbol{x} \in \{0,1\}^{2N}$, where, for $i \in \mathcal{N}$, $x_i = 1$ (resp., $x_i = 0$), if i is stored (resp., is not stored) in the cache, and we set $x_{i+N} = 1 - x_i$.[5] The set of valid cache configurations is given by:

$$\mathcal{X} \triangleq \left\{ \boldsymbol{x} \in \{0,1\}^{2N} : \sum_{i \in \mathcal{N}} x_i = h,\ x_{j+N} = 1 - x_j, \forall j \in \mathcal{N} \right\}. \tag{3.8}$$

For every request $r \in \mathcal{R}$ we define the sequence π^r as the permutation of the elements of $\mathcal{U}$, where π^r_i gives the i-th closest object to r in $\mathcal{U}$ according to the costs $c(r,o), \forall o \in \mathcal{U}$. The answer $\mathcal{A}$ provided by AÇAI (Eq. (3.6)) coincides with the first k elements of π^r for which the corresponding index in $\boldsymbol{x}$ is equal to 1. The total cost to serve r can then be expressed directly as a function of the cache state $\boldsymbol{x}$:

$$C(r,\boldsymbol{x}) = \sum_{i=1}^{2N} c(r,\pi^r_i) x_{\pi^r_i} \mathbb{1}\left(\sum_{j=1}^{i-1} x_{\pi^r_j} < k \right), \forall \boldsymbol{x} \in \mathcal{X}. \tag{3.9}$$

where $\mathbb{1}(\chi) = 1$ when the condition χ is true, and $\mathbb{1}(\chi) = 0$ otherwise.

Instead of working with the cost $C(r,\boldsymbol{x})$, we can equivalently consider the *caching gain* defined as the cost reduction due to the presence of the cache (as in [21, 63, 172]):

$$G(r,\boldsymbol{x}) \triangleq C(r, (\underbrace{0,0,\ldots,0}_{N}, \underbrace{1,1,\ldots,1}_{N})) - C(r,\boldsymbol{x}), \tag{3.10}$$

where the first term corresponds to the cost when the cache is empty (and then requests are entirely satisfied by the server). The theoretical guarantees of AÇAI are simpler to express in terms of the caching gain (Section 3.3.3.8). Observe that the caching gain is zero for any cache state when the retrieval cost $c_f = 0$, and this corresponds to the setting where the remote server and the cache are co-located and this scenario overrides the need of a cache.

The caching gain has the following compact expression (Appendix 14.3):

$$G(r,\boldsymbol{x}) = \sum_{i=1}^{K^r-1} \alpha^r_i \min\left\{ k - \sigma^r_i, \sum_{j=1}^{i} x_{\pi^r_j} - \sigma^r_i \right\}, \tag{3.11}$$

where

$$\sigma^r_i \triangleq \sum_{j=1}^{i} \mathbb{1}\left(\pi^r_j \in \mathcal{U} \setminus \mathcal{N}\right), \qquad \forall (i,r) \in \mathcal{U} \times \mathcal{R}, \tag{3.12}$$

K^r is the value of the minimum index $i \in \mathcal{U}$ such that $\sigma^r_i = k$, and $\alpha^r_i \triangleq c(r,\pi^r_{i+1}) - c(r,\pi^r_i) \geq 0$.

Let $\mathrm{conv}(\mathcal{X})$ denote the convex hull of the set of valid cache configurations $\mathcal{X}$. We observe that $G(r,\boldsymbol{y})$ is a concave function of variable $\boldsymbol{y} \in \mathrm{conv}(\mathcal{X})$. Indeed, from Eq. (3.11), $G(r,\boldsymbol{y})$ is a linear combination, with positive coefficients, of concave functions (the minimum of affine functions in $\boldsymbol{y}$).

[5] The vector $\boldsymbol{x}$ has clearly redundant components, but such redundancy leads to more compact expressions in what follows.

Algorithm 3.2 Online Mirror Ascent (OMA)

Input: $\eta \in \mathbb{R}_+$, RoundingScheme

1: **procedure** OnlineMirrorAscent
2: $\boldsymbol{y}_1 \leftarrow \underset{\boldsymbol{y} \in \text{conv}(\mathcal{X}) \cap \mathcal{D}}{\arg\min} \Phi(\boldsymbol{y})$; $\boldsymbol{x}_1 \leftarrow$ DepRound$(\boldsymbol{y}_1)$
3: **for** $t \leftarrow 1, 2, \ldots, T$ **do** ▷ Incur a gain $G(r_t, \boldsymbol{y}_t)$, and compute a subgradient $\boldsymbol{g}_t$ of $G(r_t, \boldsymbol{y})$ at point $\boldsymbol{y}_t$ (Appendix 18)
4: $\hat{\boldsymbol{y}}_t \leftarrow \nabla\Phi(\boldsymbol{y}_t)$ ▷ Map primal point to dual point
5: $\hat{\boldsymbol{z}}_{t+1} \leftarrow \hat{\boldsymbol{y}}_t + \eta \boldsymbol{g}_t$ ▷ Take gradient step in the dual
6: $\boldsymbol{z}_{t+1} \leftarrow (\nabla\Phi)^{-1}(\hat{\boldsymbol{z}}_{t+1})$ ▷ Map dual point to a primal point
7: $\boldsymbol{y}_{t+1} \leftarrow \prod^{\Phi}_{\text{conv}(\mathcal{X}) \cap \mathcal{D}}(\boldsymbol{z}_{t+1})$ ▷ Proj. new point onto feasible region
▷ Select a rounding scheme
8: **if** RoundingScheme = DepRound **then**
9: **if** $M \mid t$ **then** ▷ Round the fractional state every M requests
10: $\boldsymbol{x}_{t+1} \leftarrow$ DepRound$(\boldsymbol{y}_{t+1})$
11: **end if**
12: **else if** RoundingScheme = CoupledRounding **then**
13: $\boldsymbol{x}_{t+1} \leftarrow$ CoupledRounding$(\boldsymbol{x}_t, \boldsymbol{y}_t, \boldsymbol{y}_{t+1})$
14: **end if**
15: **end for**
16: **end procedure**

3.3.3.5 Cache Updates

We denote by $r_t \in \mathcal{R}$ the t-th request. The cache is allowed to change its state $\boldsymbol{x}_t \in \mathcal{X}$ to $\boldsymbol{x}_{t+1} \in \mathcal{X}$ in a reactive manner, after receiving the request r_t and incurring the gain $G(r_t, \boldsymbol{x}_t)$. AÇAI updates its state $\boldsymbol{x}_t$ with the goal of greedily maximizing the gain. The update of the state $\boldsymbol{x}_t$ is driven from a continuous fractional state $\boldsymbol{y}_t \in \text{conv}(\mathcal{X})$, where $y_{t,i}$ can be interpreted as the probability to store object i in the cache. At each request r_t, AÇAI increases the components of $\boldsymbol{y}_t$ corresponding to the objects that are used to answer to r_t, and decreases the other components. This could be achieved by a classic gradient method, e.g., $\boldsymbol{y}_{t+1} = \boldsymbol{y}_t + \eta \mathrm{g}_t$, where g_t is a subgradient of $G(r_t, \boldsymbol{y}_t)$ and $\eta \in \mathbb{R}_+$ is the learning rate (or stepsize), but in AÇAI we consider a more general online mirror ascent update OMA [53, Ch. 4] that is described in Algorithm 3.2.[6] OMA is parameterized by the function $\Phi(\cdot)$, that is called the *mirror map* (see Section 2.2.3.2). If the mirror map is the squared Euclidean norm, OMA coincides with the usual gradient ascent method, but other mirror maps can be selected. In particular, our experiments in Section 3.3.4 show that the negative entropy map $\Phi(\boldsymbol{y}) = \sum_{i \in \mathcal{N}} y_i \log y_i$ with domain $\mathcal{D} = \mathbb{R}^N_{>0}$ achieves better performance.

3.3.3.6 Rounding the Cache Auxiliary State

At every time slot $t \in [t]$, AÇAI can use the randomized rounding scheme DepRound [173] to generate a cache allocation $\boldsymbol{x}_{t+1} \in \mathcal{X}$ from $\boldsymbol{y}_{t+1} \in \text{conv}(\mathcal{X})$, while still satisfying the capacity constraint at any time slot t. The cache can fetch from the server the objects that are in $\boldsymbol{x}_{t+1}$ but not in $\boldsymbol{x}_t$.

As cache movements/updates introduce extra costs to the network operator, DepRound could potentially cause extra update costs that grow linearly in time. To mitigate incurring large update

[6] Properly speaking OMA, only refers to the update of $\boldsymbol{y}_t$ and does not include the randomized rounding schemes in lines 8–14.

costs, we may avoid updating the cache state at every time slot $t \in [t]$ by *freezing* the cache physical state for $M \in [t]$ time steps. In particular, we assume the update cost of the system to be proportional to the number of fetched files, which can be upper bounded by the l_1 norm of the state update: $\sum_{i\in\mathcal{N}} \max\{0, x_{t+1,i} - x_{t,i}\} \leq \|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_1$. Hence, if we denote by $C_{\text{UC},T}$ the total update cost of the system over the time horizon T, then we have

$$C_{\text{UC},T} = O\left(\sum_{t=1}^{T-1} \|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_1\right). \tag{3.13}$$

When the cache state is refreshed after a call to the rounding scheme DepRound, the incurred update cost is in the order of $O(2h)$, and $\frac{C_{\text{UC},T}}{T} = O\left(\frac{2h}{M}\right)$. Moreover, when $M = \Theta\left(T^{\beta}\right)$ for $\beta \in (0,1)$ it holds $\frac{C_{\text{UC},T}}{T} = O\left(T^{-\beta}\right)$, and for any $\epsilon > 0$ and T large enough

$$\frac{C_{\text{UC},T}}{T} \leq \epsilon. \tag{3.14}$$

The average update cost of the system is then negligible for large T. The parameter M reduces cache updates at the expense of reducing the cache reactivity (see Theorem 3.3.3).

In some applications, it is possible to slightly violate the capacity constraint with small deviations, as long as this is satisfied on average [151, 174, 175]. For example, there could be a monetary value associated to the storage reserved by the cache, and a total budget available over a target time horizon T. In this setting, the cache may violate momentarily the capacity constraint, as far as the total payment does not exceed the budget.

CoupledRounding (Algorithm 3.3) is a rounding approach which works under this relaxed capacity constraint and does not require state freezing with the parameter M. At time slot $t \in [t]$, the cache decides which files to cache through N coin tosses, where the file $i \in \mathcal{N}$ is cached with probability $y_{t,i}$, and the random state obtained is $\boldsymbol{x}_t$. By definition, the expected value of the integral state is $\mathbb{E}[\boldsymbol{x}_t] = \boldsymbol{y}_t$. The probability that the cache exceeds its storage capacity by δh is given by the Chernoff bound [176] as:

$$\mathbb{P}\left(\|\boldsymbol{x}_t\|_1 > (1+\delta)h\right) < e^{\frac{-\delta^2 h}{2}}, \delta \in (0,1], \tag{3.15}$$

where the l_1 norm is restricted to the first N components of the vector, i.e., $\|\boldsymbol{x}\|_1 \triangleq \sum_{i\in\mathcal{N}} |x_i|$. In the regime of large cache sizes $h \gg 1$, we observe from the Chernoff bound that the cache stores less than $(1+\delta)$ of its capacity h with high probability.

Theorem 3.3.1 (proof in Appendix 20.1) shows that the expected movement of CoupledRounding is equal to the movement of the fractional auxiliary states $\{\boldsymbol{y}_t\}_{t=1}^{T}$.

Theorem 3.3.1. *If the input to Algorithm 3.3 is sampled from a random variable $\boldsymbol{x}_t \in \{0,1\}^N$ with $\mathbb{E}[\boldsymbol{x}_t] = \boldsymbol{y}_t$, then we obtain as output an integral cache configuration $\boldsymbol{x}_{t+1} \in \{0,1\}^N$ satisfying $\mathbb{E}[\boldsymbol{x}_{t+1}] = \boldsymbol{y}_{t+1}$ and $\mathbb{E}\left[\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_1\right] = \|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\|_1$.*

Moreover, the movement of the fractional states is negligible for large T:

Algorithm 3.3 Coupled Rounding

Input: $\boldsymbol{x}_t, \boldsymbol{y}_t, \boldsymbol{y}_{t+1}$ ▷ $\boldsymbol{x}_t$ satisfies $\mathbb{E}[\boldsymbol{x}_t] = \boldsymbol{y}_t$

1: **procedure** CoupledRounding
2: $\delta \leftarrow \boldsymbol{y}_{t+1} - \boldsymbol{y}_t$ ▷ Compute the change in distribution
3: **for** $i \in \mathcal{N}$ **do**
4: **if** $\left(x_{t,i} = 1\right) \wedge (\delta_i < 0)$ **then**
5: $x_{t+1,i} \leftarrow 0$ w.p. $-\frac{\delta_i}{y_{t,i}}$, and $x_{t+1,i} \leftarrow 1$ w.p. $\frac{y_{t,i}+\delta_i}{y_{t,i}}$
6: **else if** $\left(x_{t,i} = 0\right) \wedge (\delta_i > 0)$ **then**
7: $x_{t+1,i} \leftarrow 0$ w.p. $\frac{1-y_{t,i}-\delta_i}{1-y_{t,i}}$, and $x_{t+1,i} \leftarrow 1$ w.p. $\frac{\delta_i}{1-y_{t,i}}$
8: **else**
9: $x_{t+1,i} \leftarrow x_{t,i}$ ▷ Keep the same state
10: **end if**
11: $x_{t+1,i+N} \leftarrow 1 - x_{t+1,i}$ ▷ Update the augmented states
12: **end for**
13: **return** $\boldsymbol{x}_{t+1}$ ▷ Return the next physical state satisfying $\mathbb{E}[\boldsymbol{x}_{t+1}] = \boldsymbol{y}_{t+1}$
14: **end procedure**

Theorem 3.3.2. *Algorithm 3.2, configured with the negative entropy mirror map and learning rate* $\eta = O\left(\frac{1}{\sqrt{T}}\right)$*, selects fractional cache states satisfying*

$$\sum_{t=1}^{T-1} \|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\|_1 = O(\sqrt{T}). \tag{3.16}$$

The proof is in Appendix 20.2. Combining the two theorems and (3.13), we also conclude that the expected average update cost of the system $\mathbb{E}\left[\frac{C_{\text{UC},T}}{T}\right]$ is negligible for large T.

3.3.3.7 Time Complexity

AÇAI uses OMA in Algorithm 3.2 coupled with a rounding procedure DepRound or CoupledRounding. The rounding step may take $O(N)$ operations (amortized every M requests when DepRound is used). In practice, AÇAI quickly sets irrelevant objects in the fractional allocation vector $\boldsymbol{y}_t$ very close to 0. Therefore, we can keep track only of objects with a fractional value above a threshold $\epsilon > 0$, and the size of this subset is practically of the order of h.

Similarly, subgradient computation may require $O(N)$ operations per each component and then have $O\left(N^2\right)$ complexity, but in practice, as the vector $\boldsymbol{y}_t$ is sparse, calculations in Appendix 3.3.3.8 Eq. 18.151 require only a constant number of operations and complexity reduces to $O(N)$.

Finally, we use the negative entropy Bregman projection in [139] (line 6 of Algorithm 3.2) that has $O(N + h\log(h))$ time complexity. The $O(N + h\log(h))$ is due to a partial sorting operation while the actual projection takes $O(h)$. Again, most of the components of $\boldsymbol{y}_t$ are equal to 0, so that, in practice, we need to sort much less points.

3.3.3.8 Theoretical Guarantees

The best static cache allocation in hindsight is the cache state $\boldsymbol{x}_*$ that maximizes the time-averaged caching gain in Eq. (3.10) over the time horizon T, i.e.,

$$\boldsymbol{x}_* \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(G_T(\boldsymbol{x}) \triangleq \frac{1}{T}\sum_{t=1}^{T} G(r_t, \boldsymbol{x}) \right). \tag{3.17}$$

We observe that solving (3.17) is NP-hard in general even for $k = 1$ under a stationary request process [127]. Nevertheless, AÇAI operates in the *online setting* and provides guarantees in terms of the ψ-regret [177]. In this scenario, the regret is defined as gain loss in comparison to the best static cache allocation $\boldsymbol{x}_*$ in (3.17). The ψ-regret discounts the best static gain by a factor $\psi \in (0, 1]$. Formally,

$$\psi\text{-Regret}_{T,\mathcal{X}}(\text{OMA}_\Phi) = \sup_{\{r_1,r_2,\ldots,r_T\}\in\mathcal{R}^T} \left\{ \psi \sum_{t=1}^{T} G(r_t, \boldsymbol{x}_*) - \mathbb{E}\left[\sum_{t=1}^{T} G(r_t, \boldsymbol{x}_t) \right] \right\}, \tag{3.18}$$

where the expectation is over the randomized choices of DepRound. Note that the supremum in (3.18) is over all possible request sequences. This definition corresponds to the so called *adversarial analysis*, imagining that an adversary selects requests in $\mathcal{R}$ to jeopardize cache performance. The definition of the regret in Eq. (3.18), which compares the gain of the policy to a static offline solution, is classic. Several bandit settings, e.g., simple multi-armed bandits [75, 80, 81], contextual bandits [82–84], and, of course, their applications to caching problems under the full-information setting [49,50,139,171,178], adopt this definition. In all these cases, the dynamic, adaptive algorithm is compared to a static policy that has full hindsight of the entire trace of actions. Moreover, as is customary in the context of online problems in which the offline problem is NP-hard [76], the regret is not w.r.t. the optimal caching gain, but the gain obtained by an offline approximation algorithm.

Obviously, regret bounds in the adversarial setting provide strong robustness guarantees in practical scenarios. AÇAI has the following regret guarantee:

Theorem 3.3.3. *Algorithm 3.2 configured with the negative entropy mirror map, learning rate* $\eta = \frac{1}{(c_d^k+c_f)}\sqrt{\frac{2\log\left(\frac{N}{h}\right)}{T+(M-1)(M+T)}}$*, and rounding scheme* CoupledRounding *or* DepRound *with freezing period* $M = \Theta\left(T^\beta\right)$ *for* $\beta \in [0, 1)$*, has a sublinear* $(1 - 1/e)$*-regret in the number of requests, i.e.,*

$$(1-1/e)\text{-Regret}_{T,\mathcal{X}}(\text{OMA}_\Phi) \leq \left(1 - \frac{1}{e}\right)(c_d^k + c_f)h\sqrt{2\log\left(\frac{N}{h}\right)((M-1)(T+M)+T)},$$

where the constant c_d^k *is an upper bound on the dissimilarity cost of the k-th closest object for any request in* $\mathcal{R}$*.*

We first prove that the expected gain of the randomly sampled allocations $\boldsymbol{x}_t$ is a $(1-1/e)$-approximation of the fractional gain. Then, we use online learning results [53] to bound the regret of OMA schemes operating on a convex decision space against concave gain functions picked by an

adversary. The two results are combined to obtain an upper bound on the $(1-1/e)$-regret. The full proof is available in Appendix 21.

The $(1-1/e)$-regret of AÇAI under CoupledRounding scheme with its corresponding freezing period $M = 1$ has order-optimal regret $O\left(\sqrt{T}\right)$ [20], whereas under the rounding scheme DepRound with a corresponding freezing period $M = \Theta\left(T^{\beta}\right)$ the reduced reactivity of AÇAI is reflected by the additional $T^{\frac{\beta}{2}}$ factor in the order of the regret. Nonetheless, the expected time-average $(1-1/e)$-regret of AÇAI can get arbitrarily close to zero for large time horizon. Hence, AÇAI performs on average as well as a $(1-1/e)$-approximation of the optimal configuration $\boldsymbol{x}_*$. This observation also suggests that our algorithm can be used as an iterative method to solve the NP-hard static allocation problem with the best approximation bound achievable for this kind of problems [179].

Corollary 3.3.4. *(offline solution) Let $\bar{\boldsymbol{y}}$ be the average fractional allocation $\bar{\boldsymbol{y}} = \frac{1}{\tilde{T}}\sum_{i=1}^{\tilde{T}} \boldsymbol{y}_i$ of* AÇAI, *and $\bar{\boldsymbol{x}}$ the random state sampled from $\bar{\boldsymbol{y}}$ through* CoupledRounding *or* DepRound. *If Algorithm 3.2 is configured with the negative entropy mirror map, and, at each iteration $t \in [\tilde{T}]$, operates with subgradients of the time-averaged caching gain* (3.17), *then $\forall \epsilon > 0$ and over a sufficiently large number of iterations $\tilde{T}$, $\bar{\boldsymbol{x}}$ satisfies*

$$\mathbb{E}\left[G_T(\bar{\boldsymbol{x}})\right] \geq \left(1 - \frac{1}{e} - \epsilon\right) G_T(\boldsymbol{x}_*).$$

where $\boldsymbol{x}_* = \underset{\boldsymbol{x}\in\mathcal{X}}{\arg\max}\, G_T(\bar{\boldsymbol{x}})$.

The proof can be found in Appendix 22.

3.3.4 Experiments

We start evaluating AÇAI in a simple scenario with a synthetic request process, for which we can compute the optimal fractional static cache allocation. We then consider real-world catalogs and traces and compare our solution with state of the art online policies proposed for kNN caching, i.e., SIM-LRU [108], CLS-LRU [108], and QCache [107] described in Section 3.3.1.

3.3.4.1 Simple Scenario

We consider a synthetic catalog as in [169] of $N = 900$ objects positioned on a 30×30 grid. The request process is generated according to the Independent Reference Model [15]. The objects' popularity is represented by a Gaussian distribution. In particular, an object $o \in \mathcal{N}$ with an l_1 (norm-1) distance d_o from the center of the grid (15, 15) is requested at any time slot t with probability $p_o \propto e^{-\frac{d_o^2}{2\times 6^2}}$. The synthetic catalog is depicted in Figure 3.15.

We consider the dissimilarity cost to be the l_1 distance. We take different values for the retrieval cost $c_f \in \{1, 2, 3, 4\}$ and the number of neighbors $k \in \{1, 2, 3, 4, 5\}$. We take the cache capacity to be $h = 15$. We configure AÇAI with DepRound rounding scheme.

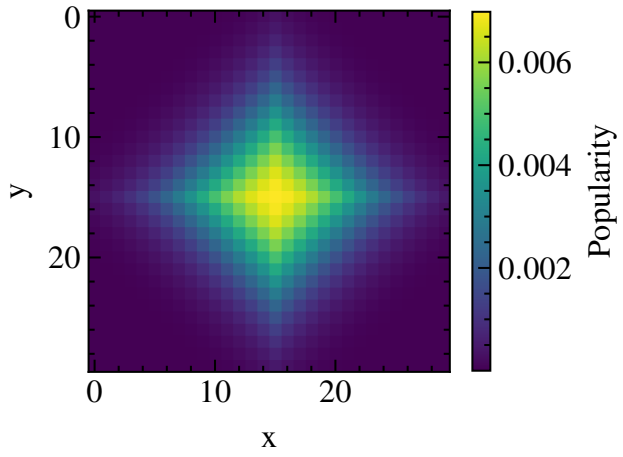
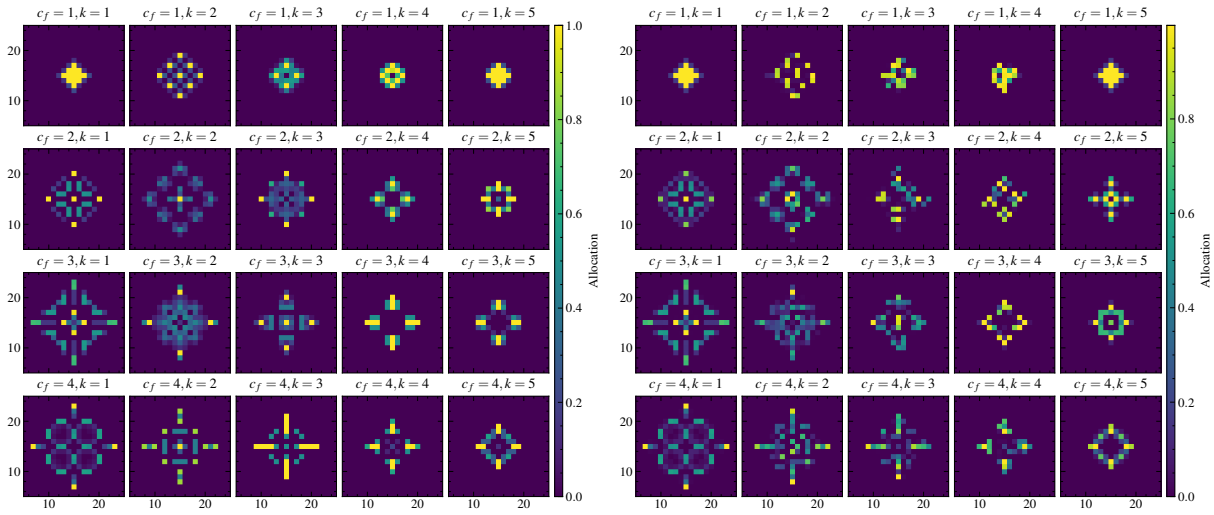


Figure 3.15: Synthetic catalog of objects located on a 30×30 grid. The heatmap depicts the popularity distribution of objects in the grid.



(a) Optimal fractional static cache allocations

(b) AÇAI's fractional static approximated allocations

Figure 3.16: The optimal fractional static cache allocations, and AÇAI's fractional static approximated allocations under different values of retrieval costs $c_f \in \{1, 2, 3, 4\}$ and number of neighbors $k \in \{1, 2, 3, 4, 5\}$.

We use CVXPY [180] to find the optimal fractional static cache allocation, and AÇAI to compute its approximation according to Corollary 3.3.4. In particular, AÇAI runs for $T = 10\,000$ iterations with a diminishing learning rate $\eta_t = \frac{2.0}{c_f}(1 + \cos(\frac{\pi t}{T}))$.

Results. Figure 3.16 shows the optimal fractional static cache allocations and the AÇAI's fractional static approximated allocations (see Corollary 3.3.4) under different retrieval costs and the number of neighbors k. We observe that the optimal fractional static cache allocations in Figure 3.16 (a)

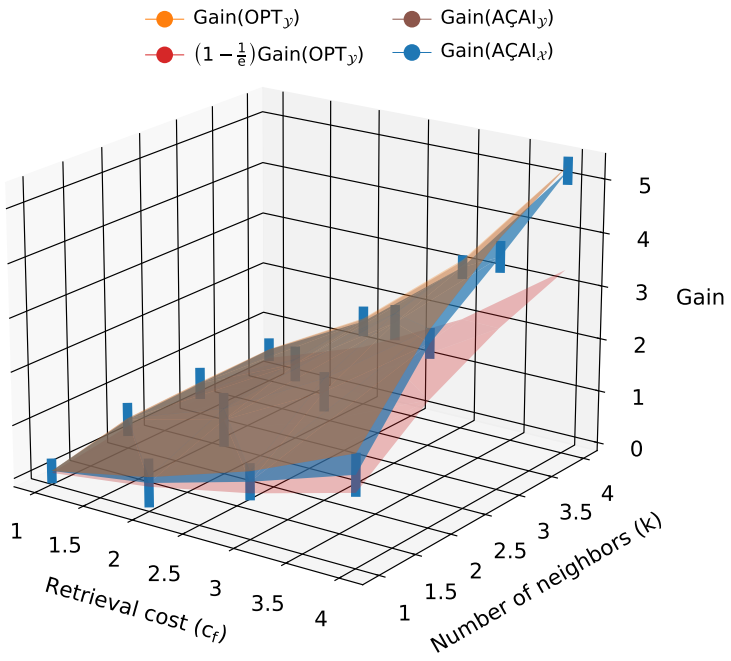


Figure 3.17: The gain of the optimal fractional static cache, its $(1-\frac{1}{e})$-approximation, and the gain obtained by AÇAI's fractional static approximated allocations and static integral approximated allocations under different values of retrieval costs $c_f \in \{1,2,3,4\}$ and number of neighbors $k \in \{1,2,3,4,5\}$. For AÇAI's static integral approximated allocations, we report 95% confidence intervals computed over 50 different runs. The gain of the fractional static approximated allocations obtained by AÇAI overlaps with gain of the optimal fractional static cache allocations.

are symmetric, while AÇAI's fractional static approximated allocations in Figure 3.16 (b) partially lose this symmetry for values of $k \in \{2,3,4\}$ primarily due to the different ways a kNN query can be satisfied over the physical catalog for such values. In fact, there are multiple objects in the catalog with the same distance from a request r, and AÇAI only selects a single permutation π^r for a request r. We observe that, when the retrieval cost is higher, the allocations are more spread to cover a larger part of the popular region. Moreover, for larger values of the number k of objects to be served, more mass is added to the neighborhood of cached objects, but when the number k of objects to be served changes from $k=1$ to $k=2$ it is not clearly observed.

In Figure 3.17, we compare the gain obtained by the optimal fractional static cache, its $(1-\frac{1}{e})$-approximation, and the gain obtained by the fractional static approximated allocations and the static integral approximated allocations of AÇAI (obtained through DEPROUND). While Figure 3.16 shows that AÇAI's fractional static approximated allocations may differ from the optimal ones, their costs are practically indistinguishable (the two corresponding surfaces overlap in Figure 3.17). We also observe that rounding comes at a cost, as there is a clear gap between the gain of AÇAI's fractional allocations and of AÇAI's integral ones. Still, the average gain of the integral allocations remains close to the fractional optimum and well above the $(1-\frac{1}{e})$-approximation. AÇAI then performs much better than what guaranteed by Corollary 3.3.4 (a potential gain reduction by a factor $1-\frac{1}{e}$).

3.3.4.2 Real-World Datasets

SIFT1M trace. SIFT1M is a classic benchmark data-set to evaluate approximate kNN algorithms [181]. It contains 1 million objects embedded as points in a 128-dimensional space. SIFT1M does not provide a request trace so we generated a synthetic one according to the Independent Reference Model [15] (similar to what done in other papers like [106,121]). Request r_t is for object i with a probability λ_i independently from previous requests. We spatially correlated requests by letting λ_i depend on the position of the embeddings in the space. In particular, we considered the barycenter of the whole dataset and set λ_i proportional to $d_i^{-\beta}$, where d_i is the distance of i from the barycenter. The parameter β was chosen such that the tail of the ranked objects popularity distribution is similar to a Zipf with parameter 0.9, as observed in some image retrieval systems [107]. We generated a trace with 10^5 requests. The number of distinct objects requested in the trace is approximately 2×10^4.

Amazon trace. We use the same trace described in Section 3.2.4.3. We truncate the trace to the interval $[2 \times 10^5, 3 \times 10^5]$.[7] The number of distinct objects requested in this trace is approximately 2×10^4.

3.3.4.3 Settings and Performance Metrics

For AÇAI, unless otherwise said, we choose the negative entropy $\Phi(\boldsymbol{y}) = \sum_{i \in \mathcal{N}} y_i \log(y_i)$ as mirror map (see Figure 3.23 and the corresponding discussion for other choices) and the rounding scheme DepRound with $M = 1$. The learning rate is set to the best value found exploring the range $[10^{-6}, 10^{-4}]$.

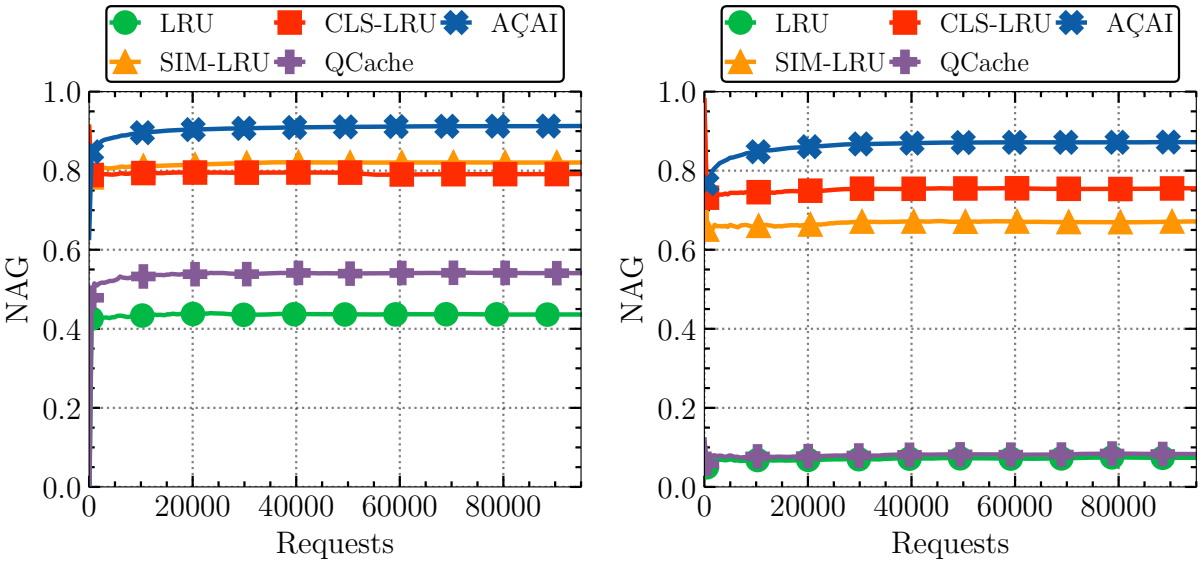
As for the state-of-the-art caching policies, SIM-LRU and CLS-LRU have two parameters, C_θ and k', that we set in each experiment to the best values we found exploring the ranges $[c_f, 2c_f]$ for C_θ and $[1, h]$ for k'. For QCache we consider $l = h/k$: the cache can then perform the kNN search over all local objects.

We also consider a simple similarity caching policy that stores previous requests and the corresponding set of k closest objects as key-value pairs, and manages the set of keys according to LRU. The cache then serves locally the request if it coincides with one of the previous requests in its memory, it forwards it to the server, otherwise. The ordered list of keys is updated as in LRU. We refer to this policy simply as LRU.

We compare the policies in terms of their normalized average caching gain per-request, where the normalization factor corresponds to the caching gain of a cache with size equal to the whole catalog. In such case, the cache could store the entire catalog locally and would achieve the same dissimilarity cost of the server without paying any fetching cost. The maximum possible caching gain is then kc_f. The normalized average gain of a policy $\mathcal{P}$ with cache states $\{\boldsymbol{x}_t\}_{t=1}^T$ over T requests can then be defined as:

$$\mathrm{NAG}(\mathcal{P}) = \frac{1}{kc_f T} \sum_{t=1}^{T} G(r_t, \boldsymbol{x}_t). \tag{3.19}$$

[7] We discard the initial part of the trace because it contains requests only for a small set of objects (likely the set of products to crawl was progressively extended during the measurement campaign in [158]).



(a) SIFT1M trace (b) Amazon trace

Figure 3.18: Caching gain for the different policies. The cache size is $h = 1000$ and $k = 10$.

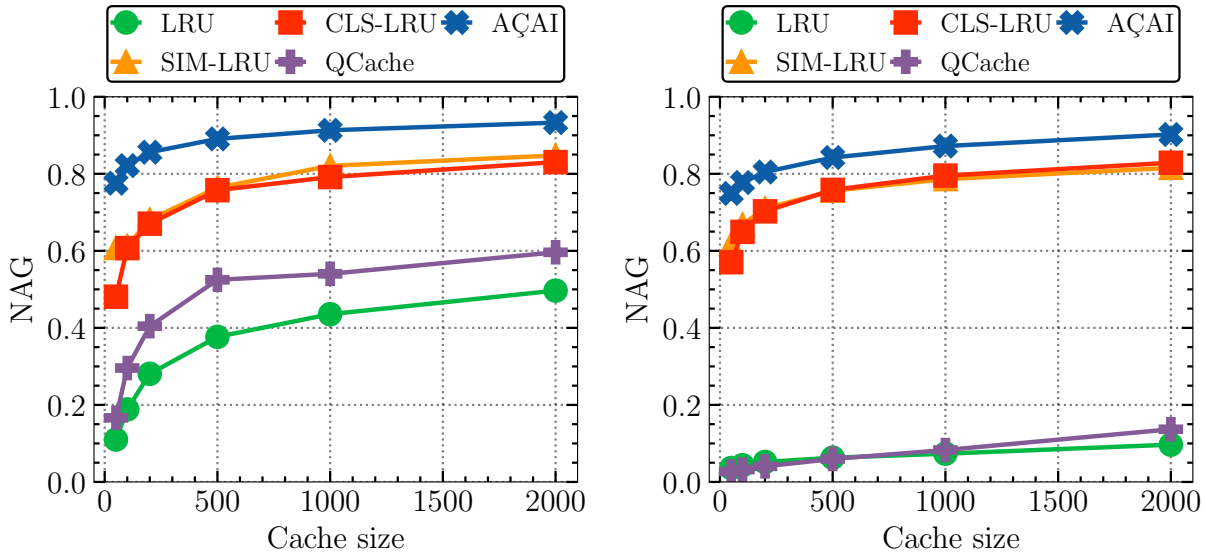
3.3.4.4 Results

We consider a dissimilarity cost proportional to the squared Euclidean distance. This is the usual metric considered for SIFT1M benchmark and also the one considered to learn the embeddings for the Amazon trace in [158].

The numerical value of the fetching cost depends on its interpretation (delay experienced by the user, load on the server or on the network) as well as on the application, because it needs to be converted into the same unit of the approximation cost. In our evaluation, we let it depend on the topological characteristics of the dataset in order to be able to compare the results for the two different traces. Unless otherwise said, we set c_f equal to the average distance of the 50-th closest neighbor in the catalog $\mathcal{N}$.

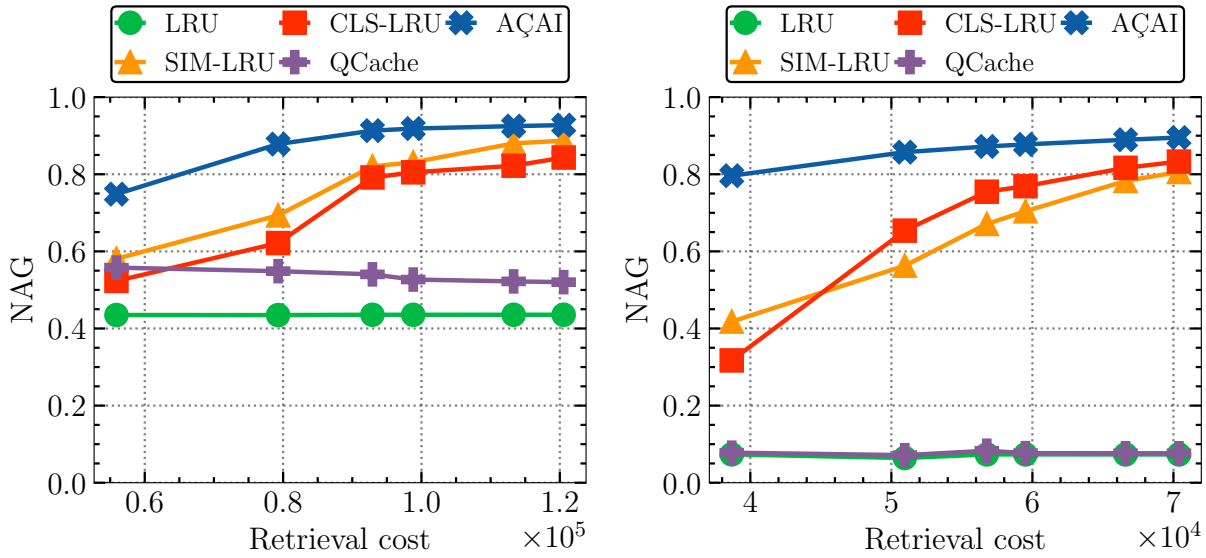
Figure 3.18 shows how the normalized average gain changes as requests arrive and the different caching policies update the local set of objects (starting from an empty configuration). The cache size is $h = 1000$ and the cache provides $k = 10$ similar objects for each request. All policies reach an almost stationary gain after at most a few thousand requests. Unsurprisingly, the naïve LRU has the lowest gain (it can only satisfy locally requests that match exactly a previous request) and similarity caching policies perform better. AÇAI has a significant improvement in comparison to the second best policy (SIM-LRU for SIFT1M and CLS-LRU for Amazon).

This advantage of AÇAI is constantly confirmed for different cache sizes (Figure 3.19), different values of the fetching cost c_f (Figure 3.20), and different values of k (Figure 3.21). The relative improvement of AÇAI, in comparison to the second best policy, is larger for small values of the cache size (+30% for SIFT1M and +25% for Amazon when $h = 50$), and small values of the fetching cost (+35% for SIFT1M and +100% for c_f equal to the average distance from the second closest object). Note how these are the settings where caching choices are more difficult (and indeed all policies have lower gains): when cache storage can accommodate only a few objects, it is critical to carefully



(a) SIFT1M trace (b) Amazon trace

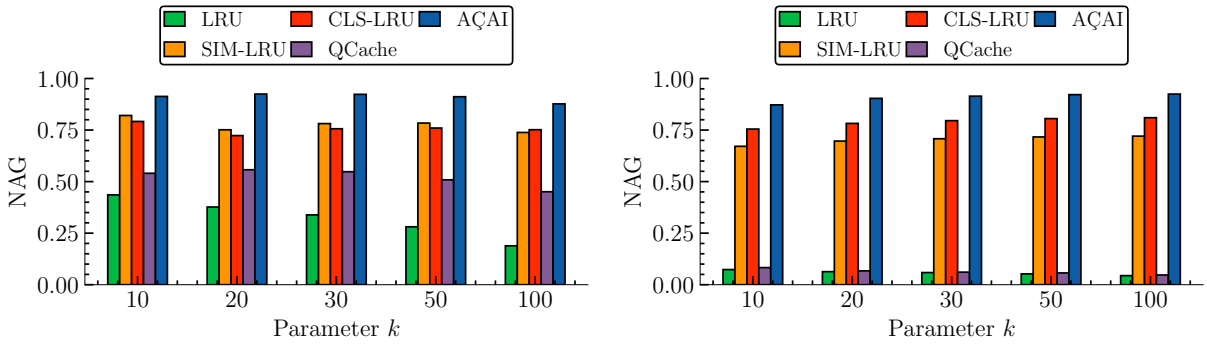
Figure 3.19: Caching gain for the different policies, for different cache sizes $h \in \{50, 100, 200, 500, 1000, 2000\}$ and $k = 10$.



(a) SIFT1M trace (b) Amazon trace

Figure 3.20: Caching gain for the different policies and different retrieval cost. The retrieval cost c_f is taken as the average distance to the i-th neighbor, $i \in \{2, 10, 50, 100, 500, 1000\}$. The cache size is $h = 1000$ and $k = 10$.

select which ones to store; when the server is close, the costs of serving requests from the cache or from the server are similar and it is difficult to correctly decide how to satisfy the request. Caching



(a) SIFT1M trace (b) Amazon trace

Figure 3.21: Caching gain for the different policies. The cache size is $h = 1000$, and $k \in \{10, 20, 30, 50, 100\}$.

policies performance are in general less dependent on the number k of similar objects to retrieve and AÇAI achieves about 10% improvement for k between 10 and 100 when $h = 1000$ (Figure 3.21).

Sensitivity analysis. We now evaluate the robustness of AÇAI to the configuration of its single parameter (the learning rate η). Figure 3.22 shows indeed that, for learning rates that are two orders of magnitude apart, we can achieve almost the same normalized average gain both for $h = 50$ and for $h = 1000$.[8]

In contrast, the performance of the second best policies (SIM-LRU and CLS-LRU) are more sensitive to the choice of their two configuration parameters k' and C_θ. For example, the optimal configuration of SIM-LRU is $k' = 10$ and $C_\theta = 1.5 \times c_f$ for a small cache ($h = 50$) but $k' = 200$ and $C_\theta = 2 \times c_f$ for a large one ($h = 1000$). Moreover, in both cases a misconfiguration of these parameters would lead to significant performance degradation.

Choice of the mirror map. If the mirror map is selected equal to the squared Euclidean norm, the OMA update coincides with a standard gradient update. Figure 3.23 shows the superiority of the negative entropy map: it allows to achieve a higher gain than the Euclidean norm map or the same gain but in a shorter time. To the best of our knowledge, ours is the first paper that shows the advantage of using non-Euclidean mirror maps for similarity caching problems. It is possible to justify theoretically this result, considering the difference of subgradients norms for similarity caching and exact caching problems (Appendix 19.1). This result does not follow from the result in [139], where they predict for single-request batch one should always select the Euclidean mirror map for OMA (i.e., OGA), and this is a consequence of the gradient being sparse. In the case of similarity caching, even for $k = 1$, multiple objects in the vicinity of a request can provide utility in serving the request, i.e., the subgradient of the gain function when request r is received is no longer a sparse vector as in the classical caching setting.

Dissecting AÇAI performance. In comparison to state-of-the-art similarity caching policies, AÇAI introduces two key ingredients: (1) the use of fast, approximate indexes to decide what to serve from the local catalog and what from the remote one, and (2) the OMA algorithm to update

[8]Under a stationary request process, a smaller learning rate would lead to converge slower but to a solution closer to the optimal one. Under a non-stationary process, a higher learning rate may allow faster adaptivity. In this trace, the two effects almost compensate, but see also Figure 3.23.

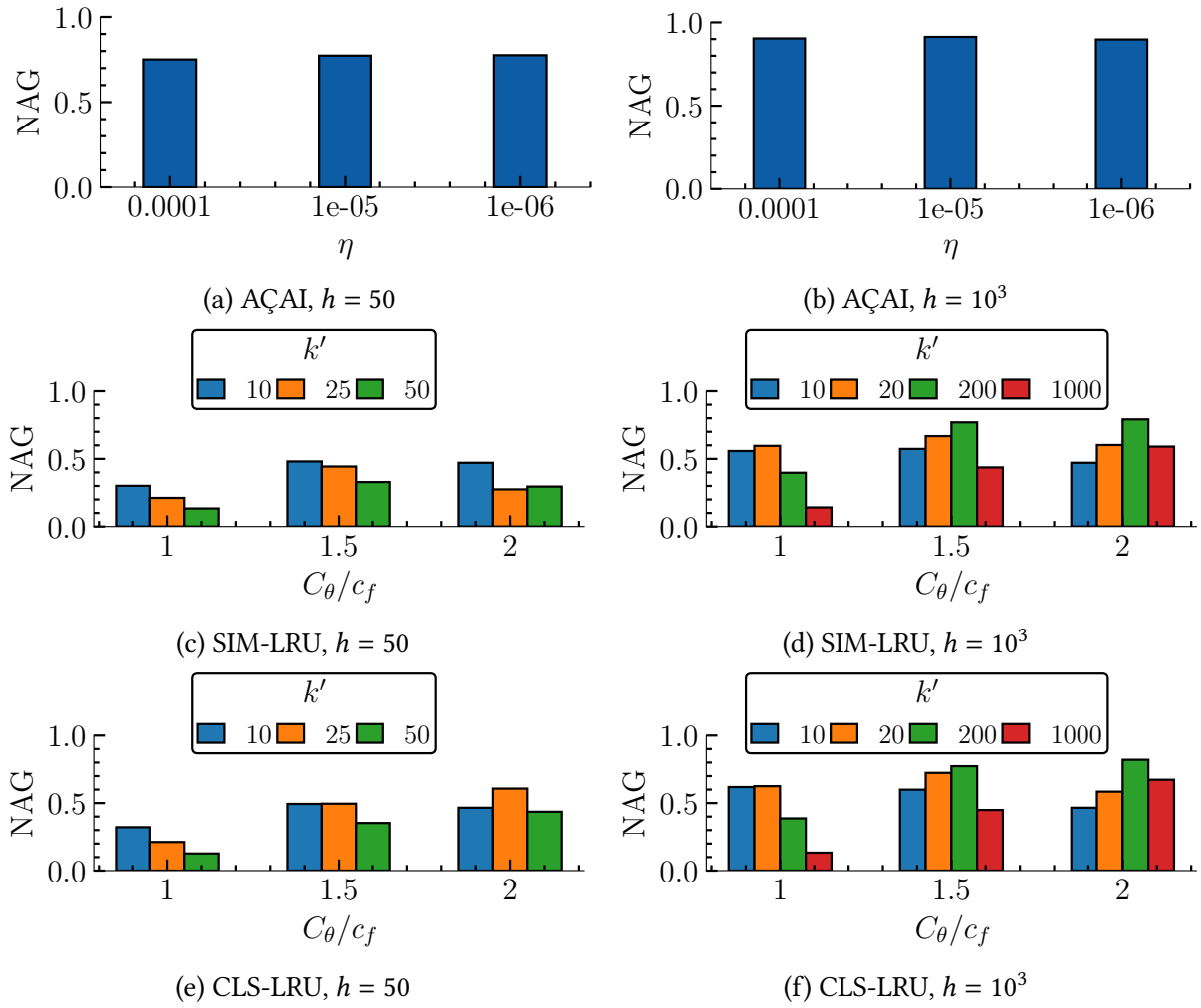


Figure 3.22: Caching gain for AÇAI for different values of η (top). Caching gain for SIM-LRU(middle) and CLS-LRU (bottom) for different values of the parameters (k', C_θ). SIFT1M trace.

the cache state. It is useful to understand how much each ingredient contributes to AÇAI improvement with respect to the other policies.

To this aim, we integrated the same indexes in the other policies allowing them to serve requests as AÇAI does, combining both local objects and remote ones on the basis of their costs (see Section 3.3.3.3), while leaving their cache updating mechanism unchanged. We then compute, in the same setting of Figure 3.21, how much the gain of the second best policy (SIM-LRU for SIFT1M and CLS-LRU for Amazon) increases because of AÇAI request service mechanism. This is the part of AÇAI improvement attributed to the use of the two indexes, the rest is attributed to the cache update mechanism through OMA. We observe from Figure 3.24 that most of AÇAI gain improvement over the second best caching policy is due to the use of approximate indexes, but OMA updates are still responsible for 15–20% of AÇAI performance improvement under SIFT1M trace and for 20–35% for the Amazon trace.

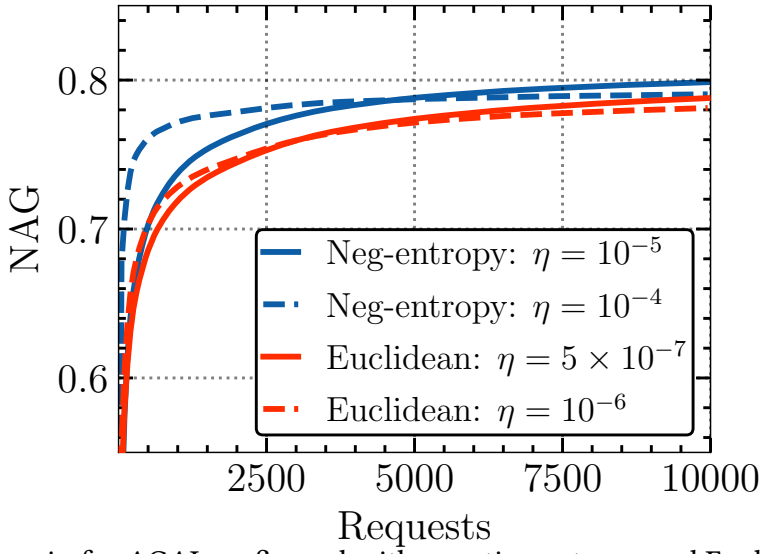
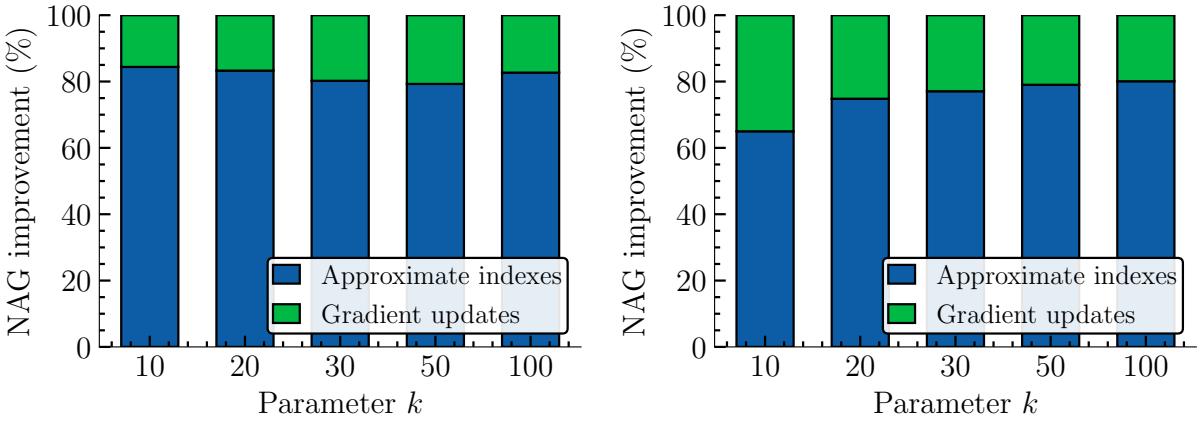


Figure 3.23: Caching gain for AÇAI configured with negative entropy and Euclidean maps (SIFT1M trace). The cache size is $h = 100$ and $k = 10$.



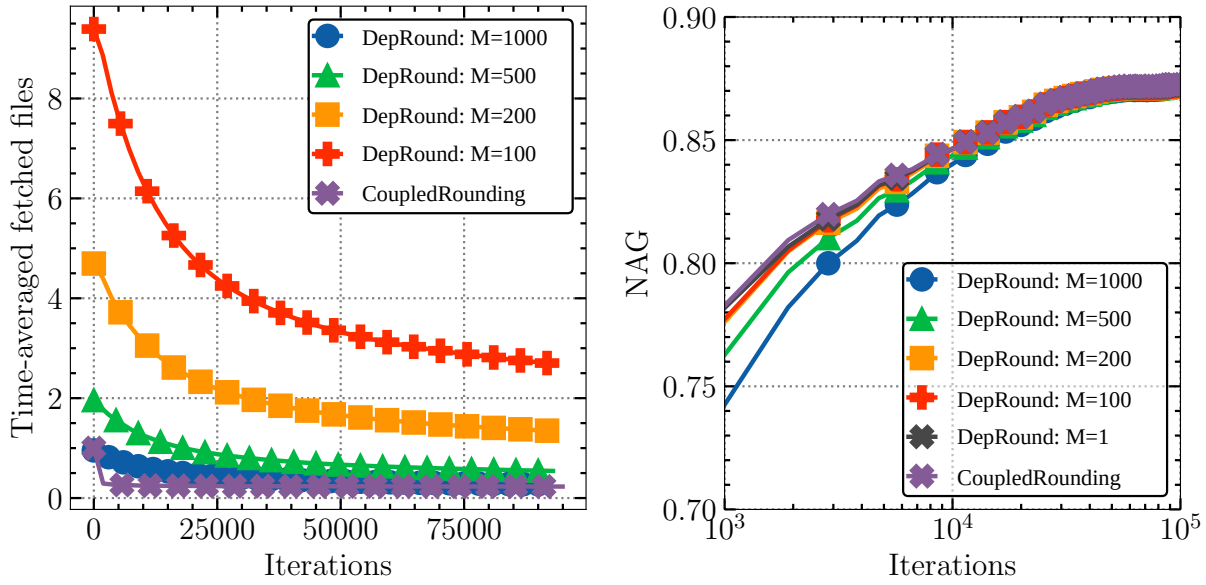
(a) SIFT1M trace (b) Amazon trace

Figure 3.24: AÇAI caching gain improvement in comparison to the second best state-of-the-art similarity caching policy: contribution of approximate indexes and gradient updates. The cache size is $h = 1000$, and $k \in \{10, 20, 30, 50, 100\}$.

Update cost. In this part, we evaluate the update cost of the different rounding schemes. We set the cache size $h = 1000$ and $k = 10$. We run AÇAI over the Amazon trace with a learning rate $\eta = 10^{-5}$.

Figure 3.25 (a) gives the time-averaged number of files fetched and Figure 3.25 (b) gives the caching gain of the different rounding schemes. We observe by increasing the cache state freezing parameter M, the system fetches fewer files per iteration at the expense of reduced reactivity at the start. The coupled rounding scheme achieves the best performance as it fetches fewer files without a reduction in reactivity.

Figure 3.26 gives the instantaneous and time averaged cache occupancy of the cache using the CoupledRounding scheme with the relaxed capacity constraint. We observe that the time averaged



(a) Time-averaged fetched files

(b) Caching gain

Figure 3.25: Number of fetched files (time average) and caching gain of AÇAI under different rounding schemes. AÇAI is run with the learning rate $\eta = 10^{-5}$ over the Amazon trace. The cache size is $h = 1000$ and $k = 10$.

cache occupancy rapidly converges to the cache capacity h, while the instantaneous occupancy is kept within 5% of the cache capacity.

3.3.5 Conclusion

Edge computing provides computing and storage resources that may enable complex applications with tight delay guarantee like augmented-reality ones, but these strategically positioned resources need to be used efficiently. To this aim, we design AÇAI, a content cache management policy that determines dynamically the best content to store on the edge server. Our solution adapts to the user requests, without any assumption on the traffic arrival pattern. AÇAI leverages two key components: (1) new efficient content indexing methods to keep track of both local and remote content, and (2) mirror ascending techniques to optimally select the content to store. The results show that AÇAI is able to outperform the state-of-the-art policies and does not need careful parameter tuning.

As future work, we plan to evaluate AÇAI in the context of machine learning classification tasks [182], in which the size of the objects in the catalog is comparable to their d-dimensional representation in the index, and, as a consequence, the index size cannot be neglected in comparison to the local catalog size. Another important future research direction is to consider dynamic regret, whereby the performance of a policy is compared to a dynamic optimum. Also, since the employed online algorithm OMA is greedy (i.e., does not keep track of the history of the requests), with careful selection of the mirror map, it may have an adaptive regret guarantee, e.g., such guarantee holds for OGD [46].

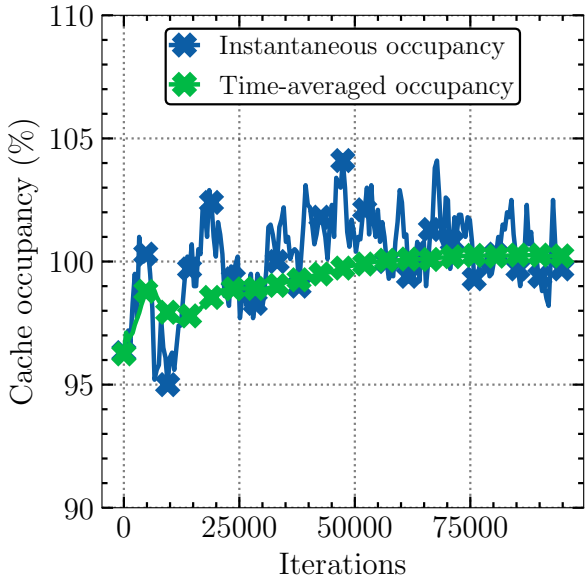


Figure 3.26: Time averaged and instantaneous cache occupancy under CoupledRounding. AÇAI is run with the learning rate $\eta = 10^{-5}$ over the Amazon trace. The cache size is $h = 1000$ and $k = 10$.

Chapter 4

Inference Delivery Networks

4.1 Introduction

Machine learning (ML) models are often trained to perform inference, that is to elaborate predictions based on input data. ML model training is a computationally and I/O intensive operation and its streamlining is the object of much research effort. Although inference does not involve complex iterative algorithms and is therefore generally assumed to be easy, it also presents fundamental challenges that are likely to become dominant as ML adoption increases [183]. In a future where AI systems are ubiquitously deployed and need to make timely and safe decisions in unpredictable environments, inference requests will have to be served in real-time and the aggregate rate of predictions needed to support a pervasive ecosystem of sensing devices will become overwhelming.

Today, two deployment options for ML models are common: inferences can be served by the end devices (smartphones, IoT equipment, smart vehicles, etc.), where only simple models can run, or by a remote cloud infrastructure, where powerful "machine learning as a service" (MLaaS) solutions rely on sophisticated models and provide inferences at extremely high throughput.

However, there exist applications for which both options may be unsuitable: local models may have inadequate accuracy, while the cloud may fail to meet delay constraints. As an example, popular applications such as recommendation systems, voice assistants, and ad-targeting, need to serve predictions from ML models in less than 200 ms. Future wireless services, such as connected and autonomous cars, industrial robotics, mobile gaming, augmented/virtual reality, have even stricter latency requirements, often below 10 ms and in the order of 1 ms for what is known as the tactile Internet [3]. In enabling such strict latency requirements, the advent of Edge Computing plays a key role, as it deployes computational resources at the edge of the network (base stations, access points, ad-hoc servers). However, edge resources have limited capacity in comparison to the cloud and need to be wisely used. Therefore, integrating ML inference in the continuum between end devices and the cloud—passing through edge servers and regional micro data-centers—will require complex resource orchestration.

We believe that, to allocate resources properly, it will be crucial to study the trade-offs between accuracy, latency and resource-utilization, adapted to the requirements of the specific application. In fact, inference accuracy and, in general, resource efficiency increase toward the cloud, but so does communication latency. In this chapter, we present the novel idea of *inference delivery networks* (IDN): networks of computing nodes that coordinate to satisfy inference requests achieving the

best trade-off. An IDN may be deployed directly by the ML application provider, or by new IDN operators that offer their service to different ML applications, similarly to what happens for content delivery networks. The same inference task can be served by a set of heterogeneous models featuring diverse performance and resource requirements (e.g., different model architectures [184], multiple downsized versions of the same pre-trained model [185], different configurations and execution setups). Therefore, we study the novel problem of how to deploy the available ML models on the available IDN nodes, where a deployment strategy consists in two coupled decisions: (1) where to place models for serving a certain task and (2) how to select their size/complexity among the available alternatives.

4.1.1 Contributions

In this chapter, we first define a specific optimization problem for ML model allocation in IDNs. We characterize the complexity of such problem and then introduce INFIDA (INFerence Intelligent Distributed Allocation), a distributed dynamic allocation policy. Following this policy, each IDN node periodically updates its local allocation of inference models on the basis of the requests observed during the recent past and limited information exchange with its neighbors. The policy offers strong performance guarantees in an adversarial setting [186], that is a worst case scenario where the environment evolves in the most unfavorable way. Numerical experiments in realistic settings show that our policy outperforms heuristics with similar complexity. Our contributions are as follows:

- We present the novel idea of inference delivery networks (IDNs).
- We frame the allocation of ML model in IDNs as an (NP-hard) optimization problem that captures the trade-off between latency and accuracy (Section 4.3).
- We propose INFIDA, a distributed and dynamic allocation algorithm for IDNs (Section 4.4), and we show it provides strong guarantees in the adversarial setting (Section 4.5).
- We evaluate INFIDA in a realistic simulation scenario and compare its performance both with an offline greedy heuristic and with its online variant under different topologies and trade-off settings (Section 4.6).

4.1.2 Organization

This chapter is organized as follows. In Section 4.2 we discuss related work and other relevant background with respect to IDNs. We formalize the problem of allocating ML model in IDNs in Section 4.3. We introduce our algorithm INFIDA in (Section 4.4) and discuss its theoretical guarantees in Section 4.5. We present our experimental results in Section 4.6.

4.2 Related Work

The problem of machine learning is often reduced to the training task, i.e., producing statistical models that can map input data to certain predictions. A considerable amount of existing works

addresses the problem of model training: production systems such as Hadoop [187] and Spark [188] provide scalable platforms for analyzing large amount of data on centralized systems, and even the problem of distributing the training task over the Internet has been largely addressed recently by many works on federated learning [189–194]. However, there is surprisingly less research on how to manage the deployment of ML models once they have been trained (inference provisioning).

Most of the existing solutions on inference provisioning (e.g., Tensorflow Serving [195], Azure ML [196], and Cloud ML [197]) address the scenario where inference queries are served by a data center. Recent works [167, 198–200] propose improvements on performance and usability of such cloud inference systems. Clipper [167] provides a generalization of TensorFlow Serving [195] to enable the usage of different ML frameworks, such as Apache Spark MLLib [201], Scikit-Learn [202], and Caffe [203]. The auhtors of [198] propose a reinforcement learning scheduler to improve the system throughput. INFaaS [199] provides a real-time scheduling of incoming queries on available model variants, and scales deployed models based on load thresholds. Last, InferLine [200] extends Clipper to minimize the end-to-end latency of a processing pipeline, both periodically adjusting the models allocation and constantly monitoring and handling unexpected query spikes; the solution can be applied to any inference serving system that features a centralized queue of queries. All these solutions address the problem of inference provisioning in the scenario where the requests are served within a data center and are not suitable for a geographically distributed infrastructure where resources are grouped in small clusters and network latency is crucial (e.g., Edge Computing). For instance, none of the previous works consider the network delay between different compute nodes, it being negligible in a data center.

For what concerns inference provisioning in constrained environments, fewer works exist. Some solutions attempt to adapt inference to the capabilities of mobile hardware platforms through the principle of model splitting, a technique that distributes a ML model by partitioning its execution across multiple discrete computing units. Model splitting was applied to accommodate the hardware constraints in multi-processor mobile devices [204], to share a workload among mobile devices attached at the same network edge [205], and to partially offload inferences to a remote cloud infrastructure [206], possibly coupled with early exit strategies [207] and conditional hierarchical distributed deployment [208]. Model splitting is orthogonal to our concerns and could be accounted for in an enhanced IDN scheme.

There has been some work on ML model placement at the edge in the framework of what is called "AI on Edge" [209], but it considers a single intermediate tier between the edge device and the cloud, while we study general networks with nodes in the entire cloud-to-the-edge continuum. Our dynamic placement INFIDA algorithm could be applied also in this more particular setting, for example in the MODI platform [210]. The work closest to ours is [211], which proposes an online learning policy, with the premise of load balancing over a pool of edge devices while maximizing the overall accuracy. INFIDA has more general applicability, as we do not make any assumption on the network topology and also employ a more flexible cost model that takes into account network delay. Another related work in this framework is VideoEdge [212], which studies how to split the analytics pipeline across different computational clusters to maximize the average inference accuracy. Beside the focus on the specific video application, the paper does not propose any dynamic allocation placement algorithm.

Even if the problem of inference provisioning is currently overlooked in the context of distributed systems, there exists a vast literature on the problem of content placement [213] where objects can be stored (cached) into different nodes in order to reduce the operational cost of content delivery. The similarities between this problem and inference provisioning inspired us in the design of Inference Delivery Networks. However, the two problems feature some crucial differences. For instance, content placement has been extended to the case of *service caching* (or placement), where an entire service can be offloaded onto nodes co-located with base-stations or mobile-micro clouds, engaging not only storage but also computational resources and energy [214, 215]; nonetheless the problem considered in this chapter differs from the above, as inference services can run under different configurations that lead to variable resource consumption and inference accuracy. In some sense, traditional services provide a binary answer to any user request: the request can be satisfied if and only if the service is deployed at the node and any two nodes deploying the service are equivalent. In ML inference, however, several models can provide an answer but the accuracy of the answer can be different [216].

A similar trade-off between resource usage and perceived quality typically emerges in the context of video caching [150, 217–222], where the same video can be cached into multiple network nodes at different qualities (or resolutions): the operator optimizes the user experience by jointly deciding the placement of videos and their quality. These works either maximize the video quality perceived by the user [217, 220, 221], minimize the download time [219, 222] and the backhaul traffic [218], or minimize a combined cost [150]. Although some of the models in these papers may be adapted to inference provisioning in IDNs, these works in general study static optimization problems under a known request process and consider simple network topologies: a single cache [218, 219], a pool of parallel caches [217, 222], bipartite networks [150, 221]. The only exception is [220], which considers an arbitrary topology and provides some online heuristics, but ignores the service latency, which is of paramount importance when placing interactive ML models (e.g., for applications like augmented reality or autonomous driving). Instead, we propose a dynamic policy that jointly optimize inference quality and latency and provides strong performance guarantees without requiring any knowledge about the request process thanks to our adversarial setting (Section 4.5).

Finally, ML model allocation in an IDN is closely related to *similarity caching* considered in Chapter 3. To the best of our knowledge, the literature on similarity caching has restricted itself to (1) a single cache (except for [110, 168, 223]), and (2) homogeneous items with identical resource requirements. A consequence is that in our setting similarity caching policies would only allocate models based on their accuracy, ignoring the trade-offs imposed by their resource requirements. Moreover, the literature on similarity caching ignores system throughput constraints, while we explicitly take into account that each model can only serve a bounded number of requests per second, according to its capacity.

4.3 Inference System Design

We consider a network of compute nodes, each capable of hosting some pre-trained ML models depending on its capabilities. Such ML models are used to serve inference requests for different

	Inference Delivery Networks
$G(\mathcal{V}, \mathcal{E})$	Weighted graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$
$w_{u,v}$	Weight of edge $(u, v) \in \mathcal{E}$
$\mathcal{N}$ / $\mathcal{M}$	Tasks / models catalog
$\mathcal{M}$	Models catalog
s_m^v	Size of model $m \in \mathcal{M}$ on node $v \in \mathcal{V}$
a_m	Prediction accuracy of model m
d_m^v	Average inference delay of model m at node v
b^v	Allocation budget constraint at node v
x_m^v	Set to 1 if model m at node v is allocated
ω_m^v	Set to 1 if model m at node v is permanently stored
$\boldsymbol{\omega}$ / $\boldsymbol{x}$	Minimal allocation vector/ allocation vector
$\boldsymbol{x}^v$ / $\boldsymbol{x}$	Allocation vector at node v / global allocation vector
$\boldsymbol{s}^v$	Vector of model sizes at node v
$\boldsymbol{p}$	Routing path $\{p_1, \ldots, p_J\}$
$\nu(\boldsymbol{p})$	Repository node associated to the path $\boldsymbol{p}$
$C_{\boldsymbol{p},m}^{p_j}$	Cost of serving on p_j along path $\boldsymbol{p}$ using model m
ρ	Request type $\rho = (i, \boldsymbol{p})$ (task i routed through $\boldsymbol{p}$)
$\mathcal{R}$	Set of all the possible request types
R	Total number of request types
r_ρ^t	Number of times ρ is requested during time slot t
$\text{load}_m^{t,v}(\rho)$	Number of type-ρ requests served by model m at node v during the t-slot
L_m^v	Maximum capacity of model m on node v
$l_{\rho,m}^{t,v}$	Potential available capacity of m on node v for request type ρ at time t
γ_ρ^k	k-th smallest cost for request type ρ along its path
$\boldsymbol{r}_t$ / $\boldsymbol{l}_t$	Request batch / potential available capacities
$\lambda_\rho^k(\boldsymbol{l}_t)$ / $z_\rho^k(\boldsymbol{l}_t, \boldsymbol{x})$	Potential / effective available capacity of the model serving ρ with cost γ_ρ^k
$Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x})$	Number of requests of type ρ that can be served by the k-th smallest cost models found along path $\boldsymbol{p}$ under allocation $\boldsymbol{x}$ at time slot t
$\kappa_\rho(v, m)$	Rank of model m allocated to node v
K_ρ	Maximum number of models can serve ρ
$\mathcal{X}^v$ / $\mathcal{X}$	Set of integral allocations at v / at $\mathcal{V}$
C / G	The overall system cost/ allocation gain
	INFIDA
Φ^v	Weighted negative entropy map
Φ	Global mirror map
y_m^v	Fractional allocation of model m at v
$\boldsymbol{y}^v$ / $\boldsymbol{y}$	Fractional allocation vector at v / at $\mathcal{V}$
$\mathcal{Y}^v$ / $\mathcal{Y}$	Set of fractional allocations at v / at $\mathcal{V}$
$\boldsymbol{g}_t$	Subgradient vector of G over $\mathcal{Y}$ at point $\boldsymbol{y}_t$
$g_{t,m}^v$	Component (v, m) of $\boldsymbol{g}_t$
$\mathcal{P}_{\mathcal{Y}^v \cap \mathcal{D}^v}^{\Phi^v}$	Projection operator onto $\mathcal{Y}^v \cap \mathcal{D}^v$
η	learning rate
B	Refresh period
T	Time horizon
t	Time slot $t \in [T]$
A	Regret constant
$L_{\max}$	Upper bound on model capacities
Δ_C	Upper bound on maximum serving cost difference
ψ	Regret discount factor equal to $1 - \frac{1}{e}$

Table 4.1: Notation Summary for Chapter 4

classification or regression *tasks*.[1] As shown in Figure 4.1, requests are generated by end devices and routed over given serving paths (e.g., from edge to cloud nodes). The goal of the system is to optimize the allocation of ML models across the network so that the aggregate serving cost is minimized. Our system model is detailed below, and the notation used across the paper is summarized in Table 4.1.

4.3.1 Compute Nodes and Models

We represent the inference delivery network (IDN) as a weighted graph $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of compute nodes, and $\mathcal{E}$ represents their interconnections. Each node $v \in \mathcal{V}$ is capable of serving inference tasks that are requested from anywhere in the network (e.g., from end-users, vehicles, IoT devices). We denote by $\mathcal{N} = \{1, 2, \ldots, |\mathcal{N}|\}$ the set of tasks the system can serve (e.g., object detection, speech recognition, classification), and assume that each task $i \in \mathcal{N}$ can be served with different quality levels (e.g., different accuracy as illustrated in Figure 4.2) and different resources' requirements by a set of suitable models $\mathcal{M}_i$. Each task is served by a separate set of models, i.e., $\mathcal{M}_i \cap \mathcal{M}_{i'} = \emptyset, \forall i, i' \in \mathcal{N}, i \neq i'$. Catalog $\mathcal{M}_i$ may encompass, for instance, independently trained models or shrunk versions of a high quality model generated through distillation [225, 226]. We denote by $\mathcal{M} = \cup_{i \in \mathcal{N}} \mathcal{M}_i = \{1, 2, \ldots, |\mathcal{M}|\}$ the catalog of all the available models.

[1] We are using the term task according to its meaning in the ML community, e.g., a task could be to detect objects in an image, to predict its future position, to recognize vocal commands.

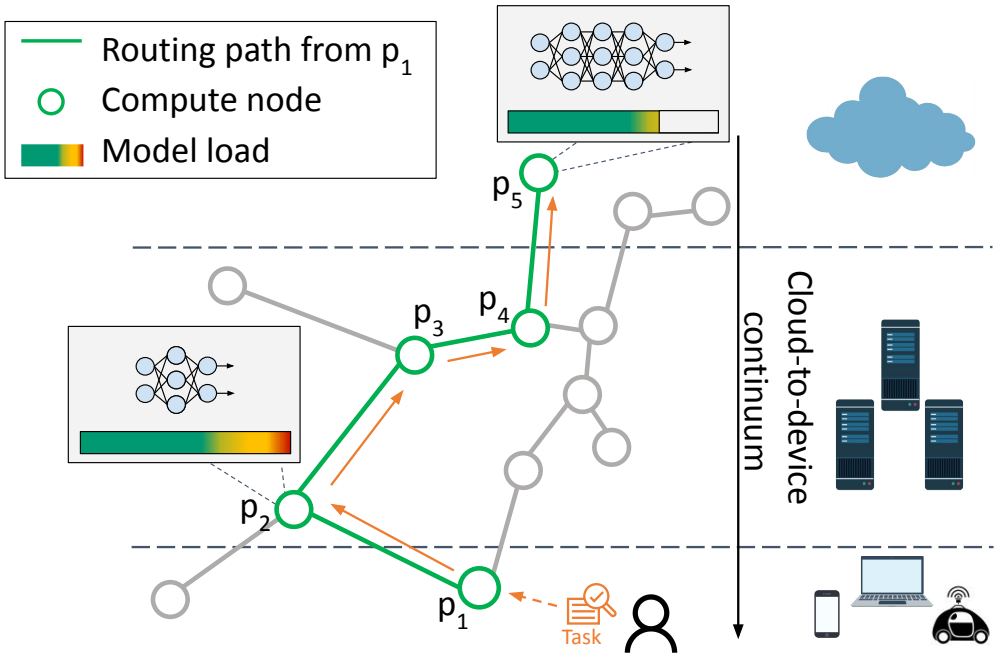


Figure 4.1: System overview: a network of compute nodes serves inference requests along predefined routing paths. A repository node at the end of each path ensures that requests are satisfied even when there are no suitable models on intermediate nodes.

Finally, every model of the catalog may provide a different throughput (i.e., number of requests it can serve in a given time period), and therefore, support a different load (we formalize this in Section 4.3.4).

For each compute node $v \in \mathcal{V}$, we denote by

$$x_m^v \in \{0, 1\}, \text{ for } m \in \mathcal{M}, \tag{4.1}$$

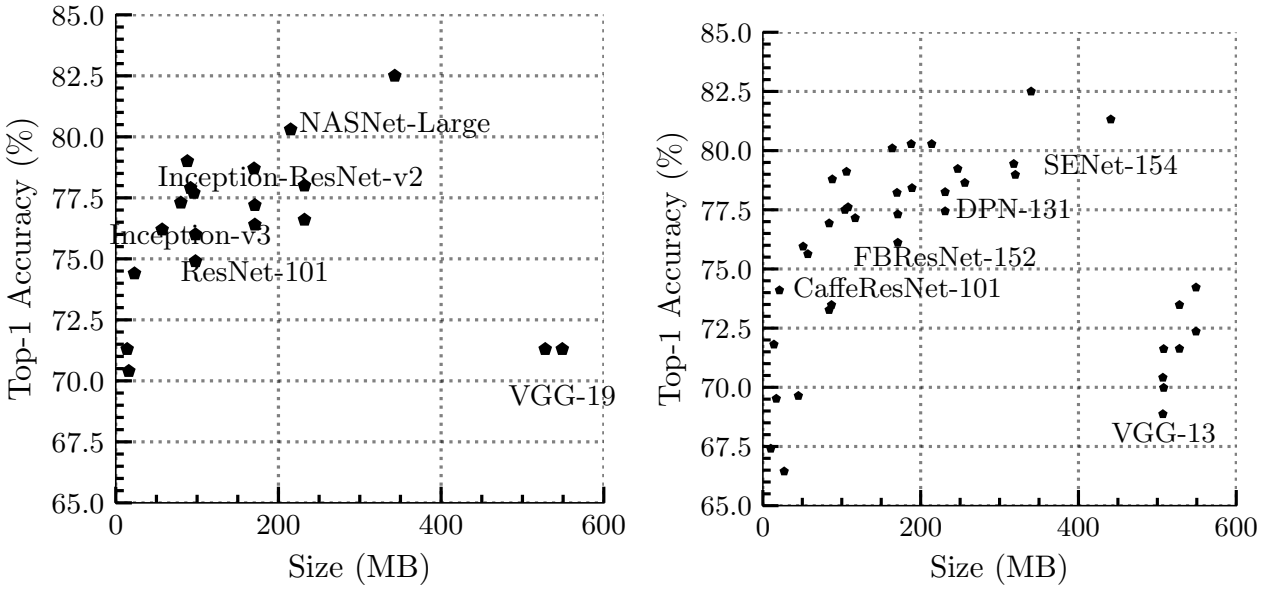
the decision variable that indicates if model $m \in \mathcal{M}$ is deployed on node v.[2] Therefore, $\boldsymbol{x}^v = [x_m^v]_{m \in \mathcal{M}}$ is the allocation vector on node v, and $\boldsymbol{x} = [\boldsymbol{x}^v]_{v \in \mathcal{V}}$ denotes the global allocation decision.

We assume that the allocation of ML models at each node is constrained by a single resource dimension, potentially different at each node. A node could be, for instance, severely limited by the amount of available GPU memory, another by the maximum throughput in terms of instructions per second. The limiting resource determines the *allocation budget* $b^v \in \mathbb{R}_+$ at node $v \in \mathcal{V}$. We also denote with $s_m^v \in \mathbb{R}_+$ the size of model $m \in \mathcal{M}$, i.e., the consumed amount of the limiting resource at node v.[3] Therefore, budget constraints are expressed as

$$\sum_{m \in \mathcal{M}} x_m^v s_m^v \leq b^v, \forall v \in V. \tag{4.2}$$

[2]Our formulation allows each node to host multiple copies of the same model to satisfy a larger number of request. For example, two copies of the same model can be represented as two distinct models with identical performance and requirements.

[3]Note that, even when the limiting resource is the same, say computing, the budget consumed by a model may be different across nodes, as they may have different hardware (e.g., GPUs, CPUs, or TPUs).



(a) Keras pre-trained models

(b) Pytorch pre-trained models

Figure 4.2: Example of pre-trained model catalog for the image classification task. Data from [224].

To every task $i \in \mathcal{N}$, we associate a fixed set of *repository nodes* that always run one model capable of serving all the requests for task i (e.g., high-performance models deployed in large data centers). We call these models *repository models* and they are statically allocated. Repository models ensure requests are satisfied even when the rest of the network is not hosting any additional model.

We discern repository models through constants $\omega_m^v \in \{0, 1\}$, each indicating if model m is permanently deployed on node v. We assume that values ω_m^v are given as input. We call the vector $\boldsymbol{\omega} = [\omega_m^v]_{(v,m)\in\mathcal{V}\times\mathcal{M}}$ the *minimal allocation*. Note that the presence of repositories introduce the following constraints to the allocation vector:

$$x_m^v \geq \omega_m^v, \forall v \in \mathcal{V}, \forall m \in \mathcal{M}. \tag{4.3}$$

The set of possible allocations at node $v \in \mathcal{V}$ is determined by the integrality constraints (4.1), budget constraints (4.2), and repository constraints (4.3), i.e.,

$$\mathcal{X}^v \triangleq \left\{ \boldsymbol{x}^v \in \{0, 1\}^{\mathcal{M}} : \boldsymbol{x}^v \text{ satisfies Eqs. (4.1)–(4.3)} \right\}. \tag{4.4}$$

The set of possible global allocations is given as $\mathcal{X} \triangleq \bigtimes_{v\in\mathcal{V}} \mathcal{X}^v$.

4.3.2 Inference Requests

We assume that every node has a predefined routing path towards a suitable repository node for each task $i \in \mathcal{N}$. The routing is therefore predetermined, and our decisions only concern placement of models (i.e., variables x_m^v).

A routing path $\boldsymbol{p}$ of length $|\boldsymbol{p}| = J$ is a sequence $\{p_1, p_2, \ldots, p_J\}$ of nodes $p_j \in \mathcal{V}$ such that edge $(p_j, p_{j+1}) \in \mathcal{E}$ for every $j \in \{1, 2, \ldots, J-1\}$. As in [21], we assume that paths are simple, i.e., they do not contain repeated nodes. A request is therefore characterized by the pair $(i, \boldsymbol{p})$, where i is the task requested and $\boldsymbol{p}$ is the routing path to be traversed. We call the pair $(i, \boldsymbol{p})$ the *request type*. We denote by $\mathcal{R}$ the set of all possible request types, and by $\mathcal{R}_i$ all possible request types for tasks i. When a request for task i is propagated from node p_1 toward the associated repository node $v(\boldsymbol{p}) \triangleq p_J$, any intermediate node along the path that hosts a suitable model $m \in \mathcal{M}_i$ can serve it. The actual serving strategy is described in Section 4.3.5.

4.3.3 Cost Model

When serving a request of type $\rho=(i, \boldsymbol{p}) \in \mathcal{R}$ on node p_j using model m, the system experiences an *inference cost* that depends on the quality of the model (i.e., on inference inaccuracy) and the inference time. Additionally, the system experiences a *network cost*, due to using the path between p_1 and p_j. Similarly to previous work [227], we can write the total cost of serving a request as

$$C^{p_j}_{\boldsymbol{p},m} = f((p_1, \ldots, p_j), m). \tag{4.5}$$

While our theoretical results hold under this very general cost model, in what follows—for the sake of concreteness—we refer to the following simpler model:

$$C^{p_j}_{\boldsymbol{p},m} = \sum_{j'=1}^{j-1} w_{p_{j'}, p_{j'+1}} + d^{p_j}_m + \alpha(1-a_m), \tag{4.6}$$

where a_m and $d^{p_j}_m$ are respectively the prediction accuracy (in a scale from 0 to 1) and the average inference delay of model m on node p_j. Indeed, the same model may provide different inference delays, depending on the hardware capabilities of the node on which it is deployed, e.g., the type of GPU or TPU [228]. Parameter $w_{v,v'} \in \mathbb{R}_+$ is the (round-trip) latency of edge $(v, v') \in \mathcal{E}$. Parameter α weights the importance of accuracy w.r.t. the overall latency and can be set depending on the application.

Note that seeking cost minimization along a serving path usually leads to a trade-off: while the network cost always increases with j, in a typical network the service cost $d^{p_j}_m + \alpha(1-a_m)$ tends to decrease, as farther nodes (e.g., data centers) are better equipped and can run more accurate models (Figure 4.2).

4.3.4 Request Load and Serving Capacity

Let us assume that time is split in slots of equal duration. We consider a time horizon equal to T slots. At the beginning of a slot t, the system receives a batch of requests $\boldsymbol{r}_t = [r^t_\rho]_{\rho \in \mathcal{R}}$, where $r^t_\rho \in \mathbb{N} \cup \{0\}$ denotes the number of requests of type $\rho \in \mathcal{R}$.

Model $m \in \mathcal{M}$ has maximum capacity $L^v_m \in \mathbb{N}$ when deployed at node $v \in \mathcal{V}$, i.e., it can serve at most L^v_m requests during one time slot $t \in [T]$, in absence of other requests for other models. We do not make specific assumptions on the time required to serve a request.

We denote by $l^{t,v}_{\rho,m} \in \mathbb{N} \cup \{0\}$ the *potential available capacity*, defined as the maximum number of type-ρ requests node v can serve at time t through model m, under the current request load $\boldsymbol{r}_t$ and allocation vector $\boldsymbol{x}^v_t$. Formally, let $\text{load}^{t,v}_m(\rho)$ denote the number of type-ρ requests served by model m at node v during the t-slot, then

$$l^{t,v}_{\rho,m} \triangleq \min\left\{L^v_m - \sum_{\rho' \in \mathcal{R}\setminus\{\rho\}} \text{load}^{t,v}_m(\rho'),\, r^t_\rho\right\}. \tag{4.7}$$

The potential available capacity depends on the request arrival order and the scheduling discipline at node v. For instance, suppose that in time slot t, requests of two types $\rho = (i, \boldsymbol{p})$ and $\rho' = (i, \boldsymbol{p}')$ arrive at node v. The arrival order and the node scheduling discipline may determine that many requests of type ρ' be served, which would leave a small $l^{t,v}_{\rho,m}$ available for requests of type ρ. Or the opposite may happen. It is useful to define the potential available capacity also for models that are not currently deployed at the node, as $l^{t,v}_{\rho,m} \triangleq \min\{L^v_m, r^t_\rho\}$. The *effective available capacity* is then equal to $l^{t,v}_{\rho,m} x^v_m$.

Our analysis in Section 4.5 considers a "pessimistic" scenario where an adversary selects both requests and available capacities for all models but the repository ones. This approach relieves us from the need to model system detailed operations, while our proposed algorithm (Section 4.4) benefits from strong guarantees in the adversarial setting. In what follows, we can then consider that values $l^{t,v}_{\rho,m}$ are exogeneously determined. The vector of potential available capacities at time $t \in [T]$ is denoted by

$$\boldsymbol{l}_t = [l^{t,v}_{\rho,m}]_{(\rho,m,v)\in\bigcup_{i\in\mathcal{N}}\mathcal{R}_i\times\mathcal{M}_i\times\mathcal{V}}. \tag{4.8}$$

As we mentioned in Section 4.3.1, any request of type $\rho = (i, \boldsymbol{p}) \in \mathcal{R}$ can always be served by the associated repository model at node $v(\boldsymbol{p})$. This requirement can be expressed as follows:

$$\sum_{\rho\in\mathcal{R}_i} r^t_\rho \leq \sum_{m\in\mathcal{M}_i} \omega^{v(\boldsymbol{p})}_m L^{v(\boldsymbol{p})}_m, \forall i \in \mathcal{N}. \tag{4.9}$$

Thus, at any time $t \in [T]$ the adversary can select a request batch $\boldsymbol{r}_t$ and potential available capacity $\boldsymbol{l}_t$ from the set

$$\mathcal{A} \triangleq \Bigg\{(\boldsymbol{r}, \boldsymbol{l}) \in (\mathbb{N}\cup\{0\})^{\mathcal{R}} \times (\mathbb{N}\cup\{0\})^{\bigcup_{i\in\mathcal{N}}\mathcal{R}_i\times\mathcal{M}_i\times\mathcal{V}} : \sum_{\rho\in\mathcal{R}_i} r^t_\rho \leq \sum_{m\in\mathcal{M}_i} \omega^{v(\boldsymbol{p})}_m L^{v(\boldsymbol{p})}_m, l^v_{\rho,m} \leq \min\{L^v_m, r_\rho\},$$
$$\forall i \in \mathcal{N}, v \in \mathcal{V}, m \in \mathcal{M}, \rho \in \mathcal{R}\Bigg\}. \tag{4.10}$$

Note the constraint on potential available capacities is looser than the definition in (4.7) corresponding to a more powerful adversary.

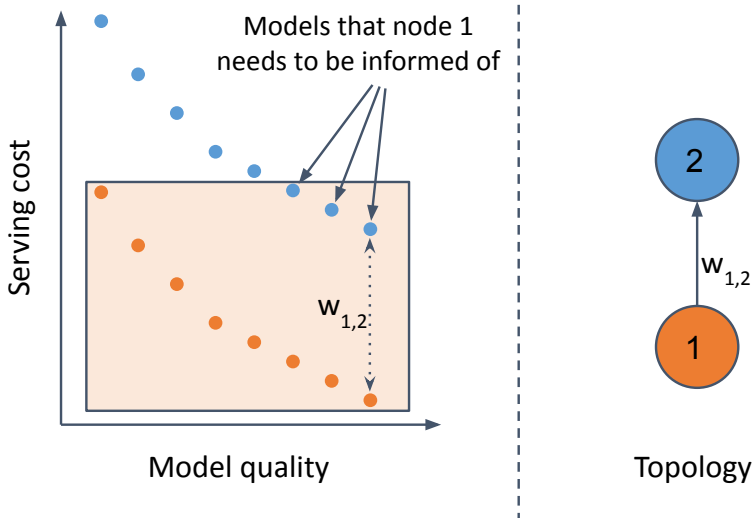


Figure 4.3: Necessity of partial synchronization in IDN among close-by computing nodes under the cost model in Eq. (4.6).

4.3.5 Serving Model

Given request $\rho{=}(i,\boldsymbol{p}){\in}\mathcal{R}$, let $K_\rho = |\boldsymbol{p}||\mathcal{M}_i|$ denote the maximum number of models that may encounter along its serving path $\boldsymbol{p}$. We order the corresponding costs $\{C^{p_j}_{\boldsymbol{p},m}, \forall m \in \mathcal{M}_i, \forall p_j \in \boldsymbol{p}\}$ in increasing order and we denote by $\kappa_\rho(v,m)$ the rank of model $m \in \mathcal{M}_i$ allocated at node v within the order defined above.[4] If $v \notin \boldsymbol{p}$ we have $\kappa_\rho(v,m) = \infty$.

If $\kappa_\rho(v,m) = k$, then model m at node v has the k-th smallest cost to serve request ρ. We denote the model service cost, its potential available capacity, and its effective capacity as γ^k_ρ, $\lambda^k_\rho(\boldsymbol{l}_t)$, and $z^k_\rho(\boldsymbol{l}_t,\boldsymbol{x})$, respectively:

$$\gamma^k_\rho = C^v_{\boldsymbol{p},m}, \quad \lambda^k_\rho(\boldsymbol{l}_t) = l^{t,v}_{\rho,m}, \quad z^k_\rho(\boldsymbol{l}_t,\boldsymbol{x}) = x^v_m l^{t,v}_{\rho,m}. \tag{4.11}$$

We assume the IDN serves requests as follows. Each request is forwarded along its serving path and served when it encounters a model with the smallest serving cost among those that are not yet saturated, i.e., that may still serve requests.

Since models do not necessarily provide increasing costs along the path, this serving strategy requires that a node that runs a model $m \in \mathcal{M}_i$ and receives a request for task i, knows whether there are better alternatives for serving task i upstream or not. In the first case, it will forward the request along the path, otherwise it will serve it locally. We argue that, in a real system, this partial knowledge can be achieved with a limited number of control messages. In fact, if node $v = p_h$ hosts the model with the k-th cost for request $(i,\boldsymbol{p})$, it only needs information about those models that (1) are located upstream on the serving path (i.e., on nodes $p_l \in \boldsymbol{p}$ with $l > h$), and (2) provide a cost smaller than γ^k_ρ. Since the cost increases with the network latency (as illustrated in Figure 4.3), the number of models satisfying these criteria is small in practice.[5] A node needs to propagate down-

[4]Note that we do not consider only the models deployed in the network, but all the possible node-model pairs.

[5]In realistic settings (Section 4.6), we experienced that each deployed model has at most 6 better alternatives on upstream nodes (worst case with α=1).

stream a control message with the information about the requests it can serve and the corresponding costs. Nodes forwarding the control message progressively remove the information about the tasks they can serve with a smaller cost, until the control message payload is empty and the message can be dropped. Every node $v \in \mathcal{V}$ generates this control message whenever the available capacity of any of the local models in v changes.

According to the presented serving strategy, the requests load is split among the currently available models giving priority to those that provide the smallest serving costs up to their saturation. In particular, model m with the k-th smallest cost will serve some requests of type ρ only if the less costly models have not been able to satisfy all of such requests (i.e., if $\sum_{k'=1}^{k-1} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) < r_\rho^t$). If this is the case, model m will serve with cost γ_ρ^k at most $z_\rho^k(\boldsymbol{l}_t, \boldsymbol{x})$ requests (its effective available capacity) out of the $r_\rho^t - \sum_{k'=1}^{k-1} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x})$ requests still to be satisfied. The aggregate cost incurred by the system at time slot t is then given by

$$C(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) = \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho} \gamma_\rho^k \cdot \min\left\{ r_\rho^t - \sum_{k'=1}^{k-1} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}), z_\rho^k(\boldsymbol{l}_t, \boldsymbol{x}) \right\} \cdot \mathbb{1}\left(\sum_{k'=1}^{k-1} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) < r_\rho^t \right). \tag{4.12}$$

Note that we introduce the $\min\{\cdot, \cdot\}$ operator, since the number of requests served by the k-th best model cannot exceed its effective capacity $z_\rho^k(\boldsymbol{l}_t, \boldsymbol{x})$. We add the indicator function $\mathbb{1}(\cdot)$ to indicate that the k-th best model does not serve any requests, in case better models (ranked from 1 to $k-1$) are able to satisfy all of them.

4.3.6 Allocation Gain and Static Optimal Allocations

We are interested in model allocations $\boldsymbol{x}$ that minimize the aggregate cost (4.12), or, equivalently, that maximize the *allocation gain* defined as

$$G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) = C(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) - C(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}). \tag{4.13}$$

The first term $C(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})$ on the right hand side is the service cost when only repository models are present in the network. Since intermediate nodes can help serving the requests at a reduced cost, $C(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})$ is an upper bound on the aggregate serving cost, and the allocation gain captures the cost reduction achieved by model allocation $\boldsymbol{x}$.

The static model allocation problem can then be formulated as finding the model allocation $\boldsymbol{x}^*$ that maximizes the time-averaged allocation gain over the time horizon T, i.e.,

$$\boldsymbol{x}_* = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \left(G_T(\boldsymbol{x}) \triangleq \frac{1}{T} \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) \right). \tag{4.14}$$

This is a submodular maximization problem under multiple knapsack constraints. In our context, this intuitively means that the problem is characterized by a diminishing return property: adding a model m to any node v gives us a marginal gain that depends on the current allocation: the more the models already deployed in the current allocation, the less the marginal gain we get by the new m. We prove submodularity in Lemma 24.2 in Appendix 24 . It is known that submodular maximization

problems are NP-hard and cannot be approximated with a ratio better than $(1 - 1/e)$ even under simpler cardinality constraints [179]. Under the multi-knapsack constraint, it is possible to solve the offline problem achieving a $(1 - 1/e - \epsilon)$-approximation through a recent algorithm proposed in [229].

Let us consider a model allocation $\boldsymbol{x}$. Within time slot t, the k smallest cost models along a path $\boldsymbol{p}$ that are suitable for request type $\rho = (i, \boldsymbol{p})$ can serve up to $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x})$ requests, where $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x})$ is defined as

$$Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) \triangleq \min\left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\}. \tag{4.15}$$

The $\min\{\cdot, \cdot\}$ operator denotes that we can never serve more than the number of requests r_ρ^t issued by users. Observe that, being the minimal allocation $\boldsymbol{\omega}$ an input parameter not dependent on our decisions, $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})$ is a constant. Additionally, since the models allocated in $\boldsymbol{x}$ always include those allocated in $\boldsymbol{\omega}$, we have $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) \geq Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})$.

Using (4.15), we provide the following alternative formulation of the allocation gain.

Lemma 4.3.1. *The allocation gain* (4.13) *has the following equivalent expression:*

$$G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) = \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \underbrace{\left(\gamma_\rho^{k+1} - \gamma_\rho^k\right)}_{\text{cost saving}} \underbrace{\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})\right)}_{\text{additional requests}}. \tag{4.16}$$

We prove this lemma in Appendix 25. This result tells us that the gain of a certain allocation $\boldsymbol{x}$ can be expressed as a sum of several components. In particular, for each request type ρ, the k-th smallest cost model along the path contributes to the gain with a component (1) proportional to its cost saving $\gamma_\rho^{k+1} - \gamma_\rho^k$ with respect to the $(k+1)$-th smallest cost model and (2) proportional to the amount of additional requests that the k-th smallest cost models in allocation $\boldsymbol{x}$ can serve with respect to the minimal allocation $\boldsymbol{\omega}$.

4.4 INFIDA Algorithm

In this section, we propose INFIDA, an online algorithm that can operate in a distributed fashion without requiring global knowledge of the allocation state and requests arrival. In Section 4.5, we show that INFIDA generates dynamically allocations experiencing average costs that converge to a $(1-1/e-\epsilon)$-approximation of the optimum, which matches the best approximation ratio achievable in polynomial time even in this online setting.

4.4.1 Algorithm Overview

On every node $v \in \mathcal{V}$, INFIDA updates the allocation $\boldsymbol{x}^v \in \mathcal{X}^v \subset \{0,1\}^{|\mathcal{M}|}$, by operating on a correspondent fractional state $\boldsymbol{y}^v \in \mathcal{Y}^v \subset [0,1]^{|\mathcal{M}|}$, and the fractional allocations satisfy the budget constraint in Eq. (4.2). Note that, if $\|\boldsymbol{s}^v\|_1 < b^v$ for a node $v \in \mathcal{V}$, we can always consider fractional

Algorithm 4.1 INFIDA distributed allocation on node v

1: **procedure** INFIDA($\boldsymbol{y}_1^v = \underset{\boldsymbol{y}^v \in \mathcal{Y}^v \cap \mathcal{D}^v}{\arg\min}\ \Phi^v(\boldsymbol{y}^v)$, $\boldsymbol{x}_1^v$=DepRound($\boldsymbol{y}_1^v$), $\eta \in \mathbb{R}_+$)
2: **for** $t = 1, 2, \ldots, T$ **do**
3: Compute $\boldsymbol{g}_t^v \in \partial_{\boldsymbol{y}^v} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t)$ through (4.18).
4: $\hat{\boldsymbol{y}}_t^v \leftarrow \nabla\Phi^v(\boldsymbol{y}_t^v)$ ▷ Map state to the dual space
5: $\hat{\boldsymbol{h}}_{t+1}^v \leftarrow \hat{\boldsymbol{y}}_t^v + \eta \boldsymbol{g}_t^v$ ▷ Take gradient step in the dual space
6: $\boldsymbol{h}_{t+1}^v \leftarrow (\nabla\Phi^v)^{-1}(\hat{\boldsymbol{h}}_{t+1}^v)$ ▷ Map dual state back to the primal space
7: $\boldsymbol{y}_{t+1}^v \leftarrow \mathcal{P}^{\Phi^v}_{\mathcal{Y}^v \cap \mathcal{D}^v}(\boldsymbol{h}_{t+1}^v)$ ▷ Project new state onto the feasible region using Algorithm .2
8: $\boldsymbol{x}_{t+1}^v \leftarrow$ DepRound($\boldsymbol{y}_{t+1}^v$) ▷ Sample a discrete allocation
9: **end for**
10: **end procedure**

allocations that consume entirely the allowed budget; otherwise, all the allocations are set to 1 (node v can store the whole catalog of models). Formally, if $\|\boldsymbol{s}^v\|_1 \geq b^v$ then

$$\mathcal{Y}^v \triangleq \left\{ \boldsymbol{y}^v \in [0,1]^{\mathcal{M}} : \sum_{m \in \mathcal{M}} y_m^v s_m^v = b^v, \forall v \in V \right\}; \tag{4.17}$$

otherwise, for the corner case $\|\boldsymbol{s}^v\|_1 < b^v$, we have $\mathcal{Y}^v \triangleq \left\{[1]^{\mathcal{M}}\right\}$.

Each variable y_m^v can be interpreted as the probability of hosting model m on node v, i.e., $y_m^v = \mathbb{P}[x_m^v = 1] = \mathbb{E}[x_m^v]$.

We define $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ as in (4.13), replacing $\boldsymbol{x}$ with $\boldsymbol{y}$. Note that $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ is a concave function of variable $\boldsymbol{y} \in \mathcal{Y} = \bigtimes_{v \in \mathcal{V}} \mathcal{Y}^v$ (see Lemma 28.1 in Appendix 28).

Within a time slot t, node v collects measurements from messages that have been routed through it (Section 4.4.2). At the end of every time slot, the node (1) computes its new fractional state $\boldsymbol{y}^v$, and (2) updates its local allocation $\boldsymbol{x}^v$ via randomized rounding (Section 4.4.3). INFIDA is summarized in Algorithm 4.1 and detailed below.

State computation. The fractional state $\boldsymbol{y}^v$ is updated through an iterative procedure aiming to maximize $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$. This could be the standard gradient ascent method, which updates the fractional state at each node as $\boldsymbol{y}_{t+1}^v = \boldsymbol{y}_t^v + \eta \boldsymbol{g}_t^v$, where $\eta_t \in \mathbb{R}_+$ is the step size and $\boldsymbol{g}_t^v$ is a subgradient of $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ with respect to $\boldsymbol{y}^v$.

In our work, we use OMA algorithm described in Section 2.2.3 in Chapter 2. OMA uses a function $\Phi^v : \mathcal{D}^v \to \mathbb{R}_+$ (mirror map) to map $\boldsymbol{y}$ to a dual space before applying the gradient ascent method; then the obtained state is mapped back to the primal space (lines 3–5 of Algorithm 4.1). OMA reduces to the classic gradient ascent method if Φ^v is the squared Euclidean norm (in this case the primal space coincides with the dual one). Instead, we use the weighted negative entropy map $\Phi^v(\boldsymbol{y}^v) = \sum_{m \in \mathcal{M}} s_m^v y_m^v \log(y_m^v)$, which is known to achieve better convergence rate in high dimensional spaces when each subgradient component is bounded.[6] To compute a feasible fractional state $\boldsymbol{y}^v$, we then perform a projection to the set $\mathcal{Y}^v$ on node v (line 6 of Algorithm 4.1). We adapt the

[6]Technically, the advantage in this setting derives from the infinite norm of the subgradient being independent from the space dimension, while the Euclidean norm grows proportionally to the squared root of the space dimension [53, Section 4.3].

projection algorithm in Algorithm 2.2 in Chapter 2 to obtain a weighted negative entropy projection $\mathcal{P}^{\Phi^v}_{\mathcal{Y}^v \cap \mathcal{D}^v}(\cdot)$. Our adaptation is described in Appendix 26.

Allocation update. Once the fractional state $\boldsymbol{y}^v$ has been updated, the final step of INFIDA is to determine a new random discrete allocation $\boldsymbol{x}^v$ and update the local models accordingly. The sampled allocation $\boldsymbol{x}^v$ should (1) comply with the budget constraint (4.2) on node v and (2) be consistent with the fractional state, i.e., $\mathbb{E}[x^v_m] = y^v_m \ \forall m \in \mathcal{M}$. To this purpose, we use the DepRound [173] subroutine (line 7 of Algorithm 4.1).

In the remainder of this section we detail how each node computes its contribution to the global subgradient, and the rounding strategy used to determine the discrete allocation.

4.4.2 Subgradient Computation

At the end of every time slot t, a subgradient $\boldsymbol{g}_t$ of the gain function in Eq. (4.16) at point $\boldsymbol{y}_t \in \mathcal{Y}$ is computed in a distributed fashion: each node v evaluates the (v,m)-th component of the subgradient for any $m \in \mathcal{M}$ as follows (see Appendix 27):

$$g^v_{t,m} = \sum_{\rho \in \mathcal{R}} l^{t,v}_{\rho,m} \cdot \left(\gamma_\rho^{K^*_\rho(\boldsymbol{y}_t)} - C^v_{\boldsymbol{p},m} \right) \cdot \mathbb{1}\left(\kappa_\rho(v,m) < K^*_\rho(\boldsymbol{y}_t) \right), \tag{4.18}$$

where $K^*_\rho(\boldsymbol{y}_t)$ is the order of the *worst needed* model, i.e., the model with the highest cost that is needed to serve all the r^t_ρ requests in the batch given the fractional state $\boldsymbol{y}_t$. Formally, $K^*_\rho(\boldsymbol{y}_t) \triangleq \min\left\{k \in [K_\rho - 1] : \sum_{k'=1}^{k} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) \geq r^t_\rho\right\}$.

For the sake of clarity assume that the mirror map is Euclidean, and then the dual and primal spaces collapse and $\hat{\boldsymbol{y}}_t = \boldsymbol{y}_t$. At each iteration, each component $y^{t,v}_m$ of the fractional allocation vector is updated by adding a mass equal to the product of $\eta > 0$ and the corresponding component of the subgradient (Algorithm 4.1, line 5). Observe that $g^v_{m,t}$ is the sum of different contributions, one per each request type ρ. Thanks to the indicator function, only the terms of the request types that are served by model m on v contribute to $g^v_{m,t}$. This contribution is proportional to the potential available capacity of model m on node v and to the relative gain $\left(\gamma_\rho^{K^*_\rho(\boldsymbol{y}_t)} - C^v_{\boldsymbol{p},m}\right)$, i.e., the cost reduction achieved when serving request type ρ with model m on v, rather than with the worst needed model. Then, gradient updates add more mass to the models that can contribute more to increase the gain. On the contrary, the projection step tends to remove the added mass from *all* components to satisfy the constraints. The overall effect is that fractional allocations of more (resp. less) useful models tend to increase (resp. decrease).

The subgradient in Eq. (4.18) can be computed at each node using only information from the control messages collected at the end of the time slot t. The steps needed to compute the subgradient are as follows.

1. At the end of the time slot, each node generates a control message for every received request type $\rho = (i, \boldsymbol{p})$ that is propagated along $\boldsymbol{p}$. The control message contains the quantity $r^t_\rho \geq 1$ (the multiplicity of the request), and a cumulative counter Z initialized to zero.

2. As the control message travels upstream, intermediate nodes add to Z the local values $z_\rho^k(\boldsymbol{l}_t, \boldsymbol{y}_t)$ (fractional effective capacity in Eq. (4.11)). These values are added following increasing values of cost. This message is propagated until $Z \geq r_\rho^t$, that is until the message reaches the $K_\rho^*(\boldsymbol{y}_t)$-th model.

3. Once the $K_\rho^*(\boldsymbol{y}_t)$-th model is detected, a control message is sent down in the opposite direction, containing the cost $\gamma_\rho^{K_\rho^*(\boldsymbol{y}_t)}$ of the last checked model. Every node v in the reverse direction reads the cost value from the control message and, for each model $m \in \mathcal{M}_i$, computes the quantity

$$h_m^v = l_{\rho,m}^{t,v} \cdot \left(\gamma_\rho^{K_\rho^*(\boldsymbol{y}_t)} - C_{\boldsymbol{p},m}^v\right). \tag{4.19}$$

4. Node v can then compute $g_{t,m}^v$ in Eq. (4.18) as follows

$$g_{t,m}^v = \sum_{m \in \mathcal{M}_i} h_m^v.$$

Note that the cost in Eq. (4.6) does not necessarily increase along the path. Therefore, a traversed node is not able to update directly the variable Z when there exist upstream nodes with lower cost. In this case, the node simply appends the information $(z_\rho^k(\boldsymbol{l}_t, \boldsymbol{y}_t), \gamma_\rho^k)$ to the message, and lets upstream nodes to apply any pending update in the correct order.

4.4.3 State Rounding

Once the new fractional state $\boldsymbol{y}_{t+1}$ is computed, each node v independently draws a random set of models to store locally in such a way that $\mathbb{E}[\boldsymbol{x}_{t+1}^v] = \boldsymbol{y}_{t+1}^v$. This sampling guarantees that the final allocation $\boldsymbol{x}_{t+1}^v$ satisfies constraint (4.2) in expectation. A naive approach is to draw each variable $x_m^{v,t+1}$ independently, but it leads to a large variance of the total size of the models selected, potentially exceeding by far the allocation budget at node v.

To construct a suitable allocation we adopt the DepRound procedure from [173]. The procedure modifies the fractional state $\boldsymbol{y}_{t+1}^v$ iteratively: at each iteration, DepRound operates on two fractional variables $y_m^{v,t+1}, y_{m'}^{v,t+1}$ so that at least one of them becomes integral and the aggregate size of the corresponding models $s_m^v y_m^{v,t+1} + s_{m'}^v y_{m'}^{v,t+1}$ does not change. This operation is iterated until all variables related to node v are rounded except (at most) one, which we call residual fractional variable. This is done in $O(|\mathcal{M}|)$ steps.

Note that, to satisfy $\mathbb{E}[\boldsymbol{x}_{t+1}^v] = \boldsymbol{y}_{t+1}^v$, the residual fractional variable, say it $y_{\bar{m}}^{v,t+1}$, needs to be rounded. At this point $x_{\bar{m}}^{v,t+1}$ can be randomly drawn. Now the final allocation can exceed the budget bound b^v by at most $s_{\bar{m}}$. These (slight) occasional violations of the constraint may not be a problem, e.g., at an edge server running multiple applications, where resources may be partially redistributed across different applications; they may be explicitly accounted for in the service level agreements. If the budget bound cannot be exceeded even temporarily, the node is not able to store the model $\bar{m}$, but it may still exploits the residual free resources to deploy the model that provides the best marginal gain among those that fit the available budget. In practice, we expect the corresponding gain decrease to be negligible.

4.5 Theoretical Guarantees

We provide the optimality guarantees of our INFIDA algorithm in terms of the ψ-regret [177]. In our scenario, the ψ-regret is defined as the gain loss in comparison to the best static allocation in hindsight, i.e., $\boldsymbol{x}_* \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x})$, discounted by a factor $\psi \in (0, 1]$. Formally,

$$\psi\text{-Regret}_{T,\mathcal{X}} \triangleq \sup_{\{\boldsymbol{r}_t,\boldsymbol{l}_t\}_{t=1}^{T}\in\mathcal{A}^T} \left\{ \psi \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_*) - \mathbb{E}\left[\sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t)\right]\right\},$$

where allocations $\boldsymbol{x}_t$ are computed using INFIDA and the expectation is over the randomized choices of DEPROUND. Note that, by taking the supremum over all request sequences and potential available capacities, we measure regret in an adversarial setting, i.e., against an adversary that selects, for every $t \in [T]$, vectors $\boldsymbol{r}_t$ and $\boldsymbol{l}_t$ to jeopardize the performance of our algorithm. Obviously, we do not expect such an adversary would exist in reality, but the adversarial analysis provides bounds on the behavior of INFIDA in the worst case.

The adversarial analysis is a modeling technique to characterize system performance under highly volatile external parameters (e.g., the sequence of requests $\boldsymbol{r}_t$) or difficult to model system interactions (e.g., the available capacities $\boldsymbol{l}_t$). This technique has been recently successfully used to model caching problems (e.g., in [139, 171]). Our main result is the following (the full proof is in Appendix 29):

Theorem 4.5.1. INFIDA *has a sublinear* $(1 - 1/e)$*-regret w.r.t. the time horizon* T*, i.e., there exists a constant* A *such that:*

$$(1 - 1/e)\text{-Regret}_{T,\mathcal{X}} \leq A\sqrt{T}, \tag{4.20}$$

where $A \propto RL_{\max}\Delta_C$*.* R*,* $L_{\max}$*, and* Δ_C *are upper bounds, respectively, on the total number of request types at any time slot, on the model capacities, and on the largest serving cost difference between serving at a repository node and at any other node.*

We first prove that the expected gain of the randomly sampled allocations $\boldsymbol{x}_t$ is a $(1-1/e)$-approximation of the fractional gain. Then, we use online learning results [53] to bound the regret of Online Mirror Ascent schemes operating on a convex decision space and against concave gain functions picked by an adversary. The two results are combined to obtain an upper bound on the $(1-1/e)$-regret. The full proof and the full characterization of A are provided in Appendix 29. in Appendix 29.

We observe that the regret bound depends crucially on the maximum number of request types R, maximum model capacity $L_{\max}$ and maximum serving cost difference Δ_C. When considering the cost model in Eq. (4.6), we can consider for Δ_C the sum of the total latency of the heaviest path, the parameter α, and the largest inference delay. This result is intuitive: when these values are bigger, the adversary has a larger room to select values that can harm the performance of the system.

As a direct consequence of Theorem 4.5.1, the expected time averaged $(1-1/e)$-regret of INFIDA can get arbitrarily close to zero for large time horizon. Hence, INFIDA achieves a time averaged

expected gain that is a $(1 - 1/e - \epsilon)$-approximation of the optimal time averaged static gain, for arbitrarily small ϵ.

Observe that INFIDA computes a different $\boldsymbol{x}_t$ at every time slot. Intuitively, this allows it to "run after" the exogenous variation of the adversarial input $\{\boldsymbol{r}_t, \boldsymbol{l}_t\}_{t=1}^{T} \in \mathcal{A}^T$. An alternative goal that can be achieved by INFIDA is to find a static allocation $\bar{\boldsymbol{y}}$. In order to do so, we need to (1) run INFIDA for $\tilde{T}$ time-slots, (2) based on the $\{\boldsymbol{y}_t\}_{t=1}^{\tilde{T}}$ computed by INFIDA, calculate $\bar{\boldsymbol{x}}$ (the exact calculation is in Proposition 4.5.2), (3) deploy in the IDN the allocation $\bar{\boldsymbol{x}}$ and keep it static, in order to avoid switches. Obviously, we would like the quality of $\bar{\boldsymbol{x}}$ to be close to the best $\boldsymbol{x}_*$, defined in (4.14). The following proposition shows that the gain achieved with our $\bar{\boldsymbol{x}}$ is boundedly close to the optimum. Moreover, since (4.14) is NP-hard, there cannot exist better bounds than the one we achieve, assuming P $\neq$ NP [179].

Proposition 4.5.2. *(offline solution) Replace in INFIDA the allocation gain $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ by $G_T(\boldsymbol{y})$ (defined in (4.14)). After $\tilde{T}$ iterations, let $\bar{\boldsymbol{y}}$ be the average fractional allocation $\bar{\boldsymbol{y}} = \frac{1}{\tilde{T}} \sum_{\tilde{t}=1}^{\tilde{T}} \boldsymbol{y}_t$, and $\bar{\boldsymbol{x}}$ the random state sampled from $\bar{\boldsymbol{y}}$ using* DepRound. *$\forall \epsilon > 0$, for $\bar{T}$ large enough, $\bar{\boldsymbol{x}}$ satisfies*

$$\mathbb{E}\left[G_T(\bar{\boldsymbol{x}})\right] \geq \left(1 - \frac{1}{e} - \epsilon\right) G_T(\boldsymbol{x}_*), \tag{4.21}$$

where $\boldsymbol{x}_* = \arg\max_{\boldsymbol{x} \in \mathcal{X}} G_T(\boldsymbol{x})$.

The proof is given in Appendix. 30.

4.6 Experimental Results

We evaluate INFIDA by simulating a realistic scenario based on the typical structure of ISP networks. We compare our solution with a greedy heuristic and its online variant (described below), as the greedy heuristic is known to achieve good performance in practice for submodular optimization [177].

Topology. We simulate a hierarchical topology similar to [230] that spans between edge and cloud, with different capacities at each tier. We consider 5 tiers: base stations (tier 4), central offices (tiers 3, 2), ISP data center (tier 1), a remote cloud (tier 0). We assume a hierarchical geographic distribution similar to LTE. We take the Round-Trip Time (RTT) across the different tiers as follows: tier 4 to tier 3 takes 6 ms, tier 3 to tier 2 takes 6 ms, tier 2 to tier 1 takes 15 ms, and tier 1 to tier 0 takes 40 ms. We execute our experiments at two different scales: *Network Topology I* counts 24 base stations and 36 nodes in total, while *Network Topology II* is a simpler 5-node scenario with 2 base stations.

Processing Units. We take GPU memory of the computing nodes as the limiting budget. The node at tier 0 can store the entire models catalog. We simulate the performance of two different processing units: the computing nodes at tiers 0 and 1 are equipped with high-end GPUs (Titan RTX), and the remaining tiers 2–4 have mid-tier GPUs (GeForce GTX 980). The budget of each computing tier is given as follows: a tier-1 node has 16GB GPUs, a tier-2 node has 12GB GPUs, a tier-3 node has 8GB GPUs, and a tier-4 node has 4GB GPUs.

Catalog and requests. We simulate performance based on state-of-the-art pre-trained models and their pruned versions [231, 232], profiled for each simulated processing unit, for a total of 10 models

variants of yolov4	accuracy (mAP@0.5)	memory (MB)	frames per second Titan RTX	frames per second GTX 980
608p	65.3	1577	41.3	14.2
512p	64.9	1185	55.5	18.9
416p	62.8	1009	73.8	25.1
320p	57.3	805	100	34.1
3.99pruned	55.1	395	209	71.0
8.09pruned	51.4	195	329	112
10.10pruned	50.9	156	371	126
14.02pruned	49.0	112	488	166
tiny-416p	38.3	187	888	302
tiny-288p	34.4	160	1272	433

Table 4.2: Catalog for variants of YOLOv4 [231] profiled on two different Processing Units. Accuracy is for the MS COCO dataset. Values for the pruned variants are adapted from [232].

(Table 4.2). We consider a task catalog with $|\mathcal{N}| = 20$ different object detection tasks. We allow 3 duplicates per model; this gives $|\mathcal{M}_i| = 30$ alternative models per task $i \in \mathcal{N}$. Note how, as model complexity decreases, the number of frames a GPU can process per second increases, and consequently the average inference delay decreases.

The time slot duration is set to 1 minute and requests arrive at a constant rate of 7,500 requests per second (rps), unless otherwise said. Each request type is assigned randomly to two base stations in tier 4. The corresponding task is selected according to two different popularity profiles: (1) in the *Fixed Popularity Profile* , a request is for task i with constant probability $p(1) = \frac{(i+1)^{-1.2}}{\sum_{i' \in \mathcal{N}} (i'+1)^{-1.2}}$ (a Zipf distribution with exponent 1.2), while (2) in the *Sliding Popularity Profile*, the l-th consecutive request is for task i with probability $\tilde{p}(i, l) = p\left((i + 5\lfloor l/W \rfloor) \mod 20\right)$, that is, the popularity of the tasks changes through a cyclic shift of 5 tasks every 1 hour for a request rate of 7,500 rps ($W = 2.3 \times 10^7$).

Static greedy. We adapt the static greedy (SG) heuristic from the cost-benefit greedy in [177]. SG operates in hindsight seeking maximization of the time averaged allocation gain over the whole time horizon T, as in Eq. (4.14). Starting from an empty allocation, this policy progressively allocates the model that provides the highest marginal gain normalized by size, among those that meet the budget constraints. This process is repeated until either the intermediate allocation is capable of serving all requests or none of the remaining valid allocations introduces a positive marginal gain.

Online load-aware greedy heuristic. As INFIDA is the first online policy for ML models' allocation in IDNs, there is no clear baseline to compare it with. We then propose an online heuristic based on SG, which we call online load-aware greedy (OLAG). A node v uses counters $\phi^v_{m,\rho}$ to keep track of the number of times a request $\rho \in \mathcal{R}$ is forwarded upstream but could have been served locally at a lower cost compared to the repository, i.e., using a model $m \in \mathcal{M}$ with positive gain that we denote by $q^v_{m,\rho}$. For every model m, an importance weight is computed as $w^v_m = \frac{1}{s^v_m} \frac{1}{|\mathcal{R}|} \sum_{\rho \in \mathcal{R}} q^v_{m,\rho} \min\{\phi^v_{m,\rho}, L^v_m\}$, where s^v_m is the size of model m and $\min\{\phi^v_{m,\rho}, L^v_m\}$ is the number of requests that could have been improved by m. At the end of a time slot, the node selects the model m_* with the highest importance while respecting the resource budget constraint, then subtracts the quantity $\min\{\phi^v_{m,\rho}, L^v_m\}$ from $\phi^v_{m_*,\rho}$ and from all the $\phi^v_{m',\rho}$: $q^v_{m',\rho} < q^v_{m_*,\rho}$, i.e., models that

provide a gain lower than m_*. This procedure is repeated until the resource budget of the node is consumed.

Offline INFIDA. Motivated by Proposition 4.5.2, we implemented also an offline version of INFIDA that we call INFIDA$_{\text{OFFLINE}}$, which maximizes the time-averaged gain (4.14) over the whole time horizon T. The potential available capacities are determined at runtime from the current allocations and request batches (rather than by an adversary).

Performance Metrics. The performance of a policy $\mathcal{P}$ with the associated sequence of allocation decisions $\{\boldsymbol{x}_t\}_{t=1}^T$ is evaluated in terms of the time-averaged gain normalized to the number of requests per time slot (NTAG):

$$\text{NTAG}(\mathcal{P}) = \sum_{t=1}^{T} \frac{1}{T \left\|\boldsymbol{r}_t\right\|_1} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t). \tag{4.22}$$

Moreover, we evaluate the update cost of a policy $\mathcal{P}$ with the associated sequence of allocation decisions $\{\boldsymbol{x}_t\}_{t=1}^T$ by quantifying the total size of fetched models over T time slots. The update cost is reflected by the Time-Averaged Model Updates (MU) metric defined as:

$$\text{MU}(\mathcal{P}) \triangleq \frac{1}{T} \sum_{t=2}^{T} \sum_{(v,m)\in\mathcal{V}\times\mathcal{M}} s_m^v \max\{0, x_{t,m}^v - x_{t-1,m}^v\}. \tag{4.23}$$

4.6.1 Trade-off between Latency and Accuracy

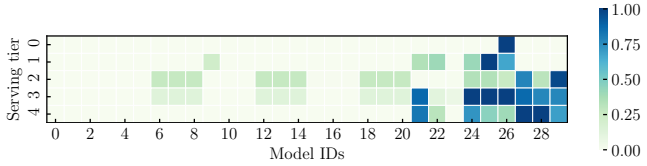
We first evaluate how INFIDA adapts to different trade-offs between end-to-end latency and inference accuracy by varying the trade-off parameter α.[7]

Figure 4.4 shows the fractional allocation decision at each tier of the network topology for different values of α (remember that the smaller α the more importance is given to the latency rather than to inaccuracy, see Eq. (4.6)). Models are ordered horizontally by increasing accuracy with 3 potential replicas for each model, and only the models able to serve the most popular request are shown. Note that the tier-0 node acts as a repository and its allocation is fixed; moreover, in Fig 4.4 the repository node picks the second most accurate model because it provides the smallest combined cost in Eq. (4.6).

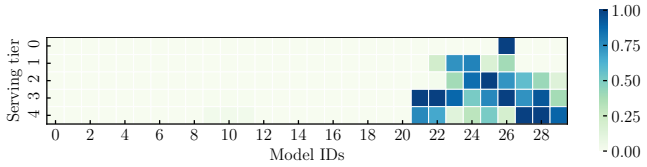
For $\alpha = 3$ (Figure 4.4a), INFIDA allocates a considerable amount of small models (which provide low accuracy) near the edge (tiers 1–3 and model IDs 0–18), as they can serve a larger number of requests compared to higher quality models with low inference delay. By giving more importance to the accuracy ($\alpha = 4$) the system tends to deploy more accurate models and rarely allocates small models (Figure 4.4b). For $\alpha = 5$, the number of models deployed on lower tiers decreases, as the system allocates no small models in practice (model IDs 0–20) and selects instead multiple replicas of the most accurate models (Figure 4.4c). Since higher quality models feature, in general, a lower serving capacity (Table 4.2), Figure 4.4c suggests that a significant number of requests is served in the cloud (Tier 0) for this value of α.

[7]The inaccuracy cost is taken in 0–100, then α picked here corresponds to 100× scaling of the parameter defined in Eq. (4.6).

(a) $\alpha = 3$



(b) $\alpha = 4$



(c) $\alpha = 5$

Figure 4.4: Fractional allocation decisions y_m^v of INFIDA on the various tiers of *Network Topology I* under *Fixed Popularity Profile*. We only show the allocations corresponding to the models capable to serve the most popular request. The model IDs are sorted by increasing accuracy.

Figure 4.5 shows the average experienced inaccuracy (inaccuracy is given by 100 − mAP and mAP is the mean average precision) and latency for different values of α under *Network Topology I* and *Fixed Popularity Profile*. When accuracy is not important (i.e., $\alpha \approx 0$), INFIDA effectively achieves very low end-to-end latency (few milliseconds) by prioritizing the deployment of small and inaccurate models near to the edge nodes. Noticeably, the trend in both curves (decreasing inaccuracy and increasing latency) suggests that, when higher accuracy is required, the system starts to prefer models deployed close to the cloud, leading to a sudden change in the trade-off and to a significant increase in latency.

In Figure 4.6 we show the normalized time-averaged gain of INFIDA compared to OLAG, SG, and INFIDA$_{\text{OFFLINE}}$ for different values of α under the *Sliding Popularity Profile*. Results are shown both for *Network Topology I* (Figure 4.6a) and for *Network Topology II* (Figure 4.6b).

The plot shows that the gain decreases by increasing α. This is expected since the gain (4.13) is defined as the improvement w.r.t. the repository allocation (tier 0). Therefore, when the latency is not important, high accuracy models at tier 0 are preferred, and there is no much room for improvement (the optimal gain eventually tends to zero for $\alpha \to +\infty$). Note that, in general, SG and INFIDA$_{\text{OFFLINE}}$ policies perform worse than their offline counterparts, as they pick a single allocation that is the best w.r.t. the whole sequence of requests. However, in the *Sliding Popularity Profile* the best decision changes periodically, and only the online policies have the ability to adapt to such change. Moreover, we observe that consistently INFIDA$_{\text{OFFLINE}}$ has better performance than SG: although both policies are offline, INFIDA$_{\text{OFFLINE}}$ manages to provide a better allocation.

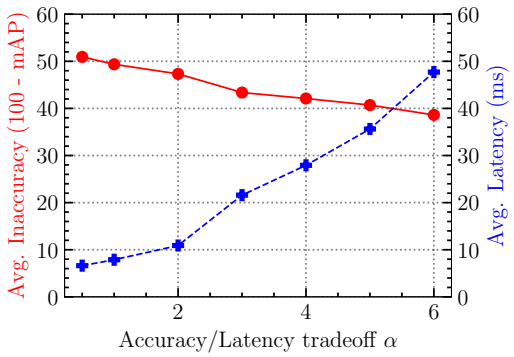


Figure 4.5: Average latency (dashed line) and inaccuracy (solid line) costs experienced with INFIDA for different values of α under *Network Topology I* and *Fixed Popularity Profile*.

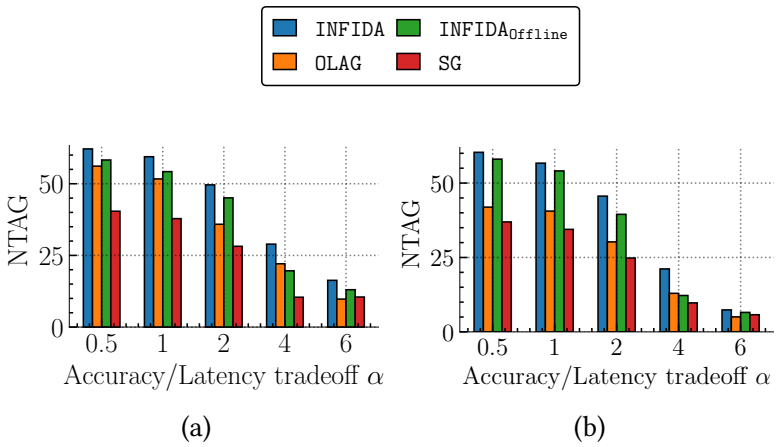
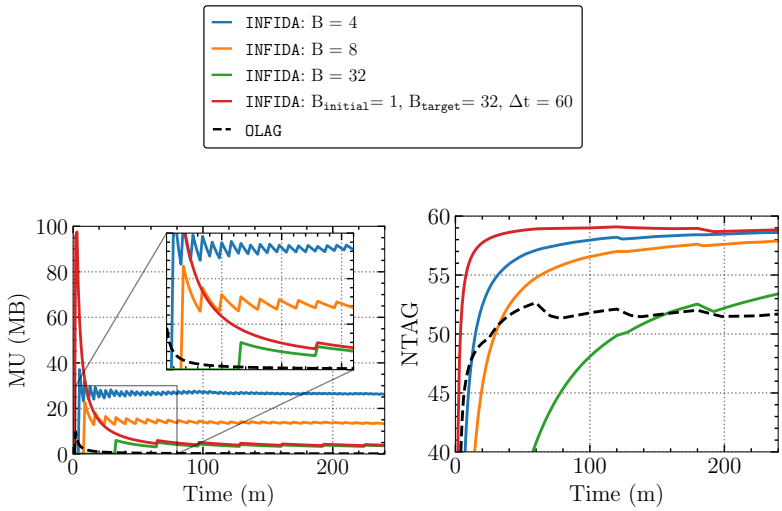


Figure 4.6: NTAG of the different policies under *Sliding Popularity Profile* and network topologies: (a) *Network Topology I*, and (b) *Network Topology II*.

4.6.2 Trade-off between model updates and service cost.

In this set of experiments, we evaluate how the frequency at which INFIDA updates the model allocation affects the stability of the system. Indeed, frequent updates could lead to massive migrations with an overhead on network bandwidth. As an evaluation metric, we measure the total size of fetched models averaged over time (see the performance metric in Eq. (4.23)). We introduce B that we call the *refresh period*, and we restrict INFIDA to only sample a physical allocation every $B \in \{4, 8, 32\}$ time slots (line 8 in Algorithm 4.1). Additionally, we experiment linear stretching of the refresh period B with initial period $B_{\text{init}} = 1$ and target period $B_{\text{target}} = 32$ in a stretching duration of Δt = 1H. We run this experiment under *Network Topology I* and *Sliding Popularity Profile*. We set the trade-off parameter $\alpha = 1$.

In particular, Figure 4.7a shows the update cost (MU) for different refresh periods, while Figure 4.7b shows the NTAG. Both plots include the performance of the OLAG heuristic. We observe that, by increasing the refresh period B, the system fetches a smaller number of models, and therefore the update cost decreases, at the expense of reactivity. Nevertheless, even for large values of B



(a) Models Updates (MU) (b) NTAG

Figure 4.7: (a) Models Updates (MU) and (b) NTAG of OLAG and INFIDA for different values of refresh period $B \in \{4, 8, 16\}$, and for a dynamic refresh period with initial value $B_{\text{init}} = 1$, target value $B_{\text{target}} = 32$ and stretching duration $\Delta t = 60$ (1H). The experiment is run under *Network Topology I* and *Sliding Popularity Profile*.

INFIDA eventually exceeds OLAG in performance: this result is expected since the algorithm continues to learn on the fractional (virtual) states and only the physical allocations are delayed and eventually catch-up for a large time horizon. On the other hand, we observe that OLAG is relatively conservative in updating its allocation, as it quickly picks a sub-optimal allocation and rarely updates it.

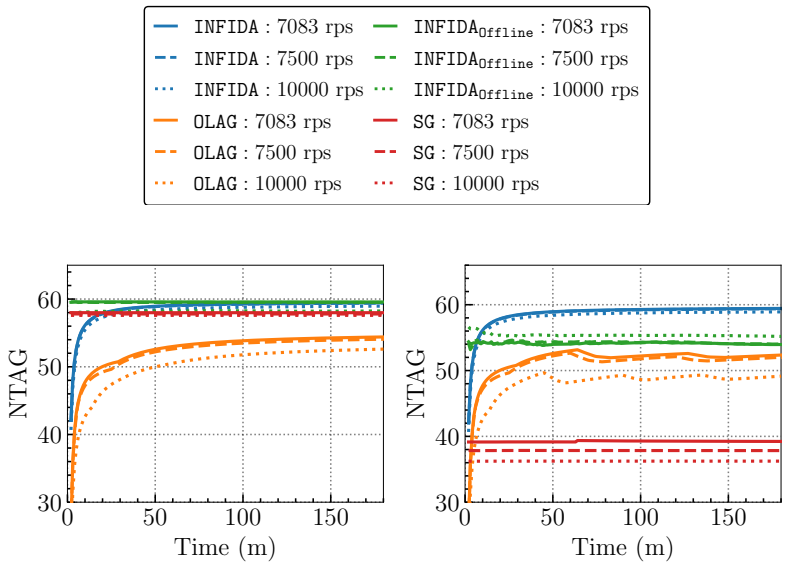
The previous observation motivates the use of a dynamic refresh period. By refreshing more frequently at the start we allow the physical allocation generated by INFIDA to catch up quickly with the fractional states as shown in Figure 4.7b: a dynamic refresh period that stretches from $B_{\text{init}} = 1$ to $B_{\text{target}} = 32$ attains much faster and more precise convergence. This is achieved at the expense of a high update cost at the start, which is, however, quickly dampened until it matches the same update cost of fixing the refresh period to B_{target}.

4.6.3 Scalability on Requests Load

We show how the system performs under different requests loads. For this set of experiments, we set $\alpha = 1$. Figure 4.8 compares the results for the different allocation policies.

We notice that, being INFIDA$_{\text{OFFLINE}}$ and SG offline policies, they perform well when the popularity profile is static (Figure 4.8a), but deteriorate under *Sliding Popularity Profile*. Notably, the performance degradation of INFIDA$_{\text{OFFLINE}}$ (≈8%) is considerably limited compared to SG (≈30%), which even gets worse when increasing the requests load.

Figure 4.8 shows that, in general, INFIDA provides a higher gain compared to the OLAG heuristic. In particular, under *Fixed Popularity Profile* INFIDA manages to converge to the same NTAG



(a) *Fixed Popularity Profile* (b) *Sliding Popularity Profile*

Figure 4.8: NTAG of the different policies for different request rates under *Network Topology I*.

provided by its offline counterpart (Figure 4.8a), which is ≈10% better than the one provided by OLAG when the load is 7,083 rps. Additionally, OLAG's performance deteriorates when the requests load increases from 7,083 rps to 10,000 rps. It is also noteworthy that OLAG visibly suffers from perturbed performance when the popularity of the tasks changes over time (Figure 4.8b). On the other hand, results show the robustness of INFIDA against changing request loads and popularity: the algorithm preserves its performance in terms of normalized time-averaged gain for the analyzed request loads and under both *Fixed Popularity Profile* and *Sliding Popularity Profile*, always converging to the highest NTAG.

Last, in Figure 4.9 we evaluate separately the average latency and inaccuracy attained by the different policies using different values of $\alpha \in \{0.5, 1, 2, 3, 4, 5, 6\}$ under *Fixed Popularity Profile* and *Network Topology II*. We observe that INFIDA and its offline counterpart INFIDA$_{\text{Offline}}$ consistently provide the lowest average inaccuracy and latency under both high request load (10,000 rps) and default request load (7,500 rps). INFIDA$_{\text{Offline}}$ is run with hindsight and serves as a lower bound on the achievable latency and inaccuracy under a fixed popularity request process (as in Figure 4.8a).

4.7 Conclusion

In this chapter, we introduce the idea of inference delivery networks (IDNs), networks of computing nodes that coordinate to satisfy inference requests in the continuum between Edge and Cloud. IDN nodes can serve inference requests with different levels of accuracy and end-to-end latency, based on their geographic location and processing capabilities. We formalize the NP-hard problem of allocating ML models on IDN nodes, capturing the trade-off between latency and accuracy. We propose INFIDA, a dynamic ML model allocation algorithm that operates in a distributed fashion and

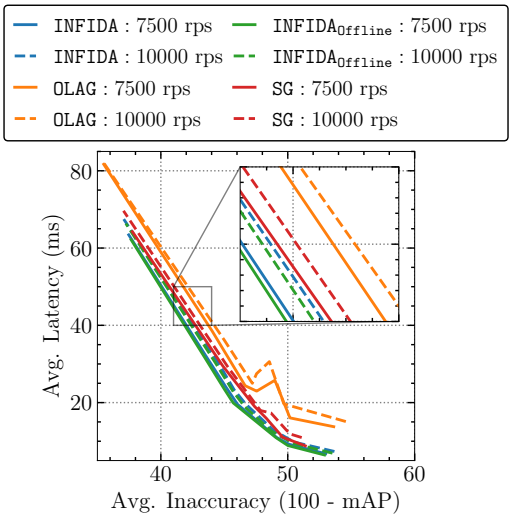


Figure 4.9: Average Latency vs. Average Inaccuracy obtained for different values of $\alpha \in \{0.5, 1, 2, 3, 4, 5, 6\}$ under *Fixed Popularity Profile* and *Network Topology II*.

provides strong guarantees in an adversarial setting. We evaluate INFIDA simulating the realistic scenario of an ISP network, and compared its performance under two different topologies with both an offline greedy heuristic and its online variant. Our results show that INFIDA adapts to different latency/accuracy trade-offs and scales well with the number of requests, outperforming the greedy policies in all the analyzed settings.

CHAPTER 5

Long-term Fairness in Dynamic Resource Allocation

5.1 Introduction

Achieving fairness when allocating resources in communication and computing systems has been a subject of extensive research, and has been successfully applied in numerous practical problems. Fairness is leveraged to perform congestion control in the Internet [233, 234], to select transmission power in multi-user wireless networks [235, 236], and to allocate multidimensional resources in cloud computing platforms [237–239]. Depending on the problem at hand, the criterion of fairness can be expressed in terms of how the service performance is distributed across the end-users, or in terms of how the costs are balanced across the servicing nodes. The latter case exemplifies the natural link between fairness and load balancing in resource-constrained systems [240, 241]. A prevalent fairness metric is α-fairness, which encompasses the utilitarian principle (Bentham-Edgeworth solution [242]), proportional fairness (Nash bargaining solution [243]), max-min fairness (Kalai–Smorodinsky bargaining solution [244]), and, under some conditions, Walrasian equilibrium [245]. All these fairness metrics have been used in different cases for the design of resource management mechanisms [246, 247].

A common limitation of the above works is that they consider *static* environments. That is, the resources to be allocated and, importantly, the users' utility functions, are fixed and known to the decision maker. This assumption is very often unrealistic for today's communication and computing systems. For instance, in small-cell mobile networks the user churn is typically very high and unpredictable, thus hindering the fair allocation of spectrum to cells [4]. Similarly, placing content files at edge caches to balance the latency gains across the served areas is non-trivial due to the non-stationary and fast-changing patterns of requests [5]. At the same time, the increasing virtualization of these systems introduces cost and performance volatility, as extensive measurement studies have revealed [6–8]. This uncertainty is exacerbated for services that process user-generated data (e.g., streaming data applications) where the performance (e.g., inference accuracy) depends also on a priori unknown input data and dynamically selected machine learning libraries [9–11].

5.1.1 Contributions

This paper makes the next step towards enabling long-term fairness in dynamic systems. We consider a system that serves a set of agents $\mathcal{I}$, where a controller selects at each timeslot $t \in \mathbb{N}$ a resource allocation profile $\boldsymbol{x}_t$ from a set of eligible allocations $\mathcal{X}$ based on past agents' utility functions $\boldsymbol{u}_{t'} : \mathcal{X} \to \mathbb{R}^{\mathcal{I}}$ for $t' < t$ and of α-fairness function $F_\alpha : \mathbb{R}^{\mathcal{I}}_{\geq 0} \to \mathbb{R}$. The utilities might change due to unknown, unpredictable, and (possibly) non-stationary perturbations that are revealed to the controller only after it decides $\boldsymbol{x}_t$. We employ the terms *horizon-fairness* (HF) and *slot-fairness* (SF) to distinguish the different ways fairness can be enforced in a such time-slotted dynamic system. Under horizon-fairness, the controller enforces fairness on the aggregate utilities for a given time horizon T, whereas under slot-fairness, it enforces fairness on the utilities at each timeslot separately. Both metrics have been studied in previous work, e.g., see [248–251] and the discussion in Section 5.2. Our focus is on horizon-fairness, which raises novel technical challenges and subsumes slot-fairness as a special case.

We design the *online horizon-fair* (OHF) policy by leveraging *online convex optimization* (OCO) [20], to handle this reduced-information setting under a powerful *adversarial* perturbation model. Adversarial analysis is a modeling technique to characterize a system's performance under unknown and hard to characterize exogenous parameters and has been recently successfully used to model caching problems (e.g., in [139, 171, 252–256]). In our context, the performance of a resource allocation policy $\boldsymbol{\mathcal{A}}$ is evaluated by the *fairness regret*, which is defined as the difference between the α-fairness, over the time-averaged utilities, achieved by a static optimum-in-hindsight (*benchmark*) and the one achieved by the policy:

$$\mathfrak{R}_T\left(F_\alpha, \boldsymbol{\mathcal{A}}\right) \triangleq \sup_{\{\boldsymbol{u}_t\}_{t=1}^T \in \mathcal{U}^T} \left\{ \max_{\boldsymbol{x} \in \mathcal{X}} F_\alpha\left(\frac{1}{T}\sum_{t\in\mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x})\right) - F_\alpha\left(\frac{1}{T}\sum_{t\in\mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x}_t)\right)\right\}. \tag{5.1}$$

If the fairness regret vanishes over time (i.e., $\lim_{T\to\infty} \mathfrak{R}_T\left(F_\alpha, \boldsymbol{\mathcal{A}}\right) = 0$), policy $\boldsymbol{\mathcal{A}}$ will attain the same fairness value as the static benchmark under any possible sequence of utility functions. A policy that achieves sublinear regret under these adversarial conditions, can also succeed in more benign conditions where the perturbations are not adversarial, or the utility functions are revealed at the beginning of each slot.

The fairness regret metric (5.1) departs from the template of OCO. In particular, the scalarization of the vector-valued utilities, through the α-fairness function, is not applied at every timeslot to allow for the controller to easily adapt its allocations, instead is only applied at the end of the time horizon T. Our first result characterizes the challenges in tackling this learning problem. Namely, Theorem 5.4.1 proves that, when utility perturbations are only subject to four mild technical conditions, such as in standard OCO, it is impossible to achieve vanishing fairness-regret. Similar negative results were obtained under different setups of primal-dual learning and online saddle point learning [257–259], but they have been devised for specific problem structures (e.g., online matrix games) and thus do not apply to our setting.

In light of this negative result, we introduce additional *necessary* conditions on the adversary to obtain a vanishing regret guarantee. Namely, the adversary can only induce perturbations to the time-averaged utilities we call budgeted-severity or partitioned-severity constrained. These conditions capture several practical utility patterns, such as non-stationary corruptions, ergodic and

periodic inputs [249, 260–262]. We proceed to propose the OHF policy which adapts dynamically the allocation decisions and provably achieves $\mathfrak{R}_T\left(F_\alpha, \boldsymbol{\mathcal{A}}\right) = o(1)$ (see Theorem 5.4.2).

The OHF policy employs a novel learning approach that operates concurrently, and in a synchronized fashion, in a primal and a dual (conjugate) space. Intuitively, OHF learns the weighted time-varying utilities in a primal space, and learns the weights accounting for the global fairness metric in some dual space. To achieve this, we develop novel techniques through a convex conjugate approach (see Lemmas 31.1, 31.2, and 31.4 in the Appendix).

Finally, we apply our fairness framework to a representative resource management problem in virtualized caching systems where different caches cooperate by serving jointly the received content requests. We evaluate the performance of OHF with its slot-fairness counterpart policy through numerical examples. We evaluate the price of fairness of OHF, which quantifies the efficiency loss due to fairness, across different network topologies and participating agents. Lastly, we apply OHF to a Nash bargaining scenario, a concept that has been widely used in resource allocation to distribute to a set of agents the utility of their cooperation [263–266].

5.1.2 Outline of Paper

The paper is organized as follows. The related literature is discussed in Section 5.2. The definitions and background are provided in Section 5.3. The adversarial model and the proposed algorithm are presented in Section 5.4. Extensions to the fairness framework are provided in Section 5.5. The resource management problem in virtualized caching systems application is provided in Section 5.6. Finally, we conclude the paper and provide directions for future work in Section 5.7.

5.2 Related Work

5.2.1 Fairness in Resource Allocation

Fairness has found many applications in wired and wireless networking [233–236, 267], and cloud computing platforms [237–239]. Prevalent fairness criteria are the max-min fairness and proportional fairness, which are rooted in axiomatic bargaining theory, namely the Kalai–Smorodinsky [244] and Nash bargaining solution [243], respectively. On the other side of the spectrum, a controller might opt to ignore fairness and maximize the aggregate utility of users, i.e., to follow the *utilitarian principle*, also referred to as the Bentham-Edgeworth solution [242]. The *Price of Fairness* (PoF) [268] is now an established metric for assessing how much the social welfare (i.e., the aggregate utility) is affected when enforcing some fairness metric. Ideally, we would like this price to be as small as possible, bridging in a way these two criteria. Atkinson [269] proposed the unifying α-fairness criterion which yields different fairness criteria based on the value of α, i.e., the utilitarian principle ($\alpha = 0$), proportional fairness ($\alpha = 1$), and max-min fairness ($\alpha \to \infty$). Due to the generality of the α-fairness criterion, we use it to develop our theory, which in turn renders our results transferrable to all above fairness and bargaining problems. In this work, the PoF, together with the metric of fairness-regret, are the two criteria we use to characterize our fairness solution.

5.2.2 Fairness in Dynamic Resource Allocation

Several works consider slot-fairness in dynamic systems [250, 251, 270]. Jalota and Ye [250] proposed a weighted proportional fairness algorithm for a system where new users arrive in each slot, having linear i.i.d. perturbed unknown utility functions at the time of selecting an allocation, and are allocated resources from an i.i.d. varying budget. Sinclair et al. [251] consider a similar setup, but assume the utilities are known at the time of selecting an allocation, and the utility parameters (number of agents and their type) are drawn from some fixed known distribution. They propose an adaptive threshold policy, which achieves a target efficiency (amount of consumed resources' budget) and fairness tradeoff, where the latter is defined w.r.t. to an offline weighted proportional fairness benchmark. Finally, Talebi and Proutiere [270] study dynamically arriving tasks that are assigned to a set of servers with unknown and stochastically-varying service rates. Using a stochastic multi-armed bandit model, the authors achieve proportional fairness across the service rates assigned to different tasks at each slot. All these important works, however, do not consider the more practical horizon-fairness metric where fairness is enforced throughout the entire operation of the system and not over each slot separately.

Horizon-fairness has been recently studied through the lens of competitive analysis [271–273], where the goal is to design a policy that achieves online fairness within a constant factor from the fairness of a suitable benchmark. Kawase and Sumita [271] consider the problem of allocating arriving items irrevocably to one agent who has additive utilities over the items. The arrival of the items is arbitrary and can even be selected by an adversary. The authors consider known utility at the time of allocation, and design policies under the max-min fairness criterion. Banerjee et al. [272] consider a similar problem under the proportional fairness criterion, and they allow the policies to exploit available predictions. We observe that the competitive ratio guarantees, while theoretically interesting, may not be informative about the fairness of the actual approximate solution achieved by the algorithm for ratios different from one. For instance, when maximizing a Nash welfare function under the proportional fairness criterion, the solution achieves some axiomatic fairness properties [243] (e.g., Pareto efficiency, individual rationality, etc.), but this welfare function is meaningless for "non-optimal" allocations [251], i.e., a policy with a high competitive ratio is not necessary less fair than a policy with a lower competitive ratio. For this reason, our work considers regret as a performance metric: when regret vanishes asymptotically, the allocations of the policy indeed achieve the exact same objective as the adopted benchmark.

A different line of work [248, 249, 274–278] considers horizon-fairness through regret analysis. Gupta and Kamble [248] study individual fairness criteria that advocate similar individuals should be treated similarly. They extend the notion of individual fairness to online contextual decision-making, and introduce: (1) fairness-across-time and (2) fairness-in-hindsight. Fairness-across-time criterion requires the treatment of individuals to be individually fair relative to the past as well as future, while fairness-in-hindsight only requires individual fairness at the time of the decision. The utilities are known at the time of selecting an allocation and are i.i.d. and drawn from an unknown fixed distribution. In this work, we make a similar distinction on the fairness criterion in the online setting, where we define the horizon-fairness and slot-fairness. Liao et al. [249] consider a similar setup to ours, with a limited adversarial model and time-varying but known utilities, and focus on proportional fairness. They consider adversarial perturbation added on a fixed item distribution

Table 5.1: Summary of related work under online fairness in resource allocation.

Paper	Criterion	HF/SF	Unknown utilities	Adversarial utilities	Metric
[250]	Weighted proportional fairness	SF	✓	×	Regret
[251]	Weighted proportional fairness	SF	×	×	Envy, Efficiency
[270]	Proportional fairness	SF	×	×	Regret
[248]	Individual fairness	HF/SF	×	×	Regret
[249]	Proportional fairness	HF	×	✓	Regret
[274]	α-fairness	HF	✓	×	Regret
[275]	Envy-freeness	HF	×	✓	Envy
[276]	Weighted proportional fairness	HF	×	✓	Envy, Pareto Efficiency
[278]	Proportional fairness	HF	×	×	Regret
[271]	Max-Min fairness	HF	×	✓	Competitive ratio
[272]	Proportional fairness	HF	×	✓	Competitive ratio
[273]	Proportional fairness	HF	×	×	Competitive ratio
This work	Weighted α-fairness	HF/SF	✓	✓	Fairness Regret

where the demand of items generally behaves predictably, but for some time steps, the demand behaves erratically. Our approach departs significantly from these interesting works in that we consider unknown utility functions, a broader adversarial model (in fact, as broad as possible while still achieving vanishing fairness regret), and by using the general α-fairness criterion that encompasses all the above criteria as special cases. This makes, we believe, our OHF algorithm applicable to a wider range of practical problems. Table 5.1 summarizes the differences between our contribution and the related works.

5.2.3 Online Learning

Achieving horizon-fairness in our setup requires technical extensions to the theory of OCO [20]. The basic template of OCO-learning (in terms of resource allocation) considers that a decision maker selects repeatedly a vector $\boldsymbol{x}_t$ from a convex set $\mathcal{X}$, without having access to the t-th slot scalar utility function $u_t(\boldsymbol{x})$, with the goal to maximize the aggregate utility $\sum_{t=1}^{T} u_t(\boldsymbol{x}_t)$. The decision maker aims to have vanishing time-averaged regret, i.e., the time-averaged distance of the aggregate utility $\sum_{t=1}^{T} u_t(\boldsymbol{x}_t)$ from the aggregate utility of the optimal-in-hindsight allocation $\max_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{T} u_t(\boldsymbol{x})$ for some time horizon T. OCO models are robust, expressive, and can be tackled with several well-studied learning algorithms [20,48,279]. However, none of those is suitable for the fairness problem at hand, as we need to optimize a global function $F_\alpha(\,\cdot\,)$ of the time-averaged vector-valued utilities. This subtle change creates additional technical complications. Indeed, optimizing functions of time-averaged utility/cost functions in learning is an open and challenging problem. In particular, Even-Dar et al. [280] introduced the concept of global functions in online learning, and devised a policy with vanishing regret using the theory of approachability [281]. However, their approach can handle only norms as global functions, and this limitation is not easy to overcome: the authors themselves stress characterizing when a global function enables a vanishing regret is an open problem (see [280, Section 7]). Rakhlin et al. [282] extend this work to non-additive global functions. However, the α-

fairness function considered in our work is not supported by their framework. To generalize the results to α-fairness global functions, we employ a convex conjugate approach conceptually similar to the approach taken in the work of Agrawal and Evanur [283] to obtain a regret guarantee with a concave global function under a stationary setting and linear utilities. In this work, we consider an adversarial setting (i.e., utilities are picked by an adversary after we select an allocation) that encompasses general concave utilities, and this requires learning over the primal space as well as the dual (conjugate) space.

5.3 Online Fairness: Definitions and Background

5.3.1 Static Fairness

Consider a system $\mathcal{S}$ that serves a set of agents $\mathcal{I}$ by selecting allocations from the set of eligible allocations $\mathcal{X}$.[1] In the general case, this set is defined as the Cartesian product of agent-specific eligible allocations' set $\mathcal{X}_i$, i.e., $\mathcal{X} \triangleq \bigtimes_{i\in\mathcal{I}} \mathcal{X}_i$. We assume that each set $\mathcal{X}_i$ is convex. The utility of each agent $i \in \mathcal{I}$ is a concave function $u_i : \mathcal{X} \to \mathbb{R}_{\geq 0}$, and depends, possibly, not only on $\boldsymbol{x}_i \in \mathcal{X}_i$, but on the entire vector $\boldsymbol{x} \in \mathcal{X}$.[2] The vector $\boldsymbol{u}(\boldsymbol{x}) \triangleq (u_i(\boldsymbol{x}))_{i\in\mathcal{I}} \in \mathcal{U}$ is the vectorized form of the agents' utilities, where $\mathcal{U}$ is the set of possible utility functions. The joint allocation $\boldsymbol{x}_\star \in \mathcal{X}$ is an α-fair allocation for some $\alpha \in \mathbb{R}_{\geq 0}$ if it solves the following convex problem:

$$\max_{\boldsymbol{x}\in\mathcal{X}} F_\alpha\left(\boldsymbol{u}(\boldsymbol{x})\right), \tag{5.2}$$

where F_α is the α-fairness criterion the system employs (e.g., when $\alpha = 1$, problem (5.2) corresponds to an Eisenberg-Gale convex problem [284]). The α-fairness function is defined as follows [269]:

Definition 5.3.1. An α-fairness function $F_\alpha : \mathcal{U} \to \mathbb{R}$ is parameterized by the inequality aversion parameter $\alpha \in \mathbb{R}_{\geq 0}$, and it is given by

$$F_\alpha(\boldsymbol{u}) \triangleq \sum_{i\in\mathcal{I}} f_\alpha(u_i), \qquad \text{where} \qquad f_\alpha(u) \triangleq \begin{cases} \frac{u^{1-\alpha}-1}{1-\alpha}, & \text{for } \alpha \in \mathbb{R}_{\geq 0} \setminus \{1\}, \\ \log(u), & \text{for } \alpha = 1, \end{cases} \tag{5.3}$$

for every $\boldsymbol{u} \in \mathcal{U}$. Note that $\mathcal{U} \subset \mathbb{R}^{\mathcal{I}}_{\geq 0}$ for $\alpha < 1$, and $\mathcal{U} \subset \mathbb{R}^{\mathcal{I}}_{>0}$ for $\alpha \geq 1$.

Note that we use the most general version of utility-based fairness where the fairness rule is defined w.r.t. to accrued utilities (as opposed to allocated resource, only), i.e., in our system $\mathcal{S}$, the utility vector $\boldsymbol{u} \in \mathcal{U}$ can be a function of the selected allocations in $\mathcal{X}$. The α-fairness function is concave and component-wise increasing, and thus exhibits diminishing returns [285]. An increase in utility to a player with a low utility results in a higher α-fairness objective. Thus, such an increase is desirable to the system controller. Moreover, the rate at which the marginal increase diminishes is controlled by α, which is then called the *inequality aversion parameter*. An allocation which maximizes the α-fairness objective is always Pareto efficient [285].

[1] Appendix 37 discusses the setting in which the set of agents $\mathcal{I}$ is unknown and agents can depart and arrive to the system.

[2] For example, in TCP congestion control, the performance of each end-node depends not only on the rate that is directly allocated to that node, but also, through the induced congestion in shared links, by the rate allocated to other nodes [233]. Similar couplings arise in wireless transmissions over shared channels [236].

5.3.2 Online Fairness

We consider the performance of the system $\mathcal{S}$ is tracked over a time horizon spanning $T \in \mathbb{N}$ timeslots. At the beginning of each timeslot $t \in \mathcal{T} \triangleq \{1, 2, \dots, T\}$, a policy selects an allocation $\boldsymbol{x}_t \in \mathcal{X}$ *before* $\boldsymbol{u}_t : \mathcal{X} \to \mathbb{R}^{\mathcal{I}}$ is revealed to the policy. The goal is to approach the performance of a properly-selected fair allocation benchmark. We consider the following two cases:

Slot-Fairness. An offline benchmark in hindsight, with access to the utilities revealed at every timeslot $t \in \mathcal{T}$, can ensure fairness at every timeslot satisfying a *slot-fairness* (SF) objective [250, 251, 270]. Formally, the benchmark selects the joint allocation $\boldsymbol{x}_\star \in \mathcal{X}$ satisfying

$$\text{SF}: \qquad \boldsymbol{x}_\star \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} \frac{1}{T} \sum_{t\in\mathcal{T}} F_\alpha\left(\boldsymbol{u}_t(\boldsymbol{x})\right). \tag{5.4}$$

Horizon-Fairness. Enforcing fairness at every timeslot can be quite restrictive, and this is especially evident for large time horizons. An alternative formulation is to consider that the agents can accept a momentary violation of fairness at a given timeslot $t \in \mathcal{T}$ as long as in the long run fairness over the total incurred utilities is achieved. Therefore, it is more natural (see Example 5.3.1) to ensure a horizon-fairness criterion over the entire period $\mathcal{T}$. Formally, the benchmark selects the allocation $\boldsymbol{x}_\star \in \mathcal{X}$ satisfying

$$\text{HF}: \qquad \boldsymbol{x}_\star \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} F_\alpha\left(\frac{1}{T} \sum_{t\in\mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x})\right). \tag{5.5}$$

Price of fairness. Bertsimas et al. [285] defined the *price of fairness* (PoF) metric to quantify the efficiency loss due to fairness as the difference between the maximum system efficiency and the efficiency under the fair scheme. In the case of α-fairness, it is defined for some utility set $\mathcal{U}$ as

$$\text{PoF}(\mathcal{U}; \alpha) \triangleq \frac{\max_{\boldsymbol{u}\in\mathcal{U}} F_0(\boldsymbol{u}) - F_0\left(\boldsymbol{u}_{\max,\alpha}\right)}{\max_{\boldsymbol{u}\in\mathcal{U}} F_0(\boldsymbol{u})}, \tag{5.6}$$

where $\boldsymbol{u}_{\max,\alpha} \in \arg\max_{\boldsymbol{u}\in\mathcal{U}} F_\alpha(\boldsymbol{u})$ and $F_0(\boldsymbol{u}) = \sum_{i\in\mathcal{I}} u_i$ measures the achieved social welfare. Note that by definition the utilitarian objective achieves maximum efficiency, i.e., $\text{PoF}(\mathcal{U}; 0) = 0$. Naturally, in our online setting, the metric is extended as follows

$$\text{PoF}(\mathcal{X}; \mathcal{T}; \alpha) \triangleq \frac{\max_{\boldsymbol{x}\in\mathcal{X}} \sum_{t\in\mathcal{T}} F_0\left(\boldsymbol{u}_t(\boldsymbol{x})\right) - \sum_{t\in\mathcal{T}} F_0\left(\boldsymbol{u}_t(\boldsymbol{x}_\star)\right)}{\max_{\boldsymbol{x}\in\mathcal{X}} \sum_{t\in\mathcal{T}} F_0\left(\boldsymbol{u}_t(\boldsymbol{x})\right)}, \tag{5.7}$$

where $\boldsymbol{x}_\star$ is obtained through either SF (5.4) or HF (5.5). We provide the following example to further motivate our choice of horizon-fairness as a performance objective. A similar argument is provided in [286, Example 7].

Example 5.3.1 – Consider a system with two agents $\mathcal{I} = \{1, 2\}$, an allocation set $\mathcal{X} = [0, x_{\max}]$ with $x_{\max} > 1$, α-fairness criterion with $\alpha = 1$, even $T \in \mathbb{N}$, and the following sequence of utilities

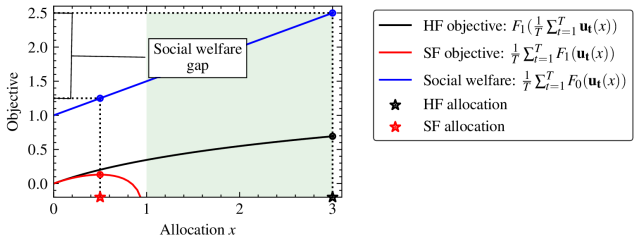


Figure 5.1: Price of Fairness under HF and SF objectives for Example 5.3.1 for $x_{\max} = 3$. The green shaded area provides the set of allocation unachievable by the SF objective but achievable by the HF objective.

$\{\boldsymbol{u}_t(x)\}_{t=1}^{T} = \{(1+x, 1-x), (1+x, 1+x), \dots\}$. It can easily be verified that PoF $= 0$ for HF objective (5.5) because the HF optimal allocation is $x_{\max}$ which matches the optimal allocation under the utilitarian objective. However, under the SF objective (5.4) we have PoF $= \frac{x_{\max}-0.5}{x_{\max}+2} \approx 1$ when $x_{\max}$ is large. Remark that the two objectives have different domains of definitions; in particular, the allocations in the set $[1, x_{\max}] \subset \mathcal{X}$ are unachievable by the SF objective because they would lead to $u_{t,2}(x) \leq 0$. The HF objective achieves lower PoF (hence, larger aggregate utility), and it allows a much larger set of eligible allocations (in particular all the allocations in the set $\mathcal{X}$), as shown in Fig. 5.1. Indeed, when the controller has the freedom to achieve fairness over a time horizon, there is an opportunity for more efficient allocations during the system operation. This example provides intuition on the robustness and practical importance of the horizon-fairness objective.

In the following section, we provide the description of an online learning model and our performance metric of interest under the HF objective.

5.3.3 Online Policies and Performance Metric

The agents' allocations are determined by an online policy $\boldsymbol{\mathcal{A}} = \{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_T\}$, i.e., a sequence of mappings. For every timeslot $t \in \mathcal{T}$, $\mathcal{A}_t : \mathcal{X}^t \times \mathcal{U}^t \to \mathcal{X}$ maps the sequence of past allocations $\{\boldsymbol{x}_s\}_{s=1}^{t} \in \mathcal{X}^t$ and utility functions $\{\boldsymbol{u}_s\}_{s=1}^{t} \in \mathcal{U}^t$ to the next allocation $\boldsymbol{x}_{t+1} \in \mathcal{X}$. We assume the initial decision $\boldsymbol{x}_1$ is feasible (i.e., $\boldsymbol{x}_1 \in \mathcal{X}$). We measure the performance of policy $\boldsymbol{\mathcal{A}}$ in terms of the *fairness regret* (5.8), i.e., the difference between the fairness objective experienced by $\boldsymbol{\mathcal{A}}$ at the time horizon T and that of the best static decision $\boldsymbol{x}_\star \in \mathcal{X}$ in hindsight. We restate the regret metric here to streamline the presentation:

$$\mathfrak{R}_T\left(F_\alpha, \boldsymbol{\mathcal{A}}\right) \triangleq \sup_{\{\boldsymbol{u}_t\}_{t=1}^{T} \in \mathcal{U}^T} \left\{ F_\alpha\left(\frac{1}{T}\sum_{t\in\mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x}_\star)\right) - F_\alpha\left(\frac{1}{T}\sum_{t\in\mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x}_t)\right)\right\}. \tag{5.8}$$

where $\boldsymbol{x}_\star$ is the HF (5.5) allocation. If the fairness regret becomes negligible for large T, then $\boldsymbol{\mathcal{A}}$ attains the same fairness objective as the optimal static decision with hindsight. Note that under the utilitarian objective ($\alpha = 0$), this fairness regret coincides with the classic time-averaged regret in OCO [20]. However, for general values of $\alpha \neq 0$, the metric is completely different, as we aim to compare α-fair functions evaluated at time-averaged vector-valued utilities.

5.4 Online Horizon-Fair (OHF) Policy

We first present in Section 5.4.1, the adversarial model considered in this work and provide a result on the impossibility of guaranteeing vanishing fairness regret (5.8) under general adversarial perturbations. We also provide a powerful family of adversarial perturbations for which a vanishing fairness regret guarantee is attainable. Secondly, we present the OHF policy in Section 5.4.2 and provide its performance guarantee. Finally, we provide in Section 5.4.3 a set of adversarial examples captured by our fairness framework.

5.4.1 Adversarial Model and Impossibility Result

We begin by introducing formally the adversarial model that characterizes the utility perturbations. In particular, we consider $\boldsymbol{\delta}_t(\boldsymbol{x}) \triangleq \left(\frac{1}{T}\sum_{s\in\mathcal{T}} \boldsymbol{u}_s(\boldsymbol{x})\right) - \boldsymbol{u}_t(\boldsymbol{x})$ to quantify how much the adversary *perturbs* the average utility by selecting a utility function $\boldsymbol{u}_t$ at timeslot $t \in \mathcal{T}$. Recall that $\boldsymbol{x}_\star \in \mathcal{X}$ denotes the optimal allocation under HF objective (5.5). We denote by $\Xi(\mathcal{T})$ the set of all possible decompositions of $\mathcal{T}$ into sets of contiguous timeslots, i.e., for every $\{\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_K\} \in \Xi(\mathcal{T})$ it holds $\mathcal{T} = \dot{\bigcup}_{k\in\{1,2,\dots,K\}} \mathcal{T}_k$ and $\max \mathcal{T}_k < \min \mathcal{T}_{k+1}$ for $k \in \{1, 2, \dots, K-1\}$. We define two types of adversarial perturbations:

$$\text{Budgeted-severity:} \quad \mathbb{V}_{\mathcal{T}} \triangleq \sup_{\{\boldsymbol{u}_t\}_{t=1}^T \in \mathcal{U}^T} \left\{ \sum_{t\in\mathcal{T}} \sum_{i\in\mathcal{I}} \left|\delta_{t,i}(\boldsymbol{x}_\star)\right| \right\}, \tag{5.9}$$

$$\text{Partitioned-severity:} \quad \mathbb{W}_{\mathcal{T}} \triangleq \sup_{\{\boldsymbol{u}_t\}_{t=1}^T \in \mathcal{U}^T} \left\{ \inf_{\substack{\{\mathcal{T}_1,\mathcal{T}_2,\dots,\mathcal{T}_K\} \\ \in \Xi(\mathcal{T})}} \left\{ \sum_{k=1}^K \sum_{i\in\mathcal{I}} \left| \sum_{t\in\mathcal{T}_k} \delta_{t,i}(\boldsymbol{x}_\star) \right| + \sum_{k=1}^K \frac{|\mathcal{T}_k|^2}{\sum_{k'<k} |\mathcal{T}_k| + 1} \right\} \right\}. \tag{5.10}$$

Our result in Theorem 5.4.2 implies that when either $\mathbb{V}_{\mathcal{T}}$ or $\mathbb{W}_{\mathcal{T}}$ grows sublinearly in the time horizon (i.e., the perturbations satisfy at least one of these two conditions), the regret of OHF policy in Algorithm 5.1 vanishes over time. We provide a detailed description of conditions (5.9) and (5.10) below.

The *budgeted-severity* $\mathbb{V}_{\mathcal{T}}$ in Eq. (5.9) bounds the total amount of perturbations of the time-averaged utility. When $\mathbb{V}_{\mathcal{T}} = 0$ the adversary is only able to select a fixed function, otherwise the adversary is able to select time-varying utilities, while keeping the total deviation no more than $\mathbb{V}_{\mathcal{T}}$. Moreover, the adversary is allowed to pick opportunely the timeslots to maximize performance degradation for the controller. This model is similar to the *adversarial corruption* setting considered in [249, 260], and it captures realistic scenarios where the utilities incurred at different timeslots are predictable, but can be perturbed for some fraction of the timeslots. For instance, Internet traffic may experience spikes due to breaking news or other unpredictable events [287].

The *partitioned-severity* $\mathbb{W}_{\mathcal{T}}$ in Eq. (5.10) may at first be less easy to understand than budgeted-severity condition (5.9), but is equally important from a practical point of view. For simplicity, consider a uniform decomposition of the timeslots, i.e., $\mathcal{T}_k = M$ for every $k \in \{1, 2, \dots, T/M\}$ assuming

w.l.g. M divides T. Then the r.h.s. term in Eq. (5.10) can be bounded as follows:

$$\sum_{k=1}^{T/M} \frac{|\mathcal{T}_k|^2}{\sum_{k'<k} |\mathcal{T}_k| + 1} = \sum_{k=1}^{T/M} \frac{M^2}{M(k-1)+1} = O\left(M^2 + M\log(T/M)\right). \tag{5.11}$$

Hence, when $M = o(\sqrt{T})$ it holds $\sum_{k=1}^{T/M} \frac{|\mathcal{T}_k|^2}{\sum_{k'<k}|\mathcal{T}_k|+1} = o(T)$. Since this term grows sublinearly in time, it remains to characterize the growth of the l.h.s. term $\sum_{k=1}^{K} \sum_{i\in\mathcal{I}} \left|\sum_{t\in\mathcal{T}_k} \delta_{t,i}(\boldsymbol{x}_\star)\right|$ in Eq. (5.10). This term is related to the perturbations selected by the adversary, however the absolute value is only evaluated at the end of each contiguous subperiod $\mathcal{T}_k$, i.e., the positive and negative deviations from the average utilities can cancel out. For example, a periodic selection of utilities from some set with cardinality M would have zero deviation for this term. This type of adversary is similar to the periodic adversary considered in [260, 262], but also includes adversarial selection of utilities from some finite set (see Example 5.4.2 in Section 5.4.3). The partitioned-severity adversary can model real-life applications that exhibit seasonal properties, e.g., the traffic may be completely different throughout the day, but daily traffic is self-similar [261]. This condition also unlocks the possibility to obtain high probability guarantees under stochastic utilities (see Corollary 5.4.4).

We formally make the following assumptions:

(A1) The allocation set $\mathcal{X}$ is convex with diameter $\operatorname{diam}(\mathcal{X}) < \infty$.

(A2) The utilities are bounded, i.e., $\boldsymbol{u}_t(\boldsymbol{x}) \in [u_{\min}, u_{\max}]^{\mathcal{I}} \subset \mathbb{R}^{\mathcal{I}}$ for every $t \in \mathcal{T}$.

(A3) The supergradients of the utilities are bounded over $\mathcal{X}$, i.e., it holds $\|\boldsymbol{g}\|_2 \leq L_{\mathcal{X}} < \infty$ for any $\boldsymbol{g} \in \partial_{\boldsymbol{x}} u_{t,i}(\boldsymbol{x})$ and $\boldsymbol{x} \in \mathcal{X}$.

(A4) The average utility of the optimal allocation (5.5) is bounded such that $\frac{1}{T}\sum_{t\in\mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x}_\star) \in \left[u_{\star,\min}, u_{\star,\max}\right]^{\mathcal{I}} \subset \mathbb{R}^{\mathcal{I}}_{>0}$.

(A5) The adversary is restricted to select utilities such that

$$\min\{\mathbb{V}_{\mathcal{T}}, \mathbb{W}_{\mathcal{T}}\} = o(T). \tag{5.12}$$

We first show that an adversary solely satisfying the mild assumptions (A1)–(A4) can arbitrarily degrade the performance of any policy $\boldsymbol{\mathcal{A}}$. Formally, we have the following negative result:

Theorem 5.4.1. *When Assumptions (A1)–(A4) are satisfied, there is no online policy $\boldsymbol{\mathcal{A}}$ attaining $\mathfrak{R}_T(F_\alpha, \boldsymbol{\mathcal{A}}) = \mathrm{o}(1)$ for $|\mathcal{I}| > 1$ and $\alpha > 0$. Moreover, there exists an adversary where Assumption (A5) is necessary for $\mathfrak{R}_T(F_\alpha, \boldsymbol{\mathcal{A}}) = o(1)$.*

The proof can be found in Appendix 32. We design an adversary with a choice over two sequences of utilities against two agents. We show that no policy can have vanishing fairness regret w.r.t. the time horizon under both sequences.

Algorithm 5.1 OHF policy

Require: $\mathcal{X}$, $\alpha \in \mathbb{R}_{\geq 0}$, $\left[u_{\star,\min}, u_{\star,\max}\right]$

1: $\Theta \leftarrow \left[-1/u_{\star,\min}^{\alpha}, -1/u_{\star,\max}^{\alpha}\right]^{\mathcal{I}}$ ▷ Initialize the dual (conjugate) subspace

2: $\boldsymbol{x}_1 \in \mathcal{X}$; $\boldsymbol{\theta}_1 \in \Theta$; ▷ Initialize allocation $\boldsymbol{x}_1$ and dual decision $\boldsymbol{\theta}_1$

3: **for** $t \in \mathcal{T}$ **do**

4: Reveal $\Psi_{t,\alpha}(\boldsymbol{\theta}_t, \boldsymbol{x}_t) = (-F_\alpha)^\star(\boldsymbol{\theta}_t) - \boldsymbol{\theta}_t \cdot \boldsymbol{u}_t(\boldsymbol{x}_t)$ ▷ Incur reward $\Psi_{t,\alpha}(\boldsymbol{\theta}_t, \boldsymbol{x}_t)$ and loss $\Psi_{t,\alpha}(\boldsymbol{\theta}_t, \boldsymbol{x}_t)$

5: $\boldsymbol{g}_{\mathcal{X},t} \in \partial_{\boldsymbol{x}} \Psi_{t,\alpha}(\boldsymbol{\theta}_t, \boldsymbol{x}_t) = \sum_{i \in \mathcal{I}} \theta_{t,i} \partial_{\boldsymbol{x}} u_{t,i}$ ▷ Compute supergradient $\boldsymbol{g}_{\mathcal{X},t}$ at $\boldsymbol{x}_t$ of reward $\Psi_{t,\alpha}(\boldsymbol{\theta}_t, \cdot)$

6: $\boldsymbol{g}_{\Theta,t} = \nabla_{\boldsymbol{\theta}} \Psi_{t,\alpha}(\boldsymbol{\theta}_t, \boldsymbol{x}_t) = \left(\left(-\theta_{t,i}\right)^{-1/\alpha} - \boldsymbol{u}_t(\boldsymbol{x}_t)\right)_{i \in \mathcal{I}}$ ▷ Compute gradient $\boldsymbol{g}_{\Theta,t}$ at $\boldsymbol{\theta}_t$ of loss $\Psi_{t,\alpha}(\cdot, \boldsymbol{x}_t)$

7: $\eta_{\mathcal{X},t} = \operatorname{diam}(\mathcal{X}) / \sqrt{\sum_{s=1}^{t} \left\|\boldsymbol{g}_{\mathcal{X},s}\right\|_2^2}$; $\eta_{\Theta,t} = \alpha u_{\min}^{-1-1/\alpha} / t$ ▷ Compute adaptive learning rates

8: $\boldsymbol{x}_{t+1} = \Pi_{\mathcal{X}}\left(\boldsymbol{x}_t + \eta_{\mathcal{X},t} \boldsymbol{g}_{\mathcal{X},t}\right)$; $\boldsymbol{\theta}_{t+1} = \Pi_{\Theta}\left(\boldsymbol{\theta}_t - \eta_{\Theta,t} \boldsymbol{g}_{\Theta,t}\right)$ ▷ Compute a new allocation and dual decision

9: **end for**

5.4.2 OHF Policy

Our policy employs a convex-concave function, composed of a convex conjugate term that tracks the global fairness metric in a dual (conjugate) space, and a weighted sum of utilities term that tracks the appropriate allocations in the primal space. This function is used by the policy to compute a gradient and a supergradient to adapt its internal state. In detail, we define the function $\Psi_\alpha : \Theta \times \mathcal{X} \to \mathbb{R}$ given by

$$\Psi_{t,\alpha}(\boldsymbol{\theta}, \boldsymbol{x}) \triangleq (-F_\alpha)^\star(\boldsymbol{\theta}) - \boldsymbol{\theta} \cdot \boldsymbol{u}_t(\boldsymbol{x}), \tag{5.13}$$

where $\Theta = \left[-1/u_{\star,\min}^{\alpha}, -1/u_{\star,\max}^{\alpha}\right]^{\mathcal{I}} \subset \mathbb{R}_{<0}^{\mathcal{I}}$ is a subspace of the dual (conjugate) space, and $(-F_\alpha)^\star$ is the *convex conjugate* (see Definition 31.1 in Appendix) of $-F_\alpha$ given by for any $\boldsymbol{\theta} \in \Theta$

$$(-F_\alpha)^\star(\boldsymbol{\theta}) = \begin{cases} \sum_{i \in \mathcal{I}} \frac{\alpha(-\theta_i)^{1-1/\alpha} - 1}{1-\alpha} & \text{for } \alpha \in \mathbb{R}_{\geq 0} \setminus \{1\}, \\ \sum_{i \in \mathcal{I}} -\log(-\theta_i) - 1 & \text{for } \alpha = 1. \end{cases} \tag{5.14}$$

The policy is summarized in Algorithm 5.1. The algorithm only requires as input: the set of eligible allocations $\mathcal{X}$, the α-fairness parameter in $\mathbb{R}_{\geq 0}^{\mathcal{I}}$, and the range $\left[u_{\star,\min}, u_{\star,\max}\right]$ of values of the average utility obtained by the optimal allocation (5.5), i.e., $\frac{1}{T} \sum_{t \in \mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x}_\star) \in \left[u_{\star,\min}, u_{\star,\max}\right]^{\mathcal{I}} \subset \mathbb{R}_{>0}^{\mathcal{I}}$. We stress that the target time horizon T is *not* an input to the policy. The utility bounds $u_{\star,\min}^{\alpha}$ and $u_{\star,\max}^{\alpha}$ depend on the specific application. For example, for the virtualized caching system considered in Section 5.6, one could simply pick a small enough $\epsilon > 0$ as $u_{\star,\min}^{\alpha}$, and the maximum batch size weighted by the largest retrieval cost in the network as $u_{\star,\max}^{\alpha}$ (see Eq. (5.27)). However, if prior information is available to tighten this range, the performance of the algorithm is ameliorated, as reflected in the regret bound in Eq. (5.15).

The policy uses its input to initialize the dual (conjugate) subspace $\Theta = \left[-1/u_{\star,\min}^{\alpha}, -1/u_{\star,\max}^{\alpha}\right]^{\mathcal{I}}$, an allocation $\boldsymbol{x}_1 \in \mathcal{X}$, and a dual decision $\boldsymbol{\theta}_1 \in \Theta$ (lines 1–2 in Algorithm 5.1). At a given timeslot $t \in \mathcal{T}$, the allocation $\boldsymbol{x}_t$ is selected; then a vector-valued utility $\boldsymbol{u}_t(\cdot)$ is revealed and in turn $\Psi_{t,\alpha}(\cdot, \cdot)$ is revealed to the policy (line 4 in Algorithm 5.1). The supergradient $\boldsymbol{g}_{\mathcal{X},t}$ of $\Psi_{t,\alpha}(\boldsymbol{\theta}_t, \cdot)$ at point

$\boldsymbol{x}_t \in \mathcal{X}$, and the gradient $\boldsymbol{g}_{\Theta,t}$ of $\Psi_{t,\alpha}(\,\cdot\,, \boldsymbol{x}_t)$ at point $\boldsymbol{\theta}_t \in \Theta$ are computed (lines 5–6 in Algorithm 5.1). The policy then finally performs an adaptation of its state variables $(\boldsymbol{x}_t, \boldsymbol{\theta}_t)$ through a descent step in the dual space and an ascent step in the primal space through online gradient descent (OGD) and online gradient ascent (OGA) policies,[3] respectively (line 8 in Algorithm 5.1). The learning rates (step size) used are "self-confident" [289] as they depend on the experienced gradients. Such a learning rate schedule is compelling because it can adapt to the adversary and provides tighter regret guarantees for "easy" utility sequences; moreover, it allows attaining an *anytime* regret guarantee, i.e., a guarantee holding for any time horizon T. In particular, OHF policy in Algorithm 5.1 enjoys the following fairness regret guarantee.

Theorem 5.4.2. *Under assumptions (A1)–(A5), OHF policy in Algorithm 5.1 attains the following fairness regret guarantee:*

$$\mathfrak{R}_T\left(F_\alpha, \boldsymbol{\mathcal{A}}\right) \leq \sup_{\{\boldsymbol{u}_t\}_{t=1}^T \in \mathcal{U}^T} \left\{ \frac{1.5\,\mathrm{diam}\left(\mathcal{X}\right)}{T} \sqrt{\sum_{t\in\mathcal{T}} \left\|\boldsymbol{g}_{\mathcal{X},t}\right\|_2^2} + \sum_{t=1}^{T} \frac{\alpha \left\|\boldsymbol{g}_{\Theta,t}\right\|_2^2}{2u_{\star,\min}^{1+\frac{1}{\alpha}} T t} \right\} + O\left(\frac{\min\left\{\mathbb{V}_{\mathcal{T}}, \mathbb{W}_{\mathcal{T}}\right\}}{T}\right) \tag{5.15}$$

$$\leq \frac{1.5\,\mathrm{diam}\left(\mathcal{X}\right) L_{\mathcal{X}}}{u_{\star,\min}^{\alpha}\sqrt{T}} + \frac{\alpha L_\Theta^2 (\log(T)+1)}{u_{\star,\min}^{1+\frac{1}{\alpha}} T} + O\left(\frac{\min\left\{\mathbb{V}_{\mathcal{T}}, \mathbb{W}_{\mathcal{T}}\right\}}{T}\right) \tag{5.16}$$

$$= O\left(\frac{1}{\sqrt{T}} + \frac{\min\left\{\mathbb{V}_{\mathcal{T}}, \mathbb{W}_{\mathcal{T}}\right\}}{T}\right) = o(1). \tag{5.17}$$

The proof is provided in Appendix 33. We prove that the fairness regret can be upper bounded with the time-averaged regrets of the primal policy operating over the set $\mathcal{X}$ and the dual policy operating over the set Θ, combined with an extra term that is upper bounded with $\min\left\{\mathbb{V}_{\mathcal{T}}, \mathbb{W}_{\mathcal{T}}\right\}$. Note that the fairness regret upper bound in Eq. (5.15) can be much tighter than the one in Eq. (5.16), because the gradients' norms can be smaller than their upper bound at a given timeslot $t \in \mathcal{T}$. Thanks to its "self-confident" learning schedule [289], which dynamically adapts to the observed utilities, our Algorithm 5.1 enjoys an *any-time* regret guarantee, i.e., it does not require the knowledge of the target time horizon T.

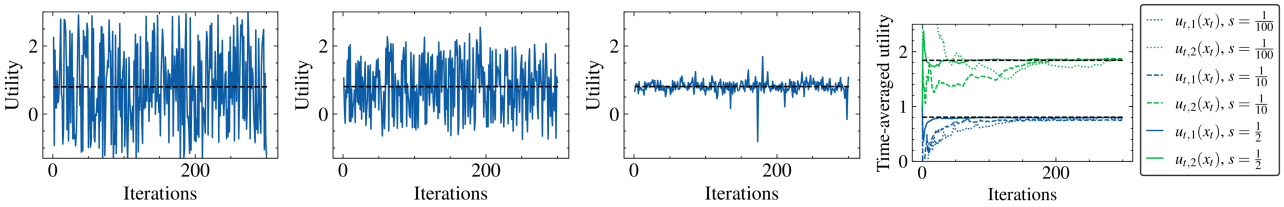
The result in Theorem 5.4.2 is tight, in the sense that no policy can have a fairness regret (5.8) with better dependency on the time horizon T. Formally,

Theorem 5.4.3. *Any policy $\boldsymbol{\mathcal{A}}$ incurs $\mathfrak{R}_T\left(F_\alpha, \boldsymbol{\mathcal{A}}\right) = \Omega\left(\frac{1}{\sqrt{T}}\right)$ fairness regret* (5.8) *for $\alpha \geq 0$.*

The proof can be found in Appendix 34. We show that the lower bound on regret in online convex optimization [20] can be transferred to the fairness regret.

We discuss in Appendix 38, the time-complexity of Algorithm 5.1 in the context of virtualized caching system application, presented in Section 5.6.

[3]Note that a different OCO policy can be used as long as it has a no-regret guarantee, e.g., online mirror descent (OMD), follow the regularized leader (FTRL), or follow the perturbed leader (FTPL) [20, 279]; moreover, one could even incorporate optimistic versions of such policies [288], to improve the regret rates when the controller has access to accurate predictions.



(a) $s = \frac{1}{100}$ (b) $s = \frac{1}{10}$ (c) $s = \frac{1}{2}$ (d) Time-averaged utility

Figure 5.2: Subfigures (a)–(c) provide the utilities of agent 2 for different values of perturbations' severity parameter $s \in \left\{\frac{1}{100}, \frac{1}{10}, \frac{1}{2}\right\}$ under the benchmark's allocation $x_\star$. Subfigure (d) provides the time-averaged utility of two agents. The dark dashed lines represent the utilities obtained by HF objective (5.5).

5.4.3 Adversarial Examples

In this section, we provide examples of adversaries satisfying Assumptions (A1)–(A5), with either $\mathbb{V}_\mathcal{T} = o(T)$ or $\mathbb{W}_\mathcal{T} = o(T)$, and of stochastic adversaries.

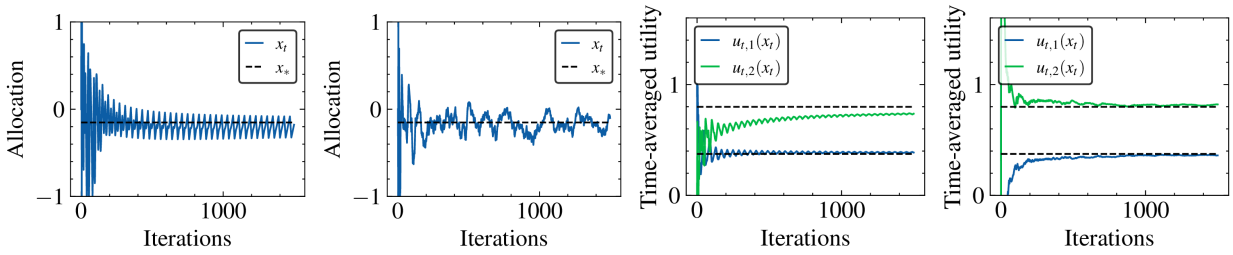
Example 5.4.1 – (Adversaries satisfying $\mathbb{V}_\mathcal{T} = o(T)$) Consider an adversary selecting utilities such that

$$\boldsymbol{u}_t(\boldsymbol{x}) = \boldsymbol{u}(\boldsymbol{x}) + \boldsymbol{\gamma}_t \odot \boldsymbol{p}_t(\boldsymbol{x}), \tag{5.18}$$

where $\boldsymbol{u} : \mathcal{X} \to \mathbb{R}^\mathcal{I}$ is a fixed utility, the time-dependent function $\boldsymbol{p}_t : \mathcal{X} \to \mathbb{R}^\mathcal{I}$ is an adversarially selected perturbation with $\left\|\boldsymbol{p}_t\right\|_\infty < \infty$, $\boldsymbol{\gamma}_t \in \mathbb{R}^\mathcal{I}$ quantifies the severity of the perturbations, and $\boldsymbol{\gamma}_t \odot \boldsymbol{p}_t(\boldsymbol{x}) = \left(\gamma_{t,i} p_{t,i}(\boldsymbol{x})\right)_{i\in\mathcal{I}}$ is the Hadamard product. The severity of the perturbations grows sublinearly in time T, i.e., $\sum_{t=1}^T \gamma_{t,i} = o(T)$ for every $i \in \mathcal{I}$. It is easy to check that, in this setting, it holds $\mathbb{V}_\mathcal{T} = o(T)$.

We provide a simple-yet-illustrative example of such an adversary. We take $\mathcal{X} = [0,1] \subset \mathbb{R}$, two agents $\mathcal{I} = \{1,2\}$, fixed utilities $\boldsymbol{u}(x) = \left(1 - x^2, 1 + x\right)$, adversarial perturbations $\boldsymbol{p}_t(x) = \left(a_{i,t} \cdot x\right)_{i\in\mathcal{I}}$ where $\boldsymbol{a}_t$ is selected uniformly at random from $[-1,1]^\mathcal{I}$ for every $t \in \mathcal{T}$. The perturbations' severity is selected as $\gamma_{\xi_{t,i},i} = t^{-s}$ where $\boldsymbol{\xi}_i : \mathcal{T} \to \mathcal{T}$ is a random permutation of the elements of $\mathcal{T}$ for $i \in \mathcal{I}$. The performance of Algorithm 5.1 is provided in Fig. 5.2. We observe that for larger values of s, corresponding to lower perturbation's severity, the policy provides faster the same utilities as the HF benchmark (5.5).

Example 5.4.2 – (Adversaries satisfying $\mathbb{W}_\mathcal{T} = o(T)$) Consider a multiset $\mathcal{M}_t$ of utilities and an adversary that selects a utility $\boldsymbol{u}_t : \mathcal{X} \to \mathbb{R}^\mathcal{I}$ from it. The multiset is updated as follows: if $\mathcal{M}_t \setminus \{\boldsymbol{u}_t\} \neq \emptyset$, $\mathcal{M}_{t+1} = \mathcal{M}_t \setminus \{\boldsymbol{u}_t\}$, otherwise, $\mathcal{M}_t = \mathcal{M}_1$. In words, the adversary selects irrevocably elements (utilities) from the set $\mathcal{M}_1$, and, when all the elements are selected, the replenished $\mathcal{M}_1$ is offered again to the adversary. Consider, without loss of generality, a time horizon T divisible by $|\mathcal{M}_1|$ and the following decomposition for the period $\mathcal{T}$: $\{1, 2, \ldots, |\mathcal{M}_1|\} \cup \{|\mathcal{M}_1| + 1, |\mathcal{M}_1| + 2, \ldots, 2\,|\mathcal{M}_1|\} \cup$



(a) Allocations (cyclic) (b) Allocations (u.a.r.) (c) Time-averaged utilities (cyclic) (d) Time-averaged utilities (u.a.r.)

Figure 5.3: Subfigures (a)–(b) provide the allocations of different agents of cyclic and u.a.r. choice of utilities over the set $\mathcal{M}_1$, respectively. Subfigures (c)–(d) provide the time-averaged utility of cyclic and u.a.r. choice of utilities over the set $\mathcal{M}_1$, respectively.

$\cdots = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_{T/|\mathcal{M}_1|}$. By construction, it holds for every $\boldsymbol{x} \in \mathcal{X}$

$$\sum_{i\in\mathcal{I}} \left| \sum_{t\in\mathcal{T}_k} \delta_{t,i}(\boldsymbol{x}) \right| = 0, \quad \forall k \in \{1, 2, \ldots, T/|\mathcal{M}_1|\}, \tag{5.19}$$

because when the multiset is fully consumed by the adversary, the average experienced utility is a fixed function. When $|\mathcal{M}_1| = \Theta(T^\epsilon)$ for $\epsilon \in [0, 1/2)$ it holds $\sum_{k=1}^{T/|\mathcal{M}_1|} \frac{|\mathcal{T}_k|^2}{\sum_{k'<k}|\mathcal{T}_k|+1} = O\left(T^{2\epsilon}\right)$ (see Eq. (5.11)); thus, combined with Eq. (5.19) it holds $\mathbb{W}_{\mathcal{T}} = o(T)$. We provide a simple example of such an adversary. Consider $\mathcal{X} = [-1, 1]$, two agents $\mathcal{I} = \{1, 2\}$, and the initial multiset

$$\mathcal{M}_1 = \{\underbrace{(1-x, 1-(1-x)^2)}_{\text{repeated 10 times}}, \underbrace{(1-(1-x)^2, 1-4x)}_{\text{repeated 20 times}}, \underbrace{(1, -2x)}_{\text{repeated 10 times}} \}. \tag{5.20}$$

We have $|\mathcal{M}_1| = 40$ and hence $\mathbb{W}_{\mathcal{T}} = o(T)$. The performance of Algorithm 5.1 is provided in Fig. 5.3 under different choice patterns over $\mathcal{M}_1$. We observe that the cyclic choice of utilities is more harmful than the u.a.r. one as it leads to slower convergence. Nonetheless, under both settings, the policy asymptotically yields the same utilities as the HF benchmark (5.5).

Example 5.4.3 – (Stochastic Adversary) Consider a scenario where $u_{t,i} : \mathcal{X} \to \mathbb{R}$ are drawn i.i.d. from an unknown distribution $\mathcal{D}_i$. Formally, the following corollary is obtained from Theorem 5.4.2.

Corollary 5.4.4. When the utilities $u_{t,i} : \mathcal{X} \to \mathbb{R}$ are drawn i.i.d. from an unknown distribution $\mathcal{D}_i$ satisfying Assumptions (A1)–(A4), the policy OHF in Algorithm 5.1 attains the following expected fairness regret guarantee:

$$\bar{\mathfrak{R}}_T(F_\alpha, \boldsymbol{\mathcal{A}}) \triangleq \sup_{\substack{\mathcal{D}_i,\, i\in\mathcal{I}}} \left\{ \mathop{\mathbb{E}}_{\substack{u_{t,i}\sim\mathcal{D}_i \\ i\in\mathcal{I},\, t\in\mathcal{T}}} \left[\max_{\boldsymbol{x}\in\mathcal{X}} F_\alpha\left(\frac{1}{T}\sum_{t\in\mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x}_t)\right) - F_\alpha\left(\frac{1}{T}\sum_{t\in\mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x}_t)\right) \right] \right\} = O\left(\frac{1}{\sqrt{T}}\right). \tag{5.21}$$

Moreover, it holds with probability one: $\mathfrak{R}_T(F_\alpha, \boldsymbol{\mathcal{A}}) \le 0$ for $T \to \infty$.

The proof is in Appendix 35. The expected fairness regret guarantee follows from Theorem 5.4.2 and observing that $\mathbb{E}[\boldsymbol{\delta}_t(\boldsymbol{x})] = \mathbf{0}$ for any $t \in \mathcal{T}$ and $\boldsymbol{x} \in \mathcal{X}$. The high probability fairness regret guarantee for large T is obtained through Hoeffding's inequality paired with Eq. (5.10).

Note that we provide additional examples of adversaries, in the context of the application of our policy to a virtualized caching system, in Section 5.6.

5.5 Extensions

In this section, we first show that our algorithmic framework extends to cooperative bargaining settings, in particular Nash bargaining [243]. Secondly, we show that our framework also extends to the weighted α-fairness criterion.

5.5.1 Nash Bargaining

Nash bargaining solution (NBS), proposed in the seminal paper [243], is a fairness criterion for dispersing to a set of agents the utility of their cooperation. The solution guarantees that, whenever the agents cooperate, each agent achieves an individual performance that exceeds its performance when operating independently. This latter is also known as the disagreement point. NBS comes from the area of cooperative game theory, and it is self enforcing, i.e., the agents will agree to apply this solution without the need for an external authority to enforce compliance. NBS has been extensively applied in communication networks, e.g., to transmission power control [263], mobile Internet sharing among wireless users [264], content delivery in ISP-CDN partnerships [265], and cooperative caching in information-centric networks [266].

Nash bargaining can be incorporated through our fairness framework when $\alpha = 1$, and utilities as redefined for every $t \in \mathcal{T}$ as follows $\boldsymbol{u}'_t(\boldsymbol{x}) = \boldsymbol{u}_t(\boldsymbol{x}) - \boldsymbol{u}_t^d$ where u_i^d is the disagreement point of agent $i \in \mathcal{I}$. In particular, OHF provides the same guarantees. We also note that the dynamic model generalizes the NBS solution by allowing both the utilities and the disagreement points to change over time, while the benchmark is defined using (5.5) and $\alpha = 1$. Hence, the proposed OHF allows the agents to collaborate without knowing in advance the benefits of their cooperation nor their disagreement points, in a way that guarantees they will achieve the commonly agreed NBS at the end of the horizon T (asymptotically).

5.5.2 The $(\boldsymbol{w}, \alpha)$-Fairness

The weighted α-fairness or simply $(\boldsymbol{w}, \alpha)$-fairness with $\alpha \geq 0$ and $\boldsymbol{w} \in \Delta_{\mathcal{I}} \subset \mathbb{R}_{\geq 0}$, where $\Delta_{\mathcal{I}}$ is the probability simplex with support $\mathcal{I}$, is defined as [234]:

Definition 5.5.1. A $(\boldsymbol{w}, \alpha)$-fairness function $F_{\boldsymbol{w},\alpha} : \mathcal{U} \to \mathbb{R}$ is parameterized by the inequality aversion parameter $\alpha \in \mathbb{R}_{\geq 0}$, weights $\boldsymbol{w} \in \Delta_{\mathcal{I}}$ and it is given by $F_{\boldsymbol{w},\alpha}(\boldsymbol{u}) \triangleq \sum_{i \in \mathcal{I}} w_i f_\alpha(u_i)$ for every $\boldsymbol{u} \in \mathcal{U}$. Note that $\mathcal{U} \subset \mathbb{R}_{\geq 0}^{\mathcal{I}}$ for $\alpha < 1$, and $\mathcal{U} \subset \mathbb{R}_{>0}^{\mathcal{I}}$ for $\alpha \geq 1$.

It is easy to check that our α-fairness framework captures the $(\boldsymbol{w}, \alpha)$-fairness by simply redefining the utilities incurred at time $t \in \mathcal{I}$ for agent $i \in \mathcal{I}$ as follows: $u'_{t,i}(\boldsymbol{x}) = w_i^{\frac{1}{1-\alpha}} u_{t,i}(\boldsymbol{x})$ for $\alpha \in \mathbb{R}_{\geq 0} \setminus \{1\}$, otherwise $u'_{t,i}(\boldsymbol{x}) = \big(u_{t,i}(\boldsymbol{x})\big)^{w_i}$. Note that for $\alpha = 1$ and uniform weights, we recover the Nash bargaining setting discussed previously; otherwise, we recover asymmetric Nash bargaining in which the different weights correspond to the bargaining powers of players [290].

5.6 Application

In order to demonstrate the applicability of the proposed fairness framework, we target a representative resource management problem in virtualized caching systems where different caches cooperate by serving jointly the received content requests. This problem has been studied extensively in its static version, where the request rates for each content file are a priori known and the goal is to decide which files to store at each cache to maximize a fairness metric of cache hits across different caches, see for instance [266, 291]. We study the more realistic version of the problem where the request patterns are unknown. This online caching model has been recently studied as a learning problem in a series of papers [139, 171, 252–255], yet none of them handles fairness metrics.

5.6.1 Multi-Agent Cache Networks

Cache network. We assume that time is slotted and the set of timeslots is denoted by $\mathcal{T} \triangleq \{1, 2, \ldots, T\}$. We consider a catalog of equally-sized files $\mathcal{F} \triangleq \{1, 2, \ldots, F\}$.[4] We model a cache network at timeslot $t \in \mathcal{T}$ as an undirected weighted graph $G_t(\mathcal{C}, \mathcal{E})$, where $\mathcal{C} \triangleq \{1, 2, \ldots, C\}$ is the set of caches, and $(c, c') \in \mathcal{E}$ denotes the link connecting cache c to c' with associated weight $w_{t,(c,c')} \in \mathbb{R}_{>0}$. Let $\mathcal{P}_{t,(c,c')} = \left\{c_1, c_2, \ldots, c_{|\mathcal{P}_{t,(c,c')}|}\right\} \in \mathcal{C}^{|\mathcal{P}_{t,(c,c')}|}$ be the shortest path at timeslot $t \in \mathcal{T}$ from cache c to cache c' with associated weight $w^{\text{sp}}_{t,(c,c')} \triangleq \sum_{k=1}^{|\mathcal{P}_{t,(c,c')}|-1} w_{t,(c_k,c_{k+1})}$.

We assume for each file $f \in \mathcal{F}$ is permanently stored at a set $\Lambda_f(\mathcal{C}) \subset \mathcal{C}$ of designated repository servers. Moreover, each cache can store fractions of the file and fractions of the same file at different caches can be additively combined.[5] We denote by $x_{t,c,f} \in [0, 1]$ the fraction of file $f \in \mathcal{F}$ stored at cache $c \in \mathcal{C}$ at timeslot $t \in \mathcal{T}$. The state of cache $c \in \mathcal{C}$ is given by $\boldsymbol{x}_{t,c}$ drawn from the set

$$\mathcal{X}_c \triangleq \left\{ \boldsymbol{x} \in [0,1]^{\mathcal{F}} : \sum_{f \in \mathcal{F}} x_f \leq k_c, x_f \geq \mathbb{1}\big(c \in \Lambda_f(\mathcal{C})\big), \forall f \in \mathcal{F} \right\}, \tag{5.22}$$

where $k_c \in \mathbb{N}$ is the capacity of cache $c \in \mathcal{C}$, and $\mathbb{1}(\chi) \in \{0, 1\}$ is the indicator function set to 1 when condition χ is true. Thus, the state of the cache network belongs to $\mathcal{X} \triangleq \bigtimes_{c \in \mathcal{C}} \mathcal{X}_c$. The system

[4] Note that we assume equally-sized files to streamline the presentation. Our model supports unequally-sized files by replacing the cardinality constraint in Eq. (5.22) with a knapsack constraint and the set $\mathcal{X}_c$ (defined in (5.22)) remains convex.

[5] This is a common assumption [63, 171], which models situations where each file can be split in a large number of small chunks and each cache can store random linear combinations of such chunks. Guarantees for this fractional setting can be readily transferred to an integral setting through randomized rounding techniques [13, 21, 57, 253].

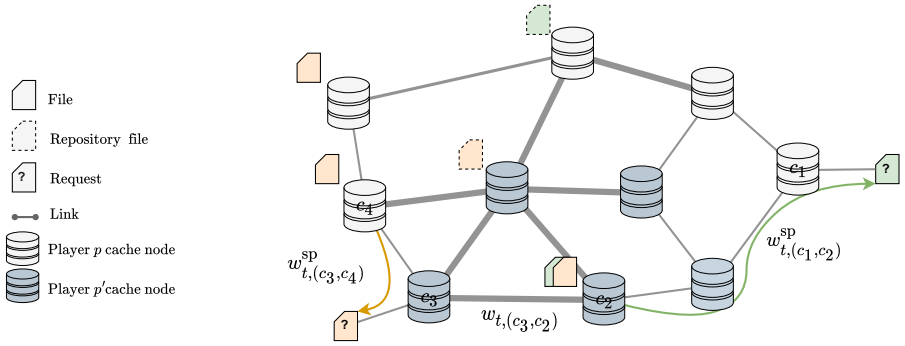


Figure 5.4: System model: a network comprised of a set of caching nodes C. A request arrives at a cache node $c \in C$, it can be partially served locally, and if needed, forwarded along the shortest retrieval path to another node to retrieve the remaining part of the file; a utility is incurred by the cache owner $i \in \mathcal{I}$. A set of permanently allocated files are spread across the network guaranteeing requests can always be served.

model is summarized in Fig. 5.4, and it is aligned with many recent papers focusing on learning for caching [13, 21, 171, 253].

Requests. We denote by $r_{t,c,f} \in \mathbb{N} \cup \{0\}$ the number of requests for file $f \in \mathcal{F}$ submitted by users associated to cache $c \in C$, during slot $t \in \mathcal{T}$. The request batch arriving at timeslot $t \in \mathcal{T}$ is denoted by $\boldsymbol{r}_t = \left(r_{t,c,f}\right)_{(c,f)\in C\times\mathcal{F}}$ and belongs to the set

$$\mathcal{R}_t \triangleq \left\{ \boldsymbol{r} \in (\mathbb{N} \cup \{0\})^{C\times\mathcal{F}} : \sum_{c\in C} \sum_{f\in\mathcal{F}} r_{c,f} \leq R_t \right\},$$

where $R_t \in \mathbb{N}$ is the total number of requests (potentially) arriving at the system at timeslot $t \in \mathcal{T}$.

Caching gain. We consider an agent $i \in \mathcal{I}$ holds a set of caches $\Gamma_i(C) \subset C$, and $\dot{\bigcup}_{i\in\mathcal{I}} \Gamma_i(C) = C$. Hence, the allocation set of agent i is given by $\mathcal{X}_i = \bigtimes_{c\in\Gamma_i(C)} \mathcal{X}_c$. Requests arriving at cache $c \in C$ can be partially served locally, and if needed, forwarded along the shortest path to a nearby cache $c' \in C$ storing the file, incurring a retrieval cost $w^{\text{sp}}_{t,(c,c')}$. Let $\phi_{t,i,c} \triangleq \arg\min_{c'\in\Lambda_i(C)} \left\{ w^{\text{sp}}_{t,(c,c')} \right\}$ and $\Phi_{t,i,c} : \left\{1, 2, \ldots, \phi_{t,i,c}\right\} \subset C \to C$ be a map providing a retrieval cost ordering for every $c \in \left\{1, 2, \ldots, \phi_{t,i,c}\right\}$, $t \in \mathcal{T}$, and $i \in \mathcal{I}$, i.e.,

$$w^{\text{sp}}_{t,(c,\Phi_{t,i,c}(\phi_{t,i,c}))} = \min \left\{ w^{\text{sp}}_{t,(c,c')} : c' \in \Lambda_f(C) \right\} \geq \cdots \geq w^{\text{sp}}_{t,(c,\Phi_{t,i,c}(2))} \geq w^{\text{sp}}_{t,(c,\Phi_{t,i,c}(1))} = 0. \tag{5.23}$$

When a request batch $\boldsymbol{r}_t \in \mathcal{R}_t$ arrives at timeslot $t \in \mathcal{T}$, agent $i \in \mathcal{I}$ incurs the following cost:

$$\text{cost}_{t,i}(\boldsymbol{x}) \triangleq \sum_{c\in\Gamma_i(C)} \sum_{f\in\mathcal{F}} r_{t,c,f} \sum_{k=1}^{\phi_{t,i,c}-1} \left(w^{\text{sp}}_{t,(c,\Phi_{t,i,c}(k+1))} - w^{\text{sp}}_{t,(c,\Phi_{t,i,c}(k))} \right) \left(1 - \min \left\{ 1, \sum_{k'=1}^{k} x_{\Phi_{t,i,c}(k'),f} \right\} \right).$$

This can be interpreted as a QoS cost paid by a user for the additional delay to retrieve part of the file from another cache, or it can represent the load on the network to provide the missing file. Note that by construction, the maximum cost is achieved for a network state, where all the caches are empty except for the repository allocations; formally, such state is given by $\boldsymbol{x}_0 \triangleq \left(\mathbb{1}\left(c \in \Lambda_f(C)\right)\right)_{(c,f)\in C\times\mathcal{F}} \in \mathcal{X}$, and the cost of the agent at this state is given by

$$\text{cost}_{t,i}(\boldsymbol{x}_0) = \sum_{c\in\Gamma_i(C)} \sum_{f\in\mathcal{F}} r_{t,c,f} \min\left\{w^{\text{sp}}_{t,(c,c')} : c' \in \Lambda_f(C)\right\} \tag{5.24}$$

$$= \sum_{c\in\Gamma_i(C)} \sum_{f\in\mathcal{F}} r_{t,c,f} \sum_{k=1}^{\phi_{t,i,c}-1} \left(w^{\text{sp}}_{t,(c,\Phi_{t,i,c}(k+1))} - w^{\text{sp}}_{t,(c,\Phi_{t,i,c}(k))}\right), \tag{5.25}$$

We can define the caching utility at timeslot $t \in \mathcal{T}$ as the cost reduction due to caching as:

$$u_{t,i}(\boldsymbol{x}) \triangleq \text{cost}_{t,i}(\boldsymbol{x}_0) - \text{cost}_{t,i}(\boldsymbol{x}) \tag{5.26}$$

$$= \sum_{c\in\Gamma_i(C)} \sum_{f\in\mathcal{F}} r_{t,c,f} \sum_{k=1}^{\phi_{t,i,c}-1} \left(w^{\text{sp}}_{t,(c,\Phi_{t,i,c}(k+1))} - w^{\text{sp}}_{t,(c,\Phi_{t,i,c}(k))}\right) \min\left\{1, \sum_{k'=1}^{k} x_{\Phi_{t,i,c}(k'),f}\right\}. \tag{5.27}$$

The caching utility is a weighted sum of concave functions with positive weights, and thus concave in $\boldsymbol{x} \in \mathcal{X}$. It is straightforward to check that this problem always satisfies Assumptions (A1)–(A4). The request batches and the time-varying retrieval costs determine whether Assumption (A5) holds. For example, this is the case when request batches are drawn i.i.d. from a fixed unknown distribution (see Example 5.4.3).

5.6.2 Results

Below we describe the experimental setup[6] of the multi-agent cache networks problem, the request traces, and competing policies. Our results are summarized as follows:

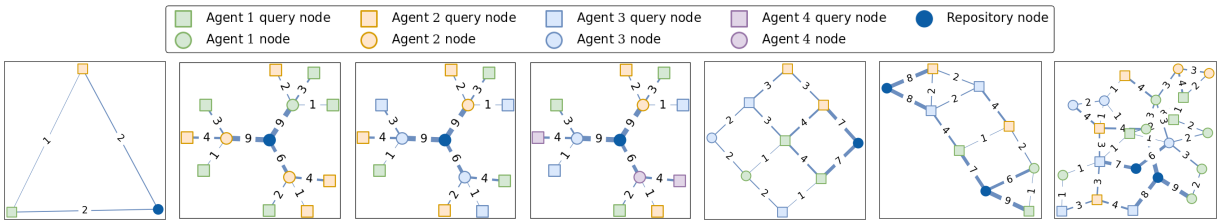
1. Under stationary requests and small batch sizes (leading to large utility deviations from one timeslot to another), OHF achieves the same time-averaged utilities as the offline benchmark, whereas OSF, a counterpart policy to OHF targeting slot-fairness (5.4), diverges and is unable to reach the Pareto front.

2. In the Nash bargaining scenario, OHF achieves the NBS in all cases, while OSF fails when the disagreement points are exigent, i.e., an agent can guarantee itself a high utility.

3. Widely used LFU and LRU might perform arbitrarily bad w.r.t. fairness, and not even achieve any point in the Pareto front (hence, they are not only unfair, but also inefficient).

4. Fairness comes at a higher price when α is increased or the number of agents is increased. This observation on the price of fairness provides experimental evidence for previous work [285].

[6] Our code is publicly available at https://github.com/tareq-si-salem/Online-Multi-Agent-Cache-Networks



(a) Cycle (b) Tree-1 (c) Tree-2 (d) Tree-3 (e) Grid (f) Abilene (g) GEANT

Figure 5.5: Network topologies used in experiments.

5. OHF is robust to different network topologies and is able to obtain time-averaged utilities that match the offline benchmark.

6. Under non-stationary requests, OHF policy achieves the same time-averaged utilities as the offline benchmark, whereas OSF can perform arbitrarily bad providing allocations that are both unfair and inefficient

General Setup. We consider three synthetic network topologies (Cycle, Tree, and Grid), and two real network topologies (Abilene and GEANT). A visualization of the network topologies is provided in Figure 5.5. The specifications of the network topologies used across the experiments are provided in Table 1 in the Appendix. A repository node permanently stores the entire catalog of files. The retrieval costs along the edges are sampled u.a.r. from $\{1, 2, \ldots, 5\}$, except for edges directly connected to a repository node which are sampled u.a.r. from $\{6, 7, \ldots, 10\}$. All the retrieval costs remain fixed for every $t \in \mathcal{T}$. The capacity of each cache is sampled u.a.r. from $\{1, 2, \ldots, 5\}$, but for the Cycle topology in which each cache has capacity 5. An agent $i \in \mathcal{I}$ has a set of query nodes denoted by $Q_i \subset \Gamma_i(C)$, and a query node can generate a batch of requests from a catalog with $|\mathcal{F}| = 20$ files. Unless otherwise said, we consider $u_{\star,\min} = 0.1$ and $u_{\star,\max} = 1.0$. The fairness benchmark refers to the maximizer of the HF objective (5.5), and the utilitarian benchmark refers to the maximizer of HF objective (5.5) for $\alpha = 0$.

Traces. Each query node generates requests according to the following:

- Stationary trace (parameters: σ, R, T, F). Requests are sampled i.i.d. from a Zipf distribution with exponent $\sigma \in \mathbb{R}_{\geq 0}$ from a catalog of files of size F. The requests are grouped into batches of size $|\mathcal{R}_t| = R, \forall t \in \mathcal{T}$.
- Non-Stationary trace (parameters: σ, R, T, F, D). Similarly, requests are sampled i.i.d. from a catalog of F files according to a Zipf distribution with exponent $\sigma \in \mathbb{R}_{\geq 0}$. Every D requests, the popularity distribution is modified in the following fashion: file $f \in \mathcal{F} = \{1, 2, \ldots, F\}$ assumes the popularity of file $f' = (f + F/2) \mod F$ (F is even). The requests are grouped into batches of size $|\mathcal{R}_t| = R, \forall t \in \mathcal{T}$.

The stationary trace corresponds to the stochastic adversary in Example 5.4.3, and the non-stationary trace corresponds to a stochastic adversary with perturbations satisfying the partitioned-severity condition in Eq. (5.10). Two sampled traces are depicted in Figure 36.13 in the Appendix.

Unless otherwise said, query nodes generate *Stationary* traces and $\sigma = 1.2$, $T = 10^4$, $R = 50$, and $D = 50$.

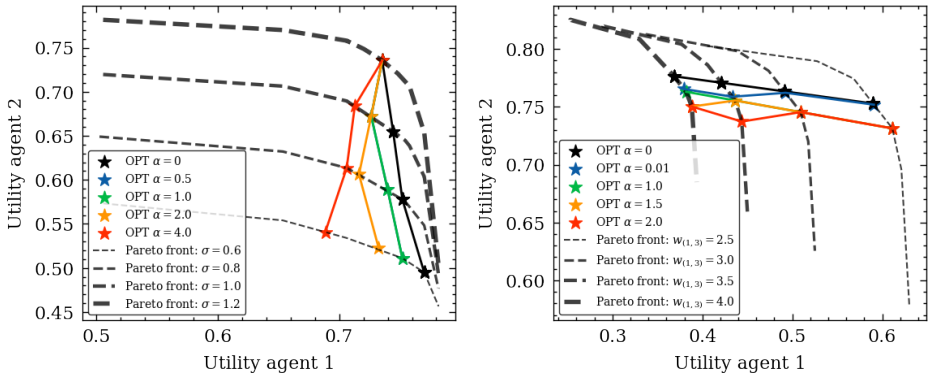
Policies. We implement the following policies and use them as comparison benchmarks for OHF.

- The classic LRU and LFU policies. A request is routed to the cache with minimal retrieval cost among those that store the requested file and this cache provides the content and updates its state corresponding to a hit. Moreover, all caches with a lower retrieval cost update their state as if a miss occurred locally. This corresponds to the popular path replication algorithm [21, 91], equipped with LRU or LFU, adapted to our setting.
- Online slot-fairness (OSF) policy. This policy is the slot-fairness (5.4) counterpart of OHF. It is obtained by configuring Algorithm 5.1 with dual (conjugate) subspace $\Theta = \{(-1)_{i\in\mathcal{I}}\}$ (i.e., taking $\alpha \longrightarrow 0$), which makes ineffective the dual policy in Algorithm 5.1. The revealed utilities at timeslot $t \in \mathcal{T}$ are the α-fairness transformed utilities $\boldsymbol{u}'_t(\,\cdot\,) = \left(f_\alpha\left(u_{t,i}(\,\cdot\,)\right)\right)_{i\in\mathcal{I}}$. The primal allocations are still determined by the same self-confident learning rates' schedule as OHF for a fair comparison. The resulting policy is a no-regret policy (see Lemma 31.3 in Appendix) w.r.t. the slot-fairness benchmark (5.4) for some $\alpha \in \mathbb{R}_{\geq 0}$.

Static analysis of symmetry-breaking parameters. We start with a numerical investigation of the potential caching gains, and how these are affected by the fairness parameter α. In Figure 5.6, we consider the CYCLE topology and different values of $\alpha \in [0, 2]$. We show the impact on the fairness benchmark of varying the request patterns ($\sigma \in \{0.6, 0.8, 1.0, 1.2\}$) for agent 2 under the *Stationary* trace in Fig. 5.6 (a), and of varying the retrieval costs between agent 1's cache and the repository ($w_{(1,3)} \in [2.5, 4]$). In Figure 5.6 (a), we observe decreasing the skewness of the popularity distribution decreases the utility of agent 2 as reflected by the downward shift of the Pareto front. We note that, as far as the file popularity distribution at agent 2 is close to the one at agent 1 ($\sigma = 1.2$), different values of alpha still provide similar utilities. However, in highly asymmetric scenarios, different values of α lead to clearly distinct utilities for each agent. We also note that higher values of α guarantees higher fairness by that increasing the utility of agent 2. Similarly, in Figure 5.6 (b), we observe increasing the retrieval cost for agent 1 decreases the utility achieved by the same agent, as reflected by the leftward shift of the Pareto front; moreover, increasing the retrieval costs (higher asymmetry) highlights the difference between different values of α.

Online analysis of symmetry-breaking parameters. In Figure 5.7, we consider the CYCLE topology, and different values of $\alpha \in \{0, 1, 2\}$. In Figure 5.7 (a)–(b) we consider the retrieval cost $w_{(1,3)} = 3.5$ between agent 1's cache node and the repository node. In Figure 5.7 (c)–(d) query node of agent 1 generates *Stationary* trace ($\sigma = 1.2$) and query node of agent 2 generates *Stationary* trace ($\sigma = 0.6$). We consider two fixed request batch sizes $R \in \{1, 50\}$.

In Figures 5.7 (a) and (c) (for batch size $R = 1$) OHF approaches the fairness benchmark's utilities for different values of α, but OSF diverges for values of $\alpha \neq 0$. For increased request batch size $R = 50$, OHF and OSF exhibit similar behavior. This is expected under stationary utilities; increasing the batch size reduces the variability in the incurred utilities at every timeslot, and the horizon-fairness



(a) (b)

Figure 5.6: Pareto front and fairness benchmark's utilities for different values of $\alpha \in [0, 2]$ under different request patterns (a) ($\sigma \in \{0.6, 0.8, 1.0, 1.2\}$) for agent 2, and different retrieval costs (b) between agent 1's cache and the repository ($w_{(1,3)} \in [2.5, 4.0]$).

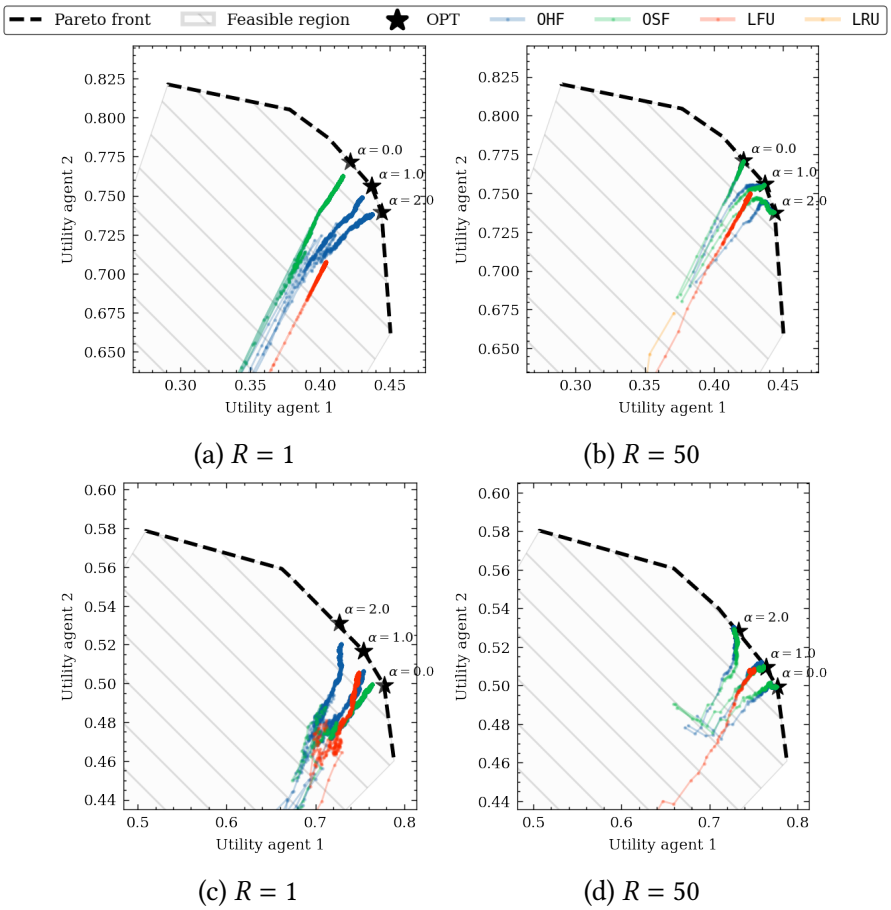
and slot-fairness objectives become closer yielding similar allocations. Note that this observation implies that OSF is only capable to converge for utilities with low variability, which is far from realistic scenarios. LFU policy outperforms LRU and both policies do not approach the Pareto front; thus, the allocations selected by such policies are inefficient and unfair.

Nash bargaining. In Figure 5.8, we consider the Cycle topology and $\alpha = 1$. We select different disagreement utilities for agent 2 in $\{0.0, 0.5, 0.7, 0.75\}$, i.e., different utility values agent 2 expects to guarantee itself even in the absence of cooperation. Note how higher values of disagreement utilities lead to higher utilities for agent 2 at the fairness benchmark. We select $u_{\star,\min} = 0.01$.

For a small batch size ($R = 1$), OHF approaches the same utilities achieved by the fairness benchmark for different disagreement points, whereas OSF fails to approach the Pareto front. Similarly, for a larger batch size $R = 50$, OHF approaches the fairness benchmark for different disagreement points, but the Pareto front is reached faster than with a batch size $R = 1$. OSF diverges for non-zero disagreement points when $R = 50$, because the allocation selected for some agent $i \in \mathcal{I}$ can be smaller than its disagreement utility (i.e., $u_{t,i}(\boldsymbol{x}_t) - u_{t,i} < 0$), while the α-fairness function is only defined for positive arguments.

Impact of agents on the price of fairness. In Figures 5.9 and 5.10, we consider the Tree 1–3 topology, $\alpha \in \{1, 2, 3\}$, and $|\mathcal{I}| \in \{2, 3, 4\}$. Agents' query nodes generate *Stationary* trace ($\sigma \in \{1.2, 0.8, 0.6\}$).

In Figures 5.9 (a)–(c), we observe for increasing the number of agents, the division of utilities differs between the fairness benchmark and utilitarian benchmark; moreover, this difference is more evident for larger values of α. Figure 5.9 (d) provides the price of fairness, and we observe the price of fairness increases with the number of agents and α. Nonetheless, under the different settings the price of fairness remains below 4%, i.e., we experience at most a 4% drop in the social welfare to provide fair utility distribution across the different agents. Figure 5.10 gives the time-averaged



(a) $R = 1$ (b) $R = 50$

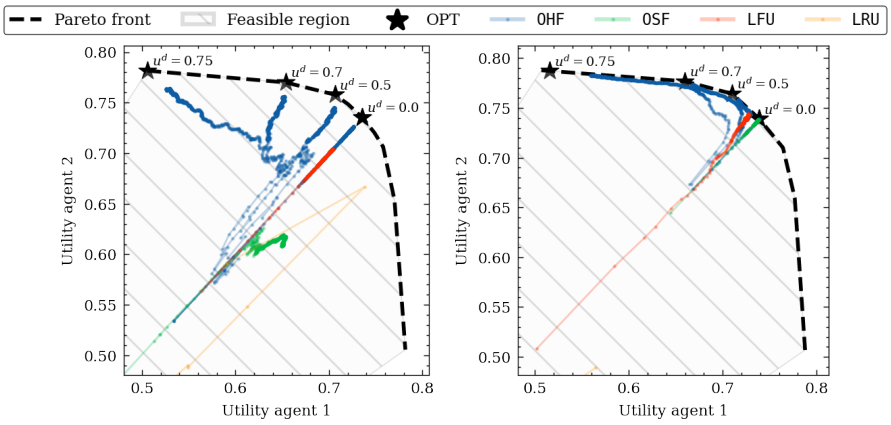
(c) $R = 1$ (d) $R = 50$

Figure 5.7: Time-averaged utilities of policies OHF, OSF, LRU, and LFU under CYCLE topology. Subfigures (a)–(b) are obtained under retrieval cost $w_{(1,3)} = 3.5$ for agent 1's query node. Subfigures (c)–(d) are obtained when agent 2's query node generates *Stationary trace* ($\sigma = 0.6$). Markers correspond to iterations in $\{100, 200, \ldots, 10^4\}$.

utilities obtained by running OHF for $\alpha = 2$. We observe the utilities obtained by OHF quickly converge to the same utilities obtained by the fairness benchmark. In this figure, we also highlight the difference between the utilities achieved by the fairness benchmark and utilitarian benchmark, is reflected by the increasing utility gap for a higher number of participating agents.

Different network topologies. In Figure 5.11 (a), we consider the network topologies TREE, GRID, ABILENE, GEANT under *Stationary* trace ($\sigma \in \{0.6, 1.0, 1.2\}$) and $\alpha = 3$. OHF achieves the same utilities as the fairness benchmark across the different topologies. Note that for larger network topologies agents achieve a higher utility because there are more resources available.

Impact of non-stationarity. In Figure 5.11 (b), we consider the CYCLE topology and $\alpha = 3$. The query node of agent 1 generates *Non-Stationary* trace, while the query node of agent 2 generates a shuffled *Non-Stationary* trace, i.e., we remove the non-stationarity from the trace for agent 2 while



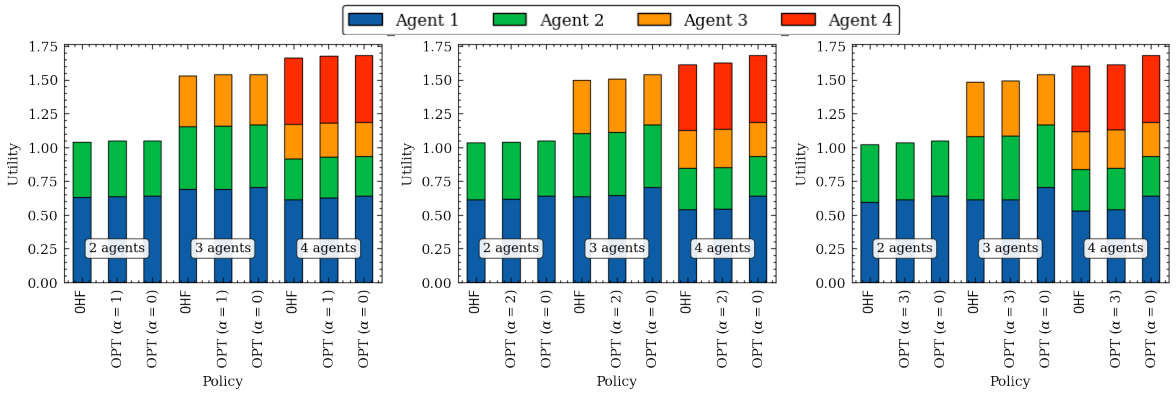
(a) $R = 1$ (b) $R = 50$

Figure 5.8: Time-averaged utilities obtained for policies OHF, OSF, LRU, and LFU for batch sizes (a) $R = 1$ and (b) $R = 100$, under Cycle network topology. Markers correspond to iterations in $\{100, 200, \ldots, 10^4\}$.

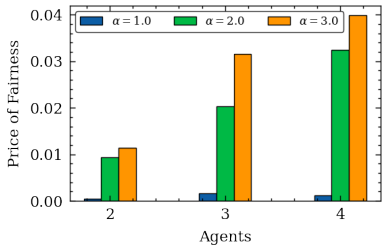
preserving the overall popularity of the requests. Therefore, on average the agents are symmetric and experience the same utilities. We observe in Figure 5.11 (b) that indeed this is the case for OHF policy; however, because OSF aims to insure fairness across the different timeslots the agents are not considered symmetric and the average utilities deviate from the Pareto front (not efficient). OSF favors agent 1 by increasing its utility by 20% compared to the utility of agent 1.

5.7 Conclusion and Future Work

In this work, we proposed a novel OHF policy that achieves horizon-fairness in dynamic resource allocation problems. We demonstrated the applicability of this policy in virtualized caching systems where different agents can cooperate to increase their caching gain. Our work paves the road for several interesting next steps. A future research direction is to consider decentralized versions of the policy under which each agent selects an allocation with limited information exchange across agents. For the application to virtualized caching systems, the message exchange techniques in [21, 255] to estimate subgradients can be exploited. Another important future research direction is to bridge the horizon-fairness and slot-fairness criteria to target applications where the agents are interested in ensuring fairness within a target time window. We observed that OHF can encapsulate the two criteria, however, it remains an open question whether a policy can smoothly transition between them. A final interesting research direction is to consider a limited feedback scenario where only part of the utility is revealed to the agents (e.g., bandit feedback). Our policy could be extended to this setting through gradient estimation techniques [20].

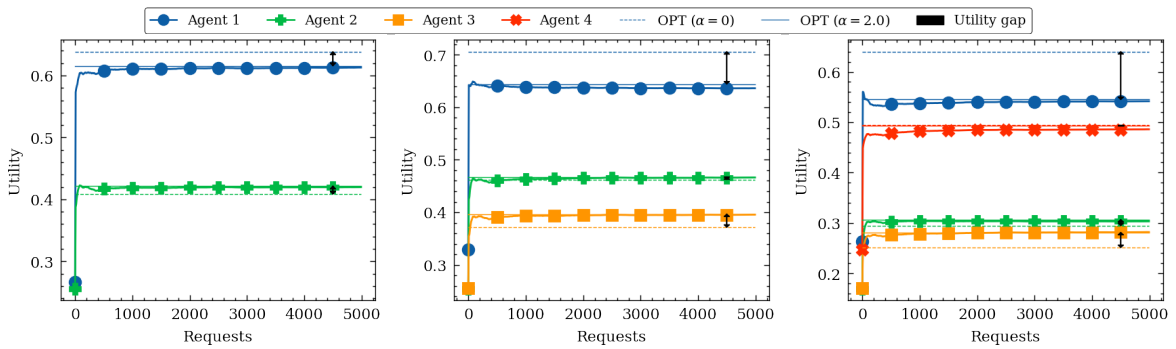


(a) $\alpha = 1$ (b) $\alpha = 2$ (c) $\alpha = 3$



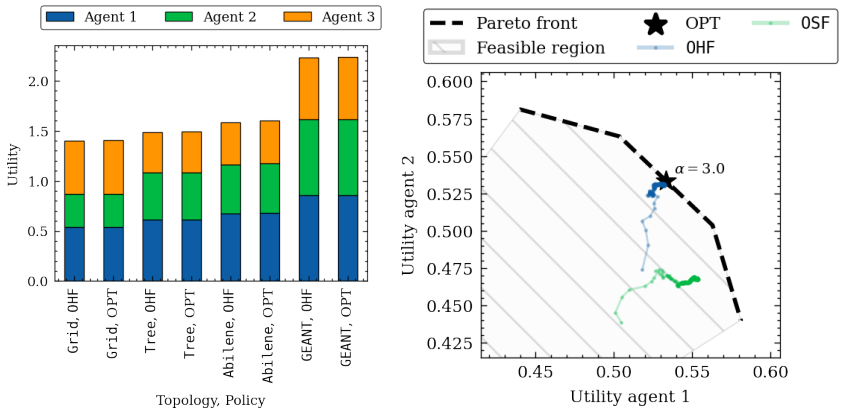
(d) PoF

Figure 5.9: Subfigures (a)–(c) provide the average utility for different agents obtained by OHF, fairness benchmark (OPT for $\alpha \neq 0$), and utilitarian benchmark (OPT for $\alpha = 0$); and Subfigure (d) provides the PoF for $\alpha \in \{0, 1, 2, 3\}$ under an increasing number of agents in $\{2, 3, 4\}$ and TREE 1–3 network topology.



(a) (b) (c)

Figure 5.10: Subfigures (a)–(c) provide the time-averaged utility across different agents obtained by OHF policy and OPT for $\alpha = 2$ under an increasing number of agents in $\{2, 3, 4\}$ and TREE 1–3 network topology.



(a) (b)

Figure 5.11: Subfigure (a) provides the average utility of OHF and fairness benchmark under network topologies Tree, Grid, Abilene, GEANT, *Stationary* trace ($\sigma \in \{0.6, 0.8, 1.2\}$), and $\alpha = 3$. Subfigure (b) provides the time-averaged utilities obtained for OHF, OSF and batch size $R = 50, t \in \mathcal{T}$, under network topology Tree (a) and *Non-stationary* trace. The markers represent the iterations in the set $\{100, 200, \ldots, 10^4\}$.

Chapter 6

Conclusion

We conclude this thesis with some brief concluding remarks and present directions for possible future research arising from the problems considered thus far.

6.1 Summary

In this thesis, we study no-regret algorithms applied to several instances of the network resource allocation problem: exact caching in Chapter 2, similarity caching in Chapter 3, and IDNs in Chapter 4. We demonstrate the versatility of gradient algorithms, that normally operate on continuous spaces, on inherently combinatorial problems (e.g., the NP-Hard problems of similarity caching and IDNs) when paired with an opportune randomized rounding scheme. We show in all these instances that regret performance guarantees extend to integral (combinatorial) settings, despite the need to account for update costs. Our extensive experimental findings support the thesis that these algorithms are robust and can adapt to changing external system's parameters (e.g., requests' popularity and retrieval costs). Chapter 5 provides a novel long-term online fairness framework for settings where the agents' utilities are subject to unknown, time-varying, and potentially adversarial perturbations. We characterize the necessary conditions that a policy needs to satisfy in order to achieve vanishing fairness-regret and prove that our proposal, OHF policy, attains this desirable objective for any α-fairness criterion. This, in turn, renders our framework suitable for enforcing also other important metrics, such as max-min fairness and (weighted) proportional fairness, and for tackling cooperative game problems under the symmetric and asymmetric Nash bargaining solutions.

6.2 Future Work

The following are a number of possible future directions that follow from this thesis:

Exact Caching. The characterization of the optimality regimes of OGD and OMD_{NE}, w.r.t. the diversity ratio, can be further improved. Also $\text{OMD}_{q\text{-norm}}$ algorithms for arbitrary values of $q \in (1, 2)$ deserve more investigation to (1) devise strongly polynomial, efficient algorithms for their Bregman projection, (2) characterize their update costs, and (3) compare their performance with OMD_{NE}.

Similarity Caching. The applicability of similarity caches to machine learning classification tasks [182] remains an open question, when the size of the objects in the catalog is comparable

to their d-dimensional representation in the index, and, as a consequence, the index size cannot be neglected in comparison to the local catalog size. Another important future research direction is to consider dynamic regret, whereby the performance of a policy is compared to a dynamic optimum. However, since the employed OMD algorithms are greedy (i.e., the algorithms do not keep track of the history of the requests), with careful selection of the mirror map and learning rate, adaptive regret guarantee are attainable, e.g., such guarantee holds for OGD [46].

Inference Delivery Networks. The system model considered in Chapter 4 models potential available capacities of allocated ML models to be fully adversarial and exogenously determined. An interesting future direction is to consider that the adversary only selects (or perturbs) the request process, then the system evolves according to a queuing model (e.g., networked counting queues [255]) to model more realistic scenarios.

Long-term Fairness in Dynamic Resource Allocation. A future research direction is to consider decentralized versions of the policy under which each agent selects an allocation with limited information exchange across agents. For the application to virtualized caching systems, the message exchange techniques in Section 2.3 in Chapter 2 can be exploited to estimate subgradients in a distributed fashion. Another important future research direction is to bridge the horizon-fairness and slot-fairness criteria to target applications where the agents are interested in ensuring fairness within a target time window. We observed that our policy OHF can encapsulate the two criteria, however, it remains an open question whether a policy can smoothly transition between them. A final interesting research direction is to consider a limited feedback scenario where only part of the utility is revealed to the agents. Our policy could be extended to this setting through gradient estimation techniques [20].

Appendix

1 Fractional Caching and Gradient-based algorithms

1.1 Proof of Proposition 2.2.1

Consider a catalog of files $\mathcal{N} = \{1, 2\}$, cache size $k = 1$, and equal service costs $w_1 = w_2 = 1$. The aggregate cost minimization policy $\mathcal{A}$ has an arbitrary initial state $\boldsymbol{x}_1 \in \mathcal{X}$. The adversary picks the following sequence of request batches $\{\boldsymbol{r}_t\}_{t=1}^T = \{\boldsymbol{e}_1, 2\boldsymbol{e}_2, 2\boldsymbol{e}_1, 2\boldsymbol{e}_2, \dots\}$, where $\boldsymbol{e}_i = [\mathbb{1}_{\{j=i\}}]_{j\in\{1,2\}}$. The aggregate cost at time slot t for a fixed cache state $\boldsymbol{x} \in \mathcal{X}$ is given by

$$\sum_{t'=1}^{t} f_{\boldsymbol{r}_{t'}}(\boldsymbol{x}) = \begin{cases} t(1-x_1) + (t-1)(1-x_2) & \text{if } t \text{ is odd,} \\ (t-1)(1-x_1) + t(1-x_2) & \text{if } t \text{ is even.} \end{cases} \tag{1.1}$$

For any time slot $t > 1$ the aggregate cost minimization policy selects the state $\boldsymbol{x}_t$ that minimizes $\sum_{t=1}^{t-1} f_{\boldsymbol{r}_t}(\boldsymbol{x})$. The policy $\mathcal{A}$ selects the following sequence of states $\{\boldsymbol{x}_t\}_{t=1}^T = \{\boldsymbol{x}_1, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_1, \boldsymbol{e}_2, \dots\}$, and it incurs over the time horizon T the total cost $\sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) = f_{\boldsymbol{r}_1}(\boldsymbol{x}_1) + 2(T-1)$. Consider a different policy that permanently selects $\boldsymbol{e}_1$, such policy incurs the total cost given by $\sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{e}_1) = 0+2+0+2+0+\cdots \leq T$. Then the optimal static allocation $\boldsymbol{x}_*$ has cost at most T; therefore, we obtain a lower bound on the regret of $\mathcal{A}$ as $\mathrm{Regret}_T(\mathcal{A}) \geq f_{\boldsymbol{r}_t}(\boldsymbol{x}_1) + 2(T-1) - T \geq T-2$. We conclude that the aggregate cost minimization policy has linear regret.

1.2 Online Mirror Descent

Theorem 1.1. *([53, Theorem 4.2]) Let (1) the map $\Phi : \mathcal{D} \to \mathbb{R}$ be a mirror map (see Section 2.2.3.2) ρ-strongly convex w.r.t a norm $\|\cdot\|$ over $\mathcal{S} \cap \mathcal{D}$ ($\mathcal{S}$ is a convex set), (2) the cost functions $f_t : \mathcal{S} \to \mathbb{R}$ be convex with bounded gradients (i.e., $\|\nabla f_t(\boldsymbol{x})\|_* \leq L, \forall \boldsymbol{x} \in \mathcal{S}$) for every $t \in [T]$, where $\|\cdot\|_*$ is the dual norm of $\|\cdot\|$, (3) and the Bregman divergence $D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1)$ be bounded by D^2 for $\boldsymbol{x} \in \mathcal{S}$ where $\boldsymbol{x}_1 = \arg\min_{\boldsymbol{x}\in\mathcal{S}\cap\mathcal{D}} \Phi(\boldsymbol{x})$. Then Algorithm 2.1 satisfies*

$$\sum_{t=1}^{T} f_t(\boldsymbol{x}_t) - \sum_{t=1}^{T} f_t(\boldsymbol{x}) \leq \frac{D^2}{\eta} + \frac{\eta L^2}{2\rho} T, \tag{1.2}$$

where $\boldsymbol{x}$ is a fixed point in $\mathcal{S}$.[1]

We remark that Theorem 1.1 is expressed differently in [53], where $f_t = f, \forall t \in [T]$ (fixed cost function). Nonetheless, as observed in [53, Section 4.6] the bound obtained in Eq. (1.2) holds as long as the dual norms of the gradients are bounded by L.

[1] Note that the fixed point $\boldsymbol{x}$ can be selected as the minimizer of the aggregate cost (i.e., $\boldsymbol{x}_* \in \arg\min_{\boldsymbol{x}\in\mathcal{S}} \sum_{t=1}^T f_t(\boldsymbol{x})$).

1.3 Proof of Theorem 2.2.3

The map $\Phi(x) = \frac{1}{2}\|x\|_q^2, q \in (1,2]$ is $\rho = q-1$ strongly convex w.r.t $\|\cdot\|_q$ over $\mathcal{D} = \mathbb{R}^N$ a direct result from [292, Lemma 17], and the dual norm of $\|\cdot\|_q$ is $\|\cdot\|_p$ (Hölder's inequality).Take $\mathcal{S} = \mathcal{X}$. The minimum value of $\Phi(x)$ over $\mathcal{X}$ is achieved when we spread the capacity mass k over the decision variable, i.e., $x_i = \frac{k}{N}, i \in \mathcal{N}$. If we select $\boldsymbol{x}_1$ to be the minimizer of $\Phi(\boldsymbol{x})$, then we have $\nabla\Phi^T(\boldsymbol{x}_1)(\boldsymbol{x} - \boldsymbol{x}_1) \geq 0, \forall \boldsymbol{x} \in \mathcal{X}$ [20, Theorem 2.2], so we obtain $D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1) = \Phi(\boldsymbol{x}) - \Phi(\boldsymbol{x}_1) - \nabla\Phi(\boldsymbol{x}_1)^T(\boldsymbol{x} - \boldsymbol{x}_1) \leq \Phi(\boldsymbol{x}) - \Phi(\boldsymbol{x}_1)$. Moreover, it is easy to check that Φ is maximized at a sparse point $\boldsymbol{x}_* \in \mathcal{X} \cap \{0,1\}^N$; thus, we have $D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1) \leq \Phi(\boldsymbol{x}_*) - \Phi(\boldsymbol{x}_1)$. By replacing $\boldsymbol{x}_1$ and $\boldsymbol{x}_*$ with their values in the previous equation we get $\Phi(\boldsymbol{x}_1) = \frac{1}{2}\left(\left(\frac{k}{N}\right)^q N\right)^{\frac{2}{q}} = \frac{1}{2}k^2 N^{-\frac{2}{p}}$, and $\Phi(\boldsymbol{x}_*) = \frac{1}{2}k^{\frac{2}{q}} = \frac{1}{2}k^2 k^{-\frac{2}{p}}$. Thus, we have

$$D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1) \leq \frac{1}{2}k^2\left(k^{-\frac{2}{p}} - N^{-\frac{2}{p}}\right) = D^2. \tag{1.3}$$

Note that the maximum of $\|\boldsymbol{r}\|_p$ is achieved when $\frac{R}{h}$ components are set to h, then the following bound holds on the gradients

$$\max_{r\in\mathcal{R}} \|\nabla f_{\boldsymbol{r}}(\boldsymbol{x})\|_p \leq \max_{\boldsymbol{r}\in\mathcal{R}} \|\boldsymbol{w}\|_\infty \|\boldsymbol{r}\|_p = \|\boldsymbol{w}\|_\infty h\left(\frac{R}{h}\right)^{\frac{1}{p}} = L. \tag{1.4}$$

The gradients are bounded in the dual norm $\|\nabla f_r(\boldsymbol{x}_t)\|_p \leq L, \forall \boldsymbol{r} \in \mathcal{R}$.

The final bound follows by Theorem 1.1, plugging (1.4) and (1.3) in (1.2), and selecting the learning rate that achieves the tightest bound $\eta = \sqrt{(q-1)k^2\left(k^{-\frac{2}{p}} - N^{-\frac{2}{p}}\right)/\left(\|w\|_\infty^2 h^2 \left(\frac{R}{h}\right)^{\frac{2}{p}} T\right)}$.

1.4 Proof of Corollary 2.2.5

Taking $\alpha = \frac{k}{N}$ and $\beta = \frac{Nh}{R}$, we can rewrite (2.15) to have $\text{Regret}_T(\text{OMD}_{q\text{-norm}}) \leq \|\boldsymbol{w}\|_\infty R\beta^{\frac{1}{q}}\sqrt{\frac{\alpha^{2/q}-\alpha^2}{q-1}T}$. We take the limit $q \to 1$, to obtain the upper bound

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

1.5 Proof of Theorem 2.2.6

We take the simplified version of the regret of the general class of q-norm mirror maps in Eq. (2.15), select $\alpha = \frac{k}{N}$ and $\beta = \frac{Nh}{R}$, so we get $\text{Regret}_T(\text{OMD}_{q\text{-norm}}) \leq \|\boldsymbol{w}\|_\infty R\phi(q)\sqrt{T}$, where $\phi(q) \triangleq$

$\beta^{\frac{1}{q}}\sqrt{\frac{\alpha^{2/q}-\alpha^2}{q-1}}$. The tightest regret bound is achieved with q^* that minimizes $\phi(q)$. We have

$$\phi'(q) = -\alpha^2\beta^{\frac{1}{q}}\frac{q^2\left(\alpha^{2/q-2}-1\right)+2(q-1)\left(\alpha^{2/q-2}\log(\alpha)+\left(\alpha^{2/q-2}-1\right)\log(\beta)\right)}{2q^2(q-1)^2\sqrt{\frac{\alpha^{2/q}-\alpha^2}{q-1}}}. \tag{1.5}$$

We study the sign $(-J(q))$ of the derivative of the minimizer in Eq (1.5)

$$J(q) = q^2\left(\alpha^{2/q-2}-1\right)+2(q-1)\left(\alpha^{2/q-2}\log(\alpha)+\left(\alpha^{2/q-2}-1\right)\log(\beta)\right) \tag{1.6}$$

$$\geq 2q(1-q)\log(\alpha)+2(q-1)\left(2\frac{1-q}{q}\log(\alpha)\log(\alpha\beta)+\log(\alpha)\right) \tag{1.7}$$

$$\geq 2q(q-1)\left((1-q)\log(\alpha)+2\frac{1-q}{q}\log(\alpha)\log(\alpha\beta)\right). \tag{1.8}$$

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

1.6 Proof of Theorem 2.2.7

We have the following regret upper bound for the q-norm mirror map, as $q \to 1$ from Corollary 2.2.5: $\text{Regret}_T(\text{OMD}_{\text{1-norm}}) \leq \|\boldsymbol{w}\|_\infty hk\sqrt{2\log\left(\frac{N}{k}\right)T}$. In [293], it is proved that the log function satisfies $\log(u+1) \leq \frac{u}{\sqrt{u+1}}, u \geq 0$. We take $u = \frac{N}{k}-1$, and note that $N \geq k > 0$, so we get $u \geq 0$. We have the following $\log\left(\frac{N}{k}\right) \leq \frac{N-k}{\sqrt{Nk}} = \sqrt{\frac{N}{k}}\left(1-\frac{k}{N}\right)$. Thus, the upper bound in Corollary 2.2.5 Eq. (2.17) can be loosened to obtain

$$\text{Regret}_T(\text{OMD}_{\text{1-norm}}) \leq \|\boldsymbol{w}\|_\infty kh\sqrt{2\log\left(\frac{N}{k}\right)T} \leq \|\boldsymbol{w}\|_\infty\sqrt{2\sqrt{Nk}h^2k\left(1-\frac{k}{N}\right)}. \tag{1.9}$$

If we take $\frac{R}{h} \geq 2\sqrt{Nk}$, then this upper bound is tighter than the upper bound on the regret of OGD in Corollary 2.2.4.

1.7 Link between Neg-entropy OMD and q-norm OMD

Theorem 1.2. *The algorithm* $\text{OMD}_{\text{1-norm}}$ *defined as the limitting algorithm obtain by taking q converges to 1 of* $\text{OMD}_{q\text{-norm}}$ *with learning rate* $\eta_q = \eta(q-1)k$*, intermediate states* $\boldsymbol{y}_t^{(q)}$*, and fractional states* $\boldsymbol{x}_t^{(q)}$ *for* $t \geq 1$ *is equivalent to* OMD_{NE} *configured with learning rate* $\eta \in \mathbb{R}_+$ *over the simplex (capped simplex* $\mathcal{X}$ *with* $k = 1$*), when both policies are initialized with same state in* $\mathbb{R}^N_{>0} \cap \mathcal{X}$*. Moreover,* $\text{OMD}_{\text{1-norm}}$ *has a multiplicative update rule over the capped simplex.*

Proof.

Let $\boldsymbol{g}_t = \nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t)$ be the gradient of the cost function at time slot t. From lines 2–3 in Algorithm 2.1 and Eq. (2.18) we obtain the following $\hat{y}_{t,i}^{(q)} = \left(\nabla\Phi(\boldsymbol{x}_t)\right)_i + \eta_q g_{t,i} = \operatorname{sign}(x_{t,i}) \frac{|x_{t,i}|^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta_q g_{t,i}$ for a given $q \in (1, 2]$. The algorithm guarantees that $\boldsymbol{x}_t \in \mathcal{X} \cap \mathbb{R}^N_{>0}$, then $x_i > 0, \forall i \in \mathcal{N}$. So, we get $\hat{y}_{t,i}^{(q)} = \frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta_q g_{t,i}, \forall i \in \mathcal{N}$. Note that $-\eta g_{t,i}$ is non-negative. The numbers p and q are conjugate numbers (see Section 2.2.3.3) and satisfy $q - 1 = \frac{1}{p-1}$. We use Eq. (2.19) to get the expression of $y_{t+1,i}^{(q)}$ as

$$\frac{\left(\frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta_q g_{t,i}\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}}\left(\frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta_q g_{t,i}\right)^p\right)^{\frac{p-2}{p}}} = \frac{\left(\frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta(q-1)k g_{t,i}\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}}\left(\frac{(x_{t,i})^{q-1}}{\|\boldsymbol{x}_t\|_q^{q-2}} - \eta(q-1)k g_{t,i}\right)^p\right)^{\frac{p-2}{p}}} \tag{1.10}$$

$$= \frac{x_{t,i}\,\|\boldsymbol{x}_t\|_q^{(2-q)}\left(1 - \eta(q-1)k g_{t,i}\frac{\|\boldsymbol{x}_t\|_q^{q-2}}{(x_{t,i})^{q-1}}\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}}(x_{t,i})^{(q-1)p}\left(1 - \eta(q-1)k g_{t,i}\frac{\|\boldsymbol{x}_t\|_q^{q-2}}{(x_{t,i})^{q-1}}\right)^p\right)^{\frac{p-2}{p}}}. \tag{1.11}$$

We rewrite the above expression solely in terms of p to obtain

$$y_{t+1,i}^{\left(\frac{p}{p-1}\right)} = \frac{x_{t,i}\,\|\boldsymbol{x}_t\|_{\frac{p}{p-1}}^{\left(2-\frac{p}{p-1}\right)}\left(1 - \eta g_{t,i}k \Big/ \left((x_{t,i})^{\frac{1}{p-1}}\,\|\boldsymbol{x}_t\|_{\frac{p}{p-1}}^{\left(2-\frac{p}{p-1}\right)}\right)\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}}(p-1)(x_{t,i})^{\left(\frac{p}{p-1}\right)}\left(1 - \eta g_{t,i}k \Big/ \left((p-1)(x_{t,i})^{\frac{1}{p-1}}\,\|\boldsymbol{x}_t\|_{\frac{p}{p-1}}^{\left(2-\frac{p}{p-1}\right)}\right)\right)^p\right)^{\frac{p-2}{p}}}. \tag{1.12}$$

Taking the limit for q converges to 1 is equivalent to let p diverges to $+\infty$, so we have

$$y_{t+1,i} \triangleq \lim_{p\to+\infty} y_{t+1,i}^{\left(\frac{p}{p-1}\right)} = x_{t,i}k\left(\lim_{p\to+\infty}\frac{\left(1 - \frac{\eta g_{t,i}}{p-1}\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}} x_{t,i}\left(1 - \frac{\eta g_{t,i}}{p-1}\right)^p\right)^{\frac{p-2}{p}}}\right) \tag{1.13}$$

$$= x_{t,i}k\left(\lim_{p\to+\infty}\frac{\left(1 - \frac{\eta g_{t,i}}{p-1}\right)^{p-1}}{\left(\sum_{i\in\mathcal{N}} x_{t,i}\left(1 - \frac{\eta g_{t,i}}{p-1}\right)^p\right)}\right) \tag{1.14}$$

$$= x_{t,i}k\frac{\exp\left(-\eta g_{t,i}\right)}{\sum_{i\in\mathcal{N}} x_{t,i}\exp\left(-\eta g_{t,i}\right)}, \forall i \in \mathcal{N}. \tag{1.15}$$

- The intermediate state of $\text{OMD}_{\text{1-norm}}$ in Eq. (1.15) is a multiplicative update rule identical to the update rule of OMD_{NE} ($y_{t+1,i} = x_{t,i}\,e^{-\eta g_{t,i}}$ in Eq. (2.23)) with an additional multiplicative factor $\frac{k}{\sum_{i\in\mathcal{N}} x_{t,i}\exp\left(-\eta g_{t,i}\right)}$.

- For $k = 1$, the intermediate state of $\text{OMD}_{\text{1-norm}}$ in Eq. (1.15) is feasible (i.e., $\boldsymbol{x}_{t+1} = \boldsymbol{y}_{t+1}$ and the projection has no effect). On the other hand, the neg-entropy projection in the case of $k = 1$ is just a normalization of the intermediate states (i.e., $x_{t+1,i} = \frac{x_{t,i}\, e^{-\eta g_{t,i}}}{\sum_{i \in \mathcal{N}} x_{t,i} \exp\left(-\eta g_{t,i}\right)}$ for $i \in \mathcal{N}$). Thus, the states obtained by the two algorithms coincide.

□

1.8 Proof of Theorem 2.2.8

The neg-entropy mirror map is $\rho = \frac{1}{k}$-strongly convex w.r.t the norm $\|\cdot\|_1$ over $\mathcal{X} \cap \mathcal{D}$ [48, Example 2.5]. The dual norm of $\|\cdot\|_1$ is $\|\cdot\|_\infty$. By taking $p \to \infty$ in Eq. (1.4) we can consider as bound for the gradient in Eq. (1.2)

$$L = \|\boldsymbol{w}\|_\infty h. \tag{1.16}$$

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

$$D_\Phi(\boldsymbol{x}, \boldsymbol{x}_1) \leq \Phi(\boldsymbol{x}) - \Phi(\boldsymbol{x}_1) \leq -\Phi(\boldsymbol{x}_1) = -\sum_{i=1}^{N} \frac{k}{N} \log\left(\frac{k}{N}\right) = k \log\left(\frac{N}{k}\right) = D^2. \tag{1.17}$$

The bound follows by Theorem 1.1, plugging (1.16) and (1.17) in (1.2), and selecting the learning rate that gives the tightest upper bound, that is $\eta = \sqrt{\frac{2\log(\frac{N}{k})}{\|w\|_\infty^2 h^2 T}}$.

1.9 Proof of Theorem 2.2.9

We adapt the Euclidean projection algorithm in [56]. Finding the projection $\mathbf{x} = \Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\mathbf{y})$ corresponds to solving a convex problem as $D_\Phi(\mathbf{x}, \mathbf{y})$ is convex in $\mathbf{x}$ and $\mathcal{X} \cap \mathcal{D}$ is a convex set. Without loss of generality, we assume the components of $\mathbf{x} = \Pi^{\Phi}_{\mathcal{X}\cap\mathcal{D}}(\mathbf{y})$ to be in non-decreasing order. Let $b \in \mathcal{N}$ be the index of the largest component of $\mathbf{x}$ smaller than 1. The KKT conditions lead to conclude that if the components of $\mathbf{y}$ are ordered in ascending order, so are the components of $\mathbf{x}$. In particular, the smallest b components of $\mathbf{x}$ can be obtained as $x_i = y_i e^\gamma$ and $y_b e^\gamma < 1 \leq y_{b+1} e^\gamma$, where γ is the Lagrangian multiplier associated with the capacity constraint. If b is known, then it follows from the capacity constraint that $m_b \triangleq e^\gamma = \frac{k+b-N}{\sum_{i=1}^{b} y_i} = \frac{k+b-N}{\|\mathbf{y}\|_1 - \sum_{i=b+1}^{N} y_i}$. We observe that necessarily $b \in \{N-k+1, \ldots, N\}$. In fact, we cannot have $b \leq N-k$. If $b \leq N-k$, we get $\sum_{i=N-k+1}^{N} x_i \geq k$ and the capacity constraint implies that $x_i = 0, \forall i \leq b$, but we must have $x_i > 0$ since $\mathbf{x} \in \mathcal{X} \cap \mathcal{D}$ and $\mathcal{D} = R^N_{>0}$. We can then find the value of b, but checking which number in $\{N-k+1, \ldots, N\}$ satisfies $y_b e^\gamma < 1 \leq y_{b+1} e^\gamma$. Note that this operation only requires the largest k components of $\mathbf{y}$. The projection corresponds to setting the components $y_{b+1}, \ldots, y_N$ to 1 and multiply the other $N-b$ components by m_b. In order to avoid updating all components at each step, we can simply set the components x_i for $i > b$ (those that should be set equal to 1) to $\frac{1}{m_b}$. Then, at any time t, we can recover the value of $x_{t,i}$, multiplying the i-th component of the vector $\mathbf{x}$ by $P = \prod_{s=1}^{t} m_{b,s}$, where $m_{b,s}$ is the returned m_b from the Bregman projection at time step s. For general values of R and h,

the projection step takes $O(k)$ steps per iteration and a partial sort is required to maintain top-k components of y_t sorted; this can be done using partial sorting in $O(N + k\log(k))$ [294]. When $R = h = 1$, Algorithm 1 leads to only a single state coordinate update, and requires $O(\log(k))$ steps to maintain top-k components of $\mathbf{x}_t$ sorted online.

1.10 Proof of Proposition 2.2.10

Every time slot $t \in [T]$, we obtain an intermediate cache state $\boldsymbol{y}_{t+1}$ through lines 2–4 in Algorithm 2.1 as $\boldsymbol{y}_{t+1} = (\nabla\Phi^{-1})(\nabla\Phi(\boldsymbol{x}_t) - \eta\nabla f_{\boldsymbol{r}_t}(\boldsymbol{x}_t))$. Let $\{\boldsymbol{x}_t\}_{t=1}^T$ and $\{\boldsymbol{x}'_t\}_{t=1}^T$ be fractional cache states obtained by OGD and OMD_{NE}, respectively, and $\{\boldsymbol{y}_t\}_{t=1}^T$ and $\{\boldsymbol{y}'_t\}_{t=1}^T$ be their intermediate fractional cache states. In the case of OGD, or equivalently OMD configured with mirror map $\Phi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}\|_2^2$, the intermediate fractional states have the same components as the previous cache state for files are not requested $\boldsymbol{y}_{t,i} = \boldsymbol{x}_{t,i}$ for every $i \notin \text{supp}(\boldsymbol{r}_t)$. Similarly, we also have $\boldsymbol{y}'_{t,i} = \boldsymbol{x}'_{t,i}$ for OMD_{NE} for every $i \notin \text{supp}(\boldsymbol{r}_t)$. The Euclidean projection algorithm onto the capped simplex [56] can only set a component of the intermediate fractional cache state to one if it exceeds it, and the remaining components are either set to zero or reduced by a constant amount $\Delta = \frac{N-b-k+\sum_{j=a+1}^{b} y_{t+1,j}}{b-a}$, where a is the number of components set to zero and b is the number of components strictly less than one, and $\Delta \geq y_{t+1,a}$ (a KKT condition in [56]) and in turn $\Delta \geq 0$ because $y_{t+1,i} \geq 0$ for any $i \in \mathcal{N}$. Therefore, all the components $i \notin \text{supp}(\boldsymbol{r}_t)$ of the resulting state are decreased or at most kept unchanged. Similarly, the neg-entropy Bregman projection onto the capped simplex sets some components to one if they exceed it, and the remaining components are scaled by a constant m_b. In our caching setting we have $y_{t+1,i} \geq x_{t,i}$ for $i \in \mathcal{N}$ in turn $\|\boldsymbol{y}_{t+1}\|_1 \geq k$, thus the equality constraint $\|\boldsymbol{x}\|_1 = k$ in the projection can be replaced by $\|\boldsymbol{x}\|_1 \leq k$. From the KKT dual feasibility condition we obtain $-\gamma \geq 0$ and $m_b = e^{\gamma} \leq 1$. Thus, we have $x_{t+1,i} \leq x_{t,i}$ for every $i \notin \text{supp}(\boldsymbol{r}_t)$. We conclude that the update cost is zero for both policies, i.e., $\text{UC}_{\boldsymbol{r}_t}(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}) = \sum_{i\notin\text{supp}(\boldsymbol{r}_t)} w'_i \max(0, x_{t+1,i} - x_{t,i}) = 0$.

2 Integral Caching

2.1 Proof of Proposition 2.2.11

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

$$\sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}) = \sum_{t=1}^T \sum_{i=1}^N (1 - x_i)(1 - x_{t,i}) = T(N-2k) + \sum_{i=1}^N x_i \sum_{t=1}^T x_{t,i}. \tag{2.18}$$

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

over time horizon T is lower bounded by

$$\text{Regret}_T(\mathcal{A}) = \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) - \sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x}_*) \geq T(N-k) - T(N-2k) - T\frac{k^2}{N} = k\left(1-\frac{k}{N}\right)T. \quad (2.19)$$

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

2.2 Proof of Proposition 2.2.12

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

2.3 Proof of Theorem 2.2.13

We consider the catalog $\mathcal{N} = \{1,2\}$, cache capacity $k = 1$, and equal service and update costs $w_i = w_i' = 1$ for $i \in \mathcal{N}$. A policy $\mathcal{A}$ selects the states $\{\boldsymbol{x}_t\}_{t=1}^T \in \mathcal{X}^T$. The randomized states obtained by Ξ are $\{\boldsymbol{z}_t\}_{t=1}^T$; thus, we have $\boldsymbol{z}_t = [1,0]$ w.p. $x_{t,1}$, and $\boldsymbol{z}_t = [0,1]$ w.p. $x_{t,2}$. An adversary selects the request batch as $\boldsymbol{r}_t = [\mathbb{1}_{\{i=i_t\}}]_{i\in\mathcal{N}}$ aiming to greedily maximize the cost of the cache, where

$$i_t = \arg\min_{i\in\mathcal{N}}\{x_{t,i}\}. \quad (2.20)$$

The expected service cost at time t is $\mathbb{E}\left[f_{\boldsymbol{r}_t}(\boldsymbol{z}_t)\right] = 1 - x_{t,i_t}$, and the expected update cost is

$$\mathbb{E}\left[\text{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] = \mathbb{P}\left(z_{t,i_t} = 1, z_{t+1,i_t} = 0\right) = \mathbb{P}\left(z_{t,i_t} = 1\right)\mathbb{P}\left(z_{t+1,i_t} = 0\right) = x_{t,i_t}(1 - x_{t+1,i_t}). \quad (2.21)$$

An update cost is incurred when i_t is requested and the state changes from $z_{t,i_t} = 1$ to $z_{t+1,i_t} = 0$; we pay a unitary cost due to fetching a single file that is not requested with probability $\mathbb{P}\left(z_{t,i_t} = 1, z_{t+1,i_t} = 0\right)$, and this gives the first equality. We use independence of the random variables $\boldsymbol{z}_t$ and $\boldsymbol{z}_{t+1}$ to obtain the second equality. A fixed state $\boldsymbol{x} = \left[\frac{1}{2}, \frac{1}{2}\right]$ incurs a cost of $\frac{1}{2}$ for every timeslot $t \in [T]$. We define the instantaneous extended regret w.r.t. the fixed state $\boldsymbol{x}$ for every timeslot $t \in [T]$ as $a_t = \mathbb{E}\left[f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) + \text{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] - f_{\boldsymbol{r}_t}(\boldsymbol{x}) = \mathbb{E}\left[f_{\boldsymbol{r}_t}(\boldsymbol{x}_t) + \text{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] - \frac{1}{2}$. Observe here that looking for a minimizer over $\mathcal{X}$ or over $\mathcal{Z}$ is equivalent. Because $\boldsymbol{x}$ is not necessarily the minimizer of the aggregate service cost $\sum_{t=1}^T f_{\boldsymbol{r}_t}(\boldsymbol{x})$, we can lower bound the extended regret (2.30) as

$$\text{E-Regret}_T(\mathcal{A}, \Xi) \geq \sum_{t=1}^T a_t. \quad (2.22)$$

Without loss of generality assume that T is even so $\sum_{t=1}^T a_t = \sum_{k=1}^{T/2}(a_{2k} + a_{2k-1})$, and we have

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

From the definition of i_{2k} in Eq. (2.20) we have $x_{2k,i_{2k}} \leq (1 - x_{2k,i_{2k-1}})$ for any $k \in [T/2]$. Thus,

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

The second and third inequalities are obtained using $x_{t,i_t} \leq \frac{1}{2}$ for every timeslot $t \in [T]$; a direct result from the definition of i_t in Eq. (2.20). We combine the above lower bound with Eq. (2.22) to obtain $\text{E-Regret}_T(\mathcal{A}, \Xi) \geq \frac{T}{8}$.

2.4 Family of Coupling Schemes with Sublinear Update Cost

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

Theorem 2.1. *Consider an OMD Algorithm and a joint distribution of $(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})$ that satisfy (a) $\mathbb{E}[\boldsymbol{z}_t] = \boldsymbol{x}_t$ and $\mathbb{E}[\boldsymbol{z}_{t+1}] = \boldsymbol{x}_{t+1}$, and (b) $\mathbb{E}\left[\text{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] = O\left(\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_1\right)$. This algorithm incurs an expected service cost equal to the service cost of the fractional sequence. Moreover, if $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$, the algorithm has also $O\left(\sqrt{T}\right)$ expected update cost and then $O\left(\sqrt{T}\right)$ extended regret.*

Proof.

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

$$\mathbb{E}\left[\text{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] \leq \gamma_1\gamma_2 \|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|. \tag{2.23}$$

As Φ is ρ-strongly convex w.r.t. the norm $\|\cdot\|$

$$\begin{aligned} D_\Phi(\boldsymbol{x}_t, \boldsymbol{y}_{t+1}) &= \Phi(\boldsymbol{x}_t) - \Phi(\boldsymbol{y}_{t+1}) - \nabla\Phi(\boldsymbol{y}_{t+1})^T(\boldsymbol{x}_t - \boldsymbol{y}_{t+1}) \\ &= \Phi(\boldsymbol{x}_t) - \Phi(\boldsymbol{y}_{t+1}) + \nabla\Phi(\boldsymbol{x}_t)^T(\boldsymbol{y}_{t+1} - \boldsymbol{x}_t) + (\nabla\Phi(\boldsymbol{x}_t) - \nabla\Phi(\boldsymbol{y}_{t+1}))^T(\boldsymbol{x}_t - \boldsymbol{y}_{t+1}) \\ &\leq -\frac{\rho}{2}\|\boldsymbol{x}_t - \boldsymbol{y}_{t+1}\|^2 + \eta g_t^T(\boldsymbol{x}_t - \boldsymbol{y}_{t+1}) \leq -\frac{\rho}{2}\|\boldsymbol{x}_t - \boldsymbol{y}_{t+1}\|^2 + \eta\|\boldsymbol{x}_t - \boldsymbol{y}_{t+1}\| L \leq \frac{\eta^2 L^2}{2\rho} \end{aligned} \tag{2.24}$$

The above inequalities are obtained using the strong convexity of Φ and the update rule, Cauchy-Schwarz inequality, and the inequality $ax - bx^2 \leq \max_x ax - bx^2 = a^2/4b$ as in the last step in the proof of [53, Theorem 4.2], respectively. We have

$$\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\| \leq \sqrt{\tfrac{2}{\rho} D_\Phi(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})} \leq \sqrt{\tfrac{2}{\rho} D_\Phi(\boldsymbol{x}_t, \boldsymbol{y}_{t+1})} \overset{(2.24)}{\leq} \sqrt{2\eta^2 \tfrac{L^2}{2\rho^2}} \leq \tfrac{L\eta}{\rho}. \tag{2.25}$$

The first inequality is obtained using the strong convexity of Φ, and the second using the generalized Pythagorean inequality [88, Lemma 11.3]. We combine Eq. (2.25) and (2.23) to obtain

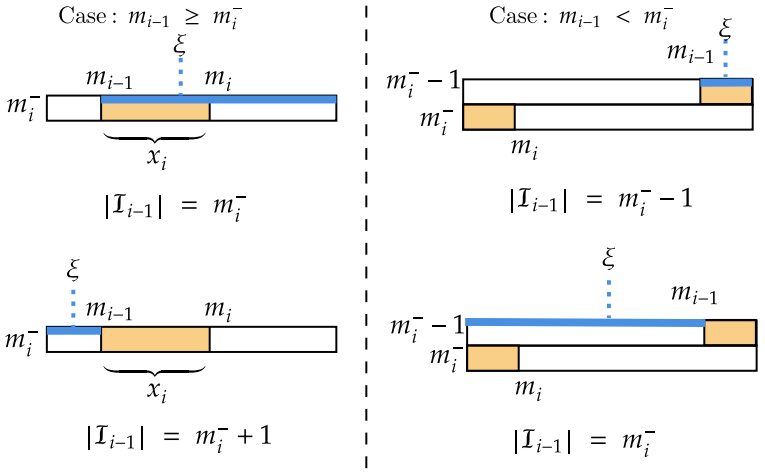


Figure 2.1: The support and distribution of the random variable $|\mathcal{I}_{i-1}|$ for $m_{i-1} \geq m_i^-$ (left) and $m_{i-1} < m_i^-$ (right). The blue shaded area represents the probability that $|\mathcal{I}_{i-1}|$ takes the value indicated below the corresponding image. This figure can be viewed as a subset of the rows in Figure 2.2.

$\mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] \leq \gamma_1\gamma_2 \|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\| \leq \gamma_1\gamma_2 \frac{L\eta}{\rho}$. The total update cost is $\sum_{t=1}^{T-1} \mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] \leq \gamma_1\gamma_2 \frac{L\eta}{\rho} T$. When OMD has a fixed learning rate $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$, we obtain $\sum_{t=1}^{T-1} \mathbb{E}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] = O\left(\sqrt{T}\right)$. The expected service cost is $\mathbb{E}\left[\sum_{t=1}^{T} f_t(\boldsymbol{z}_t)\right] = \sum_{t=1}^{T} f_t(\boldsymbol{x}_t) = O(T)$; the first equality is obtained from the linearity of the expectation operator and the function $f_{\boldsymbol{r}_t}$, and the second equality is obtained using the bound in Eq. (1.2) with $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$.

□

2.5 Proof of Theorem 2.2.14

Lemma 2.2 and Lemma 2.4 guarantee that Algorithm 2.3 used with an OMD algorithm satisfies the hypothesis of Theorem 2.1 and, hence, provides sublinear extended regret.

Lemma 2.2. *The random integral cache state $\boldsymbol{z}$ obtained by calling Algorithm 2.3 with fractional cache configuration input $\boldsymbol{x} \in \mathcal{X}$ satisfies $\boldsymbol{z} \in \mathcal{Z}$ and $\mathbb{E}_\xi[\boldsymbol{z}] = \boldsymbol{x}$.*

Proof.

We employ the shorthand notation $m_i = \sum_{j=1}^{i} x_j$ and $m_i^- = \lfloor m_i \rfloor$. The choice of ξ (see Figure 2.2) defines k different thresholds $\xi, \xi + 1, \ldots, \xi + k - 1$. For each threshold, we select the first item, whose accumulated mass exceeds the threshold. As $m_N = k$, we are guaranteed to exceeds all k thresholds, and as $x_i \leq 1$, we are guaranteed to select one item for each threshold. Therefore $\boldsymbol{z}$ belongs to $\mathcal{Z}$. From Algorithm 2.3 for any $i \in \mathcal{N}$ we have $\mathbb{P}(z_i = 1) = \mathbb{P}(\mathcal{I}_i \setminus \mathcal{I}_{i-1} = \{i\}) = \mathbb{P}\left(m_i \geq \xi + |\mathcal{I}_{i-1}|\right)$.

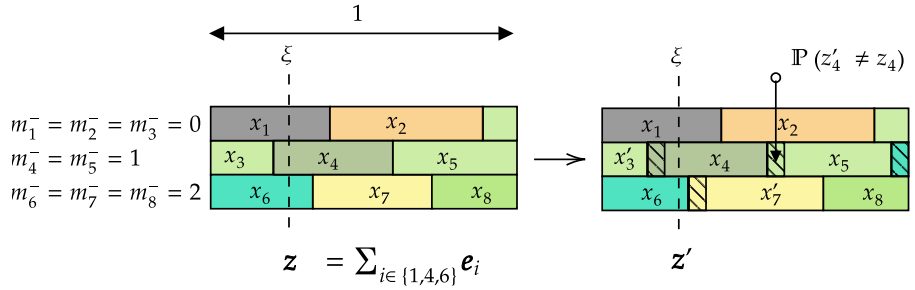


Figure 2.2: The random integral cache configuration obtained by calling ONLINE ROUNDING given the fractional cache state $\boldsymbol{x}$ and ξ (left). When ξ is kept fixed, the probability over the initial choice of ξ the random integral cache configuration rounded from a new fractional state $\boldsymbol{x}' = \boldsymbol{x} - \delta \boldsymbol{e}_3 + \delta \boldsymbol{e}_7$ is different is illustrated by the dashed areas (right).

See Figure 2.1 (left). If $m_{i-1} \geq m_i^-$, then $|\mathcal{I}_{i-1}| \in \{m_i^-, m_i^- + 1\}$ and

$$\begin{aligned}\mathbb{P}\left(m_i \geq \xi + |\mathcal{I}_{i-1}|\right) &= \mathbb{P}\left(m_i \geq \xi + |\mathcal{I}_{i-1}| \;\middle|\; |\mathcal{I}_{i-1}| = m_i^-\right) \mathbb{P}(|\mathcal{I}_{i-1}| = m_i^-) \\ &\quad + \mathbb{P}\left(m_i \geq \xi + |\mathcal{I}_{i-1}| \;\middle|\; |\mathcal{I}_{i-1}| = m_i^- + 1\right) \mathbb{P}(|\mathcal{I}_{i-1}| = m_i^- + 1) \qquad (2.26) \\ &= \frac{m_i - m_{i-1}}{(m_i^- + 1 - m_{i-1})} \cdot (m_i^- + 1 - m_{i-1}) + 0 \cdot (m_{i-1} - m_i^-) = x_i. \qquad (2.27)\end{aligned}$$

See Figure 2.1 (right). If $m_{i-1} < m_i^-$, then $|\mathcal{I}_{i-1}| \in \{m_i^- - 1, m_i^-\}$ and

$$\begin{aligned}&\mathbb{P}\left(m_i \geq \xi + |\mathcal{I}_{i-1}|\right) = \mathbb{P}\left(m_i \geq \xi + |\mathcal{I}_{i-1}| \;\middle|\; |\mathcal{I}_{i-1}| = m_i^- - 1\right) \mathbb{P}(|\mathcal{I}_{i-1}| = m_i^- - 1) \\ &+ \mathbb{P}\left(m_i \geq \xi + |\mathcal{I}_{i-1}| \;\middle|\; |\mathcal{I}_{i-1}| = m_i^-\right) \mathbb{P}(|\mathcal{I}_{i-1}| = m_i^-) \qquad (2.28) \\ &= 1 \cdot (m_i^- - m_{i-1}) + \frac{m_i - m_i^-}{m_{i-1} - m_i^- + 1} \cdot (m_{i-1} - m_i^- + 1) = m_i - m_{i-1} = x_i. \qquad (2.29)\end{aligned}$$

□

Lemma 2.3. *Consider a fractional cache configuration $\boldsymbol{x}'$ obtained by an elementary mass movement of δ from $u \in \mathcal{N}$ to $v \in \mathcal{N}$ for configuration $\boldsymbol{x} \in \mathcal{X}$, i.e., $\boldsymbol{x}' = \boldsymbol{x} - \delta \boldsymbol{e}_u + \delta \boldsymbol{e}_v$. Algorithm 2.3 outputs the random integral cache configurations $\boldsymbol{z}$, and $\boldsymbol{z}'$, given the input fractional cache states $\boldsymbol{x}$ and $\boldsymbol{x}'$, respectively. The random integral configurations satisfy $\mathbb{E}_\xi\left[\|\boldsymbol{z}' - \boldsymbol{z}\|_{1,\boldsymbol{w}'}\right] \leq 2kN \|\boldsymbol{w}'\|_\infty \delta$, where $\|\boldsymbol{x}\|_{1,\boldsymbol{w}'} \triangleq \sum_{i \in \mathcal{N}} |x_i| w'_i$, and note that $\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}) \leq \|\boldsymbol{z}_t - \boldsymbol{z}_{t+1}\|_{1,\boldsymbol{w}'}$.*

Proof.

The probability that a random integral cache state $\boldsymbol{z}'$ is changed w.r.t $\boldsymbol{z}$ can be upper bounded as $\mathbb{P}(\boldsymbol{z} \neq \boldsymbol{z}') \leq \sum_{i \in \mathcal{N}} \mathbb{P}(z_i \neq z'_i)$. Consider w.l.g that $u < v$, then $\mathbb{P}(z_i \neq z'_i) = 0$ for $i \in \mathcal{N} \setminus \{u, u+1, \ldots, v\}$, and $\mathbb{P}(z_i \neq z'_i) = \delta$ for $i \in \{u, u+1, \ldots, v\}$. We obtain $\mathbb{P}(\boldsymbol{z} \neq \boldsymbol{z}') \leq \delta(v - u + 1)$ (e.g., see Figure 2.2). More generally, for $u \neq v$ we have $\mathbb{P}(\boldsymbol{z} \neq \boldsymbol{z}') \leq \delta(|v - u| + 1)$. We conclude that

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

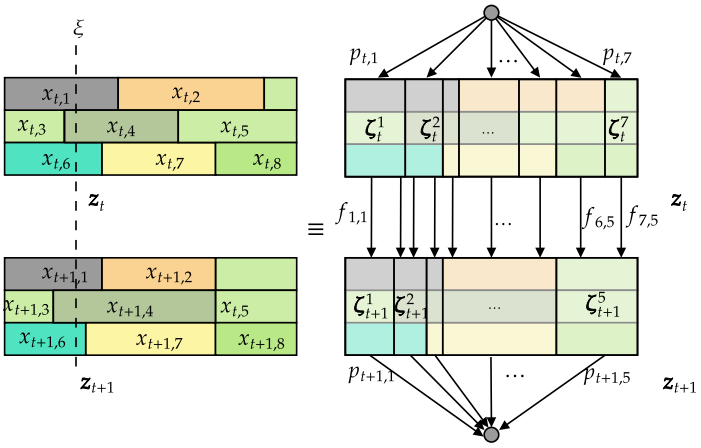


Figure 2.3: Coupling induced by Online Rounding Algorithm 2.3 when ξ is fixed. The flow $f_{i,j}$ is the joint probability $\mathbb{P}(\boldsymbol{z}_{t+1} = \zeta_{t+1,j}, \boldsymbol{z}_t = \zeta_{t,i})$, so that the next state is $\zeta_{t+1,j}$ and the previously selected state is $\zeta_{t,i}$.

□

Lemma 2.4. *The expected movement cost of the random integral cache states generate by Algorithm 2.3 is* $\mathbb{E}_\xi \left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] = O\left(\|\boldsymbol{x}_t - \boldsymbol{x}_{t+1}\|_1\right)$*, when* ξ *is sampled once u.a.r. from the interval* $[0,1]$ *and then fixed for* $t \in [T]$*.*

Proof.

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

from Lemma 2.3. Then the total expected movement cost is:

$$\mathbb{E}_{\xi}\left[\mathrm{UC}_{\boldsymbol{r}_t}(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})\right] \leq \mathbb{E}_{\xi}[\|\boldsymbol{z}_{t+1} - \boldsymbol{z}_t\|_{1,\boldsymbol{w}'}] = \mathbb{E}_{\xi}\left[\left\|\sum_{l=0}^{|\nu|-1}\left(\boldsymbol{z}^{l+1} - \boldsymbol{z}^{l}\right)\right\|_{1,\boldsymbol{w}'}\right] \tag{2.30}$$

$$\leq \sum_{l=0}^{|\nu|-1} \mathbb{E}_{\xi}\left[\left\|\boldsymbol{z}^{l+1} - \boldsymbol{z}^{l}\right\|_{1,\boldsymbol{w}'}\right] \leq \textstyle\sum_{i\in\mathcal{N}\setminus\mathcal{J}}\sum_{j\in\mathcal{J}} 2k\,\|\boldsymbol{w}'\|_{\infty} N\delta_{i,j} \tag{2.31}$$

$$= 2kN\,\|\boldsymbol{w}'\|_{\infty} \sum_{i\in\mathcal{N}\setminus\mathcal{J}} \phi_i \leq kN\,\|\boldsymbol{w}'\|_{\infty}\,\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_1\,. \tag{2.32}$$

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

□

3 Tabular Greedy Algorithm

We present here TABULARGREEDY [78], a polynomial time algorithm for solving Problem (2.38) within a $(1 - 1/e)$-approximation. This differs from the (more common) continuous greedy algorithm [66] in that it operates in the discrete rather than continuous domain, even though both algorithms involve randomization. It serves as the basis for the online algorithm by Streeter et al. [78]. A key departure from continuous greedy is the use of randomization via colors assigned to each slot, which also manifest in the online version of the algorithm.

For any set $\tilde{A} \subseteq \mathcal{S} \times C \times [M]$ and vector $\boldsymbol{m} = [m_s]_{s\in\mathcal{S}} \in [M]^{|\mathcal{S}|}$, let:

$$\texttt{sample}_{\boldsymbol{m}}(\tilde{A}) = \left\{(s, i) \in \mathcal{S} \times C : (s, i, m_s) \in \tilde{A}\right\}. \tag{3.33}$$

Intuitively, the colored allocation $\tilde{A} \subseteq \mathcal{S}\times C\times[M]$ is an allocation of items to slots, additionally parameterized by colors. Given the color vector $\boldsymbol{m}$, assigning colors to slots, $\texttt{sample}_{\boldsymbol{m}}$ acts as a selector, producing an (uncolored) allocation $A \subseteq \mathcal{S} \times C$. Let $F(\tilde{A})$ be the expected value of $f(\texttt{sample}_{\boldsymbol{m}}(\tilde{A}))$ when each color m_s is selected independently and u.a.r. from $[M]$; formally,

$$F(\tilde{A}) = \mathbb{E}\left[f(\texttt{sample}_{\boldsymbol{m}}(\tilde{A}))\right] = \frac{1}{M^{|\mathcal{S}|}} \sum_{\boldsymbol{m}'\in[M]^{|S|}} f(\texttt{sample}_{\boldsymbol{m}'}(\tilde{A})). \tag{3.34}$$

The procedure is summarized in Algorithm .1. TABULARGREEDY constructs a set of triplets $\tilde{A} \subseteq \mathcal{S}\times C\times[M]$ greedily; that is, starting from an empty set, it iterates over all colors and storage slots in an arbitrary order, and places items to (colored) slots by greedily maximizing the extended function F. Formally, in the iteration over color $m \in [M]$ and slot $s \in S$, the algorithm extends $\tilde{A}$ via:

$$i_{s,m} = \arg\max_{i\in C}\{F(\tilde{A} + (s, i, m))\} \tag{3.35a}$$

$$\tilde{A} \leftarrow \tilde{A} + (s, i_{s,m}, m), \tag{3.35b}$$

Algorithm .1 TabularGreedy

Require: Integer M, set $\mathcal{C}$, function f. set $\tilde{A} \leftarrow \emptyset$.

for $m \leftarrow 1, 2, \ldots, M$ **do**
  for each $s \in \mathcal{S}$ **do**
    Find $i_{s,m}$ s.t. $F(\tilde{A} + (s, i_{s,m}, m)) \geq \max_{i \in \mathcal{C}} F(\tilde{A} + (s, i, m)) - \epsilon_{s,m}$
    $\tilde{A} \leftarrow \tilde{A} + (s, i_{s,m}, m)$
  end for
end for
for each $s \in \mathcal{S}$ **do** independently choose m_s uniformly at random from M
end for
return $\texttt{sample}_{\boldsymbol{m}}(\tilde{A})$

where, for legibility, we use $\tilde{A} + o$ to indicate $\tilde{A} \cup \{o\}$. Finally, the algorithm returns allocation $S = \texttt{sample}_{\boldsymbol{m}}(\tilde{A})$, where colors in vector $\boldsymbol{m}$ are selected u.a.r. from $[M]$.

Note that the same node would not cache the same content multiple times. Indeed, as shown in Eq. (3.35), the algorithm extends the set of triplets $\tilde{A}$, in the iteration over color m and slot s, by the maximizer (s, i, m) of $F(\tilde{A} + (s, i, m))$. To be more specific, in the same node, if it is possible that an item i is repeatedly cached, then one of the triplets (s, i, m) could not be the maximizer in some iteration, since a different item i' could achieve greater or equal cache gain than i. This internally avoids duplicate cache in one nodes.

The following theorem characterizes the approximation guarantee of the solution produced by TabularGreedy; the theorem allows for the case where the greedy item selection $i_{s,m}$ by (3.35a) is inexact, and the selected item is suboptimal by an offest $\epsilon_{s,m}$. Let $\tilde{A}^{-}_{s,m}$ equal $\tilde{A}$ just before $(s, i_{s,m}, m)$ is added at iteration m, s, i.e., $\tilde{A}^{-}_{s,m} = \{(s', i_{s',m'}, m') : s' \in \mathcal{S}, m' < m\} \cup \{(s', i_{s',m}, m) : s' \prec s\}$:

Theorem 3.1. *(Theorem 13 in [295]) Suppose f is monotone submodular. Consider an arbitrary ordering of colors $m \in [M]$ and slots $s \in \mathcal{S}$, and consider the sequence of sets constructed by* TabularGreedy *when $i_{s,m} \in \mathcal{C}$ in Eq.* (3.35a) *is such that:*

$$F(\tilde{A} + (s, i_{s,m}, m)) \geq \max_{i \in \mathcal{C}} F(\tilde{A}^{-}_{s,m} + (s, i, m)) - \epsilon_{s,m}, \tag{3.36}$$

for some $\epsilon_{s,m} \geq 0$. Then, the final set in the sequence $\tilde{A} \subseteq \mathcal{S} \times \mathcal{C} \times [M]$ satisfies:

$$F(\tilde{A}) \geq \beta(|\mathcal{S}|, M) \cdot \max_{A \in \mathcal{D}} f(A) - \sum_{s \in \mathcal{S}} \sum_{m=1}^{M} \epsilon_{s,m}, \tag{3.37}$$

where $\beta(|\mathcal{S}|, M) = 1 - (1 - \frac{1}{M})^M - \binom{|\mathcal{S}|}{2} M^{-1}$.

The importance of accounting for inexact greedy selection lies in the fact that expectation F is hard to compute exactly, and is typically approximated by sampling, i.e., via $\tilde{F}(\tilde{A}) = \frac{1}{L} \sum_{l=1}^{L} f(\texttt{sample}_{\boldsymbol{m}_l}(\tilde{S}))$, were $\boldsymbol{m}_l$ are sampled u.a.r. The theorem implies that by selecting a large enough M (in particular, larger than $\Theta(|\mathcal{S}|^2)$, and L, the approximation guarantee can get arbitrarily close to $1 - 1/e$.

4 Formal Guarantees of Hedge Selector Algorithm 2.4

Recall that, at each time t, the hedge selector $\mathcal{E}$ defined by Algorithm 2.4 picks an action i^t from finite set C and subsequently observes an adversarially selected vector of rewards $\boldsymbol{\ell}^t = [\ell_i^t]_{i\in C} \in \mathbb{R}_+^{|C|}$, where ℓ_i^t is the reward for choosing action $i \in C$ at round t. The selector then accrues reward $\ell_{i^t}^t$, i.e., the reward associated with the action i^t it selected previously. We note that the hedge selector operates in the full-information (rather than the classic bandit) setting: all action rewards in C are observed. The regret R_T of hedge selector $\mathcal{E}$ is:

$$R_T = \sum_{t=1}^{T} \ell_{i^*}^t - \mathbb{E}[\sum_{t=1}^{T} \ell_{i^t}^t], \tag{4.38}$$

where i^* is the best selection in hindsight, i.e., $i^* = \arg\max_{i\in C} \sum_{t=1}^{T} \ell_i^t$.

The following lemma is classic; we note that it follows immediately from Theorem 1.5 in Hazan [20]. We reprove it here for completeness.

Lemma 4.1 ([20, 86]). *Assume that every action's reward is bounded by $\bar{L} \in \mathbb{R}_+$. Let $\epsilon = \frac{1}{\bar{L}}\sqrt{\frac{\log|C|}{T}}$. Then, for all $T \geq \log|C|$, the regret of hedge selector $\mathcal{E}$ defined by* (2.40) *and* (2.41) *is s.t.:*

$$R_T \leq 2\bar{L}\sqrt{T\log|C|}. \tag{4.39}$$

Proof.

Observe that for $T \geq \log|C|$, we have that

$$\epsilon = \frac{1}{\bar{L}}\sqrt{\frac{\log|C|}{T}} \in [0, \frac{1}{\bar{L}}]. \tag{4.40}$$

Let $\Phi^t = \sum_{i\in C} W_i^t$, $\boldsymbol{p}^t = [p_i^t]_{i\in C} \in \mathbb{R}^{|C|}$.

$$\begin{aligned}
\Phi^{t+1} &= \sum_{i\in C} W_i^{t+1} \overset{(2.41)}{=} \sum_{i\in C} W_i^t e^{\epsilon\ell_i^t}, \\
&\overset{(4.40)}{\leq} \sum_{i\in C} W_i^t (1 + \epsilon\ell_i^t + \epsilon^2\ell_i^t)^2), \qquad e^x \leq 1 + x + x^2, \forall x \in [0,1] \\
&= \Phi^t \sum_{i\in C} p_i^t (1 + \epsilon\ell_i^t + \epsilon^2(\ell_i^t)^2), \qquad p_i^t = \frac{W_i^t}{\sum_{j\in C} W_j^t} = \frac{W_i^t}{\Phi^t} \\
&= \Phi^t (1 + \epsilon\langle \boldsymbol{p}^t, \boldsymbol{\ell}^t\rangle + \epsilon^2\langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2\rangle), \qquad (\boldsymbol{\ell}^t)^2 = [(\ell_i^t)^2]_{i\in C} \\
&\leq \Phi^t e^{\epsilon\langle \boldsymbol{p}^t, \boldsymbol{\ell}^t\rangle + \epsilon^2\langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2\rangle}, \qquad 1 + x \leq e^x.
\end{aligned} \tag{4.41}$$

So, at round T,

$$\begin{aligned} e^{\epsilon \sum_{t=1}^{T} \ell_{i^*}^t} = W_{i^*}^T \le \Phi^T \le \Phi^0 e^{\epsilon \sum_{t=1}^{T} \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle + \epsilon^2 \sum_{t=1}^{T} \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle}, \\ \epsilon \sum_{t=1}^{T} \ell_{i^*}^t \le \ln |C| + \epsilon \sum_{t=1}^{T} \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle + \epsilon^2 \sum_{t=1}^{T} \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle, \quad \text{take logarithm} \\ \sum_{t=1}^{T} \ell_{i^*}^t - \sum_{t=1}^{T} \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle \le \frac{\ln |C|}{\epsilon} + \epsilon \sum_{t=1}^{T} \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle, \qquad \text{divided by } \epsilon \text{ and rearrange.} \end{aligned} \tag{4.42}$$

Thus,

$$\begin{aligned} R_T &= \sum_{t=1}^{T} \ell_{i^*}^t - \mathbb{E}[\sum_{t=1}^{T} \ell_{i^t}^t] = \sum_{t=1}^{T} \ell_{i^*}^t - \sum_{t=1}^{T} \langle \boldsymbol{p}^t, \boldsymbol{\ell}^t \rangle \\ &\overset{(4.42)}{\le} \frac{\ln |C|}{\epsilon} + \epsilon \sum_{t=1}^{T} \langle \boldsymbol{p}^t, (\boldsymbol{\ell}^t)^2 \rangle \le \frac{\ln |C|}{\epsilon} + \epsilon \bar{L}^2 T, \quad \boldsymbol{p}^t \text{ is probability, and } \ell_i^t \in [0, \bar{L}]. \end{aligned} \tag{4.43}$$

The latter inequality yields $R_T \le 2\bar{L}\sqrt{T \log |C|}$ as $\epsilon = \frac{1}{\bar{L}}\sqrt{\frac{\ln |C|}{T}}$.

□

5 Proof of Theorem 2.3.2

We first introduce some auxiliary lemmas to describe the properties of reward vectors. For any set $\tilde{A} \subseteq \mathcal{S} \times C \times [M]$ and given color $\boldsymbol{m} = [m_s]_{s \in \mathcal{S}}$ at round t, let $\overline{F}^t(\tilde{A}, \boldsymbol{m}) = f^t(\texttt{sample}_{\boldsymbol{m}}(\tilde{A}))$. Let $\boldsymbol{m}^t$ be the vector of colors at the beginning of round t. Let also

$$i_{s,m}^t = \begin{cases} \text{the item returned by } \mathcal{E}_{s,m}.\text{arm}() \text{ the last time it was called (including } t\text{), or} \\ \text{an arbitrary item if the selector has never been called.} \end{cases} \tag{5.44}$$

Note that, at time t, the selector $\mathcal{E}_{s,m}$.arm() is indeed called for all slots s on a path of a request $r \in \mathcal{R}^t$ when $m = m_s^t$. Let $\tilde{A}^t \subseteq \mathcal{S} \times C \times [M]$ be the triplet set constructed by Algorithm 2.5 at round t, i.e., the set comprising triplets

$$(s, i_{s,m}^t, m) \text{ for all } s \in \mathcal{S} \text{ and } m \in [M].$$

Note that such triplets are updated at all slots in paths of requests in timeslot t; all other triplets remain unaltered. We impose an ordering over all such triplets, defined by an ordering over colors first and slots second (the latter given by Eq. (2.32)). Under this ordering, similar to $\tilde{A}_{s,m}^-$ defined before Theorem 3.1, let $\tilde{A}_{s,m}^{t-}$ equal $\tilde{A}^t$ just "before" $(s, i_{s,m}^t, m)$ is added at round t; this addition is conceptual, presuming these triplets are "added" one-by-one under the aforementioned ordering to construct $\tilde{A}^t$. Under this convention,

$$\tilde{A}_{s,m}^{t-} = \{(s', i_{s',m}^t, m) : s' \in \mathcal{S}, m' < m\} \cup \{(s', i_{(s',m)}^t, m) : s' \prec s\}.$$

Lemma 5.1. *At round t, for all storage slot $s \in \bigcup_{(i,p)\in\mathcal{R}^t} \mathcal{S}_p$, the reward vector computed by Eq. (2.46), i.e., the vector $\boldsymbol{\ell}^r(s, m_s^t) \in \mathbb{R}_+^{|C|}$ with coordinates:*

$$\ell_{i'}^r(s, m_s^t) = \begin{cases} \max_{(v',j')\in\mathcal{S}_{\preceq s}+s} w_{v'}^p, & i' = i \\ \max_{(v',j')\in\mathcal{S}_{\preceq s}} w_{v'}^p, & o.w \end{cases}$$

where $r = (i,p) \in \mathcal{R}^t$ and $\mathcal{S}_{\preceq s} = \{s' \in \mathcal{S}_{i,p} : m_{s'} < m_s \text{ or } m_{s'} = m_s, s' \prec s\}$, satisfies the following property:

$$\sum_{r=(i,p)\in\mathcal{R}^t : s\in\mathcal{S}_p} \ell_{i'}^r(s, m_s^t) = \overline{F}^t(\tilde{A}^{t-}_{s,m_s^t} + (s, i', m_s^t), \boldsymbol{m}^t), \text{ for all } i' \in C. \tag{5.45}$$

Proof.

From the definition Eq. (2.36) of f_r, we have that for $r = (i,p) \in \mathcal{R}^t$:

$$\begin{aligned} f_r(A) &= \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(A \cap \left\{ \bigcup_{k'\in[k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} \neq \emptyset\right) \\ &= \sum_{k=\min\{k' :\ \exists j\in\mathcal{S}_{p_{k'}} \text{ s.t. } ((p_{k'},j),i)\in A\}}^{|p|-1} w_{p_{k+1}p_k} = \max_{v\in p:\exists j \text{ s.t. } ((v,j),i)\in A} w_v^p, \end{aligned} \tag{5.46}$$

where w_v^p is the cumulative upstream cost defined in Eq. (2.44). Then,

$$\begin{aligned} \overline{F}^t(\tilde{A}, \boldsymbol{m}) = f^t(\texttt{sample}_{\boldsymbol{m}}(\tilde{A})) &\overset{(3.33)}{=} \sum_{r\in\mathcal{R}^t} f_r(\{(s,i) \in \mathcal{S}\times C : (s,i,m_s) \in \tilde{A}\}) \\ &\overset{(5.46)}{=} \sum_{(i,p)\in\mathcal{R}^t} \sum_{k=\min\{k' :\ \exists j\in\mathcal{S}_{p_{k'}} \text{ s.t. } ((p_{k'},j),i,m_s)\in\tilde{A}\}}^{|p|-1} w_{p_{k+1}p_k} = \sum_{(i,p)\in\mathcal{R}^t} \max_{v\in p:\exists j \text{ s.t. } ((v,j),i,m_{(v,j)})\in\tilde{A}} w_v^p. \end{aligned} \tag{5.47}$$

Then, for $\tilde{A}' = \tilde{A}^{t-}_{s,m_s^t} + (s, i', m_s^t)$, we have

$$\begin{aligned} \overline{F}^t(\tilde{A}^{t-}_{s,m_s^t} + (s, i', m_s^t), \boldsymbol{m}^t) &\overset{(5.47)}{=} \sum_{(i,p)\in\mathcal{R}^t : s\in\mathcal{S}_p} \max_{((v',j'),i,m_{s'})\in\tilde{A}'} w_{v'}^p \\ = \begin{cases} \sum_{(i,p)\in\mathcal{R}^t : s\in\mathcal{S}_p} \max_{(v',j')\in\mathcal{S}_{\preceq s}+s} w_{v'}^p, & i' = i \\ \sum_{(i,p)\in\mathcal{R}^t : s\in\mathcal{S}_p} \max_{(v',j')\in\mathcal{S}_{\preceq s}} w_{v'}^p, & o.w \end{cases} &= \sum_{r=(i,p)\in\mathcal{R}^t : s\in\mathcal{S}_p} \ell_{i'}^r(s, m_s^t). \end{aligned} \tag{5.48}$$

The second equality holds because of the definition of accumulate weights by Eq. (2.44) and the fact that $v = \arg\min_v k_p(v) = \arg\max_v w_v^p$. The Third equality holds by the definitions of $\tilde{A}'$ and $\mathcal{S}_{\preceq s}$.

□

If the requests in $\mathcal{R}^t$ do not cross, it behaves same as the one request scenario. If requests do cross each other, the reward vector calculated by storage slot (v, j) is the summation of separate reward vectors deriving from each request $r^t \in \mathcal{R}^t$. Actually, it is equivalent to calculating the reward vector and calling operation feedback($\boldsymbol{\ell}^t(s, m_s)$) separately when each request arrives. We prove the above statement by the following lemma:

Lemma 5.2. *Calling feedback() with reward vector $\sum_{i=1}^{k} \boldsymbol{\ell}_i$ is equivalent to a sequence of k feedback calls, with reward vectors $\boldsymbol{\ell}_i$.*

Proof.

If we feedback a reward: $\sum_i \boldsymbol{\ell}_i$, the weight vector in it is: $\forall i \in C$, $W_i^{t+1} = W_i^t e^{\epsilon \sum_i \ell_i}$. If we feedback rewards $\boldsymbol{\ell}_i$ for all i separately, in the end, the weight vector in it is: $\forall i \in C$, $W_i^{t+1} = W_i^t \prod_i e^{\epsilon \ell_i}$. These two feedback scenario lead to same state in hedge selector.

□

Lemma 5.3. *At round t, given selected color $\boldsymbol{m}$, for $s \notin \bigcup_{(i,p)\in\mathcal{R}^t} \mathcal{S}_p$ or $m \neq m_s$, all $i', j' \in C$, $\overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m}) = \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, j', m), \boldsymbol{m})$.*

Proof.

When $s \notin \bigcup_{(i,p)\in\mathcal{R}^t} \mathcal{S}_p$ or $m \neq m_s$, according to Eq. (5.48), for all $i' \in C$,

$$\overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m}) = \sum_{(i,v)\in\mathcal{R}^t} \max_{((v',j'),i,m_{s'})\in\tilde{A}'} w^p_{v'} = \sum_{(i,v)\in\mathcal{R}^t} \max_{(v',j')\in\mathcal{S}_{\preceq s}} w^p_{v'}. \tag{5.49}$$

□ Finally, we can prove Theorem 2.3.2. *Proof.*

For all $s \in \mathcal{S}, m \in [M]$, we denote by $\mathcal{T}_{s,m}$ be the set of rounds where hedge selector $\mathcal{E}_{s,m}$ receives reward vector in our algorithm in T rounds, i.e., $\mathcal{T}_{s,m} = \{t \in [T] : v \in p^t, m = m^t_s\}$. For any $s \in \mathcal{S}, m \in [M]$, and $i' \in C$, since hedge selectors are no-regret algorithms, let $R^T_{s,m}$ be the regret of $\mathcal{E}_{s,m}$ during rounds $\mathcal{T}_{s,m}$. We denote $l^t_i(s, m)$ the i-th coordinate of total reward vector for hedge selector $\mathcal{E}_{s,m}$ at round t, i.e., $l^t_i(s, m) = \sum_{r=(i,p)\in\mathcal{R}^t : s\in\mathcal{S}_p} \ell^r_i(s, m)$. According to the definition of regret in Eq. (4.38), we have

$$R^T_{s,m} = \sum_{t\in\mathcal{T}_{s,m}} \ell^t_{i^*}(s, m) - \sum_{t\in\mathcal{T}_{s,m}} \ell^t_{i^t_{s,m}}(s, m) \geq \sum_{t\in\mathcal{T}_{s,m}} \ell^t_{i'}(s, m) - \sum_{t\in\mathcal{T}_{s,m}} \ell^t_{i^t_{s,m}}(s, m), \tag{5.50}$$

where i^* is the best selection in hindsight, i.e., $i^* = \arg\max_{i\in C} \sum_{t\in\mathcal{T}_{s,m}} \ell^t_i$. Then, by Lemma 5.1,

$$\sum_{t\in\mathcal{T}_{s,m}} \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i^t_{s,m}, m), \boldsymbol{m}) \geq \left(\sum_{t\in\mathcal{T}_{s,m}} \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m}) \right) - R^T_{s,m}. \tag{5.51}$$

For $t \in [T] \setminus \mathcal{T}_{s,m}$, all $s \in \mathcal{S}, m \in [M]$, $i' \in C$, by Lemma 5.3,

$$\overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i^t_{s,m}, m), \boldsymbol{m}) = \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m}). \tag{5.52}$$

Thus, for all $s \in \mathcal{S}, m \in [M]$, $i' \in C$,

$$\sum_{t=1}^{T} \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i^t_{s,m}, m), \boldsymbol{m}) \geq \left(\sum_{t=1}^{T} \overline{F}^t(\tilde{A}^{t-}_{s,m} + (s, i', m), \boldsymbol{m})\right) - R^T_{s,m}. \tag{5.53}$$

Taking the expectation of both sides over m, and over $i^t_{s,m}$ and choosing i to maximize the right hand side, we get

$$\sum_{t=1}^{T} F^t(\tilde{A}^{t-}_{s,m} + (s, i^t_{s,m}, m)\}) \geq \max_{i \in C} \left(\sum_{t=1}^{T} F^t(\tilde{A}^{t-}_{s,m} + (s, i', m))\right) - \epsilon_{s,m}, \tag{5.54}$$

where we define $F^t(\tilde{A}) = \mathbb{E}_{\boldsymbol{m}}[\mathbb{E}_{i^t_{s,m}}[f^t(\texttt{sample}_{\boldsymbol{m}}(\tilde{A}))]]$ and $\epsilon_{s,m} = \mathbb{E}[R^T_{s,m}]$.

We now define some additional notation. For any set $\boldsymbol{A}$ of vector in $\mathcal{S} \times C^T$, define

$$f(\boldsymbol{A}) = \sum_{t=1}^{T} f^t(\{(s, i^t) : (s, \boldsymbol{i}) \in \boldsymbol{A}\}), \tag{5.55}$$

where $\boldsymbol{i} = [i^t]_{t=1}^T$. Next, for any set $\tilde{\boldsymbol{A}} \subseteq \mathcal{S} \times C^T \times [M]$, and given $\boldsymbol{m} = [m_s]_{s \in \mathcal{S}}$, define $\texttt{sample}_{\boldsymbol{m}}(\tilde{\boldsymbol{A}}) = \{(s, \boldsymbol{i}) \in \{s\} \times C^T : (s, \boldsymbol{i}, m_s) \in \tilde{\boldsymbol{A}}\}$. Define $F(\tilde{\boldsymbol{A}}) = \mathbb{E}_{\boldsymbol{m}}[\mathbb{E}_{\boldsymbol{i}_{s,m}}[f(\texttt{sample}_{\boldsymbol{m}}(\tilde{\boldsymbol{A}}))]]$, where $\boldsymbol{i}_{s,m} = [i^t_{s,m}]_{t=1}^T \in C^T$. By linearity of expectation,

$$F(\tilde{\boldsymbol{A}}) = \sum_{t=1}^{T} F^t(\{(s, i^t, m) : (s, \boldsymbol{i}, m) \in \tilde{\boldsymbol{A}}\}). \tag{5.56}$$

Analogously to $\tilde{A}^{t-}_{s,m}$, define $\tilde{\boldsymbol{A}}^-_{s,m} = \{(s', \boldsymbol{i}_{s',m'}, m') : s' \in \mathcal{S}, m' < m\} \cup \{(s', \boldsymbol{i}_{s',m}, m) : s' \prec s\}$. Thus, for any $(s, \boldsymbol{i}, m) \in \mathcal{S} \times C^T \times [M]$, we have $F(\tilde{\boldsymbol{A}}^-_{s,m} + (s, \boldsymbol{i}, m)) = \sum_{t=1}^T F^t(\tilde{A}^{t-}_{s,m} + (s, i_t, m))$. Combining this with (5.54) , we get:

$$F(\tilde{\boldsymbol{A}}^-_{s,m} + (s, \boldsymbol{i}_{s,m}, m)) \geq \max_{i' \in C} \left(F(\tilde{\boldsymbol{A}}^-_{s,m} + (s, [i']^T, m))\right) - \epsilon_{s,m}. \tag{5.57}$$

Having proved (5.57), we can now use Theorem 3.1 to complete the proof. Define a new partition matroid over ground set $\{\mathcal{S} \times C^T\}$ with feasible solution $\mathcal{D} := \{\boldsymbol{A} \subset \mathcal{S} \times C^T : |\boldsymbol{A} \cap (\{s\} \times C^T)| = 1, \forall s \in \mathcal{S}\}$. Let $\tilde{\boldsymbol{A}} = \{(s, \boldsymbol{i}_{s,m}, m) : s \in \mathcal{S}, m \in [M]\}$. It is easy to verify that F^t is monotone submodular, and F is also monotone submodular by linearity. Thus, by Theorem 3.1,

$$F(\tilde{\boldsymbol{A}}) \geq \beta(M, |\mathcal{S}|) \cdot \max_{\boldsymbol{A} \in \mathcal{D}} \{f(\boldsymbol{A})\} - \sum_{s \in \mathcal{S}} \sum_{m=1}^{M} \epsilon_{s,m}. \tag{5.58}$$

By definition, $F(\tilde{\boldsymbol{A}}) = \mathbb{E}[\sum_{t=1}^{T} f^t(A^t)]$, and $\max_{\boldsymbol{A}\in\boldsymbol{\mathcal{D}}}\{f(\boldsymbol{A})\} \geq \max_{A\in\mathcal{D}}\{\sum_{t=1}^{T} f^t(A)\}$, we get

$$\mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t)\right] \geq \beta(|\mathcal{S}|, M) \cdot \max_{A\in\mathcal{D}}\left\{\sum_{t=1}^{T} f^t(A)\right\} - \sum_{s\in\mathcal{S}}\sum_{m=1}^{M} \epsilon_{s,m}. \tag{5.59}$$

According to Lemma 4.1, $\epsilon_{s,m} = \mathbb{E}[R_{s,m}^T] \leq 2\bar{R}\bar{L}\sqrt{|\mathcal{T}_{s,m}|\log|C|} \leq 2\bar{R}\bar{L}\sqrt{T\log|C|}$, then

$$\mathbb{E}\left[\sum_{t=1}^{T} f^t(A^t)\right] \geq \beta(|\mathcal{S}|, M) \cdot \max_{A\in\mathcal{D}}\left\{\sum_{t=1}^{T} f^t(A)\right\} - 2\bar{R}\bar{L}|\mathcal{S}|M\sqrt{T\log|C|}. \tag{5.60}$$

□

6 Proof of Lemma 2.3.5

Consider a cache network of two nodes u and v and a catalog of two files represented by the set $\{1, 2\}$. The cache node u has capacity 1 and the node v is a repository node containing the files 1 and 2. The hedge selector is initialized with a distribution of the possible states $\boldsymbol{p}^1 = (1/2, 1/2)$. According to Algorithm 2.4, the hedge selector adds $\epsilon = \Theta\left(\frac{1}{\sqrt{T}}\right)$ fraction to the component corresponding to the requested file, and reduces the same quantity from the other component. When the requested files sequence is $\{1, 2, 1, 2, \dots\}$, this gives the following distributions:

$$\{\boldsymbol{p}^1, \boldsymbol{p}^2, \boldsymbol{p}^3, \boldsymbol{p}^4, \dots\} = \{(1/2, 1/2), (1/2 + \epsilon, 1/2 - \epsilon), (1/2, 1/2), (1/2 + \epsilon, 1/2 - \epsilon), \dots\}.$$

The distribution $\boldsymbol{p}^1$, gives two integral states $(1, 0)$ w.p. $1/2$ and $(0, 1)$ w.p. $1/2$, and $\boldsymbol{p}^2$ can give two integral states $(1, 0)$ w.p $1/2 + \epsilon$ and $(0, 1)$ w.p. $1/2 - \epsilon$. The expected update cost experienced in expectation from $t = 1$ to $t = 2$ is:

$$\mathbb{E}\left[\mathrm{UC}(A^1, A^2)\right] = 1/2\,(1/2 - \epsilon) + 1/2\,(1/2 + \epsilon) = 1/2.$$

The decomposition of $\boldsymbol{p}^3$ is the same as $\boldsymbol{p}^1$. The update cost experienced in expectation from $t = 2$ to $t = 3$ is:

$$\mathbb{E}\left[\mathrm{UC}(A^2, A^3)\right] = (1/2 - \epsilon)\,1/2 + (1/2 + \epsilon)\,1/2 = 1/2.$$

The sequence repeats and the same costs are obtained. The total update cost is:

$$\mathbb{E}\left[\sum_{t=1}^{T} \mathrm{UC}(A^t, A^{t+1})\right] = \sum_{t=1}^{T} \mathbb{E}\left[\mathrm{UC}(A^t, A^{t+1})\right] = \frac{T}{2} \tag{6.61}$$

This is an update cost of $\Omega(T)$ paid in expectation.

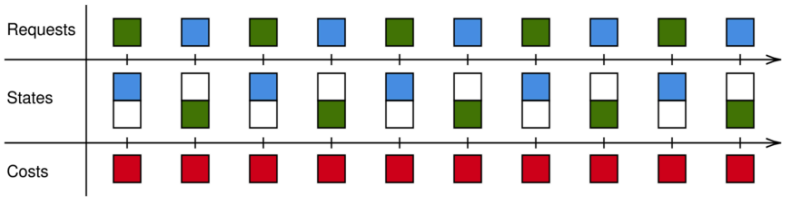


Figure 7.4: A simplified instance of GRD with linear regret. The cache can only store a single file, and at any given moment the adversary requests the file that is not stored in the cache. GRD updates its state greedily to store the requested file and oscillates between the two possible states. The optimal static strategy stores one of the files permanently incurring a total cost of $T/2$, while GRD incurs a total cost T.

7 Proof of Lemma 2.3.4

Assume a cache network formed of a designated server v and a cache with storage capacity $c_u \in \mathbb{N}$. The catalog C contains $2c_u$ items and, without lack of generality we assume that cache u initially contains the set of items $\{c_u + 1, \ldots, 2c_u\}$. Requests arrive only at node u, and we can identify a request with the requested item because there is only one possible path $(\{u, v\})$. The forwarding cost between u and v is $w \in \mathbb{R}_+$.

We consider request sequences with one request every Δ time units and one request per round ($\bar{R} = 1$) and we denote by i_t he item requested at time t. Moreover, for simplicity, we assume the time horizon T to be proportional to $2c_u$ ($T = m \times 2c_u$). The service cost without caching is wT.

The policy GRD maintains a vector $\boldsymbol{z}_t$ and updates it after every request as follows [21, Eq. (22)]:

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t e^{-\Delta\beta} + w\beta \boldsymbol{e}_{i_t}, \tag{7.62}$$

where $\boldsymbol{e}_{i_t} = \left[\mathbb{1}_{\{i=i_t\}}\right]_{i \in C}$ and $\boldsymbol{z}_0 = \boldsymbol{0}$. After the request time t GRD stores in the cache the c_u items in C that correspond to the largest c_u components of $\boldsymbol{z}_{t+1}$ at time t. Consider the request sequence

$$\{1, 2, \ldots, 2c_u, 1, 2, \ldots, 2c_u, \ldots, 1, 2, \ldots, 2c_u\}. \tag{7.63}$$

Under this request sequence, GRD behaves as LRU and simply stores at any time the c_u most recently requested items. In fact, item j is requested at time instants $hT + j$ for $h \in \{0, 1, \ldots, m-1\}$. At time $t = kT + i$, item j has been requested for the last time $(i - j) \bmod 2c_u$ time instants earlier. The corresponding component of the vector $\boldsymbol{z}_{t+1}$ has the value

$$(\boldsymbol{z}_{t+1})_j = e^{-\beta\Delta((i-j) \mod 2c_u)} \times \sum_{\substack{h \in \{0,1,\ldots,m-1\}, \\ hT+j \leq kT+i}} e^{-\beta\Delta h}.$$

The maximum value is achieved for $j = i$. The component becomes progressively smaller as j decreases from i to 1, because the first term in the product becomes smaller while the second terms

does not change. It keeps decreasing as j decreases from $2c_u$ to $i+1$, not only because the first term decreases, but also because the second term decreases as less addends are considered in the sum.

As GRD behaves as LRU, when a new request i_t arrives at node u, it is never found in the cache. GRD incurs then a total service cost wT and null caching gain. At the same time, the caching gain of any cache allocation A (with c_u different items stored at u) is $\frac{w}{2}T$, because it is able to serve half of the requests. Then, the α-regret of GRD is at least equal to $\alpha\frac{wT}{2}$. A simplified instance of GRD is shown in Figure 7.4 to provide some intuition of our proof.

8 Proof of Theorem 2.3.6

We begin by giving some intuition behind our approach. The hedge selector in Algorithm 2.4 effectively maintains a distribution $\boldsymbol{p}^t = \left[\frac{W_i^t}{\sum_{j\in C} W_j^t}\right]_{i\in C}$ for every round t. The randomized action i_t taken at time t by the selector always satisfies $\mathbb{E}[\boldsymbol{e}_{i^t}] = \boldsymbol{p}^t$ by definition, where $\boldsymbol{e}_i$ is the i-th basis vector. Consider for instance, that the rewards given to the hedge selector are always uniform. Since each action is equally important, then the distribution $\boldsymbol{p}^t$ is fixed and it is the uniform distribution. Thus, the hedge selector controlling item placements, will evict and fetch a new content with probability $1 - \frac{1}{|C|^2}$ at each time step. Clearly, since the distribution is fixed for every time step, then these movements are unnecessary. An optimal strategy in this scenario is to pick an action u.a.r at the start and stick with it.

We generalize this concept by taking minimal probabilistic jumps to a new state, only when it is necessary to maintain a change of distribution from $\boldsymbol{p}^t$ to $\boldsymbol{p}^{t+1}$. This concept is known in the literature as optimal transport or the earth mover distance [59]. The objective is to transport probability mass from a distribution to another, while minimizing the associated metric. In this scenario, it corresponds to a minimum-cost flow problem. We propose an iterative algorithm that builds the optimal flow (joint distribution). By building a feasible flow at time t from $\boldsymbol{p}^t$ to $\boldsymbol{p}^{t+1}$. Then, the algorithm takes elementary steps that generates a sequence of random variables whose marginal distribution is progressively closer to $\boldsymbol{p}^{t+1}$

The proof is split in multiple parts. We first introduce Lemma 8.1 that links the hedge selector update rule to online mirror descent [53]. This allows us to use convex optimization techniques to provide the proof of Lemma 8.2, that gives a family of coupling schemes with sublinear update cost.

In section 8.3, we dissect the hedge selector and bound the update cost of its two components, the *coupled_movement* and *elementary_δmovement* subroutines.

In some parts of the proof we switch to vector notation rather than the set notation for allocations. For any allocation $A \in \mathcal{S} \times C$ the corresponding allocation vector is denoted by $\boldsymbol{x} = [\mathbb{1}_{\{(s,i)\in A\}}]_{(s,i)\in\mathcal{S}\times C}$, and $\boldsymbol{x}_s = [\mathbb{1}_{\{(s,i)\in A\}}]_{(s,i):i\in C}$. The weighted l_1 norm is defined as:

$$||\boldsymbol{x}_s||_{1,\boldsymbol{w}'} := \sum_{i\in C} w'_{s,i}|x_{s,i}|. \tag{8.64}$$

With slight abuse of notation, we consider that for any $s \in \mathcal{S}$:

$$\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1}) := \sum_{i \in C} w'_i \max(0, x_i^{t+1} - x_i^t) \tag{8.65}$$

$$= \mathrm{UC}(A^t \cap \{(s,i) : i \in C\}, A^{t+1} \cap \{(s,i) : i \in C\}). \tag{8.66}$$

Also, note that $\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1}) \leq ||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}$.

8.1 Auxiliary Lemma

We start by introducing an auxiliary lemma.

Lemma 8.1. *$\mathcal{E}$.feedback($\boldsymbol{\ell}$) for the hedge selector in Algorithm 2.4 updates its internal weights W_t to W_{t+1} equivalently as $\nabla\Phi(\boldsymbol{W}^{t+1}) = \nabla\Phi(\boldsymbol{W}^t) + \epsilon\boldsymbol{\ell}^t$ where $\Phi(\boldsymbol{W}) = \sum_{i \in C} W_i \log(W_i)$ and $\epsilon \in \mathbb{R}_+$ is the step size.*

Proof.

We know that

$$\frac{\partial\Phi(\boldsymbol{W})}{\partial W_i} = 1 + \log(W_i) \tag{8.67}$$

From $\nabla\Phi(\boldsymbol{W}^{t+1}) = \nabla\Phi(\boldsymbol{W}^t) + \epsilon\boldsymbol{\ell}^t$ we have:

$$1 + \log(W_i^{t+1}) = 1 + \log(W_i^t) + \epsilon\ell_i^t, \tag{8.68}$$

then,

$$W_i^{t+1} = W_i^t e^{\epsilon\ell_i^t}, \tag{8.69}$$

which is exactly $\mathcal{E}$.feedback($\boldsymbol{\ell}$)

□

8.2 Family of Coupling Schemes with Sublinear Update Cost

The following Lemma provides a sufficient condition on the joint distribution of $(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})$ (the family of coupling schemes), that leads to sublinear update cost for DISTRIBUTEDTGONLINE.

Lemma 8.2. *Consider a hedge selector, shown in Algorithm 2.4, and a joint distribution of $(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})$ that satisfies for all $t \in [T-1]$:*

1. $\mathbb{E}[\boldsymbol{x}_s^t] = \boldsymbol{p}^t$ *and* $\mathbb{E}[\boldsymbol{x}_s^{t+1}] = \boldsymbol{p}^{t+1}$.
2. $\mathbb{E}[||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}] = O(||\boldsymbol{p}^{t+1} - \boldsymbol{p}^t||_{1,\boldsymbol{w}'})$.

This algorithm incurs an expected update cost of the same order of the update cost of probabilities. Selecting $\epsilon = \Theta(\frac{1}{\sqrt{T}})$, $K = \Omega(T)$, gives a $O(\bar{R}\bar{L}\sqrt{T})$ expected update cost.

Proof.

We know $\Phi(\boldsymbol{W}) = \sum_{i\in C} W_i \log(W_i)$ is 1-strong convex w.r.t. $||\cdot||_1$ on the simplex $\triangle_{|C|} = \{\boldsymbol{p} \in \mathbb{R}_+^{|C|} : \sum_{i\in C} p_i = 1\}$ [53]. Thus, $\Phi(x) - \Phi(y) \le \nabla\Phi(x)^\top(x-y) - \frac{1}{2}||x-y||_1^2$.

Then, we will prove the Lemma 8.2. Recall that, as shown in Algorithm 2.5, at every K times, each storage slot will choose a color m_s^t uniformly at random from $[M]$. At every rounds, the corresponding hedge selector $\mathcal{E}_{s,m_s^t}$ will be fed a reward vector $\boldsymbol{\ell}^t(s, m_s^t)$, and call $\mathcal{E}$.feedback($\boldsymbol{\ell}$) to update its weight vector. In the following proof, we assume that at round t and $t+1$, color m_s^t and m_s^{t+1} are chosen. The weight vector $\boldsymbol{W}^t$ is the weight vector maintained by hedge selector $\mathcal{E}_{s,m_s^t}$. The probability $\boldsymbol{p}^t$ is the normalized $\boldsymbol{W}^t$. With the assumptions in Lemma. 8.2, there exist constants $\alpha, \beta, \gamma > 0$, such that:

$$\mathbb{E}[\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})] \le \mathbb{E}[||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}] \le \alpha||\boldsymbol{p}^{t+1} - \boldsymbol{p}^t||_{1,\boldsymbol{w}'} \le \beta||\boldsymbol{p}^{t+1} - \boldsymbol{p}^t|| \le \gamma||\boldsymbol{W}^{t+1} - \boldsymbol{W}^t||, \quad (8.70)$$

where $||\cdot||$ here is $l1$ norm, the second to last inequality holds because norms can bound each other, and the last inequality holds because the inexpensive property of projection. For round t without color update:

$$\begin{aligned}
&\mathbb{E}[||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}] \le \gamma||\boldsymbol{W}^{t+1} - \boldsymbol{W}^t|| \\
\le&\gamma\sqrt{2(\Phi(\boldsymbol{W}^t) - \Phi(\boldsymbol{W}^{t+1}) + \nabla\Phi(\boldsymbol{W}^{t+1})^\top(\boldsymbol{W}^{t+1} - \boldsymbol{W}^t))}, && \text{1-strong convexity} \\
\le&\gamma\sqrt{2}\sqrt{\Phi(\boldsymbol{W}^t) - \Phi(\boldsymbol{W}^{t+1}) - \nabla\Phi(\boldsymbol{W}^t)^\top(\boldsymbol{W}^t - \boldsymbol{W}^{t+1}) + (\nabla\Phi(\boldsymbol{W}^{t+1}) - \nabla\Phi(\boldsymbol{W}^t))^\top(\boldsymbol{W}^{t+1} - \boldsymbol{W}^t)}, \\
\le&\gamma\sqrt{2}\sqrt{-\frac{1}{2}||\boldsymbol{W}^t - \boldsymbol{W}^{t+1}||^2 + (\nabla\Phi(\boldsymbol{W}^{t+1}) - \nabla\Phi(\boldsymbol{W}^t))^\top(\boldsymbol{W}^{t+1} - \boldsymbol{W}^t)}, && \text{1-strong convexity} \\
\le&\gamma\sqrt{2}\sqrt{-\frac{1}{2}||\boldsymbol{W}^t - \boldsymbol{W}^{t+1}||^2 + \epsilon\bar{R}(\boldsymbol{\ell}^t)^\top(\boldsymbol{W}^{t+1} - \boldsymbol{W}^t)}, && \text{Lemma 8.1} \\
\le&\gamma\sqrt{2}\sqrt{-\frac{1}{2}||\boldsymbol{W}^t - \boldsymbol{W}^{t+1}||^2 + \epsilon\bar{R}||\boldsymbol{\ell}^t||_\infty \cdot ||\boldsymbol{W}^{t+1} - \boldsymbol{W}^t||}, && \text{Cauchy–Schwarz inequality} \\
\le&\gamma\sqrt{2}\sqrt{\frac{(\epsilon\bar{R}||\boldsymbol{\ell}^t||_\infty)^2}{2}}, && az - bz^2 \le \frac{a^2}{4b}, a, b > 0 \\
=&\gamma\epsilon\bar{R}||\boldsymbol{\ell}^t||_\infty.
\end{aligned}$$

For round t with color update, the cache update cost is bounded by the most expensive cache update, i.e.,:

$$\mathbb{E}[||\boldsymbol{x}_s^{t+1} - \boldsymbol{x}_s^t||_{1,\boldsymbol{w}'}] \le ||\boldsymbol{w}'||_\infty. \quad (8.71)$$

The total update cost experienced for the hedge selector associated with slot s:

$$\sum_{t=1}^{T-1} \mathbb{E}[\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})] \le \sum_{t=1}^{T-1} \gamma\frac{\epsilon\bar{R}||\boldsymbol{\ell}^t||_\infty}{\rho} + \sum_{t=K,2K,\ldots} ||\boldsymbol{w}'||_\infty \le \gamma\epsilon\bar{R}\bar{L}T + \frac{T}{K}||w'||_\infty, \quad (8.72)$$

where $\bar{L} = \max_{(i,p)\in\mathcal{R}}\{\sum_{k=1}^{|p|-1} w_{p_{k+1}p_k}\} \ge \max_{t\le T}\{||\boldsymbol{\ell}^t||_\infty\}$. Assume that $K = \Omega(\sqrt{T})$, then $\exists c' > 0$,

$$\sum_{t=1}^{T-1} \mathbb{E}[\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})] \le \gamma\epsilon R\bar{L}T + \frac{||\boldsymbol{w}||'_\infty}{c'}\sqrt{T}. \quad (8.73)$$

In order to keep the hedge selectors regret sublinear $O(\sqrt{T})$ with $\epsilon = \Theta(\frac{1}{\sqrt{T}})$. The update cost for all the slots:

$$\sum_{t=1}^{T-1} \sum_{s \in \mathcal{S}} \mathbb{E}[\mathrm{UC}(\boldsymbol{x}_s^t, \boldsymbol{x}_s^{t+1})] \leq |\mathcal{S}| \gamma R \bar{L} \sqrt{T} + |\mathcal{S}| \frac{||\boldsymbol{w}||'_\infty}{c'} \sqrt{T}. \tag{8.74}$$

□

8.3 Dissection of the Coupled Hedge Selector

Assume at round t, storage slot s, item allocation $\boldsymbol{x}_s^t$ with $\mathbb{E}[\boldsymbol{x}_s^t] = \boldsymbol{p}^t$ has a particular allocation $\boldsymbol{x}_s^t$. We introduce the coupling scheme modification to the hedge selector in Algorithm2.4 given in Algorithm 2.6, which could produce $\boldsymbol{x}_s^{t+1}$ satisfying conditions in Lemma. 8.2. We first provide expected update cost of an elementary_δmovement subroutine call.

Lemma 8.3. *The elementary_δmovement subroutine outputs a random integral cache configuration X' with $\mathbb{E}[\boldsymbol{e}_{X'}] = \boldsymbol{p} - \delta \boldsymbol{e}_i + \delta \boldsymbol{e}_j$. If its input is sampled from a random variable X with $\mathbb{E}[\boldsymbol{e}_X] = \boldsymbol{p}$, then:*

$$\mathbb{E}\left[||\boldsymbol{e}_X - \boldsymbol{e}_{X'}||_{1,\boldsymbol{w}'}\right] = \delta(w'_{s,j} + w'_{s,i}). \tag{8.75}$$

Proof.

First part. Showing that $\mathbb{E}\left[\boldsymbol{e}_{X'}\right] = \boldsymbol{p} - \delta \boldsymbol{e}_i + \delta \boldsymbol{e}_j$.

$$\mathbb{E}\left[\boldsymbol{e}_{X'}\right] = \sum_{l \in C \setminus \{i\}} p_l \mathbb{E}\left[\boldsymbol{e}_{X'} | X = l\right] + p_i \mathbb{E}\left[\boldsymbol{e}_{X'} | X = i\right] \tag{8.76}$$

$$= \sum_{l \in C \setminus \{i\}} p_l \boldsymbol{e}_l + p_i \mathbb{E}\left[\boldsymbol{e}_{X'} | X = i\right], \qquad \text{Line 26, Algorithm 2.6} \tag{8.77}$$

$$= \sum_{l \in C \setminus \{i\}} p_l \boldsymbol{e}_l + p_i \left(\frac{p_i - \delta}{p_i} \boldsymbol{e}_i + \frac{\delta}{p_i} \boldsymbol{e}_j\right), \qquad \text{Line 23, Algorithm 2.6} \tag{8.78}$$

$$= \sum_{l \in C} p_l \boldsymbol{e}_l - \delta \boldsymbol{e}_i + \delta \boldsymbol{e}_j = \boldsymbol{p} - \delta \boldsymbol{e}_i + \delta \boldsymbol{e}_j. \tag{8.79}$$

Second part. The only movement that can be caused by running the subroutine is at line 23, given that $X = i$ with probability p_i, we replace this value by j with probability $\frac{\delta}{p_i}$. Hence, the expected update cost is given by:

$$\mathbb{E}[||\boldsymbol{e}_X - \boldsymbol{e}_{X'}||_{1,\boldsymbol{w}'}] = \frac{p_i \delta}{p_i}(w'_{s,j} + w'_{s,i}) = \delta(w'_{s,j} + w'_{s,i}). \tag{8.80}$$

□

We now introduce lemma 8.4 providing the expected update cost of a coupled_movement subroutine call.

Lemma 8.4. *If the input to coupled_movement subroutine in Algorithm2.6 is i_s^t with $\mathbb{E}[\boldsymbol{e}_{i_s^t}] = \boldsymbol{p}^t$, then it outputs an item i_s^{t+1}, where $\mathbb{E}[\boldsymbol{e}_{i_s^{t+1}}] = \boldsymbol{p}^{t+1}$, and $\mathbb{E}_{i_s^t}[\mathbb{E}_{i_s^{t+1}|i_s^t}[||\boldsymbol{e}_{i_s^t} - \boldsymbol{e}_{i_s^{t+1}}||_{1,\boldsymbol{w}'}]] = ||\boldsymbol{p}^t - \boldsymbol{p}^{t+1}||_{1,\boldsymbol{w}'}$.*

Proof.

The distribution over the catalog changes from a fractional state $\boldsymbol{p}_t \in \Delta_{|C|}$ to $\boldsymbol{p}^{t+1} \in \Delta_{|C|}$. The set $I = \left\{i \in C : x_{t+1,i} - x_{t,i} > 0\right\}$ in line 1 of Algorithm 2.6 is the set of components that have a fractional increase, then we get:

$$\boldsymbol{p}^{t+1} = \boldsymbol{p}^t + \sum_{j \in I} m_j e_j - \sum_{i \in C \setminus I} m_i e_i, \tag{8.81}$$

where $m_i, i \in C$ is the absolute fractional change in component i of the cache. The fractional update cost is the following:

$$||\boldsymbol{p}^t - \boldsymbol{p}^{t+1}||_{1,\boldsymbol{w}'} = \sum_{j \in I} m_j w'_{s,j} + \sum_{j \in C \setminus I} m_i w'_{s,i}. \tag{8.82}$$

A flow $[\delta_{i,j}]_{(i,j) \in C^2}$ is constructed to transport $\sum_{i \in C \setminus I} m_i$ mass from the components in I to the components in $C \setminus I$ in line 15. The expected value of the allocation generated by output variable i_s^{t+1} is given by:

$$\mathbb{E}[\boldsymbol{e}_{i_s^{t+1}}] \overset{(8.79)}{=} \mathbb{E}[\boldsymbol{e}_{i_s^t}] + \sum_{i \in C \setminus I} \sum_{j \in I} \delta_{i,j}(\boldsymbol{e}_j - \boldsymbol{e}_i) \tag{8.83}$$

$$= \boldsymbol{p}^t - \sum_{i \in C \setminus I} m_i \boldsymbol{e}_i + \sum_{j \in I} m_j \boldsymbol{e}_j = \boldsymbol{p}^{t+1}. \tag{8.84}$$

The expected movements incurred when Algorithm 2.6 is executed is the following:

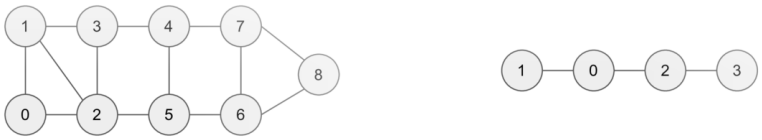
$$\mathbb{E}[||\boldsymbol{e}_{i_s^t} - \boldsymbol{e}_{i_s^{t+1}}||_{1,\boldsymbol{w}'}] \overset{(8.80)}{=} \sum_{i \in C \setminus I} \sum_{j \in I} \delta_{i,j}(w'_{s,j} + w'_{s,i}) = \sum_{j \in I} m_j w'_{s,j} + \sum_{j \in C \setminus I} m_i w'_{s,i}$$

$$\overset{(8.82)}{=} ||\boldsymbol{p}^t - \boldsymbol{p}^{t+1}||_{1,\boldsymbol{w}'}.$$

□

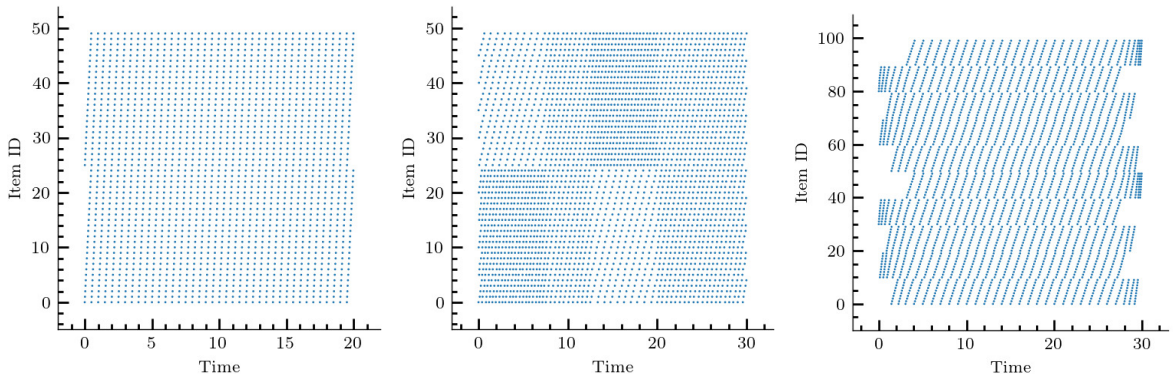
9 Adversarial Instances

In this section, we provide additional details about the topologies `path` and `abilene`. These are motivated by the proof of Lemma 2.3.4, using round-robin schemes for which greedy/myopic online algorithms would perform poorly.



(a) `Abilene` topology (b) `Path` topology

Figure 9.5: Topologies used for adversarial requests.



(a) *Stationary Adversarial* trace (b) *Sliding Popularity Adversarial* trace (c) *SN Adversarial* trace

Figure 9.6: Adversarial Request models for `abilene` and `path`. Each dot indicates an access to an item/ a request.

9.1 Topology Configuration

The `abilene` and `path` topologies are shown in Figure 9.5 (a) and (b) respectively.
Adversarial setup of `abilene`. We set the weight of each edge to be $w = 100$. The query nodes are $\{0, 5\}$. We put a cache at each node with a capacity selected u.a.r from the set $\{0, 1\}$, except for nodes $\{0, 5\}$ that have capacity 5. For every item in the catalog we select its source node u.a.r to be 7 or 8.
Adversarial setup of `path`. We set the weight of each edge to be $w = 100$. The query node is 0. We put two caches at nodes 0 and 1 with capacity 5. The whole catalog is stored at node 1.

9.2 Adversarial Traces

The adversarial traces patterns are shown in Figure 9.6. The *Stationary Adversarial* trace is generated using the sequence

$$\{0, 25, 1, 26, \ldots, 24, 49, 0, 25, \ldots\}. \tag{9.85}$$

The *Sliding Popularity Adversarial* trace is generated by mixing the two sequences

$$s_1 = \{i \mod 5 + 5k : i \in [100], k \in [5]\}, \tag{9.86}$$

$$s_2 = \{i \mod 5 + 5k + 25 : i \in [100], k \in [5]\}. \tag{9.87}$$

We generate requests from s_1 and s_2 starting at the same time, except that we generate requests from s_1 twice as fast as s_2. When we finish generating requests from a sequence we alternate the speed. This is done once, then at the final stage we generate requests with at the same speed. The *SN Adversarial* trace is generated using the same cyclic pattern as in the *Sliding Popularity Adversarial* trace, with the difference that a group of 5 items arrive according to according to a homogeneous Poisson process of rate $\gamma = 1$.

10 Jointly Optimizing Caching and Routing

In the extended model by Ioannidis and Yeh [64], a request $r = (i, b) \in \mathcal{R}$ is determined by (a) the item $i \in C$ requested, (b) the source node $b \in V$ of the request. For each request $r = (i, b)$, there exists a set of paths $\mathcal{P}_{(i,b)}$, which the request can follow towards a designated server in $\mathcal{D}_i$. The goal is to jointly determine the content allocation *as well as* the paths that requests follow.

In particular, the *path assignment* is represented by $P = \{p_r\}_{r\in\mathcal{R}} \in \prod_{r\in\mathcal{R}} \mathcal{P}_r$, where $p_r \in \mathcal{P}_r$ indicates that request $r = (i, b) \in \mathcal{R}$ follows path p_r to fetch item i. It is easy, and natural, to write the cost objective in terms of the content allocation A and the routing assignment P. However, to show that it is a submodular assignment problem, with constrains similar to the ones we encounter in caching, we deviate from [64] and express the objective in terms of the *complementary path assignment* $\overline{P}$. Formally, let

$$\overline{P} = \bigcup_{r\in\mathcal{R}} (\mathcal{P}_r \setminus \{p_r\}) \subset \bigcup_{r\in\mathcal{R}} \mathcal{P}_r. \tag{10.88}$$

Intuitively, given a path assignment P, the complementary path assignment $\overline{P}$ consists of all the paths *not* taken. We can see the routing optimization constraints as a slotted assignment problem akin to the caching problem we have studied so far in the following way. Each request $r \in \mathcal{R}$ is associated with exactly $|\mathcal{P}_r| - 1$ slots. These slots are to be occupied by paths *not taken*. That is, each such slot is to be occupied by a path p in $\mathcal{P}_r$; whenever such a path p is stored in a slot, it is added in the complementary path assignment $\overline{P}$. We denote by $\mathcal{D}'$ the set of feasible complementary path assignments under this setting, that is:[2]

$$\mathcal{D}' = \left\{ \overline{P} \subset \prod_{r\in\mathcal{R}} \mathcal{P}_r : |\overline{P} \cap \mathcal{P}_r| \leq |\mathcal{P}_r| - 1 \right\}. \tag{10.89}$$

[2]To better cast this as an assignment problem, we would need to introduce notation for slots per request, but this is equivalent to Eq. (10.89).

Then, given a content allocation A and complementary path assignment $\overline{P}$, the cost of serving a request $r = (i, b)$ is:

$$C_r(A, \overline{P}) = \sum_{p \in \mathcal{P}_r \setminus \overline{P}} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(A \cap \left\{ \bigcup_{k' \in [k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} = \emptyset \right). \tag{10.90}$$

Similarly, the caching gain of a request $r = (i, b)$ due to caching at intermediate nodes and path assignment is:

$$\begin{aligned} f_r(A, \overline{P}) &= C_r(\emptyset, \emptyset) - C_r(A, \overline{P}) \\ &= \sum_{p \in \mathcal{P}_r} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \mathbb{1}\left(p \in \overline{P} \vee A \cap \left\{ \bigcup_{k' \in [k]} \mathcal{S}_{p_{k'}} \times \{i\} \right\} \neq \emptyset \right). \end{aligned} \tag{10.91}$$

The caching gain maximization problem amounts to:

$$\underset{A, \overline{P}}{\text{maximize}} \quad f(A, \overline{P}) = \sum_{t=1}^{T} f^t(A, \overline{P}) = \sum_{t=1}^{T} \sum_{r \in \mathcal{R}^t} f_r(A, \overline{P}), \tag{10.92a}$$

$$\text{subject to} \quad A \in \mathcal{D}, \overline{P} \in \mathcal{D}'. \tag{10.92b}$$

which is a submodular maximization over an (assignment) partition matroid w.r.t. both A and $\overline{P}$ (complementary set of P). The assingment nature of the matroid follows from the representation of complementary paths as slots "taken"; we prove submodularity below:

Lemma 10.1. *Function $f : \mathcal{S} \times \mathcal{C} \times \prod_{r \in \mathcal{R}} \mathcal{P}_r \to \mathbb{R}_+$ is monotone and submodular w.r.t. both A and $\overline{P}$.*

Proof.

In this proof, we switch to vector notation rather than the set notation. For any content allocation $A \in \mathcal{S} \times \mathcal{C}$, the corresponding allocation vector is denoted by $\boldsymbol{x} = [\mathbb{1}_{\{(s,i) \in A\}}]_{(s,i) \in \mathcal{S} \times \mathcal{C}} = [x_{(s,i)}]_{(s,i) \in \mathcal{S} \times \mathcal{C}}$. For any complementary path assignment $P \in \prod_{r \in \mathcal{R}} \mathcal{P}_r$, the corresponding complementary path assignment $\bar{P}$ is represented by vector $\bar{\boldsymbol{h}} = [\mathbb{1}_{\{(r,p) \notin P\}}]_{(r,p) \in \prod_{r \in \mathcal{R}} \mathcal{P}_r} = [\bar{h}_{r,p}]_{(r,p) \in \prod_{r \in \mathcal{R}} \mathcal{P}_r}$. The caching gain of a request (i, b) is:

$$f_{(i,b)}(\bar{\boldsymbol{h}}, \boldsymbol{x}) = \sum_{p \in \mathcal{P}_{(i,b)}} \sum_{k=1}^{|p|-1} w_{p_{k+1}p_k} \left(1 - (1 - \bar{h}_{(i,b),p}) \prod_{k'=1}^{k} (1 - x_{p_{k'}i}) \right), \tag{10.93}$$

which has the same form as Eq. (2.36). By Lemma 2.3.1, f is also monotone (non-decreasing) and submodular w.r.t. both $\boldsymbol{x}$ and $\bar{\boldsymbol{h}}$. The feasible set for $\bar{\boldsymbol{h}}$ is:

$$\sum_{p \in \mathcal{P}_r} \bar{h}_{r,p} = |\mathcal{P}_r| - 1, \forall r \in \mathcal{R}, \tag{10.94}$$

which is also a partition matroid.

□ The monotonicity of the objective implies that an optimal solution exists in which all routing slots of the complementary path assignment are taken, so that indeed only one path is truly selected.[3]

This submodular maximization assignment problem can be tackled by the modified DistributedTGOnline with $1 - 1/e$ guarantee through Cor. 2.3.3 as follows. Following [64], for each request (i, b), source node b maintains an extra slot $s = (b, 0)$ determining its path assignment. Correspondingly, the information needed to compute the reward is from all possible paths due to Eq. (10.91), besides its own path. The second step in the online algorithm is, thus, modified as:

- When a request (i, b) is generated, (rather than one additional control message is generated to collect and transmit information,) $|\mathcal{P}_{i,b}|$ additional control messages are generated, one over one path $p \in \mathcal{P}_{i,b}$ to collect and transmit information.

Note that the communication cost increases linearly with $|\mathcal{P}_{i,b}|$. Nevertheless, it is possible that randomization approaches akin to the ones used in [64], can lower this dependency with a corresponding increase in regret; exploring this is beyond our scope.

11 Anytime Regret Guarantee

Under the *doubling trick* [88], the algorithm proceeds in phases. In the first phase, it sets its (short-term) horizon to a time-window $W_0 = 1$. Whenever a phase ends (i.e., the short-term horizon expires), the algorithm resets its state, and doubles the time window, so that the short term horizon at phase $n + 1$ satisfies:

$$W_{n+1} = 2W_n, \quad \text{for all } n \in \{0, 1, \ldots, k\}.$$

For the sake of notational simplicity, assume that $T = 2^{k+1} - 1$ for some $k \in \mathbb{N}$. By Theorem 2.3.2, DistributedTGOnline has bounded regret $c\sqrt{W_n}$ at the end of each short-term horizon W_n, where c is a constant independent of W_n. Thus,

$$\sum_{n=0}^{k} c\sqrt{W_n} = c \sum_{n=0}^{k} 2^{0.5n} = c\frac{2^{0.5(k+1)} - 1}{\sqrt{2} - 1} = c\frac{2^{0.5\log_2(T+1)}}{\sqrt{2} - 1} = c\frac{\sqrt{T+1}}{\sqrt{2} - 1} \leq \frac{\sqrt{2}}{\sqrt{2} - 1} c\sqrt{T}. \tag{11.95}$$

In other words, using this doubling trick, we can obtain an anytime regret bound for any algorithm designed for a fixed time horizon, while worsening the bound by a constant factor (namely, $\frac{\sqrt{2}}{\sqrt{2}-1}$).

The same argument can be used to show that the update cost is sublinear when taking update costs in to account. In particular, the modified policy resets its state at most $k - 1$ times and k is logarithmic in T, thereby contributing at most an $O(\log T)$ term to the overall update cost.

[3] Note that if an optimal solution contains fewer occupied slots, one with higher caching gain can be constructucted by adding more paths.

12 Proof of Lemma 3.2.1

We observe that for $y_t^{(1)} \in \mathcal{B}_2(R)$, $\|g_t^{(1)}\|_2$ is bounded and for $y_t^{(1)} \notin \mathcal{B}_2(R)$, $\|g_t^{(1)}\|_2 \leq 2+2\|y_t^{(1)}\|_2$, then there exists positive constants A_l and B_l for $l = 1, 2, 3, 4$, such that for $\|g_t^{(1)}(r, y_t^{(1)})\|_2^l \leq A_l + B_l\|y_t^{(1)}\|_2^l$.

We also observe that

$$\inf\{y_t^{(1)} \cdot g_t^{(1)}, \forall y_t^{(1)} \notin \mathcal{B}_2(R+1)\} > 0, \forall i = 1, \ldots, k.$$

We will prove boundedness using the function

$$f_t = \sum_{i=1}^{k} \phi(\|y_t^{(1)}\|_2^2)$$

where $\phi(x) = \mathbb{1}\left(x \geq (R+1)^2\right)(x - (R+1)^2)^2$. There exist positive constants A and B such that $\left(A_0 + B_0\|y_t^{(1)}\|_2^4\right) \leq A/k + B\phi(\|y_t^{(1)}\|_2^2)$.

Taking advantage of the inequality $\phi(x') - \phi(x) \leq (x'-x)\phi'(x') + (x'-x)^2$ applied for $x = \|y_t^{(1)}\|_2^2$ and $x' = \|y_{t+1}^{(1)}\|_2^2$, we can derive the following inequality

$$f_{t+1} - f_t \leq \sum_{i=1}^{k} \left[\left(-2\eta_t y_t^{(1)} \cdot g_t^{(1)} + \eta_t^2 \|y_t^{(1)}\|_2^2\right) \phi'(\|y_t^{(1)}\|_2^2) + \left(-2\eta_t y_t^{(1)} \cdot g_t^{(1)} + \eta_t^2 \|y_t^{(1)}\|_2^2\right)^2 \right]. \quad (12.96)$$

Using the inequalities above, we obtain

$$f_{t+1} - f_t \leq \sum_{i=1}^{k} \left[-2\eta_t y_t^{(1)} \cdot g_t^{(1)} \phi'(\|y_t^{(1)}\|_2^2) + \eta_t^2 \left(A_0 + B_0\|y_t^{(1)}\|_2^4\right) \right] \quad (12.97)$$

$$\leq \sum_{i=1}^{k} \left[-2\eta_t y_t^{(1)} \cdot g_t^{(1)} \phi'(\|y\|_2^2) \right] + \eta_t^2 \left(A + Bf_t\right). \quad (12.98)$$

We take now the conditional expectation given $H_t = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$:

$$\mathbb{E}_x[f_{t+1} - f_t | H_t] \leq \sum_{i=1}^{k} \left[-2\eta_t y_t^{(1)} \cdot \mathbb{E}_x\left[g_t^{(1)}\right] \phi'(\|y_t^{(1)}\|_2^2) \right] + \eta_t^2 \left(A + Bf_t\right) \quad (12.99)$$

$$\leq \eta_t^2 (A + Bf_t)., \quad (12.100)$$

where the last inequality follows from $\phi'(\|y_t^{(1)}\|_2^2) = 0$ for $y_t^{(1)} \in \mathcal{B}_2(R+1)$ and $y_t^{(1)} \cdot g_t^{(1)} > 0$ for $y_t^{(1)} \in \mathcal{B}_2(R+1)$.

The proof until now follows logically the same steps as in [138, Section 5.2], but 1) we did not need to suppose that $g_t^{(1)}$ have l-th moments bounded by $\|y_t^{(1)}\|_2^l$, as their norm is bounded from our definition of the algorithm, 2) we had to redefine the function f_t to take into account separately each subvector $y_t^{(1)}$ of $\boldsymbol{y}_t$. From (12.100) we can reason as in [138, Section 5.2] to show that $\{f_t\}_{t\in\mathbb{N}}$ is a

quasi-martingale and apply convergence results for quasi-martingales [138, Section 4.4] to conclude that f_t converges almost surely to a random variable $f_\infty \geq 0$ with $\mathbb{E}[f_\infty] < \infty$. We want to conclude that $f_\infty = 0$ with probability one, and then $\boldsymbol{y}_t$ is bounded almost surely.

The rest of the proof differs from [138, Section 5.2]. Moving around the terms in (12.98) and summing over t, we obtain

$$\sum_{t=1}^{\infty}\sum_{i=1}^{k} 2\eta_t y_t^{(1)} \cdot g_t^{(1)} \phi'(\|y\|_2^2) \leq f_1 - f_\infty + \sum_{t=1}^{\infty} \eta_t^2 \left(A + B f_t\right) < +\infty \text{ a.s.} \tag{12.101}$$

because $\sum_t \eta_t^2 < \infty$ and f_t converges to $f_\infty < \infty$.

Let $\mathcal{H}$ be the set of sequences such that $f_\infty > 0$. For each sequence in $\mathcal{H}$, $f_t > f_\infty/2$ for large t, and then there exists at least a value i_t and an opportune $\epsilon > 0$ such that $\|y_t^{(i_t)}\|_2 > (R+1) + \epsilon$. Then both $y_t^{(1)} \cdot g_t^{(1)}$ and $\phi'(\|y\|_2^2)$ are bounded below by positive quantities and the series in the LHS of (12.101) diverges as $\sum_t \eta_t = +\infty$. But we have concluded that this series converges a.s., then $\mathbb{P}(\mathcal{H}) = 0$ and $f_\infty = 0$ a.s.. Each sequence $(\boldsymbol{y}_1, \boldsymbol{y}_2, \dots)$ is then bounded a.s.

13 Proof of Theorem 3.2.2

We denote by $y_{t,i}$ the i-th component of the vector $\boldsymbol{y}_t \in \mathbb{R}^{k\times d}$, and by y_i the i-th component of a generic vector $\boldsymbol{y} \in \mathbb{R}^{k\times d}$.

We define

$$F(x) = \frac{x^4}{1+x^2}, \quad L(\boldsymbol{y}) = \sum_{i=1}^{k} \mathbb{1}\left(y_t^i \notin \mathcal{B}_2(R)\right) F\left(\|y_t^{(1)}\|_2 - R\right), \quad \tilde{c}(r, \boldsymbol{y}) = c(r, \boldsymbol{y}) + L(\boldsymbol{y}). \tag{13.102}$$

We observe that $\nabla_{\boldsymbol{y}} \tilde{c}(r, \boldsymbol{y}_t) = (g_t^{(1)}, \dots, g_t^{(k)})$, i.e., the dynamic in (3.4) is evolving according to the gradient of $\tilde{c}(r, \boldsymbol{y}_t)$. Similary we define $\tilde{\mathcal{C}}(\boldsymbol{y}) = \mathbb{E}_r[\tilde{c}(r, \boldsymbol{y})] = \mathcal{C}(\boldsymbol{y}) + L(\boldsymbol{y})$. The function $L(\boldsymbol{y})$ is continuously differentiable up to the third order, then $\tilde{\mathcal{C}}(\boldsymbol{y})$ is continuously differentiable up to the second/third order when $\mathcal{C}(\boldsymbol{y})$ is so. Moreover, we observe that $\|\nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y})\|_2 > 0$ for $y^{(1)} \notin \mathcal{B}_2(R)$ for some i. Then, $\liminf_{t\to\infty} \|\nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t)\|_2 = 0$, implies that $\liminf_{t\to\infty} \|\nabla_{\boldsymbol{y}} \mathcal{C}(\boldsymbol{y}_t)\|_2 = 0$.

Lemma 3.2.1 shows that the sequence $H = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots)$ is bounded a.s. Consider a bounded sequence H, such that $\boldsymbol{y}_t \in \mathcal{B}_2(R')$ for some $R' > 0$. $\nabla \tilde{c}(r, \boldsymbol{y}_t)$ exists a.s. and it is bounded for any $x \in \mathcal{X}$. By the dominated convergence theorem, it follows that we can invert the expectation and the gradient:

$$\mathbb{E}_x\left[\nabla_{\boldsymbol{y}} \tilde{c}(r, \boldsymbol{y}_t)\right] = \nabla_{\boldsymbol{y}} \mathbb{E}_x\left[\tilde{c}(r, \boldsymbol{y}_t)\right] = \nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t).$$

As $\tilde{\mathcal{C}}(\,\cdot\,)$ is continously differentiable up to the second order upon $\mathcal{B}_2(R')$, the partial derivatives are bounded, in particular, there exist two constants c_1 and c_2 such that $|\partial \tilde{\mathcal{C}}(\boldsymbol{y})/\partial y_i| \leq c_1$ and $|\partial^2 \tilde{\mathcal{C}}(\boldsymbol{y})/\partial y_i \partial y_j| \leq c_2$ for each $i, j \in \{1, 2, \dots, kp\}$.

Consider $\boldsymbol{y}_{t+1} = \boldsymbol{y}_t - \eta_t \nabla \tilde{c}(r_t, \boldsymbol{y}_t)$. Using Taylor formula we can arrive to

$$\tilde{\mathcal{C}}(\boldsymbol{y}_{t+1}) - \tilde{\mathcal{C}}(\boldsymbol{y}_t) = \sum_{i=1}^{kp} \frac{\partial \tilde{\mathcal{C}}(\boldsymbol{y}_t)}{\partial \boldsymbol{y}_i}(\boldsymbol{y}_{t+1,i} - \boldsymbol{y}_{t,i}) + \sum_{i,j=1}^{kp} \frac{\partial^2 \tilde{\mathcal{C}}(\boldsymbol{y}_t + s\boldsymbol{y}_{t+1})}{\partial \boldsymbol{y}_i \partial \boldsymbol{y}_j} \frac{(\boldsymbol{y}_{t+1,i} - \boldsymbol{y}_{t,i})(\boldsymbol{y}_{t+1,j} - \boldsymbol{y}_{t,j})}{2} \tag{13.103}$$

$$\leq -\eta_t \nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t) \cdot \nabla_{\boldsymbol{y}} \tilde{c}(r_t, \boldsymbol{y}_t) + \eta_t^2 \frac{c_2 kp}{2} \|\nabla c(r_t, \boldsymbol{y}_t)\|_2^2 \tag{13.104}$$

$$\leq -\eta_t \nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t) \cdot \nabla_{\boldsymbol{y}} \tilde{c}(r_t, \boldsymbol{y}_t) + \eta_t^2 \frac{c_2 c_1^2 (kp)^2}{2}. \tag{13.105}$$

where $s \in [0, 1]$. Denoting the constant $c_2 c_1^2 kp/2$ by b and summing for $t = 1, \ldots, T$, we obtain

$$\sum_{t=1}^{T} \eta_t \nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t) \cdot \nabla_{\boldsymbol{y}} \tilde{c}(r_t, \boldsymbol{y}_t) \leq \tilde{\mathcal{C}}(\boldsymbol{y}_1) - \tilde{\mathcal{C}}(\boldsymbol{y}_T) + \sum_{t=1}^{T} \eta_t^2 b \tag{13.106}$$

$$\leq \tilde{\mathcal{C}}(\boldsymbol{y}_1) + \sum_{t=1}^{T} \eta_t^2 b \tag{13.107}$$

Finally, taking the expected value, and letting T diverges, we obtain

$$\sum_{t=1}^{\infty} \eta_t \, \mathbb{E}\left[\|\nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t)\|_2^2\right] \leq \mathbb{E}\left[\tilde{\mathcal{C}}(\boldsymbol{y}_1)\right] + \sum_{t=1}^{\infty} \eta_t^2 b < +\infty. \tag{13.108}$$

The series on the LHS is then summable. It follows that

$$\sum_{t=1}^{\infty} \eta_t \|\nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t)\|_2^2 < +\infty \text{ a.s.} \tag{13.109}$$

and then we can complete the proof of the first thesis:

$$\liminf_{t \to \infty} \|\nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t)\|_2 = 0 \text{ a.s.}$$

Consider now that $\mathcal{C}(\,\cdot\,)$ is continuously differentiable up to the third order and then its third order partial derivatives are bounded over $\mathcal{B}_2(R')$, i.e., $|\partial^3 \tilde{\mathcal{C}}(\boldsymbol{y}_t)/\partial y_i \partial y_j \partial y_l| \leq c_3$. Let $a(\boldsymbol{y}) = \|\nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y})\|_2^2$. This is continuously differentiable up to the second order and we can then use the Taylor formula as above, considering that

$$\frac{\partial a(\boldsymbol{y})}{\partial \boldsymbol{y}_i} = 2 \sum_{j=1}^{kp} \frac{\partial \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial \boldsymbol{y}_j} \frac{\partial^2 \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial \boldsymbol{y}_i \partial \boldsymbol{y}_j}, \tag{13.110}$$

$$\frac{\partial^2 a(\boldsymbol{y})}{\partial \boldsymbol{y}_i \partial \boldsymbol{y}_l} = 2 \sum_{j=1}^{kp} \frac{\partial^2 \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_l} \frac{\partial^2 \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial \boldsymbol{y}_i \partial \boldsymbol{y}_j} + \frac{\partial \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial \boldsymbol{y}_j} \frac{\partial^3 \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial \boldsymbol{y}_i \partial \boldsymbol{y}_j \partial \boldsymbol{y}_l}. \tag{13.111}$$

Then $|\partial^2 a(\boldsymbol{y})/\partial \boldsymbol{y}_i \partial \boldsymbol{y}_l| \leq c_2' = 2kp(c_2^2 + c_1 c_3)$. We obtain:

$$a(\boldsymbol{y}_{t+1}) - a(\boldsymbol{y}_t) \leq -2\eta_t \sum_{i,j=1}^{kp} \frac{\partial \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial y_j} \frac{\partial^2 \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial y_i \partial y_j} \frac{\partial \tilde{c}(r_t, \boldsymbol{y}_t)}{\partial y_i} + \eta_t^2 \frac{c_2' c_1^2 (kp)^2}{2}. \tag{13.112}$$

If we now compute the conditional expectation given the history up to t, $H_t = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$, we obtain

$$\mathbb{E}\big[a(\boldsymbol{y}_{t+1}) - a(\boldsymbol{y}_t)|H_t\big] \leq -2\eta_t \sum_{i,j=1}^{kp} \frac{\partial \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial y_j} \frac{\partial^2 \tilde{\mathcal{C}}(\boldsymbol{y})}{\partial y_i \partial y_j} \frac{\partial \tilde{\mathcal{C}}(\boldsymbol{y}_t)}{\partial y_i} + \eta_t^2 \frac{c_2' c_1^2 (kp)^2}{2} \tag{13.113}$$

$$\leq 2c_2 kp \eta_t \|\nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t)\|_2^2 + \eta_t^2 \frac{c_2' c_1^2 (kp)^2}{2}. \tag{13.114}$$

Because of (13.109) and $\sum_t \eta_t^2 < \infty$, $\sum_t \mathbb{E}\big[a(\boldsymbol{y}_{t+1}) - a(\boldsymbol{y}_t)|H_t\big] < +\infty$ almost surely and a convergence result for quasi-martingales [138, Section 4.4] implies that $a(\boldsymbol{y}_t)$ converges almost surely to a random variable a_∞ with finite expected value. As $a_\infty > 0$ if and only if $\|\nabla_{\boldsymbol{y}} \tilde{\mathcal{C}}(\boldsymbol{y}_t)\|_2^2 > 0$ for large t, it is possible to reason as at the end of the proof of Lemma 3.2.1 and use (13.109) to conclude that $a_\infty = 0$ almost surely. It follows that

$$\lim_{t \to \infty} \nabla \tilde{\mathcal{C}}(\boldsymbol{y}_t) = 0 \text{ a.s.}$$

and this concludes the proof.

14 Equivalent Expression of the Cost Function

Lemma 14.1. *Let us fix the threshold $c \in \mathbb{N} \cup \{0\}$, $r \in \mathcal{R}$ and $i \in \mathcal{U}$. The following equality holds*

$$\min\left\{c, \sum_{j=1}^{i} x_{\pi_j^r}\right\} - \min\left\{c, \sum_{j=1}^{i-1} x_{\pi_j^r}\right\} = x_{\pi_i^r} \mathbb{1}\left(\sum_{j=1}^{i-1} x_{\pi_j^r} < c\right). \tag{14.115}$$

Proof.

We distinguish two cases:

(i) When $\sum_{j=1}^{i-1} x_{\pi_j^r} \geq c$ this implies that $\sum_{j=1}^{i} x_{\pi_j^r} = x_{\pi_j^r} + \sum_{j=1}^{i-1} x_{\pi_j^r} \geq c + x_{\pi_i^r} \geq c$ since $x_{\pi_i^r} \geq 0$. Therefore, $\sum_{j=1}^{i-1} x_{\pi_j^r} \geq c$ implies that $\min\left\{c, \sum_{j=1}^{i-1} x_{\pi_j^r}\right\} = \min\left\{c, \sum_{j=1}^{i} x_{\pi_j^r}\right\} = c$, and we have:

$$\min\left\{c, \sum_{j=1}^{i} x_{\pi_j^r}\right\} - \min\left\{c, \sum_{j=1}^{i-1} x_{\pi_j^r}\right\} = 0. \tag{14.116}$$

(ii) When $\sum_{j=1}^{i-1} x_{\pi_j^r} < c$, we have $\sum_{j=1}^{i} x_{\pi_j^r} = x_{\pi_j^r} + \sum_{j=1}^{i-1} x_{\pi_j^r} < c + x_{\pi_i^r} \leq c$ since $x_{\pi_i^r} \leq 1$, this implies that $\min\left\{c, \sum_{j=1}^{i} x_{\pi_j^r}\right\} = \sum_{j=1}^{i} x_{\pi_j^r}$ and $\min\left\{c, \sum_{j=1}^{i-1} x_{\pi_j^r}\right\} = \sum_{j=1}^{i-1} x_{\pi_j^r}$, and we have

$$\min\left\{c, \sum_{j=1}^{i} x_{\pi_j^r}\right\} - \min\left\{c, \sum_{j=1}^{i-1} x_{\pi_j^r}\right\} = x_{\pi_i^r}. \tag{14.117}$$

Combining Eqs. (14.116) and (14.117) yields Eq. (14.115).

□

Lemma 14.2. *The cost function $C(r, \boldsymbol{x})$ given by the expression in Eq.* (3.9)*, can be equivalently expressed as:*

$$C(r,\boldsymbol{x}) = -\sum_{i=1}^{K^r-1} \alpha_i^r \min\left\{k - \sigma_i^r, \sum_{j=1}^{i} x_{\pi_j^r} - \sigma_i^r\right\} + \sum_{i=1}^{K^r} c(r,\pi_i^r)\mathbb{1}\left(\pi_i^r \in \mathcal{U}\setminus\mathcal{N}\right), \tag{14.118}$$

where $\sigma_i^r = \sum_{j=1}^{i} \mathbb{1}\left(\pi_j^r \in \mathcal{U}\setminus\mathcal{N}\right)$, $\alpha_i^r = c(r,\pi_{i+1}^r) - c(r,\pi_i^r)$ and $K^r = \min\{i \in \mathcal{U} : \sigma_i^r = k\}$ for every $(r,i) \in \mathcal{R}\times\mathcal{U}$.

Proof.

Let $\tilde{c}(r,o), \forall (r,o) \in \mathcal{R}\times\mathcal{U}$ be a cost defined as $\tilde{c}(r,\pi_i^r) = c(r,\pi_i^r), \forall i \in [K^r]$ and 0 otherwise, and we also define $\tilde{\alpha}_i^r$ as $\tilde{\alpha}_i^r \triangleq \tilde{c}(r,\pi_{i+1}^r) - \tilde{c}(r,\pi_i^r)$.

When $i = 1$ and for any $r \in \mathcal{R}$, we have

$$\tilde{c}(r,\pi_i^r)\min\left\{k, \sum_{j=1}^{i-1} x_{\pi_j^r}\right\} = 0. \tag{14.119}$$

Note that $\sigma_1^r = 0$ by definition, since $\pi_1^r \in \mathcal{N}$ for any $r \in \mathcal{R}$. We have

$$-\sum_{i=1}^{K^r} \tilde{\alpha}_i^r \sigma_i^r = \sum_{i=1}^{K^r} \tilde{c}(r,\pi_i^r)\sigma_i^r - \sum_{i=1}^{K^r} \tilde{c}(r,\pi_{i+1}^r)\sigma_i^r = \sum_{i=1}^{K^r} \tilde{c}(r,\pi_i^r)\sigma_i^r - \sum_{i=2}^{K^r} \tilde{c}(r,\pi_i^r)\sigma_{i-1}^r \tag{14.120}$$

$$= \sum_{i=1}^{K^r} \tilde{c}(r,\pi_i^r)\sigma_i^r - \sum_{i=1}^{K^r} \tilde{c}(r,\pi_i^r)\sigma_{i-1}^r = \sum_{i=1}^{K^r} \tilde{c}(r,\pi_i^r)\mathbb{1}_{\left\{\pi_i^r \in \mathcal{U}\setminus\mathcal{N}\right\}} = \sum_{i=1}^{K^r} c(r,\pi_i^r)\mathbb{1}\left(\pi_i^r \in \mathcal{U}\setminus\mathcal{N}\right). \tag{14.121}$$

Observe that the indicator function $\mathbb{1}_{\left\{\sum_{j=1}^{i-1} x_{\pi_j^r} < k\right\}}$ is 0 for every $i \geq K^r + 1$; therefore, the summation in Eq. (3.9) can be limited to K^r instead of $2N$. Using Lemma 14.1 we expand the expression of

$C(r, \boldsymbol{x})$ as follows:

$$C(r,\boldsymbol{x}) = \sum_{i=1}^{K^r} c(r,\pi_i^r) x_{\pi_i^r} \mathbb{1}_{\left\{\sum_{j=1}^{i-1} x_{\pi_j^r} < k\right\}} = \sum_{i=1}^{K^r} \tilde{c}(r,\pi_i^r) x_{\pi_i^r} \mathbb{1}\left(\sum_{j=1}^{i-1} x_{\pi_j^r} < k\right) \tag{14.122}$$

$$\overset{(14.115)}{=} \sum_{i=1}^{K^r} \tilde{c}(r,\pi_i^r) \left(\min\left\{k, \sum_{j=1}^{i} x_{\pi_j^r}\right\} - \min\left\{k, \sum_{j=1}^{i-1} x_{\pi_j^r}\right\}\right) \tag{14.123}$$

$$\overset{(14.119)}{=} \sum_{i=1}^{K^r} \tilde{c}(r,\pi_i^r) \min\left\{k, \sum_{j=1}^{i} x_{\pi_j^r}\right\} - \sum_{i=1}^{K^r-1} \tilde{c}(r,\pi_{i+1}^r) \min\left\{k, \sum_{j=1}^{i} x_{\pi_j^r}\right\} \tag{14.124}$$

$$= \sum_{i=1}^{K^r} \tilde{c}(r,\pi_i^r) \min\left\{k, \sum_{j=1}^{i} x_{\pi_j^r}\right\} - \sum_{i=1}^{K^r} \tilde{c}(r,\pi_{i+1}^r) \min\left\{k, \sum_{j=1}^{i} x_{\pi_j^r}\right\} \tag{14.125}$$

$$= -\sum_{i=1}^{K^r} \tilde{\alpha}_i^r \min\left\{k, \sum_{j=1}^{i} x_{\pi_j^r}\right\} = -\sum_{i=1}^{K^r} \tilde{\alpha}_i^r \min\left\{k - \sigma_i^r, \sum_{j=1}^{i} x_{\pi_j^r} - \sigma_i^r\right\} - \sum_{i=1}^{K^r} \tilde{\alpha}_i^r \sigma_i^r \tag{14.126}$$

$$\overset{(14.121)}{=} -\sum_{i=1}^{K^r} \tilde{\alpha}_i^r \min\left\{k - \sigma_i^r, \sum_{j=1}^{i} x_{\pi_j^r} - \sigma_i^r\right\} + \sum_{i=1}^{K^r} c(r,\pi_i^r) \mathbb{1}_{\{\pi_i^r \in \mathcal{U}\setminus\mathcal{N}\}} \tag{14.127}$$

$$= -\sum_{i=1}^{K^r-1} \alpha_i^r \min\left\{k - \sigma_i^r, \sum_{j=1}^{i} x_{\pi_j^r} - \sigma_i^r\right\} + \sum_{i=1}^{K^r} c(r,\pi_i^r) \mathbb{1}_{\{\pi_i^r \in \mathcal{U}\setminus\mathcal{N}\}} \qquad (\text{since } k - \sigma_{K^r}^r = 0). \tag{14.128}$$

This gives the cost function expression in Eq. (14.118).

□

Lemma 14.3. *For any request $r \in \mathcal{R}$ the caching gain function $G(r, \boldsymbol{x})$ in Eq.* (3.10) *has the following expression*

$$G(r,\boldsymbol{x}) = \sum_{i=1}^{K^r-1} \alpha_i^r \min\left\{k - \sigma_i^r, \sum_{j=1}^{i} x_{\pi_j^r} - \sigma_i^r\right\}, \tag{14.129}$$

where $\sigma_i^r = \sum_{j=1}^{i} \mathbb{1}\left(\pi_j^r \in \mathcal{U} \setminus \mathcal{N}\right)$, $\alpha_i^r = c(r,\pi_{i+1}^r) - c(r,\pi_i^r)$ and $K^r = \min\{i \in \mathcal{U} : \sigma_i^r = k\}$ for every $(r,i) \in \mathcal{R} \times \mathcal{U}$.

Proof.

From Lemma 14.2, the cost function $C(r, \boldsymbol{x})$ can be equivalently expressed as:

$$C(r,\boldsymbol{x}) = -\sum_{i=1}^{K^r-1} \alpha_i^r \min\left\{k - \sigma_i^r, \sum_{j=1}^{i} x_{\pi_j^r} - \sigma_i^r\right\} + \sum_{i=1}^{K^r} c(r,\pi_i^r) \mathbb{1}\left(\pi_i^r \in \mathcal{U} \setminus \mathcal{N}\right). \tag{14.130}$$

Without caching the system incurs the cost $kc_f + \sum_{o \in k\text{NN}(r)}^{k} c(r,o)$ when $r \in \mathcal{R}$ is requested. This is the retrieval cost of fetching k objects and the sum of the approximation costs of the k closest objects in $\mathcal{N}$. From the definition of π^r, this cost can equivalently be expressed as $\sum_{i=1}^{K^r} c(r,\pi_i^r)\mathbb{1}\left(\pi_i^r \in \mathcal{U} \setminus \mathcal{N}\right)$. Therefore, we recover the gain expression in Eq. (14.3) as the cost reduction due to having a similarity cache. Thus, we have

$$G(r,\boldsymbol{x}) = kc_f + \sum_{o \in k\text{NN}(r)}^{k} c(r,o) - C(r,\boldsymbol{x}) \tag{14.131}$$

$$= \sum_{i=1}^{K^r} c(r,\pi_i^r)\mathbb{1}\left(\pi_i^r \in \mathcal{U} \setminus \mathcal{N}\right) - \sum_{i=1}^{K^r} c(r,\pi_i^r)\mathbb{1}\left(\pi_i^r \in \mathcal{U} \setminus \mathcal{N}\right) + \sum_{i=1}^{K^r-1} \alpha_i^r \min\left\{k - \sigma_i^r, \sum_{j=1}^{i} x_{\pi_j^r} - \sigma_i^r\right\} \tag{14.132}$$

$$= \sum_{i=1}^{K^r-1} \alpha_i^r \min\left\{k - \sigma_i^r, \sum_{j=1}^{i} x_{\pi_j^r} - \sigma_i^r\right\}. \tag{14.133}$$

This concludes the proof.

□

15 Supporting Lemmas for Proof of Proposition 17.1

Lemma 15.1. *For every request $r \in \mathcal{R}$, index $i \in \mathcal{U}$, and fractional cache state $\boldsymbol{y} \in \text{conv}(\mathcal{X})$ the index set defined as*

$$I_i^r \triangleq \left\{j \in [i] : \left(\pi_j^r \in \mathcal{N}\right) \wedge \left(\pi_j^r + N \notin \{\pi_l^r : l \in [i]\}\right)\right\} \tag{15.134}$$

satisfies the following

$$\sum_{j \in [i]} y_{\pi_j^r} - \sigma_i^r = \sum_{j \in I_i^r} y_{\pi_j^r}, \tag{15.135}$$

where $\sigma_i^r = \sum_{j=1}^{i} \mathbb{1}\left(\pi_j^r \in \mathcal{U} \setminus \mathcal{N}\right)$ (defined in Eq. (3.12)).

Proof.

$$\sum_{j \in [i]} y_{\pi_j^r} - \sigma_i^r \overset{(3.12)}{=} \sum_{j \in [i]} y_{\pi_j^r} - \sum_{j \in [i]} \mathbb{1}_{\{\pi_j^r \in \mathcal{U} \setminus \mathcal{N}\}} = \sum_{\substack{j \in [i] \\ \pi_j^r \in \mathcal{N}}} y_{\pi_j^r} + \sum_{\substack{j \in [i] \\ \pi_j^r \in \mathcal{U} \setminus \mathcal{N}}} \left(y_{\pi_j^r} - 1\right) = \sum_{\substack{j \in [i] \\ \pi_j^r \in \mathcal{N}}} y_{\pi_j^r} - \sum_{\substack{l \in [i] \\ \pi_l^r \in \mathcal{U} \setminus \mathcal{N}}} y_{(\pi_l^r - N)} \tag{15.136}$$

Remark that if $l \in [i]$ and $\pi_l^r \in \mathcal{U} \setminus \mathcal{N}$, then the object $\pi_l^r - N$ has a strictly smaller cost and it appears earlier in the permutation π^r, that is there exists $j < l$ such that $\pi_j^r = \pi_l^r - N$. In this case, the variable $y_{(\pi_l^r - N)}$ cancels out $y_{\pi_j^r}$ in the RHS of (15.136). Then, we have

$$\sum_{\substack{j \in [i] \\ \pi_j^r \in \mathcal{N}}} y_{\pi_j^r} - \sum_{\substack{l \in [i] \\ \pi_l^r \in \mathcal{U} \setminus \mathcal{N}}} y_{(\pi_l^r - N)} = \sum_{\substack{j \in [i] \\ \pi_j^r \in \mathcal{N} \\ \pi_j^r + N \notin \left\{\pi_l^r : (l \in [i]) \wedge \left(\pi_l^r \in \mathcal{U} \setminus \mathcal{N}\right)\right\}}} y_{\pi_j^r}. \tag{15.137}$$

Note that if $\pi_l^r \notin \mathcal{U} \setminus \mathcal{N}$ (i.e., $\pi_l^r \in \mathcal{N}$), then $\pi_j^r + N \neq \pi_l^r$. Then the sets $\left\{\pi_l^r : (l \in [i]) \wedge \left(\pi_l^r \in \mathcal{U} \setminus \mathcal{N}\right)\right\}$ and $\left\{\pi_l^r : l \in [i]\right\}$ coincide. Therefore, the above equality can be simplified as follows

$$\sum_{\substack{j \in [i] \\ \pi_j^r \in \mathcal{N}}} y_{\pi_j^r} - \sum_{\substack{l \in [i] \\ \pi_l^r \in \mathcal{U} \setminus \mathcal{N}}} y_{(\pi_l^r - N)} = \sum_{\substack{j \in [i] \\ \pi_j^r \in \mathcal{N} \\ \pi_j^r + N \notin \left\{\pi_l^r : l \in [i]\right\}}} y_{\pi_j^r} = \sum_{j \in I_i^r} y_{\pi_j^r}. \tag{15.138}$$

Eq. (15.136) and Eq. (15.138) are combined to get

$$\sum_{j \in [i]} y_{\pi_j^r} - \sigma_i^r = \sum_{j \in I_i^r} y_{\pi_j^r}, \tag{15.139}$$

and this concludes the proof.

□

16 Bounds on the Auxiliary Function

We define $\Lambda : \mathcal{R} \times \text{conv}(\mathcal{X}) \to \mathbb{R}_+$, an auxiliary function, that will be utilized in bounding the value of the gain function

$$\Lambda(r, \boldsymbol{y}) \triangleq \sum_{i=1}^{K^r - 1} \alpha_i^r (k - \sigma_i^r) \left(1 - \prod_{j \in I_i^r} \left(1 - \frac{y_{\pi_j^r}}{k - \sigma_i^r}\right)\right), \forall r \in \mathcal{R}, \boldsymbol{y} \in \text{conv}(\mathcal{X}). \tag{16.140}$$

The DepRound [173] subroutine outputs a rounded variable $\boldsymbol{x} \in \mathcal{X}$ from a fractional input $\boldsymbol{y} \in \text{conv}(\mathcal{X})$, by iteratively modifying the fractional input $\boldsymbol{y}$. At each iteration the subroutine Simplify that is part of DepRound is executed on two yet unrounded variables $y_i, y_j \in (0, 1)$ with $i, j \in \mathcal{N}$, until all the variables are rounded in $O(N)$ steps. Note that only $y_i, \forall i \in \mathcal{N}$ is rounded, since $x_i \in \mathcal{U} \setminus \mathcal{N}$ is determined directly from x_{i-N}. The random output of DepRound [173] subroutine $\boldsymbol{x} \in \mathcal{X}$ given the input $\boldsymbol{y} \in \text{conv}(\mathcal{X})$ has the following properties:

P1 $\mathbb{E}[x_i] = y_i, \forall i \in \mathcal{N}$.

P2 $\sum_{i\in\mathcal{N}} x_i = k$.

P3 $\forall S \subset \mathcal{N}, \mathbb{E}\left[\prod_{i\in S}(1-x_i)\right] \leq \prod_{i\in S}(1-y_i)$.

Lemma 16.1. *The random output $\boldsymbol{x} \in \mathcal{X}$ of* DEPROUND *given the fractional cache state input $\boldsymbol{y} \in \mathrm{conv}(\mathcal{X})$, or the random output $\boldsymbol{x} \in \{0,1\}^{\mathcal{U}}$ of* COUPLEDROUNDING *given an integral cache state $\boldsymbol{x}'$, and fractional cache states $\boldsymbol{y}, \boldsymbol{y}' \in \mathrm{conv}(\mathcal{X})$ with $\mathbb{E}[\boldsymbol{x}'] = \boldsymbol{y}'$, satisfy the following for any request $r \in \mathcal{R}$*

$$\mathbb{E}\left[\Lambda(r,\boldsymbol{x})\right] \geq \Lambda(r,\boldsymbol{y}). \tag{16.141}$$

Proof.

$$\mathbb{E}\left[\Lambda(r,\boldsymbol{x})\right] \overset{(16.140)}{=} \mathbb{E}\left[\sum_{i=1}^{K^r-1}\alpha_i^r(k-\sigma_i^r)\left(1-\prod_{j\in I_i^r}\left(1-\frac{x_{\pi_j^r}}{k-\sigma_i^r}\right)\right)\right] = \sum_{i=1}^{K^r-1}\alpha_i^r(k-\sigma_i^r)\left(1-\mathbb{E}\left[\prod_{j\in I_i^r}\left(1-\frac{x_{\pi_j^r}}{k-\sigma_i^r}\right)\right]\right) \tag{16.142}$$

$$\leq \sum_{i=1}^{K^r-1}\alpha_i^r(k-\sigma_i^r)\left(1-\prod_{j\in I_i^r}\left(1-\frac{y_{\pi_j^r}}{k-\sigma_i^r}\right)\right) = \Lambda(r,\boldsymbol{y}). \tag{16.143}$$

The second equality is obtained using the linearity of the expectation operator. The inequality is obtained using [12, Lemma E.10] with $S = I_i^r$, $c_m = \frac{1}{k-\sigma_i^r}$, as $\sigma_i^r < k$ for $i < K^r$ in the case when $\boldsymbol{x}$ is the output of DEPROUND, and in the case when $\boldsymbol{x}$ is an output of COUPLEDROUNDING, the inequality holds with equality since every x_i for $i \in \mathcal{N}$ is an independent random variable.

□

17 Bounds on the Gain function

Proposition 17.1. *The caching gain function $G(r,\boldsymbol{x})$ defined in Eq.* (3.10) *has the following lower and upper bound for any request $r \in \mathcal{R}$ and fractional cache state $\boldsymbol{y} \in \mathrm{conv}(\mathcal{X})$:*

$$\Lambda(r,\boldsymbol{y}) \leq G(r,\boldsymbol{y}) \leq \left(1-\frac{1}{e}\right)^{-1}\Lambda(r,\boldsymbol{y}). \tag{17.144}$$

Proof.

We have the following

$$G(r,\boldsymbol{y}) = \sum_{i=1}^{K^r-1}\alpha_i^r \min\left\{k-\sigma_i^r, \sum_{j\in[i]} y_{\pi_j^r} - \sigma_i^r\right\} \overset{(15.135)}{=} \sum_{i=1}^{K^r-1}\alpha_i^r \min\left\{k-\sigma_i^r, \sum_{j\in I_i^r}^{i} y_{\pi_j^r}\right\} \tag{17.145}$$

$$\geq \sum_{i=1}^{K^r-1}\alpha_i^r(k-\sigma_i^r)\left(1-\prod_{j\in I_i^r}\left(1-\frac{y_{\pi_j^r}}{k-\sigma_i^r}\right)\right) \tag{17.146}$$

$$= \Lambda(r,\boldsymbol{y}), \tag{17.147}$$

and

$$G(r,\boldsymbol{y}) = \sum_{i=1}^{K^r-1} \alpha_i^r \min\left\{k-\sigma_i^r, \sum_{j\in I_i^r} y_{\pi_j^r}\right\} \tag{17.148}$$

$$\leq \left(1-\frac{1}{e}\right)^{-1} \sum_{i=1}^{K^r-1} \alpha_i^r (k-\sigma_i^r)\left(1-\prod_{j\in I_i^r}\left(1-\frac{y_{\pi_j^r}}{k-\sigma_i^r}\right)\right) \tag{17.149}$$

$$= \Lambda(r,\boldsymbol{y}). \tag{17.150}$$

The inequalities in Eq. (17.146) and Eq. (17.149) are obtained through [12, Lemma E.7], and [12, Lemma E.8], respectively, by setting $c = k-\sigma_i^r$, and $q_i = 1$ for every $i \in [K^r-1]$.

□

18 Subgradients Computation

Theorem 18.1. *For any time slot $t \in [T]$, the vectors $\boldsymbol{g}_t$ given by Eq.* (18.151) *are subgradients of the caching gain function $G(r,\boldsymbol{y})$ for request $r_t \in \mathcal{R}$ at fractional cache state $\boldsymbol{y}_t \in \mathrm{conv}(\mathcal{X})$.*

$$\boldsymbol{g}_t = \left[\left(c\left(r_t, \pi^{r_t}_{i_*^{r_t}+1}\right) - c\left(r_t, l\right)\right)\mathbb{1}\left(l_*^{r_t} \leq i_*^{r_t}\right)\right]_{l\in\mathcal{N}}, \tag{18.151}$$

where $i_*^{r_t} \triangleq \max\left\{i \in [K^{r_t}-1] : \left(\sum_{j=1}^{i} y_{t,\pi_j^r} \leq k\right) \wedge \left(l+N \notin \{\pi_v^{r_t} : v \in [i]\}\right)\right\}$, $l_*^{r_t} \triangleq (\pi^{r_t})^{-1}(l), \forall l \in \mathcal{N}$, and $(\pi^{r_t})^{-1}$ is the inverse permutation of π^{r_t}.

Proof.

For any request $r \in \mathcal{R}$, the function $f^{(r,i)}(\boldsymbol{y}) \triangleq \min\left\{k-\sigma_i^r, \sum_{j=1}^{i} y_{\pi_j^r} - \sigma_i^r\right\}$ is a concave function, i.e., a minimum of two concave functions (a constant and an affine function). The subdifferential of the function at point $\boldsymbol{y}$, using Theorem [296, Theorem 8.2] is given as

$$\partial f^{(r,i)}(\boldsymbol{y}) = \begin{cases} \boldsymbol{0} & \text{if } \sum_{j=1}^{i} y_{\pi_j^r} > k, \\ \mathrm{conv}\left(\left\{\boldsymbol{0}, \nabla\left(\sum_{j=1}^{i} y_{\pi_j^r}\right)\right\}\right) & \text{if } \sum_{j=1}^{i} y_{\pi_j^r} = k, \\ \nabla\left(\sum_{j=1}^{i} y_{\pi_j^r}\right) & \text{otherwise,} \end{cases} \tag{18.152}$$

where $\mathrm{conv}(\cdot)$ is the convex hull of a set. Thus, a valid subgradient $\boldsymbol{g}^{(r,i)}(\boldsymbol{y})$ of $f^{(r,i)}$ at point $\boldsymbol{y}$ can be picked as

$$\boldsymbol{g}^{(r,i)}(\boldsymbol{y}) = \begin{cases} \boldsymbol{0} & \text{if } \sum_{j=1}^{i} y_{\pi_j^r} \geq k, \\ \nabla\left(\sum_{j=1}^{i} y_{\pi_j^r}\right) & \text{otherwise.} \end{cases} \tag{18.153}$$

Note that

$$\frac{\partial}{\partial y_l}\sum_{j=1}^{i} y_{\pi_j^r} = \mathbb{1}\left(y_l \text{ appears in the sum and } y_{l+N} = 1 - y_l \text{ does not}\right) \tag{18.154}$$

$$= \mathbb{1}\left(\left(l \in \{\pi_v^r : v \in [i]\}\right) \ \wedge \ \left(l + N \notin \{\pi_v^r : v \in [i]\}\right)\right) \tag{18.155}$$

$$= \mathbb{1}\left(\left(l_*^r \le i\right) \ \wedge \ \left(l + N \notin \{\pi_v^r : v \in [i]\}\right)\right). \tag{18.156}$$

The l-th component of the subgradient $\boldsymbol{g}^{(r,i)}(\boldsymbol{y})$ is given by

$$g_l^{(r,i)}(\boldsymbol{y}) = \begin{cases} 0, & \text{if } \sum_{j=1}^{i} y_{\pi_j^r} \ge k, \\ \frac{\partial}{\partial y_l}\sum_{j=1}^{i} y_{\pi_j^r} & \text{otherwise.} \end{cases} \tag{18.157}$$

$$= \mathbb{1}\left(\left(\sum_{j=1}^{i} y_{\pi_j^r} < k \ \wedge \ l_*^r \le i\right) \ \wedge \ \left(l + N \notin \{\pi_v^r : v \in [n]\}\right)\right). \tag{18.158}$$

For any non-negative factor α_i^r, we have $\partial\left(\alpha_i^r f^{(r,i)}(\boldsymbol{y})\right) = \alpha_i^r \partial\left(f^{(r,i)}(\boldsymbol{y})\right)$ (multiply both sides of the subgradient inequality by a non-negative constant [53, Definition 1.2]), and using [297, Theorem 23.6] we get

$$\partial G(r, \boldsymbol{y}) = \partial\left(\sum_{i=1}^{K^r-1} \alpha_i^{r_t} f^{(r,i)}(\boldsymbol{y})\right) = \sum_{i=1}^{K^r-1} \alpha_i^{r_t} \partial f^{(r,i)}(\boldsymbol{y}). \tag{18.159}$$

Let $i_*^r \triangleq \max\{i \in [K^r - 1] : \left(\sum_{j=1}^{i} y_{\pi_j^r} \le k\right) \wedge \left(l + N \notin \{\pi_v^r : v \in [i]\}\right)\}$. Now we can define a subgradient $\boldsymbol{g}_t$ of the function $G(r, \boldsymbol{y})$ at point $\boldsymbol{y}_t \in \text{conv}(\mathcal{X})$ and request $r_t \in \mathcal{R}$ for any $t \in [T]$, where every component $l \in \mathcal{N}$ of $\boldsymbol{g}_t$ is given by

$$g_{t,l} = \sum_{i=1}^{K^{r_t}-1} \alpha_i^{r_t} g_l^{(r_t,i)}(\boldsymbol{y}_t) = \sum_{i=1}^{K^{r_t}-1} \alpha_i^{r_t} \mathbb{1}\left(\left(\sum_{j=1}^{i} y_{\pi_j^{r_t}} < k\right) \ \wedge \ \left(l_*^{r_t} \le i\right) \ \wedge \ \left(l + N \notin \{\pi_v^{r_t} : v \in [n]\}\right)\right) \tag{18.160}$$

$$= \sum_{i=l_*^{r_t}}^{K^{r_t}-1} \alpha_i^{r_t} \mathbb{1}\left(\left(\sum_{j=1}^{i} y_{\pi_j^{r_t}} < k\right) \ \wedge \ \left(l + N \notin \{\pi_v^{r_t} : v \in [n]\}\right)\right) = \sum_{i=l_*^{r_t}}^{i_*^{r_t}} \alpha_i^{r_t} = \sum_{i=l_*^{r_t}}^{i_*^{r_t}} \left(c(r_t, \pi_{i+1}^{r_t}) - c(r_t, \pi_i^{r_t})\right) \tag{18.161}$$

$$= \left(c(r_t, \pi_{i_*^{r_t}+1}^{r_t}) - c(r_t, l)\right) \mathbb{1}\left(l_*^{r_t} \le i_*^{r_t}\right), \forall l \in \mathcal{N}. \tag{18.162}$$

Note that in the last equality we used the definition of $l_*^{r_t}$ to obtain $c(r_t, l) = c(r_t, \pi_{l_*^{r_t}}^r)$. This concludes the proof.

□

19 Supporting Lemmas for Proof of Theorem 3.3.3

19.1 Subgradient Bound

Lemma 19.1. *For any time slot $t \in [T]$, fractional cache state $\boldsymbol{y}_t \in \mathrm{conv}(\mathcal{X})$, and request $r_t \in \mathcal{R}$ the subgradients $\boldsymbol{g}_t$ of the gain function in Eq. (3.11) are bounded w.r.t the norm $\|\cdot\|_\infty$ by the constant*

$$L \triangleq c_d^k + c_f. \tag{19.163}$$

The constant c_d^k is an upper bound on the dissimilarity cost of the k-th closest object for any request in $\mathcal{R}$, and c_f is the retrieval cost.

Proof.

For any time slot $t \in [T]$ we have

$$\|\boldsymbol{g}_t\|_\infty = \max\left\{g_{t,l} : l \in \mathcal{N}\right\} \overset{(18.151)}{=} \max\left\{c(r_t, \pi^{r_t}_{i^{r_t}_*+1}) - c(r_t, l) : l \in \mathcal{N}\right\} \tag{19.164}$$

$$\leq c(r_t, \pi^{r_t}_{K^{r_t}}) - c(r_t, \pi^{r_t}_1) \leq c(r_t, \pi^{r_t}_{K^{r_t}}) \tag{19.165}$$

$$= c_f + c(r_t, \pi^{r_t}_{K^{r_t}-N}) \leq c_f + c(r_t, \pi^{r_t}_{K^{r_t}-N}) \leq c_f + c_d^k. \tag{19.166}$$

□ Note that $L_2 \triangleq \left\|\partial_{\boldsymbol{y}} G(r, \boldsymbol{y})\right\|_2$ can be as high as $\sqrt{N}L$ and N can be very large; moreover, the regret upper bound is proportional to L_2 instead of L when the Euclidean map is used as a mirror map (see [53, Theorem 4.2]). This justifies why it is preferable to work with the negative entropy instantiation of OMA rather than the classical Euclidean setting.

19.2 Bregman Divergence Bound

Lemma 19.2. *Let $\boldsymbol{y}^* = \underset{\boldsymbol{y}\in\mathrm{conv}(\mathcal{X})}{\arg\max} \sum_{t=1}^T G(r, \boldsymbol{y})$ and $\boldsymbol{y}_1 = \underset{\boldsymbol{y}\in\mathrm{conv}(\mathcal{X})\cap\mathcal{D}}{\arg\min} \Phi(\boldsymbol{y})$, the value of the Bregman divergence $D_\Phi(\boldsymbol{y}_*, \boldsymbol{y}_1)$ associated with the negative entropy mirror map Φ is upper bounded by the constant*

$$D \triangleq h \log\left(\frac{N}{h}\right). \tag{19.167}$$

Proof.

It is easy to check that $y_{1,i} = \frac{h}{N}, \forall i \in \mathcal{N}$ ($\boldsymbol{y}_1$ has maximum entropy); moreover, we have $\Phi(\boldsymbol{y}) \leq 0, \forall \boldsymbol{y} \in \mathrm{conv}(\mathcal{X})$. The first order optimality condition [53, Proposition 1.3] gives $-\nabla\Phi(\boldsymbol{y}_1)^T(\boldsymbol{y} - \boldsymbol{y}_1) \leq 0, \forall \boldsymbol{y} \in \mathrm{conv}(\mathcal{X})$. We have

$$D_\Phi(\boldsymbol{y}_*, \boldsymbol{y}_1) = \Phi(\boldsymbol{y}_*) - \Phi(\boldsymbol{y}_1) - \nabla\Phi(\boldsymbol{y}_1)^T(\boldsymbol{y}_* - \boldsymbol{y}_1) \leq \Phi(\boldsymbol{y}_*) - \Phi(\boldsymbol{y}_1) \leq -\Phi(\boldsymbol{y}_1) = h\log\left(\frac{N}{h}\right). \tag{19.168}$$

□

20 Update Costs

20.1 Proof of Theorem 3.3.1

First part. We show that $\mathbb{E}\left[\boldsymbol{x}_{t+1}\right] = \boldsymbol{y}_{t+1}$. Take $\boldsymbol{\delta} \triangleq \boldsymbol{y}_{t+1} - \boldsymbol{y}_t$.

If $\delta_i > 0$ for $i \in \mathcal{N}$, then:

$$\mathbb{E}\left[x_{t+1,i}\right] = \mathbb{E}\left[x_{t+1,i}|x_{t,i}=1\right]\mathbb{P}(x_{t,i}=1) + \mathbb{E}\left[x_{t+1,i}|x_{t,i}=0\right]\mathbb{P}(x_{t,i}=0) \tag{20.169}$$

$$= y_{t,i} + \left(\frac{\delta_i}{1-y_{t,i}} + 0\right)(1-y_{t,i}) = y_{t,i} + \delta_i. \tag{20.170}$$

If $\delta_i < 0$ for $i \in \mathcal{N}$, then:

$$\mathbb{E}\left[x_{t+1,i}\right] = \mathbb{E}\left[x_{t+1,i}|x_{t,i}=1\right]\mathbb{P}(x_{t,i}=1) + \mathbb{E}\left[x_{t+1,i}|x_{t,i}=0\right]\mathbb{P}(x_{t,i}=0) \tag{20.171}$$

$$= \left(\frac{y_{t,i}+\delta_i}{y_{t,i}}\right) y_{t,i} + 0 = y_{t,i} + \delta_i. \tag{20.172}$$

Otherwise, when $\delta_i = 0$ for $i \in \mathcal{N}$ we have $\mathbb{E}[x_{t+1,i}] = \mathbb{E}[x_{t,i}] = y_{t,i} = y_{t,i} + \delta_i$. Therefore we have for any $i \in \mathcal{N}$

$$\mathbb{E}[\boldsymbol{x}_{t+1}] = \boldsymbol{y}_t + \boldsymbol{\delta} = \boldsymbol{y}_{t+1}. \tag{20.173}$$

Second part. For any $i \in \mathcal{N}$, we can have two types of movements: If $\delta_i < 0$, then given that $x_{t,i} = 1$, we evict with probability $\frac{-\delta_i}{y_{t,i}}$. If $\delta_i > 0$, then given that $x_{t,i} = 0$, we retrieve a file with probability $\frac{\delta_i}{1-y_{t,i}}$. Thus the expected movement is given by:

$$\mathbb{E}\left[\|\boldsymbol{x}_{t+1} - \boldsymbol{x}_t\|_1\right] = \sum_{i\in\mathcal{N}} \mathbb{E}[|x_{t+1,i} - x_{t+1,i}|] \tag{20.174}$$

$$= \sum_{i\in\mathcal{N}} \mathbb{E}\left[|x_{t+1,i} - x_{t,i}|\middle|x_{t,i}=0\right]\mathbb{P}(x_{t,i}=0) + \mathbb{E}\left[|x_{t+1,i} - x_{t,i}|\middle|x_{t,i}=1\right]\mathbb{P}(x_{t,i}=1) \tag{20.175}$$

$$= \sum_{i\in\mathcal{N}} \left(\frac{\delta_i}{1-y_{t,i}}\mathbb{1}\left(\delta_i > 0\right)\cdot(1-y_{t,i}) + \frac{-\delta_i}{y_{t,i}}\mathbb{1}\left(\delta_i < 0\right)\cdot y_{t,i}\right) \tag{20.176}$$

$$= \sum_{i\in\mathcal{N}} |\delta_i| = \sum_{i\in\mathcal{N}} |y_{t+1,i} - y_{t,i}| = \|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\|_1. \tag{20.177}$$

20.2 Proof of Theorem 3.3.2

The negative entropy mirror map Φ is $\rho = \frac{1}{h}$ strongly convex w.r.t the norm $\|\cdot\|_1$ over $\mathcal{D} \cap \text{conv}(\mathcal{X})$ (see [48, Ex. 2.5]), and the subgradients are bounded under the dual norm $\|\cdot\|_\infty$ by L, i.e., for any $r_t \in \mathcal{R}$, $\boldsymbol{y}_t \in \text{conv}(\mathcal{X})$, and $t \in [T]$ we have $\|\boldsymbol{g}_t\|_\infty \leq L$ (Lemma 19.1). For any time slot $t \in [T-1]$,

it holds:

$$\begin{aligned} D_\Phi(\boldsymbol{y}_t, \boldsymbol{z}_{t+1}) &= \Phi(\boldsymbol{y}_t) - \Phi(\boldsymbol{z}_{t+1}) - \nabla\Phi(\boldsymbol{z}_{t+1})^T(\boldsymbol{y}_t - \boldsymbol{z}_{t+1}) \\ &= \Phi(\boldsymbol{y}_t) - \Phi(\boldsymbol{z}_{t+1}) + \nabla\Phi(\boldsymbol{y}_t)^T(\boldsymbol{z}_{t+1} - \boldsymbol{y}_t) + (\nabla\Phi(\boldsymbol{y}_t) - \nabla\Phi(\boldsymbol{z}_{t+1}))^T(\boldsymbol{y}_t - \boldsymbol{z}_{t+1}) \\ &\leq -\frac{\rho}{2}\left\|\boldsymbol{y}_t - \boldsymbol{z}_{t+1}\right\|_1^2 + \eta \boldsymbol{g}_t^T(\boldsymbol{y}_t - \boldsymbol{z}_{t+1}) \qquad (20.178) \\ &\leq -\frac{\rho}{2}\left\|\boldsymbol{y}_t - \boldsymbol{z}_{t+1}\right\|^2 + \eta L\left\|\boldsymbol{y}_t - \boldsymbol{z}_{t+1}\right\|_1^2 \qquad (20.179) \\ &\leq \frac{\eta^2 L^2}{2\rho}. \qquad (20.180) \end{aligned}$$

Eqs. (20.178)–(20.180) are obtained using the strong convexity of Φ and the update rule, Cauchy-Schwarz inequality, and the inequality $ax - bx^2 \leq \max_x ax - bx^2 = a^2/4b$ as in the last step in the proof of [53, Theorem 4.2], respectively. Moreover, for any $t \in [T-1]$ it holds

$$\begin{aligned} \left\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\right\|_1 &\leq \sqrt{\frac{2}{\rho}D_\Phi(\boldsymbol{y}_t, \boldsymbol{y}_{t+1})} \leq \sqrt{\frac{2}{\rho}D_\Phi(\boldsymbol{y}_t, \boldsymbol{z}_{t+1}) - \frac{2}{\rho}D_\Phi(\boldsymbol{y}_{t+1}, \boldsymbol{z}_{t+1})} \leq \sqrt{\frac{2}{\rho}D_\Phi(\boldsymbol{y}_t, \boldsymbol{z}_{t+1})} \qquad (20.181) \\ &\leq \sqrt{2\eta^2\frac{L^2}{2\rho^2}} \leq \frac{L\eta}{\rho}. \qquad (20.182) \end{aligned}$$

The above chain of inequalities is obtained through: the strong convexity of Φ, the generalized Pythagorean inequality [53, Lemma 4.1], non-negativity of the Bregman divergence of a convex function, and Eq. (20.180), in respective order. The learning rate is $\eta = O\left(\frac{1}{\sqrt{T}}\right)$; therefore, we finally get

$$\sum_{t=1}^{T-1}\left\|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\right\|_1 \overset{(20.182)}{\leq} \frac{L\eta T}{\rho} = O(\sqrt{T}). \qquad (20.183)$$

21 Proof of Theorem 3.3.3

To prove the ψ-regret guarantee: (1) we first establish an upper bound on the regret of the AÇAI policy over its fractional cache states domain $\text{conv}(\mathcal{X})$ against a fractional optimum, then (2) the guarantee is transformed in a ψ-regret guarantee over the integral cache states domain $\mathcal{X}$ in expectation.

Fractional domain guarantee. We establish first the regret of running Algorithm 3.2 with decisions taken over the fractional domain $\text{conv}(\mathcal{X})$. The following properties are satisfied:

(i) The caching gain function $G(r, \boldsymbol{y})$ is concave over its fractional domain $\text{conv}(\mathcal{X})$ for any $r \in \mathcal{R}$ (see Section 3.3.3.4).

(ii) The negative entropy mirror map $\Phi : \mathcal{D} \to \mathcal{R}$ is $\frac{1}{h}$ strongly convex w.r.t the norm $\|\cdot\|_1$ over $\mathcal{D} \cap \text{conv}(\mathcal{X})$ (see [48, Ex. 2.5]).

(iii) The subgradients are bounded under the dual norm $\|\cdot\|_\infty$ by L, i.e., for any $r_t \in \mathcal{R}$, $\boldsymbol{y}_t \in \operatorname{conv}(\mathcal{X})$, and $t \in [T]$ we have $\|\boldsymbol{g}_t\|_\infty \leq L$ (Lemma 19.1).

(iv) The Bregman divergence $D_\Phi(\boldsymbol{y}^*, \boldsymbol{y}_1)$ is bounded by a constant D where $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y}\in\operatorname{conv}(\mathcal{X})} \sum_{t=1}^T G(r, \boldsymbol{y})$ and $\boldsymbol{y}_1 = \operatorname*{arg\,min}_{\boldsymbol{y}\in\operatorname{conv}(\mathcal{X})\cap\mathcal{D}} \Phi(\boldsymbol{y})$ is the initial fractional cache state (Lemma 19.2).

With the above properties satisfied, the regret of Algorithm 3.2 with the gains evaluated over the fractional cache states $\{\boldsymbol{y}_t\}_{t=1}^T \in \operatorname{conv}(\mathcal{X})^T$ is [53, Theorem 4.2]

$$\operatorname{Regret}_{T,\operatorname{conv}(\mathcal{X})}(\mathrm{OMA}_\Phi) = \sup_{\{\boldsymbol{r}_t\}_{t=1}^T\in\mathcal{R}^T} \left\{ \sum_{t=1}^T G(r_t, \boldsymbol{y}_*) - \sum_{t=1}^T G(r_t, \boldsymbol{y}_t) \right\} \tag{21.184}$$

$$\leq \frac{D_\Phi(\boldsymbol{y}_*, \boldsymbol{y}_1)}{\eta} + \frac{\eta h}{2} \sum_{t=1}^T \|\boldsymbol{g}_t\|_\infty^2 \leq \frac{D}{\eta} + \frac{\eta L^2 h T}{2}. \tag{21.185}$$

Integral domain guarantee. Let $\boldsymbol{x}_* = \operatorname*{arg\,max}_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^T G(r_t, \boldsymbol{x})$ and $\boldsymbol{y}_* = \operatorname*{arg\,max}_{\boldsymbol{y}\in\operatorname{conv}(\mathcal{X})} \sum_{t=1}^T G(r_t, \boldsymbol{y})$. The fractional cache state $\boldsymbol{y}_* \in \operatorname{conv}(\mathcal{X})$ is obtained by maximizing $\sum_{t=1}^T G(r_t, \boldsymbol{y})$ over the domain $\boldsymbol{y} \in \operatorname{conv}(\mathcal{X})$. We can only obtain a lower gain by restricting the maximization to a subset of the domain $\mathcal{X} \subset \operatorname{conv}(\mathcal{X})$. Therefore, we obtain:

$$\sum_{t=1}^T G(r_t, \boldsymbol{y}_*) \geq \sum_{t=1}^T G(r_t, \boldsymbol{x}_*). \tag{21.186}$$

Note that the components for $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathcal{U} \setminus \mathcal{N}$ are completely determined by the components in $\mathcal{N}$. Take $\psi = 1 - 1/e$. For every $t \in \{1, M, 2M, \ldots, M\lfloor T/M \rfloor\}$ and $r \in \mathcal{R}$ it holds

$$\mathbb{E}\left[G(r, \boldsymbol{x}_t)\right] \overset{(17.144)}{\geq} \mathbb{E}\left[\Lambda(r, \boldsymbol{x}_t)\right] \overset{(16.141)}{\geq} \Lambda(r, \boldsymbol{y}_t) \overset{(17.144)}{\geq} \psi G(r, \boldsymbol{y}_t). \tag{21.187}$$

Moreover, consider the following decomposition of the time slots $\{1, 2, \ldots, T\} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_{\lfloor T/M \rfloor + 1}$, where each $\mathcal{T}_i$ represents the i-th freezing period, i.e., $\boldsymbol{x}_t = \boldsymbol{x}_{\min(\mathcal{T}_i)}$ for $t \in \mathcal{T}_i$. Note that $|\mathcal{T}_i| \leq M$ for $i \in \{1, 2, \ldots, \lfloor T/M \rfloor + 1\}$. Now we can decompose the total expected gain of the policy as

$$\begin{aligned}
&\psi \sum_{t=1}^T G(r_t, \boldsymbol{x}_*) - \sum_{t=1}^T \mathbb{E}\left[G(r_t, \boldsymbol{x}_t)\right] \overset{(21.186)}{\leq} \psi \sum_{t=1}^T G(r_t, \boldsymbol{y}_*) - \sum_{t=1}^T \mathbb{E}\left[G(r_t, \boldsymbol{x}_t)\right] \\
&= \psi \sum_{t=1}^T G(r_t, \boldsymbol{y}_*) - \sum_{i=1}^{\lfloor T/M \rfloor + 1} \sum_{t\in\mathcal{T}_i} \mathbb{E}\left[G(r_t, \boldsymbol{x}_t)\right] \overset{(21.187)}{\leq} \psi \sum_{t=1}^T G(r_t, \boldsymbol{y}_*) - \psi \sum_{i=1}^{\lfloor T/M \rfloor + 1} \sum_{t\in\mathcal{T}_i} G(r_t, \boldsymbol{y}_{\min(\mathcal{T}_i)}) \\
&= \psi \sum_{t=1}^T G(r_t, \boldsymbol{y}_*) - \psi \sum_{t=1}^T G(r_t, \boldsymbol{y}_t) + \psi \sum_{t=1}^T G(r_t, \boldsymbol{y}_t) - \psi \sum_{i=1}^{\lfloor T/M \rfloor + 1} \sum_{t\in\mathcal{T}_i} G(r_t, \boldsymbol{y}_{\min(\mathcal{T}_i)}) \\
&\overset{(21.185)}{\leq} \psi \cdot \operatorname{Regret}_{T,\operatorname{conv}(\mathcal{X})}(\mathrm{OMA}_\Phi) + \psi \left(\sum_{t=1}^T G(r_t, \boldsymbol{y}_t) - \sum_{i=1}^{\lfloor T/M \rfloor + 1} \sum_{t\in\mathcal{T}_i} G(r_t, \boldsymbol{y}_{\min(\mathcal{T}_i)}) \right).
\end{aligned} \tag{21.188}$$

The first equality is obtained through a decomposition of the time slots to $\mathcal{T}_i$ for $i \in \{1, 2, \ldots, \lfloor T/M \rfloor + 1\}$. It remains to bound the r.h.s of Eq. (21.188), i.e.,

$$\sum_{t=1}^{T} G(r_t, \boldsymbol{y}_t) - \sum_{i=1}^{\lfloor T/M \rfloor+1} \sum_{t\in\mathcal{T}_i} G(r_t, \boldsymbol{y}_{\min(\mathcal{T}_i)}) = \sum_{i=1}^{\lfloor T/M \rfloor+1} \sum_{t\in\mathcal{T}_i} G(r_t, \boldsymbol{y}_t) - G(r_t, \boldsymbol{y}_{\min(\mathcal{T}_i)}) \tag{21.189}$$

$$\leq \sum_{i=1}^{\lfloor T/M \rfloor+1} \sum_{t\in\mathcal{T}_i} \partial_{\boldsymbol{y}} G(r_t, \boldsymbol{y}_{\min(\mathcal{T}_i)}) \cdot \left(\boldsymbol{y}_t - \boldsymbol{y}_{\min(\mathcal{T}_i)}\right) \qquad \text{concavity of } G(r_t, \cdot)$$

$$\leq \sum_{i=1}^{\lfloor T/M \rfloor+1} \sum_{t\in\mathcal{T}_i} \left\|\partial_{\boldsymbol{y}} G(r_t, \boldsymbol{y}_{\min(\mathcal{T}_i)})\right\|_\infty \left\|\boldsymbol{y}_t - \boldsymbol{y}_{\min(\mathcal{T}_i)}\right\|_1 \qquad \text{Hölder's inequality}$$

$$\leq L \sum_{i=1}^{\lfloor T/M \rfloor+1} \sum_{t\in\mathcal{T}_i} \sum_{t'=\min(\mathcal{T}_i)}^{t-1} \|\boldsymbol{y}_{t+1} - \boldsymbol{y}_t\|_1 \qquad \text{triangle inequality and definition of } L$$

$$\leq \frac{L^2}{\rho} \sum_{i=1}^{\lfloor T/M \rfloor+1} \sum_{t\in\mathcal{T}_i} \sum_{t'=\min(\mathcal{T}_i)}^{t-1} \eta \leq \frac{L^2\eta}{\rho} \cdot \left(\frac{T}{M} + 1\right) \cdot \frac{M(M-1)}{2} \qquad \text{update cost upper bound in Eq. (20.182)}$$

$$= \frac{L^2\eta}{2\rho}(M-1)\,(T+M) = \frac{hL^2\eta}{2}(M-1)\,(T+M) \qquad \text{we have } \rho = 1/h. \tag{21.190}$$

Thus, by bounding r.h.s of Eq. (21.188) in Eq. (21.190) we get

$$\psi \sum_{t=1}^{T} G(r_t, \boldsymbol{x}_*) - \sum_{t=1}^{T} \mathbb{E}\left[G(r_t, \boldsymbol{x}_t)\right] \leq \psi \left(\frac{D}{\eta} + \frac{\eta L^2 h}{2} T + \frac{\eta L^2 h}{2}(M-1)\,(T+M)\right) \tag{21.191}$$

By selecting the learning rate $\eta = \frac{1}{L}\sqrt{\frac{2D}{h(T+(M-1)(M+T))}} = \frac{1}{(c_d^k + c_f)}\sqrt{\frac{2\log\left(\frac{N}{h}\right)}{(T+(M-1)(M+T))}}$ giving the tightest upper bound we obtain

$$\psi \sum_{t=1}^{T} G(r_t, \boldsymbol{x}_*) - \sum_{t=1}^{T} \mathbb{E}\left[G(r_t, \boldsymbol{x}_t)\right] \leq \psi L \sqrt{2Dh(T + (M-1)(T+M))} \tag{21.192}$$

$$\overset{(19.163),(19.167)}{=} \left(1 - \frac{1}{e}\right)(c_d^k + c_f) h \sqrt{2\log\left(\frac{N}{h}\right)(T + (M-1)(T+M))}. \tag{21.193}$$

This concludes the proof.

22 Proof of Corollary 3.3.4

We have the following

$$\mathbb{E}\left[G_T(\bar{\boldsymbol{x}})\right] \overset{(17.144)}{\geq} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} \Lambda(r_t, \boldsymbol{x}_t)\right] \overset{(16.141)}{\geq} \frac{1}{T}\sum_{t=1}^{T} \Lambda(r_t, \boldsymbol{x}_t) \overset{(17.144)}{\geq} \psi G_T(\bar{y}). \tag{22.194}$$

We apply Jensen's inequality to obtain

$$G_T(\bar{y}) \geq \frac{1}{\tilde{T}} \sum_{i=1}^{\tilde{T}} G_T(\boldsymbol{y}_i). \tag{22.195}$$

It is easy to verify that G_T (3.17) is concave, and has bounded subgradients under the l_∞ norm over the fractional caching domain $\text{conv}(\mathcal{X})$; moreover, the remaining properties are satisfied for the same mirror map and decision set. The regret of Algorithm 3.2 with the gains evaluated over the fractional cache states $\{\boldsymbol{y}_i\}_{i=1}^{\tilde{T}} \in \text{conv}(\mathcal{X})^T$ is [53, Theorem 4.2] is given by

$$\sum_{i=1}^{\tilde{T}} G_T(\boldsymbol{y}_*) - \sum_{i=1}^{\tilde{T}} G_T(\boldsymbol{y}_i) = \tilde{T} G_T(\boldsymbol{y}_*) - \sum_{i=1}^{\tilde{T}} G_T(\boldsymbol{y}_i) \overset{(21.185)}{\leq} (c_d^k + c_f) h \sqrt{2 \log\left(\frac{N}{h}\right) \tilde{T}}. \tag{22.196}$$

Divide both sides of the equality by $\tilde{T}$, and move the gain attained by AÇAI to the l.h.s to get

$$\frac{1}{\tilde{T}} \sum_{i=1}^{\tilde{T}} G_T(\boldsymbol{y}_i) \geq G_T(\boldsymbol{y}_*) - (c_d^k + c_f) h \sqrt{2 \log\left(\frac{N}{h}\right) \tilde{T}}, \tag{22.197}$$

where $\boldsymbol{y}_* = \underset{\boldsymbol{y} \in \text{conv}(\mathcal{X})}{\arg\max} G_T(\boldsymbol{y})$.

We combine Eq. (22.194), Eq. (22.195), and Eq. (22.197) to obtain

$$\mathbb{E}\left[G_T(\bar{\boldsymbol{x}})\right] \geq G_T(\boldsymbol{y}_*) - (c_d^k + c_f) h \sqrt{2 \log\left(\frac{N}{h}\right) \tilde{T}}, \tag{22.198}$$

and $G_T(\boldsymbol{y}_*)$ can only be larger than $G_T(\boldsymbol{x}_*)$; thus, we also obtain

$$\mathbb{E}\left[G_T(\bar{\boldsymbol{x}})\right] \geq G_T(\boldsymbol{x}_*) - (c_d^k + c_f) h \sqrt{\frac{2 \log\left(\frac{N}{h}\right)}{\tilde{T}}}. \tag{22.199}$$

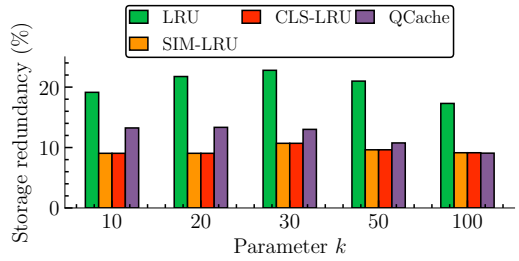
We conclude $\forall \epsilon > 0$ for a sufficiently large number of iterations $\tilde{T}$, $\bar{\boldsymbol{x}}$ satisfies

$$\mathbb{E}\left[G_T(\bar{\boldsymbol{x}})\right] \geq \left(1 - \frac{1}{e} - \epsilon\right) G_T(\boldsymbol{x}_*). \tag{22.200}$$

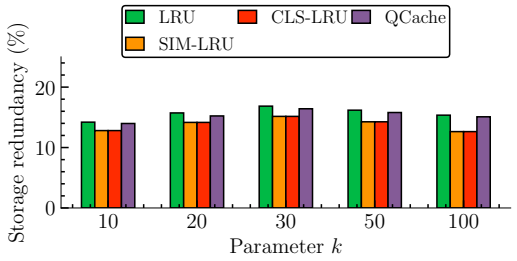
23 Additional Experiments

23.1 Redundancy

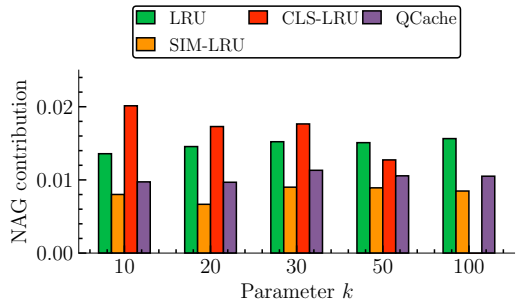
We quantify the redundancy present in the caches in Figure 23.7 (a), as the percentage of added objects to fill the physical cache. We also show the contribution of the dangling objects to the gain in Figure 23.7 (b), that does not exceed 2.0% under both traces.



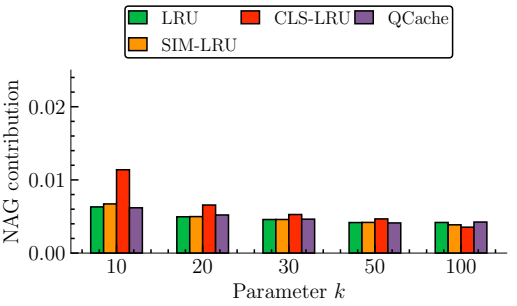
(a) Storage redundancy of the LRU-based policies under *SIFT1M trace*.



(b) Storage redundancy of the LRU-based policies under *Amazon trace*.



(c) The average gain contribution of the dangling objects under SIFT1M trace.



(d) The average gain contribution of the dangling objects under Amazon trace.

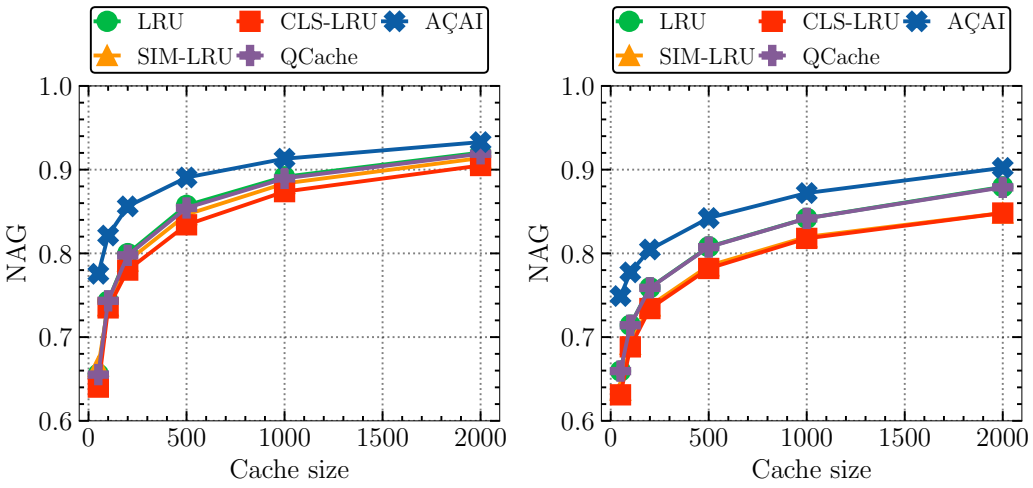
Figure 23.7: Storage redundancy percentage and gain contribution for the different policies. The cache size $h = 1000$, and $k \in \{10, 20, 30, 50, 100\}$.

23.2 Approximate Index Augmentation

We repeat the sensitivity analysis and show the caching gain when the different policies are augmented with an approximate index, and are allowed to mix the best object that can be served locally and from the server. Figure 23.8 shows the caching gain for the different caching policies and different values of the cache size $h \in \{50, 100, 200, 500, 1000\}$ and $k = 10$. Figure 23.9 shows the caching gain for the different caching policies and different values of the retrieval cost c_f, that is taken as the average distance to the i-th neighbor, $i \in \{2, 50, 100, 500, 1000\}$. The cache size is $h = 1000$ and $k = 10$. Figure 23.10 shows the caching gain for the different caching policies and different values of $k \in \{10, 20, 30, 50, 100\}$ and $h = 1000$.

23.3 Compute Time

We provide a comparison of the compute time of the different algorithms in Figure 23.11. When the different LRU-like policies are not augmented with a global catalog index in Figure 23.11 (a), AÇAI experiences a higher compute time per iteration. When the different LRU-like policices are augmented with a global catalog index in Figure 23.11 (b), AÇAI has similar compute time to the different policies except for the simple vanilla lru policy. Nonetheless, in both settings AÇAI compute time remains comparable and approximately within factor 4 w.r.t the compute time of the different policies. Due to space limitation we cannot add this figure to the main text.



(a) SIFT1M trace (b) Amazon trace

Figure 23.8: Caching gain of the different policies when augmented with the approximate index, for different cache sizes $h \in \{50, 100, 200, 500, 1000\}$ and $k = 10$.

24 Submodularity of the Gain Function

As submodularity is defined for set functions, let us associate to the gain function $G(\cdot)$ an opportune set function as follows. Given a set $S \subseteq \mathcal{V} \times \mathcal{M}$ of pairs $(v, m) \in \mathcal{V} \times \mathcal{M}$ (nodes and models), we define the corresponding associated vector $\boldsymbol{x}(S)$ with $x_m^v(S) = 1$ if $((v, m) \in S \vee \omega_m^v = 1)$, $x_m^v(S) = 0$ otherwise. Now we can define the set function $f_t \colon 2^{\mathcal{V}\times\mathcal{M}} \to \mathbb{R}$:

$$f_t(S) \triangleq G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}(S)). \tag{24.201}$$

We can also define the set function $F_T : 2^{\mathcal{V}\times V} \to \mathbb{R}$ associated to the time-averaged gain in Eq. (4.14) as

$$F_T(S) \triangleq \frac{1}{T}\sum_{t=1}^{T} f_t(S) = \frac{1}{T}\sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}(S)). \tag{24.202}$$

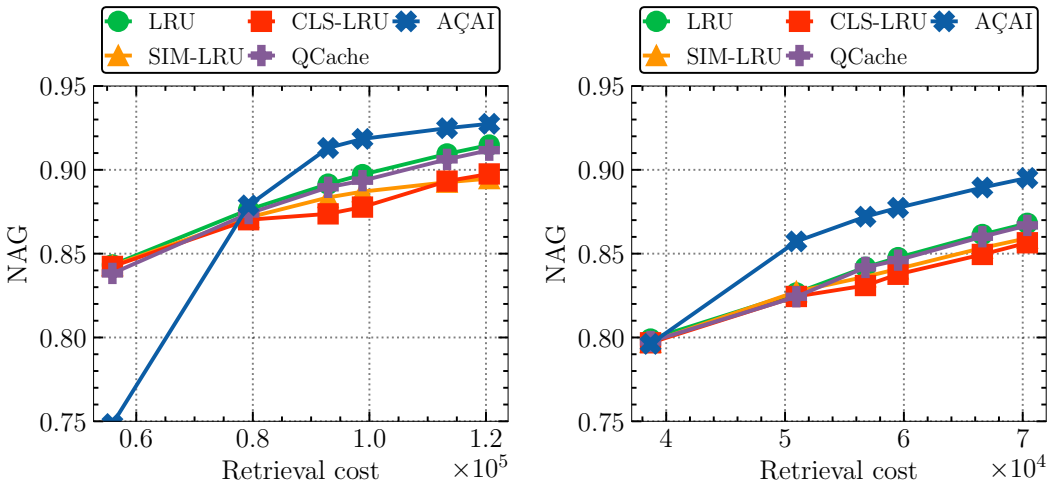
We first start proving that f_t is submodular in the following lemma.

Lemma 24.1. *The set function $f_t \colon 2^{\mathcal{V}\times\mathcal{M}} \to \mathbb{R}$ in Eq. (24.201) is normalized (i.e., $f_t(\emptyset) = 0$), submodular, and monotone.*

Proof.

Normalization. The constructed set function is normalized (as in $f_t(\emptyset) = 0$), we have

$$f_t(\emptyset) = G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}(\emptyset)) = G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = C(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) - C(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0. \tag{24.203}$$



(a) SIFT1M trace (b) Amazon trace

Figure 23.9: Caching gain for the different policies and different retrieval cost when augmented with the approximate index. The retrieval cost c_f is taken as the average distance to the i-th neighbor, $i \in \{2, 10, 50, 100, 500, 1000\}$. The cache size is $h = 1000$ and $k = 10$.

Submodularity. A function $f_t \colon 2^{\mathcal{V}\times\mathcal{M}} \to \mathbb{R}$ is submodular [177] if for every $S' \subset S'' \subset \mathcal{V} \times \mathcal{M}$ and $(\bar{v}, \bar{m}) \in (\mathcal{V} \times \mathcal{M}) \setminus S''$ it holds that

$$f_t(S'' \cup \{(\bar{v}, \bar{m})\}) - f_t(S'') \leq f_t(S' \cup \{(\bar{v}, \bar{m})\}) - f_t(S').$$

Let us consider $(\bar{v}, \bar{m}) \in (\mathcal{V} \times \mathcal{M}) \setminus S''$. We take $\boldsymbol{x}'$, $\boldsymbol{x}''$, $\bar{\boldsymbol{x}}'$, and $\bar{\boldsymbol{x}}''$ as short hand notation for $\boldsymbol{x}(S')$, $\boldsymbol{x}(S'')$, $\boldsymbol{x}(S' \cup \{(\bar{v}, \bar{m})\})$, and $\boldsymbol{x}(S'' \cup \{(\bar{v}, \bar{m})\})$, respectively.

Since $S' \subset S''$, we have that if ${x'}^v_m = 1 \implies {x''}^v_m = 1$, and thus $z^k_\rho(\boldsymbol{l}_t, \boldsymbol{x}') \leq z^k_\rho(\boldsymbol{l}_t, \boldsymbol{x}'')$, for any k (see Eq. (4.11)). Therefore

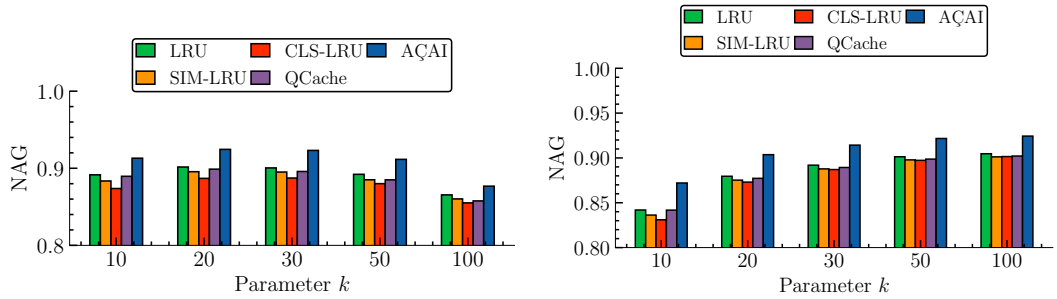
$$Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}') \leq Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}''), \forall k \in [K_\rho - 1], \forall \rho \in \mathcal{R}. \tag{24.204}$$

Let us denote $\bar{k}_\rho \triangleq \kappa_\rho(\bar{v}, \bar{m})$. Due to Eq. (24.204), $\forall k \in [K_\rho - 1], \forall \rho \in \mathcal{R}$ the following inequality holds:

$$\min\{r^t_\rho - Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}''), \lambda^{\bar{k}_\rho}_\rho(\boldsymbol{l}_t)\} \leq \min\{r^t_\rho - Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'), \lambda^{\bar{k}_\rho}_\rho(\boldsymbol{l}_t)\}, \tag{24.205}$$

or equivalently:

$$\min\{r^t_\rho, Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') + \lambda^{\bar{k}_\rho}_\rho(\boldsymbol{l}_t)\} - Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') \leq \min\{r^t_\rho, Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}') + \lambda^{\bar{k}_\rho}_\rho(\boldsymbol{l}_t)\} - Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'). \tag{24.206}$$



(a) SIFT1M trace (b) Amazon trace

Figure 23.10: Caching gain for the different policies when augmented with the approximate index. The cache size is $h = 1000$, and $k \in \{10, 20, 30, 50, 100\}$.

Observe that

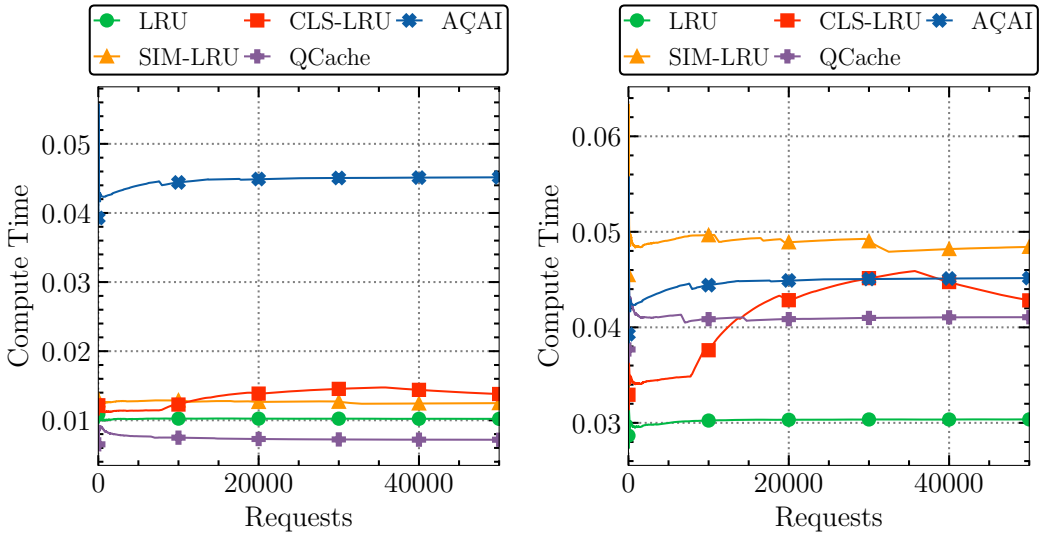
$$z_\rho^k(\boldsymbol{l}_t, \bar{\boldsymbol{x}}'') = \begin{cases} z_\rho^k(\boldsymbol{l}_t, \boldsymbol{x}'') & \text{if } k \neq \bar{k}_\rho, \\ z_\rho^k(\boldsymbol{l}_t, \boldsymbol{x}'') + \underbrace{x_{\bar{m}}^{\bar{v}}}_{=1} \cdot l_{\rho,\bar{m}}^{t,\bar{v}} \overset{(4.11)}{=} z_\rho^k(\boldsymbol{l}_t, \boldsymbol{x}'') + \lambda_\rho^{\bar{k}_\rho}(\boldsymbol{l}_t) & \text{if } k = \bar{k}_\rho, \end{cases}$$

and thus

$$Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \bar{\boldsymbol{x}}'') = \begin{cases} Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') & \text{if } k < \bar{k}_\rho \\ \min\left\{r_\rho^t, \sum_{k'=1}^k z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}'') + \lambda_\rho^{\bar{k}_\rho}(\boldsymbol{l}_t)\right\} = \min\{r_\rho^t, Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') + \lambda_\rho^{\bar{k}_\rho}(\boldsymbol{l}_t)\} & \text{if } k \geq \bar{k}_\rho. \end{cases} \tag{24.207}$$

Note that the same equality holds between $\bar{\boldsymbol{x}}'$ and $\boldsymbol{x}'$ since $(\bar{v}, \bar{m}) \notin S'$.

The marginal gain of adding pair $(\bar{v}, \bar{m}) \in \mathcal{V} \times \mathcal{M}$ to the allocation set S'' is



(a) (b)

Figure 23.11: Time-averaged compute time of AÇAI and the different policies LRU, SIM-LRU, CLS-LRU, and QCache(a) w/o an approximate global catalog index, and (b) w/ an approximate global catalog index. Experiment run over the Amazon trace, cache size $h = 1000$, parameter $k = 10$, $\eta = 10^{-4}$, retrieval cost c_f is set to be the distance to the 50-th closest neighbor in $\mathcal{N}$.

$$
\begin{aligned}
&f_t(S'' \cup \{(\bar{v}, \bar{m})\}) - f_t(S'') = G(\boldsymbol{r}_t, \boldsymbol{l}_t, \bar{\boldsymbol{x}}'') - G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') \\
&\overset{(4.16)}{=} \sum_{\rho \in \mathcal{R}} \Bigg[\sum_{k=1}^{\bar{k}_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})\right) \\
&\qquad + \sum_{k=\bar{k}_\rho}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \left(\min\left\{r_\rho^t, Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') + \lambda_\rho^{\bar{k}_\rho}(\boldsymbol{l}_t)\right\} - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})\right) \\
&\qquad - \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})\right) \Bigg] \\
&= \sum_{\rho \in \mathcal{R}} \sum_{k=\bar{k}_\rho}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \left[\left(\min\left\{r_\rho^t, Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') + \lambda_\rho^{\bar{k}_\rho}(\boldsymbol{l}_t)\right\} - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})\right) - \left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})\right) \right] \\
&= \sum_{\rho \in \mathcal{R}} \sum_{k=\bar{k}_\rho}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \left(\min\left\{r_\rho^t, Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') + \lambda_\rho^{\bar{k}_\rho}(\boldsymbol{l}_t)\right\} - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'')\right). \qquad (24.208)
\end{aligned}
$$

By bounding up each term of the marginal gain (24.208) as in (24.206) we get the following:

$$f_t(S'' \cup \{(\bar{v}, \bar{m})\}) - f_t(S'') \leq f_t(S' \cup \{(\bar{v}, \bar{m})\}) - f_t(S').$$

We conclude that f_t is a submodular set function.

Monotonicity. The function $f_t \colon 2^{\mathcal{V}\times\mathcal{M}} \to \mathbb{R}$ is monotone [177] if for every $S' \subset S'' \subset \mathcal{V}\times\mathcal{M}$ it holds that

$$f_t(S'') \geq f_t(S'). \tag{24.209}$$

We have

$$f_t(S'') - f_t(S') = G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') - G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}') \tag{24.210}$$

$$= \sum_{\rho\in\mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}'') - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}')\right) \tag{24.211}$$

$$\geq 0. \tag{24.212}$$

The last inequality is obtained using Eq. (24.204). We conclude that f_t is monotone.

□

Lemma 24.2. *The set function $F_T : 2^{\mathcal{V}\times\mathcal{M}} \to \mathbb{R}$ in Eq. (24.202) is normalized (i.e., $f_T(\emptyset) = 0$), submodular, and monotone.*

Proof.

The set function F_T is a nonnegative linear combination of submodular and monotone functions f_t (see Lemma 24.1), then F_T is also submodular and monotone [177]. Moreover, each f_t is normalized, then it follows that F_T is normalized.

□

25 Equivalent Expression of the Gain Function

Lemma 25.1. *Let us fix the threshold $c \in \mathbb{N} \cup \{0\}$, request type $\rho \in \mathcal{R}$, model rank $k \in [K_\rho]$, time slot t, load vector $\boldsymbol{l}_t$ and allocation vector $\boldsymbol{x}$. For brevity, let us denote $z_\rho^{k'} = z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x})$. The following formula holds:*

$$\min\left\{c, \sum_{k'=1}^{k} z_\rho^{k'}\right\} - \min\left\{c, \sum_{k'=1}^{k-1} z_\rho^{k'}\right\} = \min\left\{c - \sum_{k'=1}^{k-1} z_\rho^{k'}, z_\rho^k\right\} \cdot \mathbb{1}\left(\sum_{k'=1}^{k-1} z_\rho^{k'} < c\right). \tag{25.213}$$

Proof.

We distinguish two cases:

1. When the $k-1$ less costly models have at least c effective capacity, i.e., $\sum_{k'=1}^{k-1} z_\rho^{k'} \geq c$, we obtain:

$$\sum_{k'=1}^{k} z_\rho^{k'} = \sum_{k'=1}^{k-1} z_\rho^{k'} + z_\rho^k \geq c + z_\rho^k \geq c. \tag{25.214}$$

The last inequality is obtained using $z_\rho^k \geq 0$. Therefore, the left term of Eq. (25.213) becomes $c - c = 0$, and the indicator function of the right term becomes zero. Hence, Eq. (25.213) is verified in this case.

2. When $\sum_{k'=1}^{k-1} z_\rho^{k'} < c$, we obtain:

$$\min\left\{c, \sum_{k'=1}^{k} z_\rho^{k'}\right\} - \min\left\{c, \sum_{k'=1}^{k-1} z_\rho^{k'}\right\} = \min\left\{c, \sum_{k'=1}^{k} z_\rho^{k'}\right\} - \sum_{k'=1}^{k-1} z_\rho^{k'} = \min\left\{c - \sum_{k'=1}^{k-1} z_\rho^{k'}, z_\rho^{k}\right\}. \tag{25.215}$$

Hence Eq. (25.213) is verified, being the indicator function equal to 1 in this case.

By combining Eq. (25.214) and Eq. (25.215) we obtain Eq. (25.213).

□

Lemma 25.2. *The cost function given by Eq.* (4.12) *can be expressed as:*

$$C(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) = \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^k - \gamma_\rho^{k+1}\right) \min\left\{r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x})\right\} + \gamma_\rho^{K_\rho} r_\rho^t. \tag{25.216}$$

Proof.

The sum $\sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x})$ for $k = K_\rho$ surely includes a repository model as it sums all the models along the path of request type ρ; thus, we have $\sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \geq r_\rho^t$ (see Eq. (4.9)) and

$$\gamma_\rho^{K_\rho} \min\left\{r_\rho^t, \sum_{k'=1}^{K_\rho} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x})\right\} = \gamma_\rho^{K_\rho} r_\rho^t. \tag{25.217}$$

Now we use Lemma 25.1 to express the cost function in Eq. (4.12) as a sum of the difference of min functions.

$$C(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) = \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho} \gamma_\rho^k \cdot \min \left\{ r_\rho^t - \sum_{k'=1}^{k-1} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}), z_\rho^k(\boldsymbol{l}_t, \boldsymbol{x}) \right\} \cdot \mathbb{1} \left(\sum_{k'=1}^{k-1} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) < r_\rho^t \right) \tag{25.218}$$

$$= \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho} \gamma_\rho^k \left(\min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} - \min \left\{ r_\rho^t, \sum_{k'=1}^{k-1} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} \right) \tag{25.219}$$

$$\overset{(25.217)}{=} \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \gamma_\rho^k \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} - \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho} \gamma_\rho^k \min \left\{ r_\rho^t, \sum_{k'=1}^{k-1} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} + \gamma_\rho^{K_\rho} r_\rho^t \tag{25.220}$$

$$= \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \gamma_\rho^k \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} - \sum_{\rho \in \mathcal{R}} \sum_{k=2}^{K_\rho} \gamma_\rho^k \min \left\{ r_\rho^t, \sum_{k'=1}^{k-1} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} + \gamma_\rho^{K_\rho} r_\rho^t \tag{25.221}$$

$$= \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \gamma_\rho^k \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} - \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \gamma_\rho^{k+1} \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} + \gamma_\rho^{K_\rho} r_\rho^t \tag{25.222}$$

$$= \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^k - \gamma_\rho^{k+1} \right) \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} + \gamma_\rho^{K_\rho} r_\rho^t. \tag{25.223}$$

□

Proof of Lemma 4.3.1.
Proof.

By using the expression Eq. (25.216) for a generic allocation vector $\boldsymbol{x}$ and for $\boldsymbol{\omega}$, we obtain:

$$G(\boldsymbol{r}, \boldsymbol{l}_t, \boldsymbol{x}) = C(\boldsymbol{r}, \boldsymbol{l}_t, \boldsymbol{\omega}) - C(\boldsymbol{r}, \boldsymbol{l}_t, \boldsymbol{x}) \tag{25.224}$$

$$= \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^k - \gamma_\rho^{k+1} \right) \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{\omega}) \right\} - \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^k - \gamma_\rho^{k+1} \right) \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} \tag{25.225}$$

$$= \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^k - \gamma_\rho^{k+1} \right) \cdot \left\{ \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{\omega}) \right\} - \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} \right\} \tag{25.226}$$

$$= \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k \right) \cdot \left\{ \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{x}) \right\} - \min \left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{\omega}) \right\} \right\} \tag{25.227}$$

$$= \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k \right) \left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) \right). \tag{25.228}$$

□

26 Projection Algorithm

In order to project a fractional allocation $\boldsymbol{y}'$ lying outside the constraint set $\mathcal{Y}$ to a feasible allocation $\boldsymbol{y}$, we perform a Bregman projection associated to the global mirror map $\Phi : \mathcal{D} \rightarrow \mathbb{R}$, where the Bregman divergence associated to the mirror map $\Phi : \mathcal{D} \rightarrow \mathbb{R}$ is given by

$$D_{\Phi}(\boldsymbol{y}, \boldsymbol{y}') = \Phi(\boldsymbol{y}) - \Phi(\boldsymbol{y}') - \nabla\Phi(\boldsymbol{y}')^{T}(\boldsymbol{y} - \boldsymbol{y}'), \tag{26.229}$$

and the Bregman divergences associated to the mirror maps $\Phi^{v} : \mathcal{D}^{v} \rightarrow \mathbb{R}$ are also given by

$$D_{\Phi^{v}}(\boldsymbol{y}^{v}, \boldsymbol{y}'^{v}) = \Phi^{v}(\boldsymbol{y}^{v}) - \Phi^{v}(\boldsymbol{y}'^{v}) - \nabla\Phi^{v}(\boldsymbol{y}'^{v})^{T}(\boldsymbol{y}^{v} - \boldsymbol{y}'^{v}). \tag{26.230}$$

The projection operation yields a constrained minimization problem, i.e.,

$$\boldsymbol{y} = \prod\nolimits^{\Phi}_{\mathcal{Y}\cap\mathcal{D}}(\boldsymbol{y}') = \operatorname*{argmin}_{\boldsymbol{y}\in\mathcal{Y}\cap\mathcal{D}} D_{\Phi}(\boldsymbol{y}, \boldsymbol{y}'). \tag{26.231}$$

The global mirror map $\Phi : \mathcal{D} \rightarrow \mathbb{R}$ is defined as the sum of the weighted negative entropy maps $\Phi^{v} : \mathcal{D}^{v} \rightarrow \mathbb{R}$, where $\mathcal{D} = \mathbb{R}_{+}^{\mathcal{V}\times\mathcal{M}}$ is the domain of Φ, and the set $\mathcal{D}^{v} = \mathbb{R}_{+}^{\mathcal{V}}$ is the domain of Φ^{v} for all $v \in \mathcal{V}$. Thus, it follows that the global Bregman divergence is the sum of the Bregman divergences local to each node $v \in \mathcal{V}$, i.e.,

$$D_{\Phi}(\boldsymbol{y}, \boldsymbol{y}') = \sum_{v\in\mathcal{V}} D_{\Phi^{v}}(y^{v}, y'^{v}) \tag{26.232}$$

where $\boldsymbol{y} \in \mathcal{Y} = \bigtimes_{v\in\mathcal{V}} \mathcal{Y}^{v}$, and $\boldsymbol{y}^{v} \in \mathcal{Y}^{v}, \forall v \in \mathcal{V}$. In order to minimize the value $D_{\Phi}(\boldsymbol{y}, \boldsymbol{y}')$ for $\boldsymbol{y} \in \mathcal{Y} \cap \mathcal{D}$, we can independently minimize the values $D_{\Phi^{v}}(y^{v}, y'^{v})$ for $\boldsymbol{y}^{v} \in \mathcal{Y}^{v} \cap \mathcal{D}^{v}$ giving $|\mathcal{V}|$ subproblems; for every $v \in \mathcal{V}$ we perform the following projection

$$\boldsymbol{y}^{v} = \prod\nolimits^{\Phi^{v}}_{\mathcal{Y}^{v}\cap\mathcal{D}^{v}}(\boldsymbol{y}'^{v}) = \operatorname*{argmin}_{\boldsymbol{y}^{v}\in\mathcal{Y}^{v}\cap\mathcal{D}^{v}} D_{\Phi^{v}}(\boldsymbol{y}^{v}, \boldsymbol{y}'^{v}). \tag{26.233}$$

Theorem 26.1. *Algorithm .2 when executed at node $v \in \mathcal{V}$ returns $\Pi^{\Phi^{v}}_{\mathcal{Y}^{v}\cap\mathcal{D}^{v}}(\boldsymbol{y}'^{v})$, i.e., the projection of the vector $\boldsymbol{y}'^{v}$ onto the weighted capped simplex $\mathcal{Y}^{v} \cap \mathcal{D}^{v}$ under the weighted negative entropy $\Phi^{v}(\boldsymbol{y}^{v}) = \sum_{m\in\mathcal{M}} s^{v}_{m} y^{v}_{m} \log(y^{v}_{m})$. The time complexity of the projection is $O\left(|\mathcal{M}| \log(|\mathcal{M}|)\right)$.*

Proof.

$$\prod\nolimits^{\Phi^{v}}_{\mathcal{Y}^{v}\cap\mathcal{D}^{v}}(\boldsymbol{y}'^{v}) = \operatorname*{argmin}_{\boldsymbol{y}^{v}\in\mathcal{Y}^{v}\cap\mathcal{D}^{v}} D_{\Phi^{v}}(\boldsymbol{y}^{v}, \boldsymbol{y}'^{v}) \tag{26.234}$$

$$= \operatorname*{argmin}_{\boldsymbol{y}^{v}\in\mathcal{Y}^{v}\cap\mathcal{D}^{v}} \sum_{m\in\mathcal{M}} s^{v}_{m}\left(y^{v}_{m}\log\left(\frac{y^{v}_{m}}{y'^{v}_{m}}\right) - y^{v}_{m} + y'^{v}_{m}\right). \tag{26.235}$$

Algorithm .2 Weighted negative entropy Bregman projection onto the weighted capped simplex

Require: $|\mathcal{M}|$; b^v; $\boldsymbol{s}^v$; Sorted $\boldsymbol{y'}^v$ where $y'^v_{|\mathcal{M}|} \geq \cdots \geq y'^v_1$

$y'^v_{|\mathcal{M}|+1} \leftarrow +\infty$
for $k \in \{|\mathcal{M}|, |\mathcal{M}|-1, \ldots, 1\}$ **do**
  $m_k \leftarrow \frac{b^v - \sum_{m=k+1}^{|\mathcal{M}|} s^v_m}{\sum_{m=1}^{k} s^v_m y'^v_m}$
  if $y'^v_k m_k < 1 \leq y'^v_{k+1} m_k$ **then** ▷ Appropriate k is found
    for $k' \in \{1, 2, \ldots, k\}$ **do**
      $y^v_{k'} \leftarrow m_k y'^v_{k'}$ ▷ Scale the variable's components
    end for
    for $k' \in \{k+1, k+2, \ldots, |\mathcal{M}|\}$ **do**
      $y^v_{k'} \leftarrow 1$ ▷ Cap the variable's components to 1
    end for
    return $\boldsymbol{y}^v$ ▷ $\boldsymbol{y}^v$ is the result of the projection
  end if
end for

We adapt the negative entropy projection algorithm in [139]. The constraints $y^v_m > 0, \forall m \in \mathcal{M}$ are implicitly enforced by the negentropy mirror map Φ^v and $D_{\Phi^v}(\boldsymbol{y}^v, \boldsymbol{y'}^v)$ is convex in $\boldsymbol{y}^v$. The Lagrangian function of the above problem:

$$\mathcal{J}(\boldsymbol{y}^v, \beta, \tau) = \sum_{m\in\mathcal{M}} s^v_m \left(y^v_m \log\left(\frac{y^v_m}{y'^v_m}\right) - y^v_m + y'^v_m \right) - \sum_{m\in\mathcal{M}} \beta_m \left(1 - y^v_m\right) - \tau \left(\sum_{m\in\mathcal{M}} s^v_m y^v_m - b^v \right). \quad (26.236)$$

At optimal point $\hat{\boldsymbol{y}}^v$ the following KKT conditions hold:

$$s^v_m \log(\hat{y}^v_m) - s^v_m \log(y'^v_m) + \beta_m - s^v_m \tau = 0, \quad (26.237a)$$
$$\hat{y}^v_m \leq 1, \quad (26.237b)$$
$$\beta_m \geq 0, \quad (26.237c)$$
$$\sum_{m\in\mathcal{M}} s^v_m \hat{y}^v_m = b^v, \quad (26.237d)$$
$$\beta_m (1 - \hat{y}^v_m) = 0. \quad (26.237e)$$

Without loss of generality, assume the components of $\hat{\boldsymbol{y}}^v$ are in non-decreasing order. Let k be the index of the largest component of $\hat{\boldsymbol{y}}^v$ strictly smaller than , i.e.,

$$\hat{y}^v_1 \leq \ldots \leq \hat{y}^v_k < \hat{y}^v_{k+1} = \ldots = \hat{y}^v_{|\mathcal{M}|} = 1 \ \text{ if } k < |\mathcal{M}|, \quad (26.238)$$
$$\hat{y}^v_1 \leq \hat{y}^v_2 \leq \ldots \leq \hat{y}^v_{|\mathcal{M}|} < 1 \ \text{ if } k = |\mathcal{M}|. \quad (26.239)$$

The goal here is to identify a valid value for k (number of components of $\hat{\boldsymbol{y}}^v$ different from 1) and τ. For now assume that τ is known, so a valid $k \in \mathcal{M}$ should satisfy the following:

- For $m_L = 1, ..., k$, we have from (26.237e) that $\beta_{m_L} = 0$, and then from (26.237a), $s^v_{m_L}\log(y'^v_{m_L}) + s^v_{m_L}\tau = s^v_{m_L}\log(\hat{y}^v_{m_L}) < s^v_{m_L}\log(1) = 0$, and can be simplified to

$$y'^v_{m_L} e^\tau < 1, \forall m_L \in \{1, \dots, k\}. \tag{26.240}$$

- For $m_U = k+1, ..., |\mathcal{M}|$: as $\beta_{m_U} \geq 0$ from (26.237c), we get $0 = s^v_{m_U}\log(\hat{y}^v_{m_U}) = s^v_{m_U}\log(y'^v_{m_U}) - \beta_{m_U} + s^v_{m_U}\tau \leq s^v_{m_U}\log(y'^v_{m_U}) + s^v_{m_U}\tau$, and can be simplified to

$$y'^v_{m_U} e^\tau \geq 1, \forall m_U \in \{k+1, \cdots, |\mathcal{M}|\}. \tag{26.241}$$

Consider Eqs. (26.238) and (26.239), and since for $m_L \in \{1, \dots, k\}$ we have $y'^v_{m_L} e^\tau = \hat{y}^v_{m_L}$ (the order is preserved), then the conditions in Eq. (26.240) are

$$y'^v_1 e^\tau \leq ... \leq y'^v_k e^\tau < 1. \tag{26.242}$$

If the components of $\boldsymbol{y}'^v$ are ordered in ascending order, then it is enough to check if $y'^v_k e^\tau < 1$ holds for Eq. (26.242) to be true. Moreover, for $m_U \in \{k+1, \dots, |\mathcal{M}|\}$, we have $\hat{y}^v_{m_U} = 1$ and $y'^v_{m_U} e^\tau \geq 1$. Then, by taking $y'^v_{|\mathcal{M}|+1} \triangleq +\infty$ (k can be equal to $|\mathcal{M}|$ as in Eq. (26.239)) it is enough to check with the smallest $y'^v_{m_U}$ to summarize all the conditions in Eq. (26.241). Thus, all the needed conditions can be further simplified to:

$$y'^v_k e^\tau < 1 \leq y'^v_{k+1} e^\tau.$$

Note that the r.h.s inequality is ignored when $k = |\mathcal{M}|$ by construction ($y'^v_{|\mathcal{M}|+1} = +\infty$).

Now we established how to verify if a given $k \in \mathcal{M}$ is valid, what remains is to give the expression of τ using the knapsack constraint in Eq. (26.237d):

$$b^v = \sum_{m=1}^{|\mathcal{M}|} s^v_m \hat{y}^v_m = \sum_{m=k+1}^{|\mathcal{M}|} s^v_m + e^\tau \sum_{m=1}^{k} s^v_m y'^v_m.$$

For a given $k \in \mathcal{M}$, we define

$$m_k \triangleq e^\tau = \frac{b^v - \sum_{m=k+1}^{|\mathcal{M}|} s^v_m}{\sum_{m=1}^{k} s^v_m y'^v_m}. \tag{26.243}$$

Thus, a valid k is the value satisfying the following inequalities (line 7 of Algorithm .2):

$$y'^v_k m_k < 1 \leq y'^v_{k+1} m_k. \tag{26.244}$$

The appropriate k satisfying the KKT conditions is contained in $\mathcal{M}$, and due to the sorting operation this gives total time complexity of $O(|\mathcal{M}|\log(|\mathcal{M}|))$ per iteration. In practice, the online mirror ascent method quickly sets irrelevant items in the fractional allocation vector $\boldsymbol{y}'^v$ very close to 0. Therefore, we can keep track only of items with a fractional value above a threshold $\epsilon > 0$, and the size of this subset is practically $\ll |\mathcal{M}|$. Therefore, the projection can be very efficient in practice.

□

27 Subgradient Expression

Lemma 27.1. *The gain function in Eq.* (4.16) *has a subgradient* $\boldsymbol{g}_t$ *at point* $\boldsymbol{y}_t \in \mathcal{Y}$ *given by*

$$\boldsymbol{g}_t = \left[\sum_{\rho \in \mathcal{R}} l^{t,v}_{\rho,m} \left(\gamma_\rho^{K^*_\rho(\boldsymbol{y}_t)} - C^v_{\boldsymbol{p},m} \right) \mathbb{1}\left(\kappa_\rho(v,m) < K^*_\rho(\boldsymbol{y}_t) \right) \right]_{(v,m)\in\mathcal{V}\times\mathcal{M}}, \tag{27.245}$$

where $K^*_\rho(\boldsymbol{y}_t) = \min\left\{ k \in [K_\rho - 1] : \sum_{k'=1}^{k} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) \geq r^t_\rho \right\}$.

Proof.

The function given by $Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) = \min\left\{ r^t_\rho, \sum_{k'=1}^{k} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) \right\}$ is a minimum of two concave differentiable functions (a constant, and a linear function). We can characterize its subdifferential (set of all possible subgradients), using [296, Theorem 8.2], at point $\boldsymbol{y}_t \in \mathcal{Y}$ as

$$\partial Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) = \begin{cases} \left\{ \nabla\left(\sum_{k'=1}^{k} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) \right) \right\} & \text{if } Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) < r^t_\rho, \\ \operatorname{conv}\left(\left\{ \boldsymbol{0}, \nabla\left(\sum_{k'=1}^{k} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) \right) \right\} \right) & \text{if } Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) = r^t_\rho, \\ \{\boldsymbol{0}\} & \text{otherwise}, \end{cases} \tag{27.246}$$

where $\operatorname{conv}(\cdot)$ is the convex hull of a set, and the gradient ∇ is given by $\nabla(\cdot) = [\frac{\partial}{\partial y^v_m}(\cdot)]_{(v,m)\in\mathcal{V}\times\mathcal{M}}$. The operator $\frac{\partial}{\partial y^v_m}(\cdot)$ is the partial derivative w.r.t y^v_m (not to be confused with the subdifferential notation).

We restrict ourselves to the valid subgradient $\tilde{\boldsymbol{g}}^k_{\rho,t} \in \partial Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t)$ given by

$$\tilde{\boldsymbol{g}}^k_{\rho,t} = \begin{cases} \nabla\left(\sum_{k'=1}^{k} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) \right) & \text{if } Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) < r^t_\rho, \\ \boldsymbol{0}, & \text{otherwise.} \end{cases} \tag{27.247}$$

Note that for every $(v,m) \in \mathcal{V}\times\mathcal{M}$ we have

$$\frac{\partial}{\partial y^v_m} \sum_{k'=1}^{k} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) = \sum_{k'=1}^{k} \frac{\partial}{\partial y^v_m} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) \overset{(4.11)}{=} l^{t,v}_{\rho,m} \cdot \mathbb{1}\left(\kappa_\rho(v,m) \leq k \right).$$

The indicator variable $\mathbb{1}\left(\kappa_\rho(v,m) \leq k\right)$ is introduced since the partial derivative of $\sum_{k'=1}^{k} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t)$ w.r.t. y^v_m is non-zero only if model m at node v is among the k best models to serve requests of type ρ (in this case, the variable y^v_m appears once in the summation).

We obtain from Eq. (27.247)

$$\tilde{g}^{k,v}_{\rho,t,m} = \begin{cases} l^{t,v}_{\rho,m} \cdot \mathbb{1}\left(\kappa_\rho(v,m) \leq k \right) & \text{if } Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) < r^t_\rho, \\ 0 & \text{otherwise.} \end{cases} \tag{27.248}$$

$$= l^{t,v}_{\rho,m} \cdot \mathbb{1}\left(\kappa_\rho(v,m) \leq k \ \wedge \ Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) < r^t_\rho \right), \forall (v,m) \in \mathcal{V}\times\mathcal{M}. \tag{27.249}$$

By considering the subdifferential

$$\partial G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) = \partial \left(\sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k \right) \left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) \right) \right), \tag{27.250}$$

and using [297, Theorem 23.6], we get

$$\partial G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) = \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \partial \left(\left(\gamma_\rho^{k+1} - \gamma_\rho^k \right) \left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) \right) \right). \tag{27.251}$$

The constant factors $\left(\gamma_\rho^{k+1} - \gamma_\rho^k \right)$ are non-negative, so we can multiply both sides of the subgradient inequality by a non-negative constant [297, Section 23]; furthermore, the subgradient of the constants $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})$ is $\mathbf{0}$. We get

$$\partial G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) = \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k \right) \partial \left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) \right) \tag{27.252}$$

$$= \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k \right) \partial Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}). \tag{27.253}$$

Then, a subgradient $\boldsymbol{g}_t \in \partial G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t)$ at point $\boldsymbol{y}_t \in \mathcal{Y}$ is given by

$$\boldsymbol{g}_t = \sum_{\rho \in \mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k \right) \tilde{\boldsymbol{g}}_{\rho,t}^k. \tag{27.254}$$

The (v, m)-th component of the subgradient $\boldsymbol{g}_t$ is

$$g^v_{t,m} = \sum_{\rho\in\mathcal{R}} \sum_{k=1}^{K_\rho-1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \cdot \tilde{g}^{k,v}_{\rho,t,m} \tag{27.255}$$

$$= \sum_{\rho\in\mathcal{R}} \sum_{k=1}^{K_\rho-1} l^{t,v}_{\rho,m} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \cdot \mathbb{1}\left(\kappa_\rho(v,m) \le k \ \wedge \ Z^k_\rho(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) < r^t_\rho\right) \tag{27.256}$$

$$= \sum_{\rho\in\mathcal{R}} \sum_{k=\kappa_\rho(v,m)}^{K_\rho-1} l^{t,v}_{\rho,m} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \cdot \mathbb{1}\left(\sum_{k'=1}^{k} z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) < r^t_\rho\right) \tag{27.257}$$

$$= \sum_{\rho\in\mathcal{R}} \sum_{k=\kappa_\rho(v,m)}^{K^*_\rho(\boldsymbol{y}_t)-1} l^{t,v}_{\rho,m} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \tag{27.258}$$

$$= \sum_{\rho\in\mathcal{R}} l^{t,v}_{\rho,m} \cdot \sum_{k=\kappa_\rho(v,m)}^{K^*_\rho(\boldsymbol{y}_t)-1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \tag{27.259}$$

$$= \sum_{\rho\in\mathcal{R}} l^{t,v}_{\rho,m} \left(\gamma_\rho^{K^*_\rho(\boldsymbol{y}_t)} - \gamma_\rho^{\kappa_\rho(v,m)}\right) \cdot \mathbb{1}\left(\kappa_\rho(v,m) < K^*_\rho(\boldsymbol{y}_t)\right) \tag{27.260}$$

$$\overset{(4.11)}{=} \sum_{\rho\in\mathcal{R}} l^{t,v}_{\rho,m} \left(\gamma_\rho^{K^*_\rho(\boldsymbol{y}_t)} - C^v_{\boldsymbol{p},m}\right) \cdot \mathbb{1}\left(\kappa_\rho(v,m) < K^*_\rho(\boldsymbol{y}_t)\right), \forall (v,m) \in \mathcal{V}\times\mathcal{M}, \tag{27.261}$$

where $K^*_\rho(\boldsymbol{y}_t) = \min\left\{k \in [K_\rho - 1] : \sum_{k'=1}^k z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{y}_t) \ge r^t_\rho\right\}$.

□

28 Supporting Lemmas for the Proof of Theorem 4.5.1

28.1 Concavity of the Gain Function

Lemma 28.1. *The gain function given by Eq.* (4.16) *is concave over its domain $\mathcal{Y}$ of possible fractional allocations.*

Proof.

Since λ^k_ρ is defined to be the k-th smallest cost for any $k \in [K_\rho]$ (see Eq. (4.11)), then the factors $\gamma^{k+1}_\rho - \gamma^k_\rho$ are always non-negative. Moreover $z^k_\rho(\boldsymbol{l}_t, \boldsymbol{y}) = y^v_m l^{t,v}_{\rho,m}$, where v, m are such that $\kappa_\rho(v,m) = k$. Therefore, $Z^k_\rho(\boldsymbol{l}_t, \boldsymbol{y})$ is the minimum between a constant r^t_ρ and a sum $\sum_{k'=1}^k z^{k'}_\rho(\boldsymbol{l}_t, \boldsymbol{x})$ of linear functions of $\boldsymbol{y}$. Such minimum is thus a concave function of $\boldsymbol{y}$. Therefore, the gain in Eq. (4.16) is a weighted sum with positive weights of concave functions in $\boldsymbol{y}$, which is concave.

□

28.2 Strong convexity of the Mirror Map

Lemma 28.2. *The global mirror map* $\Phi(\boldsymbol{y}) = \sum_{v\in\mathcal{V}} \Phi(\boldsymbol{y}^v) = \sum_{v\in\mathcal{V}} \sum_{m\in\mathcal{M}} s_m^v y_m^v \log(y_m^v)$ *defined over the domain* $\mathcal{D} = \mathbb{R}_{>0}^{|\mathcal{M}|\times|\mathcal{V}|}$ *is* θ*-strongly convex w.r.t. the norm* $\|\cdot\|_{l_1(\boldsymbol{s})}$ *over* $\mathcal{Y}\cap\mathcal{D}$*, where*

$$\theta \triangleq \frac{1}{s_{\max}|\mathcal{V}||\mathcal{M}|}, \tag{28.262}$$

$$\|\boldsymbol{y}\|_{l_1(\boldsymbol{s})} \triangleq \sum_{(v,m)\in\mathcal{V}\times\mathcal{M}} s_m^v |y_m^v| \quad \textit{(weighted } l_1 \textit{ norm)}, \tag{28.263}$$

and $s_{\max} \triangleq \left\{s_m^v : (v,m)\in\mathcal{V}\times\mathcal{M}\right\}$ *is the maximum model size. In words, this means that the mirror map* Φ*'s growth is lower bounded by a quadratic with curvature* $\frac{1}{s_{\max}|\mathcal{V}||\mathcal{M}|}$.

Proof.

We extend the proof of the strong convexity of the negative entropy w.r.t. to the l_1 norm over the simplex given in [298, Lemma 16]. The map $\Phi(\boldsymbol{y})$ is differentiable over $\mathcal{Y}\cap\mathcal{D}$, so a sufficient (and also necessary) condition for $\Phi(\boldsymbol{y})$ to be θ-strongly convex w.r.t. $\|\cdot\|_{l_1(\boldsymbol{s})}$ is:

$$(\nabla\Phi(\boldsymbol{y'}) - \nabla\Phi(\boldsymbol{y}))^T(\boldsymbol{y'}-\boldsymbol{y}) \geq \theta \left\|\boldsymbol{y'}-\boldsymbol{y}\right\|_{l_1(\boldsymbol{s})}^2, \quad \forall \boldsymbol{y'},\boldsymbol{y}\in\mathcal{Y}\cap\mathcal{D}. \tag{28.264}$$

We have

$$(\nabla\Phi(\boldsymbol{y'}) - \nabla\Phi(\boldsymbol{y}))^T(\boldsymbol{y'}-\boldsymbol{y}) = \sum_{(v,m)\in\mathcal{V}\times\mathcal{M}} s_m^v(\log(y'^v_m) - \log(y_m^v))(y'^v_m - y_m^v). \tag{28.265}$$

Take $\mu_m^v \triangleq s_m^v(\log(y'^v_m) - \log(y_m^v))(y'^v_m - y_m^v)$, and note that $\mu_m^v \geq 0$ (because log is an increasing function).

$$\begin{aligned}
\left\|\boldsymbol{y'}-\boldsymbol{y}\right\|_{l_1(\boldsymbol{s})}^2 &= \left(\sum_{(v,m)\in\mathcal{V}\times\mathcal{M}} s_m^v|y'^v_m - y_m^v|\right)^2 = \left(\sum_{(v,m)\in\mathcal{V}\times\mathcal{M}:\mu_m^v\neq 0} \sqrt{\mu_m^v}\frac{s_m^v|y'^v_m - y_m^v|}{\sqrt{\mu_m^v}}\right)^2 \\
&\leq \left(\sum_{(v,m)\in\mathcal{V}\times\mathcal{M}:\mu_m^v\neq 0} \mu_m^v\right)\left(\sum_{(v,m)\in\mathcal{V}\times\mathcal{M}:\mu_m^v\neq 0} (s_m^v)^2\frac{(y'^v_m - y_m^v)^2}{\mu_m^v}\right) \\
&= \left(\sum_{(v,m)\in\mathcal{V}\times\mathcal{M}:\mu_m^v\neq 0} s_m^v(\log(y'^v_m)-\log(y_m^v))(y'^v_m - y_m^v)\right)\left(\sum_{(v,m)\in\mathcal{V}\times\mathcal{M}:\mu_m^v\neq 0} s_m^v\frac{y'^v_m - y_m^v}{\log(y'^v_m)-\log(y_m^v)}\right).
\end{aligned}$$

The inequality is obtained using Cauchy–Schwarz inequality. Take $s_m^v \leq s_{\max}, \forall (v,m) \in \mathcal{V} \times \mathcal{M}$, we obtain:

$$\begin{aligned}
\sum_{(v,m)\in\mathcal{V}\times\mathcal{M}:\mu_m^v\neq 0} s_m^v \frac{y'^v_m - y_m^v}{\log(y'^v_m) - \log(y_m^v)} &\leq s_{\max} \sum_{(v,m)\in\mathcal{V}\times\mathcal{M}:\mu_m^v\neq 0} \frac{y'^v_m - y_m^v}{\log(y'^v_m) - \log(y_m^v)} \\
&= s_{\max} \sum_{v\in\mathcal{V}} \sum_{m\in\mathcal{M}:\mu_m^v\neq 0} \frac{y'^v_m - y_m^v}{\log(y'^v_m) - \log(y_m^v)} \\
&\leq s_{\max} \sum_{v\in\mathcal{V}} \sum_{m\in\mathcal{M}} \frac{y'^v_m + y_m^v}{2} \\
&\leq s_{\max}|\mathcal{V}||\mathcal{M}|.
\end{aligned}$$

The second inequality is shown in [298, Eq. (A.16)]. We find that $\forall \boldsymbol{y'}, \boldsymbol{y} \in \mathcal{Y} \cap \mathcal{D}$:

$$\frac{1}{s_{\max}|\mathcal{V}|U} \left\|\boldsymbol{y'} - \boldsymbol{y}\right\|^2_{l_1(\boldsymbol{s})} \leq \sum_{(v,m)\in\mathcal{V}\times\mathcal{M}:\mu_m^v\neq 0} s_m^v{}^v (\log(y'^v_m) - \log(y_m^v))(y'^v_m - y_m^v) \tag{28.266}$$

$$= (\nabla\Phi(\boldsymbol{y'}) - \nabla\Phi(\boldsymbol{y}))^T(\boldsymbol{y'} - \boldsymbol{y}). \tag{28.267}$$

The strong convexity constant θ is $\frac{1}{s_{\max}|\mathcal{V}||\mathcal{M}|}$.

□

28.3 Subgradient Bound

Lemma 28.3. *For any $(\boldsymbol{r}_t, \boldsymbol{l}_t) \in \mathcal{A}$, the subgradients $\boldsymbol{g}_t$ of the gain function in Eq. (4.16) at point $\boldsymbol{y}_t \in \mathcal{Y}$ are bounded under the norm $\|\cdot\|_{l_\infty(\frac{1}{\boldsymbol{s}})}$ by $\sigma = \frac{RL_{\max}\Delta_C}{s_{\min}}$, where $s_{\min} \triangleq \min\{s_m^v : \forall (v,m) \in \mathcal{V} \times \mathcal{M}\}$, $L_{\max} \triangleq \max\{L_m^v : \forall (v,m) \in \mathcal{V} \times \mathcal{M}\}$, $R = |\mathcal{R}|$, and $\Delta_C \triangleq \max\left\{\left(\sum_{m\in\mathcal{M}} \omega_{m'}^{v(\boldsymbol{p})} C_{\boldsymbol{p},m'}^{v(\boldsymbol{p})}\right) - C_{\boldsymbol{p},m}^v : \forall (i,\boldsymbol{p}) \in \mathcal{R}, (v,m) \in \boldsymbol{p} \times \mathcal{M}\right\}$ is the maximum serving cost difference between serving at a repository node $v(\boldsymbol{p})$ and at any other node $v \in \boldsymbol{p}$. The norm $\|\cdot\|_{l_\infty(\frac{1}{\boldsymbol{s}})}$ is defined as*

$$\|\boldsymbol{y}\|_{l_\infty(\frac{1}{\boldsymbol{s}})} \triangleq \max\left\{\frac{|y_m^v|}{s_m^v} : (v,m) \in \mathcal{V} \times \mathcal{M}\right\}. \tag{28.268}$$

Proof.

We have for any $t \in [T]$

$$\|\boldsymbol{g}_t\|_{l_\infty(\frac{1}{\boldsymbol{s}})} = \max\left\{\frac{|g_{t,m}^v|}{s_m^v}, \forall (v,m) \in \mathcal{V}\times\mathcal{M}\right\} \leq \max\left\{\frac{|g_{t,m}^v|}{s_{\min}}, \forall (v,m) \in \mathcal{V}\times\mathcal{M}\right\} \tag{28.269}$$

$$\overset{(4.18)}{\leq} \frac{L_{\max}}{s_{\min}} \max\left\{\sum_{\rho\in\mathcal{R}} \left(\gamma_\rho^{K_\rho^*(\boldsymbol{y}_t)} - C_{\boldsymbol{p},m}^v\right) \cdot \mathbb{1}\left(\kappa_\rho(v,m) < K_\rho^*(\boldsymbol{y}_t)\right), \forall (v,m) \in \mathcal{V}\times\mathcal{M}\right\} \tag{28.270}$$

$$\overset{(4.11)}{\leq} \frac{L_{\max} R}{s_{\min}} \max\{\gamma_\rho^{K_\rho} - \gamma_\rho^1, \forall \rho \in \mathcal{R}\} \leq \frac{L_{\max} R \Delta_C}{s_{\min}} = \sigma. \tag{28.271}$$

□

28.4 Dual Norm

Lemma 28.4. *$\|\cdot\|_{l_\infty(\frac{1}{\boldsymbol{s}})}$ is the dual norm of $\|\cdot\|_{l_1(\boldsymbol{s})}$ defined in* (28.268) *and* (28.263)*, respectively.*

Proof.

The dual norm $\|\cdot\|_*$ of $\|\cdot\|_{l_1(\boldsymbol{s})}$ is defined as (e.g., [53])

$$\|\boldsymbol{z}\|_* \triangleq \sup_{\boldsymbol{y}\in\mathbb{R}^{\mathcal{V}\times\mathcal{M}}} \left\{\boldsymbol{z}^T\boldsymbol{y} : \|\boldsymbol{y}\|_{l_1(\boldsymbol{s})} \leq 1\right\}, \forall \boldsymbol{z} \in \mathbb{R}^{\mathcal{V}\times\mathcal{M}}. \tag{28.272}$$

We thus need to show that $\|\boldsymbol{z}\|_{l_\infty(\frac{1}{\boldsymbol{s}})} = \sup_{\boldsymbol{y}\in\mathbb{R}^{\mathcal{V}\times\mathcal{M}}} \left\{\boldsymbol{z}^T\boldsymbol{y} : \|\boldsymbol{y}\|_{l_1(\boldsymbol{s})} \leq 1\right\}, \forall \boldsymbol{z} \in \mathbb{R}^{\mathcal{V}\times\mathcal{M}}$.

Consider any two vectors $\boldsymbol{y}$ and $\boldsymbol{z}$ in $\mathbb{R}^{\mathcal{V}\times\mathcal{M}}$. We have

$$\boldsymbol{z}^T\boldsymbol{y} = \sum_{(v,m)\in\mathcal{V}\times\mathcal{M}} y_m^v z_m^v = \sum_{(v,m)\in\mathcal{V}\times\mathcal{M}} (s_m^v y_m^v)\left(\frac{z_m^v}{s_m^v}\right) \leq \sum_{(v,m)\in\mathcal{V}\times\mathcal{M}} (s_m^v \cdot |y_m^v|)\left(\frac{|z_m^v|}{s_m^v}\right) \tag{28.273}$$

$$\leq \left(\textstyle\sum_{(v,m)\in\mathcal{V}\times\mathcal{M}} s_m^v |y_m^v|\right) \max\left\{\frac{|z_m^v|}{s_m^v} : (v,m) \in \mathcal{V}\times\mathcal{M}\right\} = \|\boldsymbol{y}\|_{l_1(\boldsymbol{s})} \|\boldsymbol{z}\|_{l_\infty(\frac{1}{\boldsymbol{s}})}. \tag{28.274}$$

Observe that

$$\boldsymbol{y}^T\boldsymbol{z} \leq \|\boldsymbol{z}\|_{l_\infty(\frac{1}{\boldsymbol{s}})}, \quad \forall \boldsymbol{y} : \|\boldsymbol{y}\|_{l_1(\boldsymbol{s})} \leq 1. \tag{28.275}$$

Let $(v_*, m_*) = \arg\max_{(v,m)\in\mathcal{V}\times\mathcal{M}} \left\{\frac{|z_m^v|}{s_m^v}\right\}$. The equality is achieved in (28.275) when $\boldsymbol{y}_* = \left[\frac{\text{sign}(z_m^v)}{s_m^v}\mathbb{1}\left((v,m) = (v_*,m_*)\right)\right]_{(v,m)\in\mathcal{V}\times\mathcal{M}}$. Note that $\|\boldsymbol{y}_*\|_{l_1(\boldsymbol{s})} = 1 \leq 1$, then the supremum in (28.272) is attained for $\boldsymbol{y} = \boldsymbol{y}_*$ and has value $\|\boldsymbol{z}\|_{l_\infty(\frac{1}{\boldsymbol{s}})}$; therefore, $\|\cdot\|_{l_\infty(\frac{1}{\boldsymbol{s}})}$ is the dual norm of $\|\cdot\|_{l_1(\boldsymbol{s})}$.

□

28.5 Bregman Divergence Bound

Lemma 28.5. *The value of the Bregman divergence $D_\Phi(\boldsymbol{y}, \boldsymbol{y}_1)$ in Eq. (26.229) associated with the mirror map $\Phi(\boldsymbol{y}) = \sum_{v\in\mathcal{V}} \Phi^v(\boldsymbol{y}^v) = \sum_{v\in\mathcal{V}} \sum_{m\in\mathcal{M}} s_m^v y_m^v \log(y_m^v)$ is upper bounded by a constant*

$$D_{\max} \triangleq \sum_{v\in\mathcal{V}} \min\{b^v, \|\boldsymbol{s}^v\|_1\} \log\left(\frac{\|\boldsymbol{s}^v\|_1}{\min\{b^v, \|\boldsymbol{s}^v\|_1\}}\right) \geq D_\Phi(\boldsymbol{y}, \boldsymbol{y}_1). \tag{28.276}$$

where $y_{1,m}^v = \frac{\min\{b^v, \|\boldsymbol{s}^v\|_1\}}{\|\boldsymbol{s}^v\|_1}, \forall (v,m) \in \mathcal{V}\times\mathcal{M}$ and $\boldsymbol{s}^v = [s_m^v]_{m\in\mathcal{M}}$ for every $v \in \mathcal{V}$.

Proof.

We prove that $\boldsymbol{y}_1^v$ is the minimizer of Φ^v over $\mathcal{Y}^v$. As Φ^v is convex over $\mathcal{Y}^v$ and differentiable in $\boldsymbol{y}_1^v$, $\boldsymbol{y}_1^v$ is a minimizer if and only if $\nabla\Phi^v(\boldsymbol{y}_1^v)^T(\boldsymbol{y}_1^v - \boldsymbol{y}) \leq 0, \forall \boldsymbol{y}^v \in \mathcal{Y}^v$ [53, Proposition 1.3] (first order optimality condition). Note that from the definition of $\mathcal{Y}^v$ (see Section 4.4) we have for any $\boldsymbol{y}^v \in \mathcal{Y}^v$

$$\sum_{m\in\mathcal{M}} s_m^v y_m^v = \min\{b^v, \|\boldsymbol{s}^v\|_1\}. \tag{28.277}$$

Let $c = \frac{\min\{b^v, \|\boldsymbol{s}^v\|_1\}}{\|\boldsymbol{s}^v\|_1}$, we get

$$\nabla\Phi^v(\boldsymbol{y}_1^v)^T(\boldsymbol{y}_1^v - \boldsymbol{y}) = \sum_{m\in\mathcal{M}} s_m^v(\log(c)+1)(c - y_m^v) = (\log(c)+1)(c\,\|\boldsymbol{s}^v\|_1 - \sum_{m\in\mathcal{M}} s_m^v y_m^v) \tag{28.278}$$

$$\overset{(28.277)}{=} (\log(c)+1)\left(c\,\|\boldsymbol{s}^v\|_1 - \sum_{m\in\mathcal{M}} s_m^v y_m^v\right) \tag{28.279}$$

$$= (\log(c)+1)\left(\min\{b^v, \|\boldsymbol{s}^v\|_1\} - \min\{b^v, \|\boldsymbol{s}^v\|_1\}\right) \tag{28.280}$$

$$= 0. \tag{28.281}$$

We confirmed that $\boldsymbol{y}_1^v$ is a minimizer of Φ^v over $\mathcal{Y}^v$. We have $\Phi^v(\boldsymbol{y}^v) \leq 0, \forall \boldsymbol{y}^v \in \mathcal{Y}^v$, and using the first order optimality condition we obtain

$$D_{\Phi^v}(\boldsymbol{y}^v, \boldsymbol{y}_1^v) \overset{(26.230)}{=} \Phi^v(\boldsymbol{y}^v) - \Phi^v(\boldsymbol{y}_1^v) + \nabla\Phi^v(\boldsymbol{y}_1^v)^T(\boldsymbol{y}_1^v - \boldsymbol{y}^v) \leq \Phi^v(\boldsymbol{y}^v) - \Phi^v(\boldsymbol{y}_1^v) \leq -\Phi^v(\boldsymbol{y}_1^v) \tag{28.282}$$

$$= \min\{b^v, \|\boldsymbol{s}^v\|_1\} \log\left(\frac{\|\boldsymbol{s}^v\|_1}{\min\{b^v, \|\boldsymbol{s}^v\|_1\}}\right). \tag{28.283}$$

Thus, we obtain

$$\sum_{v\in\mathcal{V}} D_{\Phi^v}(\boldsymbol{y}^v, \boldsymbol{y}_1^v) \overset{(26.232)}{=} D_\Phi(\boldsymbol{y}, \boldsymbol{y}_1) \leq \sum_{v\in\mathcal{V}} \min\{b^v, \|\boldsymbol{s}^v\|_1\} \log\left(\frac{\|\boldsymbol{s}^v\|_1}{\min\{b^v, \|\boldsymbol{s}^v\|_1\}}\right). \tag{28.284}$$

□

28.6 Bounds on the Gain Function

Upper and lower bounds on the gain function in Eq. (4.16) will be established using the following bounding function

$$\Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) \triangleq \sum_{\rho \in \operatorname{supp} \boldsymbol{r}^t} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) r_\rho^t \left(1 - \prod_{k'=1}^{k} \left(1 - z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})/r_\rho^t\right)\right) \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0\right), \forall \boldsymbol{y} \in \mathcal{X} \cup \mathcal{Y}, \tag{28.285}$$

where

$$\operatorname{supp} \boldsymbol{r}^t \triangleq \left\{\rho \in \mathcal{R} : r_\rho^t \neq 0\right\} \tag{28.286}$$

is the set of request types for which there is a non-zero number of requests in the request batch $\boldsymbol{r}^t$.

Lemma 28.6. *The gain function in Eq.* (4.16) *can be equivalently expressed as*

$$G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) = \sum_{\rho \in \operatorname{supp} \boldsymbol{r}^t} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \min\left\{r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})\right\} \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0\right), \forall \boldsymbol{y} \in \mathcal{X} \cup \mathcal{Y}. \tag{28.287}$$

Proof.

Remember from the definition in Eq. (4.15) that $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) = \min\left\{r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})\right\}$. We observe that $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})$ is not a function of $\boldsymbol{y}$ and it is equal to 0, when there is no repository with model's rank smaller or equal to k, and to r_ρ^t, otherwise; therefore, $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) \in \{0, r_\rho^t\}$.

When $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) \neq 0$, and thus $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = r_\rho^t$, the following holds

$$Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = \min\left\{r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})\right\} - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) \tag{28.288}$$

$$= \min\left\{0, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})\right\} = 0. \tag{28.289}$$

The last equality holds because $\sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) \geq 0, \forall \boldsymbol{y} \in \mathcal{X} \cup \mathcal{Y}$ from Eq. (4.3). Otherwise, when $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0$, we have $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) \overset{(4.15)}{=} \min\left\{r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})\right\}$.

Hence, we can succinctly write, for any value of $Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})$, that

$$Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = \min\left\{r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})\right\} \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0\right). \tag{28.290}$$

Since $\sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})$ is non negative, the previous formula implies that

$$Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0, \quad \text{if } r_\rho^t = 0. \tag{28.291}$$

By combining Eq. (28.290) and (28.291) that

$$Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = \min\left\{r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})\right\} \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0 \ \wedge \ r_\rho^t \neq 0\right). \tag{28.292}$$

Hence, applying the above equalities on the gain expression we get

$$G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) \overset{(4.16)}{=} \sum_{\rho\in\mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) - Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega})\right) \tag{28.293}$$

$$\overset{(28.292)}{=} \sum_{\rho\in\mathcal{R}} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \min\left\{r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})\right\} \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0 \ \wedge \ r_\rho^t \neq 0\right) \tag{28.294}$$

$$\overset{(28.286)}{=} \sum_{\rho\in\operatorname{supp}\boldsymbol{r}^t} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \min\left\{r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y})\right\} \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0\right). \tag{28.295}$$

□

Lemma 28.7. *Consider $n \in \mathbb{N}$, $\boldsymbol{y} \in [0,1]^n$, $\boldsymbol{q} \in \mathbb{N}^n$, and $c \in \mathbb{N}$. We assume that $q_i \leq c, \forall i \in [n]$. The following holds*

$$\min\left\{c, \sum_{i\in[n]} y_i q_i\right\} \geq c - c \prod_{i\in[n]} (1 - y_i q_i / c). \tag{28.296}$$

Proof.

We define $a_n \triangleq c - c\prod_{i\in[n]}(1 - y_i q_i/c)$ and $b_n \triangleq \min\left\{c, \sum_{i\in[n]} y_i q_i\right\}$.

We first show by induction that, if $a_n \leq b_n$, then this inequality holds also for $n+1$.

Base case ($n = 1$).

$$a_1 = c - c + y_1 q_1 = y_1 q_1 = \min\{c, q_1 y_1\} = b_1. \tag{28.297}$$

Induction step.

$$a_{n+1} = c - c \prod_{i\in[n+1]} (1 - y_i q_i/c) \tag{28.298}$$

$$= c - c \prod_{i\in[n]} (1 - y_i q_i/c)(1 - y_{n+1} q_{n+1}/c) \tag{28.299}$$

$$= c - c \prod_{i\in[n]} (1 - y_i q_i/c) + (c y_{n+1} q_{n+1}/c) \prod_{i\in[n]} (1 - y_i q_i/c) \tag{28.300}$$

$$= a_n + y_{n+1} q_{n+1} \prod_{i\in[n]} (1 - y_i q_i/c) \tag{28.301}$$

$$\leq a_n + y_{n+1} q_{n+1}. \tag{28.302}$$

The last inequality holds since by construction $q_i \leq c$ and thus $0 \leq y_i q_i/c \leq 1$, and $0 \leq \prod_{i\in[n]} (1 - y_i q_i/c) \leq 1$. For the same reason, $0 \leq \prod_{i\in[n+1]}(1 - y_i q_i/c) \leq 1$ and thus, by (28.298), we have $a_{n+1} \leq c$. Moreover, note that if $b_n = c$ then $b_{n+1} = c$. Therefore:

$$a_{n+1} \leq \min\{c, a_n + y_{n+1} q_{n+1}\} \leq \min\{c, b_n + y_{n+1} q_{n+1}\} \tag{28.303}$$

$$= \begin{cases} \min\left\{c, \sum_{i=1}^{n+1} y_i q_i\right\} = b_{n+1}, & \text{if} \quad b_n \leq c, \\ \min\{c, c + y_{n+1} q_{n+1}\} = c = b_{n+1}, & \text{if} \quad b_n = c, \end{cases} \tag{28.304}$$

and the proof by induction is completed.

□

Lemma 28.8. *Consider* $\boldsymbol{y} \in [0,1]^n$, $\boldsymbol{q} \in \mathbb{N}^n$, $c \in \mathbb{N}$ *and* $n \in \mathbb{N}$. *We assume that* $q_i \leq c, \forall i \in [n]$. *The following holds*

$$c - c \prod_{i\in[n]} (1 - y_i q_i/c) \geq (1 - 1/e) \min\left\{c, \sum_{i\in[n]} y_i q_i\right\}. \tag{28.305}$$

Proof.

Our proof follows the same lines of the proof of [299, Lemma 3.1]. We use the arithmetic/geometric mean inequality [300] on the non-negative variables $1 - y_i q_i/c, i \in [n]$ to obtain:

$$\frac{1}{n} \sum_{i\in[n]} (1 - y_i q_i/c) \geq \left(\prod_{i\in[n]} (1 - y_i q_i/c)\right)^{\frac{1}{n}}. \tag{28.306}$$

We reformulate the above as:

$$1 - \prod_{i\in[n]} (1 - y_i q_i/c) \geq 1 - \left(\frac{1}{n} \sum_{i\in[n]} (1 - y_i q_i/c) \right)^n = 1 - \left(1 - \frac{1}{n} \sum_{i\in[n]} y_i q_i/c \right)^n \tag{28.307}$$

$$\geq 1 - \left(1 - \frac{1}{n} \min \left\{ 1, \sum_{i\in[n]} y_i q_i/c \right\} \right)^n . \tag{28.308}$$

To obtain the last inequality, consider that, for any number z, we have $z \geq \min\{1, z\}$, and thus $\sum_{i\in[n]} y_i q_i/c \geq \min\left\{1, \sum_{i\in[n]} y_i q_i/c\right\}$.

The function $f(z) = 1 - (1 - z/n)^n$ is concave for $z \in [0, 1]$, then, for $z \in [0, 1]$, $f(z) \geq f(0) + z\frac{f(1)-f(0)}{1-0} = zf(1)$, as $f(0) = 0$. Setting $z = \min\left\{1, \sum_{i\in[n]} y_i q_i/c\right\}$, we obtain the following:

$$1 - \prod_{i\in[n]} (1 - y_i q_i/c) \overset{(28.308)}{\geq} 1 - \left(1 - \frac{1}{n} z\right)^n \geq (1 - (1 - 1/n)^n)\, z \geq (1 - 1/e) z. \tag{28.309}$$

The last inequality is obtained since $1 - (1 - 1/n)^n$ decreases in n, and it is lower bounded by $1 - 1/e$. By multiplying both sides of the above inequality by $c \in \mathbb{N}$, and replacing z with its value we conclude the proof.

□

Lemma 28.9. *for any request batch $\boldsymbol{r}_t$ and potential available capacity $\boldsymbol{l}_t$ such that $(\boldsymbol{r}_t, \boldsymbol{l}_t) \in \mathcal{A}$, the allocation gain $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ has the following lower and upper bounds*

$$\left(1 - \tfrac{1}{e}\right)^{-1} \Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) \geq G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) \geq \Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}), \qquad \forall \boldsymbol{y} \in \mathcal{X} \cup \mathcal{Y}. \tag{28.310}$$

Proof.

We have the following

$$G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}) \overset{(28.287)}{=} \sum_{\rho\in\operatorname{supp}\boldsymbol{r}^t} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) \min\left\{ r_\rho^t, \sum_{k'=1}^{k} z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y}) \right\} \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0\right) \tag{28.311}$$

$$\overset{(28.296)}{\geq} \sum_{\rho\in\operatorname{supp}\boldsymbol{r}^t} \sum_{k=1}^{K_\rho - 1} \left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) r_\rho^t \left(1 - \prod_{k'=1}^{k} \left(1 - z_{\rho,k'}(\boldsymbol{l}_t, \boldsymbol{y})/r_\rho^t\right)\right) \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0\right) \tag{28.312}$$

$$\overset{(28.285)}{=} \Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}), \forall \boldsymbol{y} \in \mathcal{X} \cup \mathcal{Y}, \tag{28.313}$$

and

$$(1-1/e)G(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{y}) \overset{(28.287)}{=} (1-1/e)\sum_{\rho\in\operatorname{supp}\boldsymbol{r}^t}\sum_{k=1}^{K_\rho-1}\left(\gamma_\rho^{k+1}-\gamma_\rho^k\right)\min\left\{r_\rho^t,\sum_{k'=1}^{k}z_\rho^{k'}(\boldsymbol{l}_t,\boldsymbol{y})\right\}\mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{\omega})=0\right) \tag{28.314}$$

$$\overset{(28.305)}{\leq}\sum_{\rho\in\operatorname{supp}\boldsymbol{r}^t}\sum_{k=1}^{K_\rho-1}\left(\gamma_\rho^{k+1}-\gamma_\rho^k\right)r_\rho^t\left(1-\prod_{k'=1}^{k}\left(1-z_{\rho,k'}(\boldsymbol{l}_t,\boldsymbol{y})/r_\rho^t\right)\right)\mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{\omega})=0\right) \tag{28.315}$$

$$\overset{(28.285)}{=}\Lambda(\boldsymbol{r}_t,\boldsymbol{l}_t,\boldsymbol{y}),\forall\boldsymbol{y}\in\mathcal{X}\cup\mathcal{Y}. \tag{28.316}$$

Inequalities in Eq. (28.312) and Eq. (28.315) follow from Eq. (28.296) and Eq. (28.305), respectively, by replacing $c=r_\rho^t$, $q_{k'}=\lambda_\rho^{k'}(\boldsymbol{l}_t)$, $x_{k'}=z_{\rho,k'}(\boldsymbol{l}_t,\boldsymbol{y})/\lambda_\rho^{k'}(\boldsymbol{l}_t)$, and $n=k$.

□

Lemma 28.10. *Let the allocation $\boldsymbol{x}^v$ be the random output of* DEPROUND *on node $v\in\mathcal{V}$ given the fractional allocation $\boldsymbol{y}^v\in\mathcal{Y}^v$. For any subset of the model catalog $S\subset\mathcal{M}$ and any number $c_m\in[0,1],\forall m\in S$,* DEPROUND *satisfies the following:*

$$\mathbb{E}\left[\prod_{m\in S}(1-x_m^v c_m)\right]\leq\prod_{m\in S}(1-y_m^v c_m). \tag{28.317}$$

Proof.

DEPROUND uses a subroutine SIMPLIFY, which, given input variables $y_m,y_{m'}\in(0,1)$, outputs $x_m,x_{m'}\in[0,1]$ with at least one of them being integral (0 or 1). Note that the input to SIMPLIFY is never integral since it is only called on fractional and yet unrounded variables. The property (B3) in [173, Lemma 2.1] implies that the output variables x_m and $x_{m'}$ satisfy the following inequality:

$$\mathbb{E}[x_m x_{m'}]\leq y_m y_{m'}. \tag{28.318}$$

We have for any $c_m,c_{m'}\in[0,1]$:

$$\begin{aligned}\mathbb{E}[(1-x_m c_m)(1-x_{m'}c_{m'})] &= \mathbb{E}[1-x_m c_m-x_{m'}c_{m'}+x_m x_{m'}c_m c_{m'}] &(28.319)\\ &= 1-y_m c_m-y_{m'}c_{m'}+\mathbb{E}[x_m x_{m'}]c_m c_{m'} &(28.320)\\ &\leq 1-y_m c_m-y_{m'}c_{m'}+y_m y_{m'}c_m c_{m'}\\ &= (1-y_m c_m)(1-y_{m'}c_{m'}). &(28.321)\end{aligned}$$

where the second equality is obtained recalling that, by construction, $\mathbb{E}[x_m^v]=y_m,\forall m\in\mathcal{M}$ (Section 4.4.3).

Thus, the two fractional inputs y_m and $y_{m'}$ to the SIMPLIFY subroutine, return x_m and $x_{m'}$ satisfying the property (28.321). By induction as in the proof in [173, Lemma 2.2], we obtain for any $S \subset \mathcal{M}$:

$$\mathbb{E}\left[\prod_{m\in S}(1 - x_m c_m)\right] \leq \prod_{m\in S}(1 - y_m c_m). \tag{28.322}$$

Note that the above property is satisfied with equality if the components of $\boldsymbol{x} \in \{0,1\}^{|\mathcal{M}|}$ are sampled independently with $\mathbb{E}[x_m] = y_m$.

□

Lemma 28.11. *Let the allocation $\boldsymbol{x}^v$ be the random output of* DEPROUND *on node $v \in \mathcal{V}$ given the fractional allocation $\boldsymbol{y}^v \in \mathcal{Y}^v$. The following holds*

$$\mathbb{E}\left[\Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t)\right] \geq \Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t). \tag{28.323}$$

Proof.

For any k', assume $m \in \mathcal{M}$ and $v \in \mathcal{V}$ are such that $\kappa_\rho(v, m) = k'$ (Section 4.3.5). Since $z_\rho^{k'}(\boldsymbol{l}_t, \boldsymbol{y}) = y_m^v l_{\rho,m}^{t,v}$ for a given $(m, v) \in \mathcal{M} \times \mathcal{V}$, then $z_{\rho,k'}(\boldsymbol{l}_t, \boldsymbol{y})/r_\rho^t$ can be written as:

$$z_{\rho,k'}(\boldsymbol{l}_t, \boldsymbol{y})/r_\rho^t = \frac{l_{\rho,m}^{t,v}}{r_\rho^t} y_m^v. \tag{28.324}$$

where $\frac{l_{\rho,m}^{t,v}}{r_\rho^t}$ is a constant in $[0,1]$ ($l_{\rho,m}^{t,v} \leq \min\{r_\rho^t, L_m^v\}$ - see Section 4.3.4) that scales variable y_m^v ;therefore, by applying Lemma 28.10, we obtain the following upper bound on the bounding function. Consider for all $v \in \mathcal{V}$ and $t \in [T]$ that $\boldsymbol{x}_t^v$ is the random allocation obtained by running DEPROUND on the fractional allocation $\boldsymbol{y}_t^v$, then

$$\mathbb{E}\left[\Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t)\right] \overset{(28.285)}{=} \mathbb{E}\left[\sum_{\rho\in\operatorname{supp}\boldsymbol{r}^t}\sum_{k=1}^{K_\rho-1}\left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) r_\rho^t \left(1 - \prod_{k'=1}^{k}\left(1 - z_{\rho,k'}(\boldsymbol{l}_t, \boldsymbol{x}_t)/r_\rho^t\right)\right) \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0\right)\right] \tag{28.325}$$

$$= \sum_{\rho\in\operatorname{supp}\boldsymbol{r}^t}\sum_{k=1}^{K_\rho-1}\left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) r_\rho^t \left(1 - \mathbb{E}\left[\prod_{k'=1}^{k}\left(1 - z_{\rho,k'}(\boldsymbol{l}_t, \boldsymbol{x}_t)/r_\rho^t\right)\right]\right) \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0\right) \tag{28.326}$$

$$\geq \sum_{\rho\in\operatorname{supp}\boldsymbol{r}^t}\sum_{k=1}^{K_\rho-1}\left(\gamma_\rho^{k+1} - \gamma_\rho^k\right) r_\rho^t \left(1 - \prod_{k'=1}^{k}\left(1 - z_{\rho,k'}(\boldsymbol{l}_t, \boldsymbol{y}_t)/r_\rho^t\right)\right) \mathbb{1}\left(Z_\rho^k(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{\omega}) = 0\right) \tag{28.327}$$

$$= \Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t). \tag{28.328}$$

The equality is obtained using the linearity of the expectation, and the inequality is obtained by applying directly Lemma 28.10.

□

29 Proof of Theorem 4.5.1

To prove the ψ-regret guarantee: (1) we first establish an upper bound on the regret of the INFIDA policy over its fractional allocations domain $\mathcal{Y}$ against a fractional optimum, then (2) we use it to derive a corresponding ψ-regret guarantee over the integral allocations domain $\mathcal{X}$.

Fractional domain regret guarantee. To establish the regret guarantee of running Algorithm 4.1 at the level of each computing node $v \in \mathcal{V}$, we showed that the following properties hold:

1. The function G is concave over its domain $\mathcal{Y}$ (Lemma 28.1).
2. The mirror map $\Phi : \mathcal{D} \rightarrow \mathbb{R}$ is θ-strongly convex w.r.t. the norm $\|\cdot\|_{l_1(\boldsymbol{s})}$ over $\mathcal{Y} \cap \mathcal{D}$, where θ is equal to Eq. (28.262) (Lemma 28.2).
3. The gain function $G : \mathcal{Y} \rightarrow \mathbb{R}$ is σ-Lipchitz w.r.t $\|\cdot\|_{l_1(\boldsymbol{s})}$: the subgradients are bounded under the norm $\|\cdot\|_{l_\infty(\frac{1}{\boldsymbol{s}})}$ by σ, i.e., the subgradient of $G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ at point $\boldsymbol{y}_t \in \mathcal{Y}$ is upper bounded ($\|\boldsymbol{g}_t\|_{l_\infty(\frac{1}{\boldsymbol{s}})} \leq \sigma$) for any $(\boldsymbol{r}_t, \boldsymbol{l}_t) \in \mathcal{A}$ (Lemma 28.3).
4. $\|\cdot\|_{l_\infty(\frac{1}{\boldsymbol{s}})}$ is the dual norm of $\|\cdot\|_{l_1(\boldsymbol{s})}$ (Lemma 28.4).
5. The Bregman divergence $D_\Phi(\boldsymbol{y}_*, \boldsymbol{y}_1)$ in Eq. (26.229) is upper bounded by a constant $D_{\max}$ where $\boldsymbol{y}_* = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y})$ and $\boldsymbol{y}_1 = \arg\min_{\boldsymbol{y} \in \mathcal{Y} \cap \mathcal{D}} \Phi(\boldsymbol{y})$ is the initial allocation (Lemma 28.5).

Because of properties 1–5 above, the following bound holds for the regret of INFIDA over its fractional domain $\mathcal{Y}$ (vector field point of view of Mirror Descent in [53, Section 4.2] combined with [53, Theorem 4.2]):

$$\text{Regret}_{T,\mathcal{Y}} = \sup_{\{(\boldsymbol{r}_t, \boldsymbol{l}_t)\}_{t=1}^{T} \in \mathcal{A}^T} \left\{ \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_*) - \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) \right\} \tag{29.329}$$

$$\leq \frac{D_\Phi(\boldsymbol{y}_*, \boldsymbol{y}_1)}{\eta} + \frac{\eta}{2\theta} \sum_{t=1}^{T} \|\boldsymbol{g}_t\|^2_{l_\infty(\frac{1}{\boldsymbol{s}})} \leq \frac{D_{\max}}{\eta} + \frac{\eta \sigma^2 T}{2\theta}. \tag{29.330}$$

where η is the learning rate of INFIDA (Algorithm 4.1, line 6). By selecting the learning rate $\eta = \frac{1}{\sigma}\sqrt{\frac{2\theta D_{\max}}{T}}$ giving the tightest upper bound we obtain

$$\text{Regret}_{T,\mathcal{Y}} \leq \sigma \sqrt{\frac{2 D_{\max}}{\theta} T}. \tag{29.331}$$

Integral domain regret guarantee. Note that, by restricting the maximization to the subset of integral allocations $\boldsymbol{x} \in \mathcal{X}$, the optimal allocation $\boldsymbol{x}_* = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x})$ can only lead to a lower gain, i.e.,

$$\sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_*) \leq \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_*). \tag{29.332}$$

By taking $\psi = 1 - \frac{1}{e}$, and using the bounding function Λ defined in Eq. (28.285), with the expectation taken over the random choices of the policy (DepRound at line 8 in Algorithm 4.1) we obtain

$$\mathbb{E}\left[\sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t)\right] \overset{(28.310)}{\geq} \mathbb{E}\left[\sum_{t=1}^{T} \Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t)\right] \overset{(28.323)}{\geq} \sum_{t=1}^{T} \Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t) \overset{(28.310)}{\geq} \psi \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_t)$$
$$\overset{(29.331)}{\geq} \psi \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}_*) - \psi\sigma\sqrt{\frac{2D_{\max}}{\theta}T} \overset{(29.332)}{\geq} \psi \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_*) - \psi\sigma\sqrt{\frac{2D_{\max}}{\theta}T}. \tag{29.333}$$

Thus, we have

$$\psi \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_*) - \mathbb{E}\left[\sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t)\right] \leq \psi\sigma\sqrt{\frac{2D_{\max}}{\theta}T}. \tag{29.334}$$

The above inequality holds for any sequence $\{(\boldsymbol{r}_t, \boldsymbol{l}_t)\}_{t=1}^{T} \in \mathcal{A}^T$. Thus, the ψ-regret is given by

$$\psi\text{-Regret}_{T,\mathcal{X}} \overset{(4.20)}{=} \sup_{\{\boldsymbol{r}_t, \boldsymbol{l}_t\}_{t=1}^{T} \in \mathcal{A}^T} \left\{\psi \sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_*) - \mathbb{E}\left[\sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{x}_t)\right]\right\} \leq A\sqrt{T}, \tag{29.335}$$

where

$$A = \psi\sigma\sqrt{\frac{2D_{\max}}{\theta}} = \psi\frac{RL_{\max}\Delta_C}{s_{\min}}\sqrt{s_{\max}|\mathcal{V}||\mathcal{M}|}\sqrt{2\sum_{v\in\mathcal{V}} \min\{b^v, \|\boldsymbol{s}^v\|_1\} \log\left(\frac{\|\boldsymbol{s}^v\|_1}{\min\{b^v, \|\boldsymbol{s}^v\|_1\}}\right)},$$

using the upper bounds on θ, σ, and $D_{\max}$ determined in Lemmas 28.2, 28.3, and 28.5, respectively.

This proves Theorem 4.5.1.

30 Proof of Proposition 4.5.2

Let $\bar{\boldsymbol{y}}$ be the average fractional allocation $\bar{\boldsymbol{y}} = \frac{1}{\tilde{T}}\sum_{t=1}^{\tilde{T}} \boldsymbol{y}_t$ of INFIDA, and $\bar{\boldsymbol{x}}$ the random state sampled from $\bar{\boldsymbol{y}}$ using DepRound. We take $G_T(\boldsymbol{y}) = \frac{1}{T}\sum_{t=1}^{T} G(\boldsymbol{r}_t, \boldsymbol{l}_t, \boldsymbol{y}), \forall \boldsymbol{y} \in \mathcal{Y}$. We have

$$\mathbb{E}\left[G_T(\bar{\boldsymbol{x}})\right] \overset{(28.310)}{\geq} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} \Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \bar{\boldsymbol{x}})\right] \overset{(28.323)}{\geq} \frac{1}{T}\sum_{t=1}^{T} \Lambda(\boldsymbol{r}_t, \boldsymbol{l}_t, \bar{\boldsymbol{y}}) \overset{(28.310)}{\geq} \psi G_T(\bar{\boldsymbol{y}}). \tag{30.336}$$

Using Jensen's inequality we get

$$\psi G_T(\bar{\boldsymbol{y}}) \geq \psi\frac{1}{\tilde{T}}\sum_{t=1}^{\tilde{T}} G_T(\boldsymbol{y}_t). \tag{30.337}$$

It straightforward to check that G_T satisfies the same properties 1 (concavity) and 3 (subgradient boundedness) as G and the remaining properties are preserved under the same mirror map and convex decision set. With properties 1–5 satisfied, we can apply [53, Theorem 4.2] to obtain

$$\sum_{t=1}^{\tilde{T}} G_T(\boldsymbol{y}_*) - \sum_{t=1}^{\tilde{T}} G_T(\boldsymbol{y}_t) = \tilde{T} G_T(\boldsymbol{y}_*) - \sum_{t=1}^{\tilde{T}} G_T(\boldsymbol{y}_t) \leq \sigma \sqrt{\frac{2D_{\max}}{\theta} \tilde{T}}. \tag{30.338}$$

Dividing both sides of the above inequality by $\tilde{T}$ gives

$$\frac{1}{\tilde{T}} \sum_{t=1}^{\tilde{T}} G_T(\boldsymbol{y}_t) \geq G_T(\boldsymbol{y}_*) - \sigma \sqrt{\frac{2D_{\max}}{\theta \tilde{T}}}. \tag{30.339}$$

Using the same argument to obtain Eq. (29.332), i.e., restricting the maximization to the integral domain gives a lower value, we get

$$\frac{1}{\tilde{T}} \sum_{t=1}^{\tilde{T}} G_T(\boldsymbol{y}_t) \geq G_T(\boldsymbol{x}_*) - \sigma \sqrt{\frac{2D_{\max}}{\theta \tilde{T}}}. \tag{30.340}$$

Using Eq. (30.336), and Eq. (30.340) we obtain

$$\mathbb{E}\left[G_T(\bar{\boldsymbol{x}})\right] \geq \psi G_T(\boldsymbol{x}_*) - \psi \sigma \sqrt{\frac{2D}{\theta \tilde{T}}}. \tag{30.341}$$

Thus, $\forall \epsilon > 0$ and over a sufficiently large running time $\tilde{T}$ for INFIDA, $\bar{\boldsymbol{x}}$ satisfies

$$\mathbb{E}\left[G_T(\bar{\boldsymbol{x}})\right] \geq \left(1 - \frac{1}{e} - \epsilon\right) G_T(\boldsymbol{x}_*). \tag{30.342}$$

31 Technical Lemmas and Definitions

31.1 Convex Conjugate

Definition 31.1. Let $F : \mathcal{U} \subset \mathbb{R}^I \to \mathbb{R} \cup \{-\infty, +\infty\}$ be a function. Define $F^\star : \mathbb{R}^I \to \mathbb{R} \cup \{-\infty, +\infty\}$ by

$$F^\star(\boldsymbol{\theta}) = \sup_{\boldsymbol{u} \in \mathcal{U}} \left\{\boldsymbol{u} \cdot \boldsymbol{\theta} - F(\boldsymbol{u})\right\}, \tag{31.343}$$

for $\boldsymbol{\theta} \in \mathbb{R}^I$. This is the *convex conjugate* of F.

31.2 Convex Conjugate of α-Fairness Function

Lemma 31.1. *Let* $\mathcal{U} = \left[u_{\star,\min}, u_{\star,\max}\right]^{\mathcal{I}} \subset \mathbb{R}^{\mathcal{I}}_{>0}$, $\Theta = \left[-1/u^{\alpha}_{\star,\min}, -1/u^{\alpha}_{\star,\max}\right]^{\mathcal{I}} \subset \mathbb{R}^{\mathcal{I}}_{<0}$, *and* $F_\alpha : \mathcal{U} \to \mathbb{R}$ *be an* α*-fairness function* (5.3). *The convex conjugate of* $-F_\alpha$ *is given by*

$$(-F_\alpha)^\star(\boldsymbol{\theta}) = \begin{cases} \sum_{i\in\mathcal{I}} \dfrac{\alpha(-\theta_i)^{1-1/\alpha} - 1}{1-\alpha} & \text{for } \alpha \in \mathbb{R}_{\geq 0} \setminus \{1\}, \\ \sum_{i\in\mathcal{I}} -\log(-\theta_i) - 1 & \text{for } \alpha = 1, \end{cases} \tag{31.344}$$

where $\boldsymbol{\theta} \in \Theta$.

Proof.

The convex conjugate of $-f_\alpha(u) \triangleq -\frac{u^{1-\alpha}-1}{1-\alpha}$ for $u \in \left[u_{\star,\min}, u_{\star,\max}\right]$ and $\alpha \in \mathbb{R}_{\geq 0} \setminus \{1\}$ is given by

$$(-f_\alpha)^\star(\theta) = \max_{u\in\left[u_{\star,\min},u_{\star,\max}\right]} \left\{u\theta + \frac{u^{1-\alpha}-1}{1-\alpha}\right\}. \tag{31.345}$$

We evaluate the derivative to characterize the maxima of r.h.s. term in the above equation

$$\frac{\partial}{\partial u}\left(u\theta + \frac{u^{1-\alpha}-1}{1-\alpha}\right) = \theta + \frac{1}{u^\alpha}. \tag{31.346}$$

The function $\theta + \frac{1}{u^\alpha}$ is a decreasing function in u; thus $\theta + \frac{1}{u^\alpha} \geq 0$ when $u \leq \left(-\frac{1}{\theta}\right)^{\frac{1}{\alpha}}$, and $\theta + \frac{1}{u^\alpha} < 0$ otherwise. The maximum is achieved at $u = \left(-\frac{1}{\theta}\right)^{\frac{1}{\alpha}}$. It holds through Eq. (31.345)

$$(-f_\alpha)^\star(\theta) = \frac{\alpha(-\theta)^{1-1/\alpha} - 1}{1-\alpha} \qquad \text{for} \qquad \theta \in \left[-1/u^{\alpha}_{\star,\min}, -1/u^{\alpha}_{\star,\max}\right]. \tag{31.347}$$

Moreover, it is easy to check that the same argument holds for $f_1(u) = \log(u)$ and we have

$$(-f_1)^\star(\theta) = -1 - \log(-\theta) \qquad \text{for} \qquad \theta \in \left[-1/u_{\star,\min}, -1/u_{\star,\max}\right]. \tag{31.348}$$

The convex conjugate of $-F_\alpha(\boldsymbol{u}) = \sum_{i\in\mathcal{I}} f_\alpha(u_i)$ for $\boldsymbol{u} \in \mathcal{U}$, using Eq. (31.347) and Eq. (31.348), is given by

$$(-F_\alpha)^\star(\boldsymbol{\theta}) = \sum_{i\in\mathcal{I}} (-f_\alpha)^\star(\theta_i) = \begin{cases} \sum_{i\in\mathcal{I}} \dfrac{\alpha(-\theta_i)^{1-1/\alpha} - 1}{1-\alpha} & \text{for } \alpha \in \mathbb{R}_{\geq 0} \setminus \{1\}, \\ \sum_{i\in\mathcal{I}} -\log(-\theta_i) - 1 & \text{for } \alpha = 1, \end{cases} \tag{31.349}$$

for $\boldsymbol{\theta} \in \Theta$, because $F_\alpha(\boldsymbol{u})$ is separable in $\boldsymbol{u} \in \mathcal{U}$.

□

31.3 Convex Biconjugate of α-Fairness Functions

The following Lemma provides a stronger condition on $\boldsymbol{\theta}$ compared to [283, Lemma 2.2], i.e., we restrict $\boldsymbol{\theta} \in \Theta$ instead of $\|\boldsymbol{\theta}\|_\star \leq L$ where $L \geq \|\nabla_{\boldsymbol{u}} F_\alpha(\boldsymbol{u})\|_\star$ for all $\boldsymbol{u} \in \mathcal{U}$ and $\|\cdot\|_\star$ is the dual norm of $\|\cdot\|$.

Lemma 31.2. *Let* $\mathcal{U} = \left[u_{\star,\min}, u_{\star,\max}\right]^{\mathcal{I}} \subset \mathbb{R}^{\mathcal{I}}_{>0}$, $\Theta = \left[-1/u^\alpha_{\star,\min}, -1/u^\alpha_{\star,\max}\right]^{\mathcal{I}} \subset \mathbb{R}^{\mathcal{I}}_{<0}$, *and* $F_\alpha : \mathcal{U} \to \mathbb{R}$ *be an* α*-fairness function* (5.3). *The function* F_α *can be always be recovered from the convex conjugate* $(-F_\alpha)^\star$, *i.e.,*

$$F_\alpha(\boldsymbol{u}) = \min_{\boldsymbol{\theta}\in\Theta} \left\{(-F_\alpha)^\star(\boldsymbol{\theta}) - \boldsymbol{\theta}\cdot\boldsymbol{u}\right\}, \tag{31.350}$$

for $\boldsymbol{u} \in \mathcal{U}$.

Proof.

This proof follows the same lines of the proof of [283, Lemma 2.2]. Since $\boldsymbol{u} \in \mathcal{U}$, therefore the gradient of F_α at point $\boldsymbol{u}$ is given as $\nabla_{\boldsymbol{u}} F_\alpha(\boldsymbol{u}) = \left[1/u_i^\alpha\right]_{i\in\mathcal{I}} \in -\Theta = \left[1/u^\alpha_{\star,\min}, 1/u^\alpha_{\star,\max}\right]^{\mathcal{I}}$. Moreover, it holds

$$\min_{\boldsymbol{\theta}\in\Theta} \left\{(-F_\alpha)^\star(\boldsymbol{\theta}) - \boldsymbol{\theta}\cdot\boldsymbol{u}\right\} = \min_{\boldsymbol{\theta}\in\Theta} \left\{\max_{\boldsymbol{u}'\in\mathcal{U}} \{\boldsymbol{\theta}\cdot\boldsymbol{u}' + F_\alpha(\boldsymbol{u})\} - \boldsymbol{\theta}\cdot\boldsymbol{u}\right\} \tag{31.351}$$

$$= \max_{\boldsymbol{u}'\in\mathcal{U}} \min_{\boldsymbol{\theta}\in\Theta} \{\boldsymbol{\theta}\cdot\boldsymbol{u}' + F_\alpha(\boldsymbol{u}') - \boldsymbol{\theta}\cdot\boldsymbol{u}\}. \qquad \text{Minmax theorem} \tag{31.352}$$

We take

$$\begin{aligned}
\min_{\boldsymbol{\theta}\in\Theta} \{\boldsymbol{\theta}\cdot\boldsymbol{u}' + F_\alpha(\boldsymbol{u}') - \boldsymbol{\theta}\cdot\boldsymbol{u}\} &= \min_{\boldsymbol{\theta}\in\Theta} \{F_\alpha(\boldsymbol{u}') + \boldsymbol{\theta}\cdot(\boldsymbol{u}' - \boldsymbol{u})\} \\
&\leq F_\alpha(\boldsymbol{u}') - \nabla F_\alpha(\boldsymbol{u})\cdot(\boldsymbol{u}' - \boldsymbol{u}) && \text{Because } -\nabla F_\alpha(\boldsymbol{u}) \in \Theta \\
&\leq F_\alpha(\boldsymbol{u}). && \text{Use concavity of } F_\alpha
\end{aligned}$$

The equality is achieved when $\boldsymbol{u} = \boldsymbol{u}'$ and the maximum value in (31.352) is attained for this value. We conclude the proof.

□

31.4 Online Gradient Descent (OGD) with Self-Confident Learning Rates

Lemma 31.3 provides the regret guarantee of OGD oblivious to the time horizon T and bound on subgradients' norm for any $t \in \mathcal{T}$. This adopts the idea of [289] which denominate such learning schemes as self-confident.

Lemma 31.3. *Consider a convex set* $\mathcal{X}$, *a sequence of* σ*-strongly convex functions* $f_t : \mathcal{X} \to \mathbb{R}$ *with subgradient* $\boldsymbol{g}_t \in \partial f_t(\boldsymbol{x}_t)$ *at* $\boldsymbol{x}_t$, *and OGD update rule* $\boldsymbol{x}_{t+1} = \Pi_{\mathcal{X}}\left(\boldsymbol{x}_t - \eta_t \boldsymbol{g}_t\right) = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \left\|\boldsymbol{x} - \left(\boldsymbol{x}_t - \eta_t \boldsymbol{g}_t\right)\right\|_2$ *initialized at* $\boldsymbol{x}_1 \in \mathcal{X}$. *Let* $\operatorname{diam}(\mathcal{X}) \triangleq \max\{\|\boldsymbol{x} - \boldsymbol{x}'\|_2 : \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}\}$.

Selecting the learning rates as $\boldsymbol{\eta} : \mathcal{T} \to \mathbb{R}$ *such that* $\eta_t \leq \eta_{t-1}$ *for all* $t > 1$ *gives the following regret guarantee against a fixed decision* $\boldsymbol{x} \in \mathcal{X}$*:*

$$\sum_{t\in\mathcal{T}} f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}) \leq \operatorname{diam}^2(\mathcal{X}) \sum_{t=1}^{T} \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}} - \sigma \right) + \sum_{t=1}^{T} \eta_t \left\| \boldsymbol{g}_t \right\|_2^2 . \tag{31.353}$$

- *When* $\sigma > 0$*, selecting the learning rate schedule* $\eta_t = \frac{1}{\sigma t}$ *for* $t \in \mathcal{T}$ *gives*

$$\sum_{t\in\mathcal{T}} f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}) \leq \sum_{t=1}^{T} \frac{\left\| \boldsymbol{g}_t \right\|_2^2}{t\sigma} = O\left(\log(T)\right). \tag{31.354}$$

- *When* $\sigma = 0$*, selecting the learning rate schedule* $\eta_t = \frac{\operatorname{diam}(\mathcal{X})}{\sqrt{\sum_{s=1}^{t} \left\| \boldsymbol{g}_s \right\|_2^2}}$ *for* $t \in \mathcal{T}$ *gives*

$$\sum_{t\in\mathcal{T}} f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}) \leq 1.5 \operatorname{diam}\left(\mathcal{X}\right) \sqrt{\sum_{t\in\mathcal{T}} \left\| \boldsymbol{g}_s \right\|_2^2} = O\left(\sqrt{T}\right). \tag{31.355}$$

Proof.

This proof follows the same lines of the proof of [20]. We do not assume a bound on the gradients is known beforehand and the time horizon T. Take a fixed $\boldsymbol{x} \in \mathcal{X}$. Applying the definition of σ-strong convexity to the pair of points $\boldsymbol{x}_t$ and $\boldsymbol{x}$, we have

$$2\left(f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x})\right) \leq 2\boldsymbol{g}_t \cdot (\boldsymbol{x}_t - \boldsymbol{x}) - \sigma \left\| \boldsymbol{x}_t - \boldsymbol{x} \right\|_2^2 . \tag{31.356}$$

Pythagorean theorem implies

$$\left\| \boldsymbol{x}_{t+1} - \boldsymbol{x} \right\|_2^2 = \left\| \Pi_{\mathcal{X}} \left(\boldsymbol{x}_t - \eta_t \boldsymbol{g}_t \right) - \boldsymbol{x} \right\|_2^2 \leq \left\| \boldsymbol{x}_t - \eta_t - \boldsymbol{x} \right\|_2^2 , \tag{31.357}$$

Expanding the r.h.s. term gives

$$\left\| \boldsymbol{x}_{t+1} - \boldsymbol{x} \right\|_2^2 \leq \left\| \boldsymbol{x}_t - \boldsymbol{x} \right\|_2^2 + \eta_t^2 \left\| \boldsymbol{g}_t \right\|_2^2 - 2\eta_t \boldsymbol{g}_t \cdot (\boldsymbol{x}_t - \boldsymbol{x}) , \tag{31.358}$$

$$2\boldsymbol{g}_t \cdot (\boldsymbol{x}_t - \boldsymbol{x}) \leq \frac{\left\| \boldsymbol{x}_t - \boldsymbol{x} \right\|_2^2 - \left\| \boldsymbol{x}_{t+1} - \boldsymbol{x} \right\|_2^2}{\eta_t} + \eta_t \left\| \boldsymbol{g}_t \right\|_2^2 . \tag{31.359}$$

Combine Eq. (31.356) and Eq. (31.359) and for $t = 1$ to $t = T$:

$$\begin{aligned}
2\sum_{t=1}^{T} f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}) &\leq \sum_{t=1}^{T} \frac{\left\| \boldsymbol{x}_t - \boldsymbol{x} \right\|_2^2 \left(1 - \sigma\eta_t\right) - \left\| \boldsymbol{x}_{t+1} - \boldsymbol{x} \right\|_2^2}{\eta_t} + \sum_{t=1}^{T} \eta_t \left\| \boldsymbol{g}_t \right\|_2^2 \\
&\leq \sum_{t=1}^{T} \left\| \boldsymbol{x}_t - \boldsymbol{x} \right\|_2^2 \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}} - \sigma \right) + \sum_{t=1}^{T} \eta_t \left\| \boldsymbol{g}_t \right\|_2^2 && \frac{1}{\eta_0} \triangleq 0 \\
&\leq \operatorname{diam}^2(\mathcal{X}) \left(\frac{1}{\eta_T} - \sigma T \right) + \sum_{t=1}^{T} \eta_t \left\| \boldsymbol{g}_t \right\|_2^2 . && \text{Telescoping series}
\end{aligned}$$

When $\sigma > 0$ and $\eta_t = \frac{1}{\sigma t}$, from Eq. (31.360) we have

$$\sum_{t=1}^{T} f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}) \leq 0 + \sum_{t=1}^{T} \frac{\left\|\boldsymbol{g}_t\right\|_2^2}{2\sigma t} \leq \max_{t\in\mathcal{T}} \left\{\left\|\boldsymbol{g}_t\right\|_2^2\right\} \sum_{t=1}^{T} \frac{1}{2\sigma} \leq \frac{\max_{t\in\mathcal{T}} \left\{\left\|\boldsymbol{g}_t\right\|_2^2\right\}}{2\sigma} \mathrm{H}_T = O\left(\log(T)\right), \tag{31.360}$$

where H_T is the T-th harmonic number.

When $\sigma = 0$ and $\eta_t = \frac{\operatorname{diam}(\mathcal{X})}{\sqrt{\sum_{s=1}^{t} \left\|\boldsymbol{g}_s\right\|_2^2}}$, from Eq. (31.360) we have

$$\sum_{t=1}^{T} f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}) \leq \frac{\operatorname{diam}(\mathcal{X})}{2} \sqrt{\sum_{t=1}^{T} \left\|\boldsymbol{g}_s\right\|_2^2} + \frac{\operatorname{diam}(\mathcal{X})}{2} \sum_{t=1}^{T} \frac{\left\|\boldsymbol{g}_t\right\|_2^2}{\sqrt{\sum_{s=1}^{t} \left\|\boldsymbol{g}_s\right\|_2^2}} \tag{31.361}$$

$$\leq 1.5 \operatorname{diam}(\mathcal{X}) \sqrt{\sum_{t=1}^{T} \left\|\boldsymbol{g}_s\right\|_2^2} = O\left(\sqrt{T}\right). \tag{31.362}$$

Last inequality is obtained using [289, Lemma 3.5], i.e., $\sum_{t=1}^{T} \frac{|a_t|}{\sum_{s=1}^{t} |a_s|} \leq 2\sqrt{\sum_{t=1}^{T} |a_t|}$. This concludes the proof.

□

31.5 Saddle-Point Problem Formulation of α-Fairness

Lemma 31.4. *Let $\mathcal{X}$ be a convex set, $\mathcal{U} = \left[u_{\star,\min}, u_{\star,\max}\right]^{\mathcal{I}} \subset \mathbb{R}_{>0}^{\mathcal{I}}$, $u_i : \mathcal{X} \to \mathcal{U}$ be a concave function for every $i \in \mathcal{I}$, $\Theta = \left[-1/u_{\star,\min}^{\alpha}, -1/u_{\star,\max}^{\alpha}\right]^{\mathcal{I}} \subset \mathbb{R}_{<0}^{\mathcal{I}}$, and $\Psi_\alpha : \Theta \times \mathcal{X} \to \mathbb{R}$ be a function given by*

$$\Psi_\alpha(\boldsymbol{\theta}, \boldsymbol{x}) \triangleq (-F_\alpha)^\star(\boldsymbol{\theta}) - \boldsymbol{\theta} \cdot \boldsymbol{u}(\boldsymbol{x}). \tag{31.363}$$

The following holds:

- *The solution of the saddle-point problem formed by Ψ_α is a maximizer of the α-fairness function*

$$\max_{\boldsymbol{x}\in\mathcal{X}} \min_{\boldsymbol{\theta}\in\Theta} \Psi_\alpha(\boldsymbol{\theta}, \boldsymbol{x}) = \max_{\boldsymbol{x}\in\mathcal{X}} F_\alpha(\boldsymbol{u}(\boldsymbol{x})). \tag{31.364}$$

- *The function $\Psi_\alpha : \Theta \times \mathcal{X} \to \mathbb{R}$ is concave over $\mathcal{X}$.*
- *The function $\Psi_\alpha : \Theta \times \mathcal{X} \to \mathbb{R}$ is $\frac{u_{\star,\min}^{1+1/\alpha}}{\alpha}$-strongly convex over Θ w.r.t. $\|\cdot\|_2$ for $\alpha > 0$.*

Proof.

Equation (31.364) is a direct result of Lemma 31.2. The function Ψ_α is concave over $\mathcal{X}$ because $-\boldsymbol{\theta} \cdot \boldsymbol{u}(\boldsymbol{x})$ is a weighted sum of concave functions with non-negative weights. To prove the strong convexity of Ψ_α w.r.t. $\|\cdot\|_2$, a sufficient condition [292, Lemma 14] is given by $\boldsymbol{\theta'}^T \left(\nabla^2_{\boldsymbol{\theta}} \Psi_\alpha(\boldsymbol{\theta}, \boldsymbol{x})\right) \boldsymbol{\theta'} \geq \sigma \left\|\boldsymbol{\theta'}\right\|_2^2$ for all $\boldsymbol{\theta}, \boldsymbol{\theta'} \in \Theta$, and it holds

$$\boldsymbol{\theta'}^T \left(\nabla^2_{\boldsymbol{\theta}} \Psi_\alpha(\boldsymbol{\theta}, \boldsymbol{x})\right) \boldsymbol{\theta'} = \sum_{i \in \mathcal{I}} \theta_i'^2 \frac{\partial^2}{\partial \theta_i} \left(-F_\alpha\right)^\star (\boldsymbol{\theta}) = \sum_{i \in \mathcal{I}} \frac{\theta_i'^2}{\alpha(-\theta_i)^{1+1/\alpha}} \geq \frac{u_{\star,\min}^{1+1/\alpha}}{\alpha} \left\|\boldsymbol{\theta'}\right\|_2^2. \tag{31.365}$$

This concludes the proof.

□

32 Proof of Theorem 5.4.1

Proof.

Consider two players $\mathcal{I} = \{1, 2\}$, allocation set $\mathcal{X} = [-1, 1]$ for all $t \in \mathcal{T}$. We define $\gamma_T \in [0.4, 1]$, $\psi_T \triangleq \frac{1}{T} \sum_{t=1}^{\gamma_T T} x_t$. We assume w.l.g. $\gamma_T T$ is a natural number. We consider two strategies selected by the adversary:

Strategy 1. The adversary reveals the following utilities:

$$\boldsymbol{u}_t(x) = \begin{cases} (1 + x, 2 - x) & \text{if } t \leq \gamma_T T, \\ (1, 1) & \text{otherwise.} \end{cases} \tag{32.366}$$

Under the selected utilities, the static optimum attains the following objective

$$\begin{aligned} \mathrm{OPT}^{\mathrm{S1}} &= \max_{x \in \mathcal{X}} \; f_\alpha((1 + x)\gamma_T + (1 - \gamma_T)) + f_\alpha((2 - x)\gamma_T + (1 - \gamma_T)) && (32.367) \\ &= \max_{x \in \mathcal{X}} \; f_\alpha(1 + \gamma_T x) + f_\alpha(1 + \gamma_T - \gamma_T x). && (32.368) \end{aligned}$$

The above objective is concave in x. We can perform a derivative test to characterize its maximum

$$\frac{\partial f_\alpha(1 + \gamma_T x) + F_\alpha(1 + \gamma_T - \gamma_T x)}{\partial x} = \frac{\gamma_T}{(1 + \gamma_T x)^\alpha} - \frac{\gamma_T}{(1 + \gamma_T - \gamma_T x)^\alpha} = 0, \qquad \text{for } x = \frac{1}{2}. \tag{32.369}$$

Thus, it holds

$$\mathrm{OPT}^{\mathrm{S1}} = 2 f_\alpha(1 + 0.5\gamma_T). \tag{32.370}$$

The fairness regret denoted by $\mathfrak{R}_T^{\mathrm{S1}}(F_\alpha, \boldsymbol{A})$ under this strategy of a policy $\mathcal{A}$ is given by

$$\begin{aligned} \mathfrak{R}_T^{\mathrm{S1}}(F_\alpha, \boldsymbol{A}) &= \mathrm{OPT}^{\mathrm{S1}} - f_\alpha\left(\frac{1}{T}\left(\sum_{t=1}^{\gamma_T T} 1 + x_t\right) + 1 - \gamma_T\right) - f_\alpha\left(\frac{1}{T}\left(\sum_{t=1}^{\gamma_T T} 2 - x_t\right) + 1 - \gamma_T\right) && (32.371) \\ &= 2 f_\alpha(1 + 0.5\gamma_T) - f_\alpha(1 + \psi_T) - f_\alpha(1 + \gamma_T - \psi_T). && (32.372) \end{aligned}$$

Strategy 2. The adversary reveals the following utilities:

$$\boldsymbol{u}_t(x) = \begin{cases} (1+x, 2-x) & \text{if } t \le \gamma_T T, \\ (2, 0) & \text{otherwise.} \end{cases} \tag{32.373}$$

Under the selected utilities, the static optimum attains the following objective

$$\begin{aligned} \mathrm{OPT}^{\mathrm{S2}} &= \max_{x \in \mathcal{X}} f_\alpha((1+x)\gamma_T + (1-\gamma_T)2) + f_\alpha((2-x)\gamma_T) && (32.374) \\ &= \max_{x \in \mathcal{X}} f_\alpha(2 - \gamma_T + \gamma_T x) + f_\alpha(2\gamma_T - \gamma_T x). && (32.375) \end{aligned}$$

Similar to the previous strategy, we can perform a derivative test to characterize the maximum of the the above objective

$$\frac{\partial f_\alpha(2 - \gamma_T + \gamma_T x) + f_\alpha(2\gamma_T - \gamma_T x)}{\partial x} = \frac{\gamma_T}{(2 - \gamma_T + \gamma_T x)^\alpha} - \frac{\gamma_T}{(2\gamma_T - \gamma_T x)^\alpha} = 0, \qquad \text{for } x = 1.5 - \frac{1}{\gamma_T}.$$

Therefore, it holds

$$\mathrm{OPT}^{\mathrm{S2}} = 2 f_\alpha(1 + 0.5\gamma_T). \tag{32.376}$$

and the fairness regret $\mathfrak{R}_T^{\mathrm{S2}}(F_\alpha, \boldsymbol{A})$ under this strategy is

$$\begin{aligned} \mathfrak{R}_T^{\mathrm{S2}}(F_\alpha, \boldsymbol{A}) &= \mathrm{OPT}^{\mathrm{S2}} - f_\alpha\left(\frac{1}{T}\left(\sum_{t=1}^{\gamma_T T} 1 + x_t\right) + 2 - 2\gamma_T\right) - f_\alpha\left(\frac{1}{T}\left(\sum_{t=1}^{\gamma_T T} 2 - x_t\right)\right) && (32.377) \\ &= 2 f_\alpha(1 + 0.5\gamma_T) - f_\alpha(2 - \gamma_T + \psi_T) - f_\alpha(2\gamma_T - \psi_T). && (32.378) \end{aligned}$$

We take the average fairness regret $\frac{1}{2}\left(\mathfrak{R}_T^{\mathrm{S1}}(F_\alpha, \boldsymbol{A}) + \mathfrak{R}_T^{\mathrm{S2}}(F_\alpha, \boldsymbol{A})\right)$ across the two strategies

$$\begin{aligned} &\frac{1}{2}\left(\mathfrak{R}_T^{\mathrm{S1}}(F_\alpha, \boldsymbol{A}) + \mathfrak{R}_T^{\mathrm{S2}}(F_\alpha, \boldsymbol{A})\right) && (32.379) \\ &= 2 f_\alpha(1 + 0.5\gamma_T) - \frac{1}{2}\left(f_\alpha(2 - \gamma_T + \psi_T) + f_\alpha(2\gamma_T - \psi_T) + f_\alpha(1 + \psi_T) + f_\alpha(1 + \gamma_T - \psi_T)\right). && (32.380) \end{aligned}$$

The r.h.s. of the above equation is convex in ψ_T, so its minimizer can be characterized through the derivative as follows

$$\begin{aligned} &\frac{\partial f_\alpha(2 - \gamma_T + \psi) + f_\alpha(2\gamma_T - \psi) + f_\alpha(1 + \psi) + f_\alpha(1 + \gamma_T - \psi)}{\partial \psi} && (32.381) \\ &= \frac{1}{(2 - \gamma_T + \psi)^\alpha} - \frac{1}{(2\gamma_T - \psi)^\alpha} + \frac{1}{(1 + \psi)^\alpha} - \frac{1}{(1 + \gamma_T - \psi)^\alpha} = 0, \qquad \text{for } \psi = \gamma_T - 0.5. && (32.382) \end{aligned}$$

We replace $\psi_T = \gamma_T - 0.5$ in Eq. (32.380) to get

$$\frac{1}{2}\left(\mathfrak{R}_T^{\mathrm{S1}}(F_\alpha, \boldsymbol{A}) + \mathfrak{R}_T^{\mathrm{S2}}(F_\alpha, \boldsymbol{A})\right) \ge 2 f_\alpha(1 + 0.5\gamma_T) - \left(f_\alpha(1.5) + f_\alpha(0.5 + \gamma_T)\right). \tag{32.383}$$

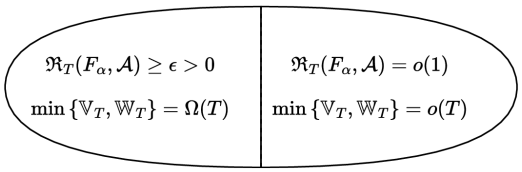


Figure 32.12: Assumption (A5) and fairness regret (5.8) under scenarios 1 and 2.

We take the derivative of the lower bound

$$\frac{\partial 2 f_\alpha(1+0.5\gamma_T) - (f_\alpha(1.5) + f_\alpha(0.5+\gamma_T))}{\partial \gamma_T} = \frac{(0.5+\gamma)^\alpha - (1+0.5\gamma)^\alpha}{(0.5+\gamma)^\alpha (1+0.5\gamma)^\alpha}. \tag{32.384}$$

Note that the sign of the derivative is determined by the numerator $(0.5+\gamma)^\alpha - (1+0.5\gamma)^\alpha$. It holds $(0.5+\gamma)^\alpha - (1+0.5\gamma)^\alpha < 0$ for $\gamma_T < 1$, otherwise $(0.5+\gamma)^\alpha - (1+0.5\gamma)^\alpha = 0$. Hence, the lower bound in Eq. (32.383) is strictly decreasing for $\gamma_T < 1$, and it holds for $\gamma_T \leq 1-\epsilon$ for $\epsilon > 0$

$$\frac{1}{2}\left(\mathfrak{R}_T^{\mathrm{S1}}(F_\alpha, \boldsymbol{A}) + \mathfrak{R}_T^{\mathrm{S2}}(F_\alpha, \boldsymbol{A})\right) \geq 2 f_\alpha(1.5 - 0.5\epsilon) - (f_\alpha(1.5) + f_\alpha(1.5+0.5\epsilon)) > 0. \tag{32.385}$$

In other words, the fairness regret guarantee is not attainable[4] for values of $\gamma_T \leq 1-\epsilon$ for any T and $\epsilon > 0$. We can also verify that (A5) is violated when $\gamma_T \leq 1-\epsilon$ for any T and $\epsilon > 0$. Note that γ_T is defined to be in the set $[0.4, 1]$.

Under strategy 1 we have $x_\star = \frac{1}{2}$ and it holds

$$\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{u}_t(x_\star) = (1+0.5\gamma_T, 1+0.5\gamma_T), \text{ and } \boldsymbol{u}_t(x_\star) = \begin{cases} (1.5, 1.5) & \text{if } t \leq \gamma_T T, \\ (1,1) & \text{otherwise.} \end{cases} \tag{32.386}$$

Then, it holds

$$\mathbb{V}_{\mathcal{T}} \geq 2(1-\gamma_t)\gamma_T T \geq 2\epsilon\gamma_T T \geq 0.8\epsilon T = \Omega(T). \tag{32.387}$$

Moreover, it can easily be checked that $\mathbb{W}_{\mathcal{T}} = \Omega(T)$ because there is no decomposition $\{1, 2, \ldots, T\} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_K$ where $\max\{\mathcal{T}_k : k \in [K]\} = o\left(T^{\frac{1}{2}}\right)$ under which $\sum_{k=1}^{K} \sum_{i \in \mathcal{I}} \left|\sum_{t \in \mathcal{T}_k} \delta_{t,i}(\boldsymbol{x}_\star)\right| = o(T)$.

To conclude, when $\gamma_T = 1 - o(1)$, we have $\min\{\mathbb{V}_{\mathcal{T}}, \mathbb{W}_{\mathcal{T}}\} \leq \mathbb{V}_{\mathcal{T}} = o(T)$; thus, Assumption (A5) only holds when $\gamma_T = 1 - o(1)$ for which the vanishing fairness regret guarantee is attainable. Figure 32.12 provides a summary of the connection between the fairness regret under scenarios 1 and 2 and Assumption (A5).

□

[4]Note that the fairness regret must vanish for any adversarial choice of sequence of utilities.

33 Proof of Theorem 5.4.2

Proof.

Note that $\Psi_{\alpha,t} : \Theta \times \mathcal{X} \to \mathbb{R}$ is the function given by

$$\Psi_{\alpha,t}(\boldsymbol{\theta}, \boldsymbol{x}) = (-F_\alpha)^\star(\boldsymbol{\theta}) - \boldsymbol{\theta} \cdot \boldsymbol{u}_t(\boldsymbol{x}), \tag{33.388}$$

where $F_\alpha : \mathcal{U} \to \mathbb{R}$ is an α-fairness function (5.3). From Lemma 31.3, OGD operating over the set Θ under the $\frac{u_{\star,\min}^{1+1/\alpha}}{\alpha}$-strongly convex (Lemma 31.4) cost functions $\Psi_{\alpha,t}(\underline{\boldsymbol{\theta}_t}, \boldsymbol{x}_t)$ has the following regret guarantee against any fixed $\boldsymbol{\theta} \in \Theta$

$$\frac{1}{T}\sum_{t=1}^{T} \Psi_{\alpha,t}(\boldsymbol{\theta}_t, \boldsymbol{x}_t) - \frac{1}{T}\sum_{t=1}^{T} \Psi_{\alpha,t}(\boldsymbol{\theta}, \boldsymbol{x}_t) \leq \frac{1}{T} \cdot \underbrace{\frac{1}{2}\sum_{t=1}^{T} \frac{\alpha}{u_{\star,\min}^{1+1/\alpha} t} \left\|\boldsymbol{g}_{\Theta,t}\right\|_2^2}_{\mathfrak{R}_{T,\Theta}}, \tag{33.389}$$

From Lemma 31.2, it holds

$$\min_{\boldsymbol{\theta}\in\Theta} \frac{1}{T}\sum_{t=1}^{T} \Psi_{\alpha,t}(\boldsymbol{\theta}, \boldsymbol{x}_t) = F_\alpha\left(\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{u}_t(\boldsymbol{x}_t)\right). \tag{33.390}$$

Combine Eq. (33.389) and Eq. (33.390) to obtain the lower bound

$$F_\alpha\left(\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{u}_t(\boldsymbol{x}_t)\right) + \frac{\mathfrak{R}_{T,\Theta}}{T} \geq \frac{1}{T}\sum_{t=1}^{T} \Psi_{\alpha,t}(\boldsymbol{\theta}_t, \boldsymbol{x}_t). \tag{33.391}$$

From Lemma 31.3, OGD operating over the set $\mathcal{X}$ under the reward functions $\Psi_{\alpha,t}(\boldsymbol{\theta}_t, \boldsymbol{x})$ has the following regret guarantee for any fixed $\boldsymbol{x}_\star \in \mathcal{X}$:

$$\frac{1}{T}\sum_{t=1}^{T} \Psi_{\alpha,t}(\boldsymbol{\theta}_t, \boldsymbol{x}_\star) - \frac{1}{T}\sum_{t=1}^{T} \Psi_{\alpha,t}(\boldsymbol{\theta}_t, \boldsymbol{x}_t) \leq \frac{1}{T} \cdot \underbrace{1.5 \operatorname{diam}(\mathcal{X}) \sqrt{\sum_{t\in\mathcal{T}} \left\|\boldsymbol{g}_{\mathcal{X},t}\right\|_2^2}}_{\mathfrak{R}_{T,\mathcal{X}}}, \tag{33.392}$$

Hence, we have the following

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\Psi_{\alpha,t}(\boldsymbol{\theta}_t,\boldsymbol{x}_t)+\frac{\mathfrak{R}_{T,\mathcal{X}}}{T}\geq\frac{1}{T}\sum_{t=1}^{T}\Psi_{\alpha,t}(\boldsymbol{\theta}_t,\boldsymbol{x}_\star)\\
&=\frac{1}{T}\sum_{t=1}^{T}F^\star(\boldsymbol{\theta}_t)-\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\theta}_t\cdot\boldsymbol{u}_t(\boldsymbol{x}_\star) && \text{Replace } \Psi_{\alpha,t}(\boldsymbol{\theta}_t,\boldsymbol{x}_\star) \text{ using Eq. (33.388)}\\
&\geq F^\star\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\theta}_t\right)-\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\theta}_t\cdot\boldsymbol{u}_t(\boldsymbol{x}_\star) && \text{Jensen's inequality \& convexity of } F^\star\\
&\geq F^\star\left(\bar{\boldsymbol{\theta}}\right)-\bar{\boldsymbol{\theta}}\cdot\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{u}_t(\boldsymbol{x}_\star)\right)-\frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{\theta}_t-\bar{\boldsymbol{\theta}})\cdot\boldsymbol{u}_t(\boldsymbol{x}_\star)\\
&\geq\min_{\boldsymbol{\theta}\in\Theta}\left\{F^\star\left(\boldsymbol{\theta}\right)-\boldsymbol{\theta}\cdot\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{u}_t(\boldsymbol{x}_\star)\right)\right\}-\frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{\theta}_t-\bar{\boldsymbol{\theta}})\cdot\boldsymbol{u}_t(\boldsymbol{x}_\star)\\
&=F_\alpha\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{u}_t(\boldsymbol{x}_\star)\right)-\frac{1}{T}\sum_{t=1}^{T}\left(\boldsymbol{\theta}_t-\bar{\boldsymbol{\theta}}\right)\cdot\boldsymbol{u}_t(\boldsymbol{x}_\star). && (33.393)
\end{aligned}$$

We combine the above equation and Eq. (33.391) to obtain

$$\begin{aligned}
F_\alpha\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{u}_t(\boldsymbol{x}_\star)\right)-F_\alpha\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{u}_t(\boldsymbol{x}_t)\right)&\leq\frac{\mathfrak{R}_{T,\mathcal{X}}}{T}+\frac{\mathfrak{R}_{T,\Theta}}{T}+\frac{1}{T}\sum_{t=1}^{T}\left(\boldsymbol{\theta}_t-\bar{\boldsymbol{\theta}}\right)\cdot\boldsymbol{u}_t(\boldsymbol{x}_\star)\\
&=\frac{\mathfrak{R}_{T,\mathcal{X}}}{T}+\frac{\mathfrak{R}_{T,\Theta}}{T}+\underbrace{\frac{1}{T}\sum_{t=1}^{T}\left(\bar{\boldsymbol{\theta}}-\boldsymbol{\theta}_t\right)\cdot\boldsymbol{\delta}_t(\boldsymbol{x}_\star)}_{\Sigma} && (33.394)
\end{aligned}$$

We provide two approaches to bound the r.h.s. term Σ in Eq. (33.396), and this gives the two conditions in Assumption (A5):

Bound 1. We can bound the r.h.s. term Σ in the above equation as follows

$$\begin{aligned}
\Sigma=\bar{\boldsymbol{\theta}}\cdot\sum_{t=1}^{T}\boldsymbol{\delta}_t(\boldsymbol{x}_\star)-\sum_{t=1}^{T}\boldsymbol{\theta}_t\cdot\boldsymbol{\delta}_t(\boldsymbol{x}_\star)=-\sum_{t=1}^{T}\boldsymbol{\theta}_t\cdot\boldsymbol{\delta}_t(\boldsymbol{x}_\star) && (33.395)\\
\leq\frac{1}{u_{\star,\min}}\sum_{i\in\mathcal{I}}\sum_{t=1}^{T}\delta_{t,i}(\boldsymbol{x}_\star)\mathbb{1}_{\left\{\delta_{t,i}(\boldsymbol{x}_\star)\geq 0\right\}}=O\left(\mathbb{V}_{\mathcal{T}}\right). && (33.396)
\end{aligned}$$

Bound 2. We alternatively bound Σ as follows

$$\begin{aligned}
\Sigma&=\sum_{k=1}^{K}\sum_{t\in\mathcal{T}_k}\left(\bar{\boldsymbol{\theta}}-\boldsymbol{\theta}_t\right)\cdot\boldsymbol{\delta}_t(\boldsymbol{x}_\star)=\sum_{k=1}^{K}\sum_{t\in\mathcal{T}_k}\left(\bar{\boldsymbol{\theta}}-\boldsymbol{\theta}_{\min(\mathcal{T}_k)}\right)\cdot\boldsymbol{\delta}_t(\boldsymbol{x}_\star)+\sum_{k=1}^{K}\sum_{t\in\mathcal{T}_k}\left(\boldsymbol{\theta}_{\min(\mathcal{T}_k)}-\boldsymbol{\theta}_t\right)\cdot\boldsymbol{\delta}_t(\boldsymbol{x}_\star)\\
&\leq\Delta_\alpha\sum_{k=1}^{K}\left\|\sum_{t\in\mathcal{T}_k}\boldsymbol{\delta}_t(\boldsymbol{x}_\star)\right\|_1+u_{\max}\sum_{k=1}^{K}\sum_{t\in\mathcal{T}_k}\left\|\boldsymbol{\theta}_{\min(\mathcal{T}_k)}-\boldsymbol{\theta}_t\right\|_1, && (33.397)
\end{aligned}$$

where $\Delta_\alpha = \max\left\{\left\|\boldsymbol{\theta}-\boldsymbol{\theta}'\right\|_\infty : \boldsymbol{\theta},\boldsymbol{\theta}' \in \Theta\right\}$. We bound the term $\sum_{k=1}^K \sum_{t\in\mathcal{T}_k} \left\|\boldsymbol{\theta}_{\min(\mathcal{T}_k)} - \boldsymbol{\theta}_t\right\|_1$ in the above equation as

$$\sum_{k=1}^K \sum_{t\in\mathcal{T}_k} \left\|\boldsymbol{\theta}_{\min(\mathcal{T}_k)} - \boldsymbol{\theta}_t\right\|_1 \leq L_\Theta \sum_{k=1}^K \eta_{\Theta,\min(\mathcal{T}_k)} \sum_{t\in\mathcal{T}_k} (t - \min(\mathcal{T}_k)) \leq L_\Theta \sum_{k=1}^K \eta_{\Theta,\min(\mathcal{T}_k)} |\mathcal{T}_k|^2 \tag{33.398}$$

$$= L_\Theta \frac{u_{\star,\min}^{1+\frac{1}{\alpha}}}{\alpha} \sum_{k=1}^K \frac{|\mathcal{T}_k|^2}{\min(\mathcal{T}_k)}, \tag{33.399}$$

and replacing this upper-bound in Eq. (33.397) gives

$$\Sigma \leq \Delta_\alpha \sum_{k=1}^K \left\|\sum_{t\in\mathcal{T}_k} \boldsymbol{\delta}_t(\boldsymbol{x}_\star)\right\|_1 + u_{\max} L_\Theta \frac{\alpha}{u_{\star,\min}^{1+\frac{1}{\alpha}}} \sum_{k=1}^K \underbrace{\frac{|\mathcal{T}_k|^2}{\min(\mathcal{T}_k)}}_{\sum_{k'<k}|\mathcal{T}_k|+1} = O\left(\mathbb{W}_\mathcal{T}\right). \tag{33.400}$$

We combine Eq. (33.396), Eq. (33.400), and Eq. (33.394) to obtain

$$\mathfrak{R}_T(F_\alpha,\boldsymbol{\mathcal{A}}) \leq \sup_{\{\boldsymbol{u}_t\}_{t=1}^T\in\mathcal{U}^T} \left\{\frac{1}{T}\left(\mathfrak{R}_{T,\mathcal{X}} + \mathfrak{R}_{T,\Theta}\right)\right\} + O\left(\frac{\min\{\mathbb{V}_\mathcal{T},\mathbb{W}_\mathcal{T}\}}{T}\right) \tag{33.401}$$

$$\leq \sup_{\{\boldsymbol{u}_t\}_{t=1}^T\in\mathcal{U}^T} \left\{\frac{1}{T}\left(1.5\,\mathrm{diam}(\mathcal{X})\sqrt{\sum_{t\in\mathcal{T}} \left\|\boldsymbol{g}_{\mathcal{X},t}\right\|_2^2} + \frac{\alpha}{u_{\star,\min}^{1+\frac{1}{\alpha}}} \sum_{t=1}^T \frac{\left\|\boldsymbol{g}_{\Theta,t}\right\|_2^2}{t}\right)\right\} + O\left(\frac{\min\{\mathbb{V}_\mathcal{T},\mathbb{W}_\mathcal{T}\}}{T}\right). \tag{33.402}$$

The following upper bounds hold

$$\left\|\boldsymbol{g}_{\Theta,t}\right\|_2 = \left\|\left(\frac{1}{\left(-\theta_{t,i}\right)^{1/\alpha}} - (\boldsymbol{u}_t(\boldsymbol{x}_t))\right)_{i\in\mathcal{I}}\right\|_2 \leq \sqrt{I}\max\left\{\frac{1}{u_{\star,\min}^{1/\alpha}} - u_{\min}, u_{\max} - \frac{1}{u_{\star,\max}^{1/\alpha}}\right\} = L_\Theta,$$

$$\left\|\boldsymbol{g}_{\mathcal{X},t}\right\|_2 = \left\|\boldsymbol{\theta}_t \cdot \partial_{\boldsymbol{x}}\boldsymbol{u}_t(\boldsymbol{x}_t)\right\|_2 \leq \frac{1}{u_{\star,\min}^\alpha}\left\|\partial_{\boldsymbol{x}}\boldsymbol{u}_t(\boldsymbol{x}_t)\right\|_2 \leq \frac{L_\mathcal{X}}{u_{\star,\min}^\alpha}.$$

Thus, the regret bound in Eq. (33.402) can be upper bounded as

$$\mathfrak{R}_T(F_\alpha,\boldsymbol{\mathcal{A}}) = \frac{1}{T}\sup_{\{\boldsymbol{u}_t\}_{t=1}^T\in\mathcal{U}^T}\left\{1.5\,\mathrm{diam}(\mathcal{X})\frac{L_\mathcal{X}\sqrt{T}}{u_{\star,\min}^\alpha} + \frac{\alpha}{u_{\star,\min}^{1+\frac{1}{\alpha}}}\sum_{t=1}^T \frac{L_\Theta^2}{t}\right\} + \frac{\min\{\mathbb{V}_\mathcal{T},\mathbb{W}_\mathcal{T}\}}{T}$$

$$\leq \frac{1}{T}\sup_{\{\boldsymbol{u}_t\}_{t=1}^T\in\mathcal{U}^T}\left\{1.5\,\mathrm{diam}(\mathcal{X})\frac{L_\mathcal{X}\sqrt{T}}{u_{\star,\min}^\alpha} + \frac{\alpha}{u_{\star,\min}^{1+\frac{1}{\alpha}}}L_\Theta^2(\log(T)+1)\right\} + \frac{\min\{\mathbb{V}_\mathcal{T},\mathbb{W}_\mathcal{T}\}}{T}$$

$$= O\left(\frac{1}{\sqrt{T}} + \frac{\min\{\mathbb{V}_\mathcal{T},\mathbb{W}_\mathcal{T}\}}{T}\right).$$

This concludes the proof.

□

34 Proof of Theorem 5.4.3 (Lower Bound)

Proof.

Consider a scenario with a single player $\mathcal{I} = \{1\}$, $\mathcal{X} = \{x \in \mathbb{R}, |x| \leq 1\}$, and the utility selected by an adversary at time slot $t \in \mathcal{T}$ is given by

$$u_t(x) = w_t x + 1, \quad \text{where } w_t \in \{-1, +1\}. \tag{34.403}$$

The weight w_t is selected in $\{-1, +1\}$ uniformly at random for $t \in \mathcal{T}$. A policy $\mathcal{A}$ selects the sequence of decisions $\{x_t\}_{t=1}^T$ and has the following fairness regret

$$\mathbb{E}\left[\max_{x\in\mathcal{X}} f_\alpha\left(\frac{1}{T}\sum_{t=1}^T u_t(x)\right) - f_\alpha\left(\frac{1}{T}\sum_{t=1}^T u_t(x_t)\right)\right] \geq \mathbb{E}\left[\max_{x\in\mathcal{X}} f_\alpha\left(\frac{1}{T}\sum_{t=1}^T u_t(x)\right)\right] - \underbrace{f_\alpha\left(\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^T u_t(x_t)\right]\right)}_{=0}$$

$$= \mathbb{E}\left[f_\alpha\left(\max_{x\in\mathcal{X}} \frac{1}{T}\sum_{t=1}^T u_t(x)\right)\right] = \mathbb{E}\left[f_\alpha\left(\frac{1}{T}\left|\sum_{t=1}^T w_{t,1}\right| + 1\right)\right] \overset{\text{(a)}}{\geq} \mathbb{E}\left[\frac{1}{T}\left|\sum_{t=1}^T w_{t,1}\right|\right]\left(\frac{2^{1-\alpha}-1}{1-\alpha}\right) \overset{\text{(b)}}{\geq} \frac{\left(\frac{2^{1-\alpha}-1}{1-\alpha}\right)}{\sqrt{2T}}$$

$$= \Omega\left(\frac{1}{\sqrt{T}}\right).$$

Inequality (a) is obtained considering $f_\alpha(x+1)$ is concave in x, $f_\alpha(0+1) = 0$, and $f_\alpha(x+1) \geq f_\alpha(2)x$ for $x \in [0, 1]$. Inequality (b) is obtained through Khintchine inequality. A lower bound on the fairness regret (5.8) can be established:

$$\mathfrak{R}_T\left(F_\alpha, \boldsymbol{\mathcal{A}}\right) \geq \mathbb{E}\left[\max_{x\in\mathcal{X}} f_\alpha\left(\frac{1}{T}\sum_{t=1}^T u_t(x)\right) - f_\alpha\left(\frac{1}{T}\sum_{t=1}^T u_t(x_t)\right)\right] = \Omega\left(\frac{1}{\sqrt{T}}\right). \tag{34.404}$$

This concludes the proof.

□

35 Proof of Corollary 5.4.4

Proof.

Expected regret. When the utilities are i.i.d., we have the following

$$\mathbb{E}\left[\boldsymbol{u}_t(\boldsymbol{x})\right] = \boldsymbol{u}, \forall t \in \mathcal{T}, \tag{35.405}$$

for some fixed utility $\boldsymbol{u} \in \mathcal{U}$. In the proof Theorem 33, in particular, in Eq. (33.394) it holds

$$F_\alpha\left(\frac{1}{T}\sum_{t=1}^T \boldsymbol{u}_t(\boldsymbol{x}_\star)\right) - F_\alpha\left(\frac{1}{T}\sum_{t=1}^T \boldsymbol{u}_t(\boldsymbol{x}_t)\right) \leq \frac{\mathfrak{R}_{T,\mathcal{X}}}{T} + \frac{\mathfrak{R}_{T,\Theta}}{T} + \frac{1}{T}\sum_{t=1}^T \left(\boldsymbol{\theta}_t - \bar{\boldsymbol{\theta}}\right) \cdot \boldsymbol{u}_t(\boldsymbol{x}_\star). \tag{35.406}$$

Taking the expectation of both sides gives

$$\mathbb{E}\left[F_\alpha\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{u}_t(\boldsymbol{x}_\star)\right)-F_\alpha\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{u}_t(\boldsymbol{x}_t)\right)\right]\leq\mathbb{E}\left[\frac{\mathfrak{R}_{T,\mathcal{X}}}{T}+\frac{\mathfrak{R}_{T,\Theta}}{T}\right]+\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\left(\boldsymbol{\theta}_t-\bar{\boldsymbol{\theta}}\right)\cdot\boldsymbol{u}_t(\boldsymbol{x}_\star)\right]. \tag{35.407}$$

The variables $\boldsymbol{\theta}_t$ and $\boldsymbol{u}_t$ are independent for $t\in\mathcal{T}$, thus we have

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\left(\boldsymbol{\theta}_t-\bar{\boldsymbol{\theta}}\right)\cdot\boldsymbol{u}_t(\boldsymbol{x}_\star)\right]=\mathbb{E}\left[\left(\bar{\boldsymbol{\theta}}-\bar{\boldsymbol{\theta}}\right)\cdot\boldsymbol{u}(\boldsymbol{x}_\star)\right]=0. \tag{35.408}$$

Through Eq. (34.403), it holds

$$\mathbb{E}\left[F_\alpha\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{u}_t(\boldsymbol{x}_\star)\right)-F_\alpha\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{u}_t(\boldsymbol{x}_t)\right)\right]=O\left(\frac{1}{\sqrt{T}}\right). \tag{35.409}$$

This concludes the first part of the proof.

Almost-sure zero-regret. Let $\Delta=(u_{\max}-u_{\min})$, $\mathcal{T}=\mathcal{T}_1\cup\mathcal{T}_2\cup\cdots\cup\mathcal{T}_K$ where $K=T^{2/3}$ and $|\mathcal{T}_k|=\kappa=T^{1/3}$ for $k\in\{1,2,\ldots,K\}$, and let $\beta\in(0,1/6)$. Employing Hoeffding's inequality we can bound the l.h.s. term in Eq. (5.9) for $i\in\mathcal{I}$ as

$$\mathbb{P}\left(\left|\sum_{t\in\mathcal{T}_k}\delta_{t,i}(\boldsymbol{x})\right|\leq\Delta T^{1/6+\beta}\right)\geq 1-2\exp\left(\frac{-2T^{1/3+2\beta}}{((T-\kappa)\kappa^2/T^2+\kappa(\kappa/T-1)^2)}\right)=1-2\exp\left(\frac{-2T^{1/3+2\beta}}{(\kappa-\kappa^2/T)}\right) \tag{35.410}$$

$$=1-2\exp\left(\frac{-2T^{1/3+2\beta}}{\left(T^{1/3}-T^{-1/3}\right)}\right). \tag{35.411}$$

Hence, it follows

$$\begin{aligned}\mathbb{P}\left(\sum_{k=1}^{K}\sum_{i\in\mathcal{I}}\left|\sum_{t\in\mathcal{T}_k}\delta_{t,i}(\boldsymbol{x})\right|\leq\Delta T^{5/6+\beta}\right)&\geq\left(1-2\exp\left(\frac{-2T^{1/3+2\beta}}{\left(T^{1/3}-2T^{-1/3}\right)}\right)\right)^{IT^{2/3}}\\&\geq 1-2IT^{2/3}\exp\left(\frac{-2T^{1/3+2\beta}}{\left(T^{1/3}-T^{-1/3}\right)}\right)\qquad\text{Bernoulli's inequality}\\&\geq 1-2IT^{2/3}\exp\left(\frac{-2T^{1/3+2\beta}}{T^{1/3}}\right)\\&\geq 1-2IT^{2/3}\exp(-2T^{2\beta}).\end{aligned}$$

It follows from the above equation paired with Eq. (5.9)

$$\mathbb{W}_{\mathcal{T}}=O\left(T^{5/6+\beta}+T^{2/3}\right)=O\left(T^{5/6+\beta}\right),\qquad\text{w.p.}\qquad p\geq 1-2IT^{2/3}\exp(-2T^{2\beta}). \tag{35.412}$$

Thus, for any $\beta \in (0, 1/6)$ and $T \to \infty$, it holds

$$\frac{\mathbb{W}_{\mathcal{T}}}{T} \leq 0, \qquad \text{w.p.} \qquad p \geq 1. \tag{35.413}$$

Note that given that $\mathbb{W}_{\mathcal{T}} \geq 0$ in Eq. (5.10), it holds $\lim_{T\to\infty} \mathbb{W}_{\mathcal{T}} = 0$ w.p. $p = 1$. Therefore, it follows from Theorem 33 for $T \to \infty$

$$\mathfrak{R}_T\left(F_\alpha, \boldsymbol{\mathcal{A}}\right) = O\left(\frac{1}{\sqrt{T}} + \frac{\min\left\{\mathbb{V}_{\mathcal{T}}, \mathbb{W}_{\mathcal{T}}\right\}}{T}\right) = O\left(\frac{1}{\sqrt{T}} + \frac{\mathbb{W}_{\mathcal{T}}}{T}\right) \leq 0, \qquad \text{w.p.} \qquad 1. \tag{35.414}$$

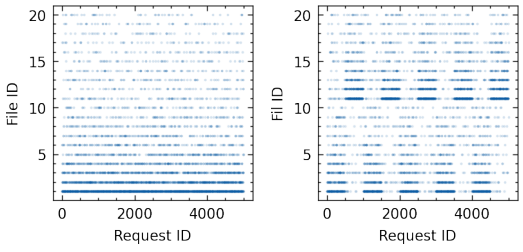
This concludes the proof.

□

36 Additional Experimental Details

Table 1: Specification of the network topologies used in experiments.

Topologies	$\lvert C\rvert$	$\lvert\mathcal{E}\rvert$	k_c	$\lvert Q_i\rvert$	$\lvert\cup_{f\in\mathcal{F}}\Lambda_f(C)\rvert$	w	Figure
Cycle	3	3	5–5	1	1	1–2	Fig. 5.5 (a)
Tree-1–Tree-3	13	12	1–5	2–5	1	1–9	Fig. 5.5 (b)–(d)
Grid	9	12	1–5	2	1	1–7	Fig. 5.5 (e)
Abilene	12	13	1–5	2	2	1–8	Fig. 5.5 (f)
GEANT	22	33	1–5	3	2	1–9	Fig. 5.5 (g)



(a) Stationary (b) Non-Stationary

Figure 36.13: Request traces stationary (a) and non-stationary (b) configured with $\sigma = 1.2$, $T = 5000$, $F = 20$, $D = 100$. Each dot indicates a requested file.

37 Departing and Arriving Agents

The system model in Section 5.3.3 supports departing and arriving agents. Consider a population of agents $\mathcal{I}$, the system may only observe a subset of the agents as the *participating* agents, $\mathcal{I}_t \subset \mathcal{I}$ at time t, and the utility of *absent* agents is simply $u_{t,i}(\cdot) = 0$. For example, in the extreme scenario where a single agent $t \in \mathcal{T}$ is participating at a given time slot, the long-term fairness objective (5.5) falls back to the slot-fairness objective (5.4), i.e., $F_\alpha\left(\sum_{t\in\mathcal{T}} \boldsymbol{u}_t(\boldsymbol{x}_t)\right) = \sum_{t\in\mathcal{T}} f_\alpha\left(u_{t,t}(\boldsymbol{x}_t)\right)$ where the fairness is ensured across the different agents arriving at different timeslots $t \in \mathcal{T}$. It is easy to verify that even in the case when the set of agents $\mathcal{I}$ is unknown to the controller in advance, one could augment the dual space with an extra dimension each time a new user appears, and the same guarantees hold.

38 Time-Complexity of Algorithm 5.1

Algorithm 1 applied to the virtualized caching system application has a time complexity $O\left(CF^2\right)$, where C is number of caches and F is the number of files in the catalog; the most expensive operation in Algorithm 1 is the projection step in line 8 that corresponds to the Euclidean projection onto a capped simplex, and this can be performed in $O\left(F^2\right)$ steps [56] for each cache state. Despite the high time complexity (F is typically large), in practice solvers (e.g., CVXPY [180]) support *warm-start* that speeds up the projection when the warm-start parameters are close to the ones of obtained by the solution, and since Algorithm 1 is iterative and the cache states do not severely change, typically a lower computational cost is achieved. Moreover, the proposed caching model in Section 5.6 supports request batching, where a batch includes the requests arriving between two consecutive cache updates. Batching amortizes the computational cost of the different policies, reducing the cost per request by the batch size (R_t at time slot t).

Bibliography

[1] Jiadai Wang, Jiajia Liu, and Nei Kato. Networking and Communications in Autonomous Driving: A Survey. *IEEE Communications Surveys & Tutorials*, 21(2):1243–1274, 2019.

[2] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. A Survey on Mobile Augmented Reality With 5G Mobile Edge Computing: Architectures, Applications, and Technical Aspects. *IEEE Communications Surveys & Tutorials*, 23(2):1160–1192, 2021.

[3] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis. 5G-Enabled Tactile Internet. *IEEE Journal on Selected Areas in Communications*, 34(3):460–473, 2016.

[4] J. G. Andrews, et al. What Will 5G be? *IEEE Journal on Selected Areas in Communications*, 32(6):1065–1082, 2014.

[5] G. S. Paschos, E. Bastug, I. Land, G. Caire, M. Debbah. Wireless Caching: Technical Misconceptions and Business Barriers. *IEEE Commun. Mag.*, 54(8):16–22, 2016.

[6] Stefano Traverso et al. Temporal Locality in Today's Content Caching: Why It Matters and How to Model It. *ACM SIGCOMM Computer Communication Review*, 43(5):5–12, November 2013.

[7] Mathieu Leconte, Georgios Paschos, Lazaros Gkatzikis, Moez Draief, Spyridon Vassilaras, and Symeon Chouvardas. Placing Dynamic Content in Caches with Small Population. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[8] Salah-Eddine Elayoubi and James Roberts. Performance and Cost Effectiveness of Caching in Mobile Access Networks. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, pages 79–88, 2015.

[9] Jose A. Ayala-Romero, Andres Garcia-Saavedra, Xavier Costa Pérez, George Iosifidis. EdgeBOL: Automating Energy-savings for Mobile Edge AI. In *ACM CoNEXT*, pages 397–410, 2021.

[10] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In *MobiCom '19: The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16. Association for Computing Machinery, New York, NY, USA, August 2019.

[11] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: adaptively unearthing the best cloud configurations for big data analytics. In *NSDI'17: Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, pages 469–482. USENIX Association, USA, March 2017.

[12] Tareq Si Salem, Gabriele Castellano, Giovanni Neglia, Fabio Pianese, and Andrea Araldo. Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees. *preprint arXiv:2105.02510*, 2021.

[13] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. No-Regret Caching via Online Mirror Descent. *preprint Arxiv:2101.12588*, 2021.

[14] AWS. Amazon Web Service ElastiCache, 2021.

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

[16] Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202–208, February 1985.

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

[18] Elias Koutsoupias. The k-server Problem. *Computer Science Review*, 3(2):105–118, May 2009.

[19] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis. Learning to Cache With No Regrets. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 235–243, 2019.

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

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

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

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

[24] Ronald Fagin. Asymptotic Miss Ratios over Independent References. *Journal of Computer and System Sciences*, 14(2):222–250, 1977.

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

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

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

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

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

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

[31] Emilio Leonardi and Giovanni Neglia. Implicit Coordination of Caches in Small Cell Networks Under Unknown Popularity Profiles. *IEEE Journal on Selected Areas in Communications*, 36(6):1276–1285, June 2018.

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

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

[34] 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.

[35] Weibo Chu, Mostafa Dehghan, John C.S. 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.

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

[37] Giovanni Neglia, Damiano Carra, and Pietro Michiardi. Cache Policies for Linear Utility Maximization. *IEEE/ACM Transactions on Networking*, 26(1):302–313, February 2018.

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

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

[40] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire. FemtoCaching: Wireless Content Delivery Through Distributed Caching Helpers. *IEEE Transactions on Information Theory*, 59(12):8402–8413, 2013.

[41] 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.

[42] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive Algorithms for On-Line Problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 322–333, New York, NY, USA, 1988. Association for Computing Machinery.

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

[44] Sébastien Bubeck, Michael B Cohen, Yin Tat Lee, James R Lee, and Aleksander Mądry. *K*-server via Multiscale Entropic Regularization. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*, pages 3–16, 2018.

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

[46] Martin Zinkevich. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 928–935. AAAI Press, 2003.

[47] N. Littlestone and M. K. Warmuth. The Weighted Majority Algorithm. *Information and computation*, 108(2):212–261, 1994.

[48] Shai Shalev-Shwartz. Online Learning and Online Convex Optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, February 2012.

[49] 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), June 2020.

[50] 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.

[51] Jun-Lin Lin. On the Diversity Constraints for Portfolio Optimization. *Entropy*, 15(11):4607–4621, 2013.

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

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

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

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

[56] Weiran Wang and Canyi Lu. Projection onto the Capped Simplex. *preprint arXiv:1503.01002*, 2015.

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

[58] William G Madow and Lillian H Madow. On the Theory of Systematic Sampling. *Ann. Math. Statist.*, 15(4):1–24, 1944.

[59] Gabriel Peyré, Marco Cuturi, et al. Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[60] Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. Access-Time-Aware Cache Algorithms. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2(4), November 2017.

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

[62] Nitish K. Panigrahy, Philippe Nain, Giovanni Neglia, and Don Towsley. A New Upper Bound on Cache Hit Probability for Non-Anticipative Caching Policies. *SIGMETRICS Performance Evaluation Review*, 48(3):138–143, March 2021.

[63] 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.

[64] Stratis Ioannidis and Edmund Yeh. Jointly Optimal Routing and Caching for Arbitrary Network Topologies. *IEEE Journal on Selected Areas in Communications*, 36(6):1258–1275, 2018.

[65] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. *An Analysis of Approximations for Maximizing Submodular set Functions—II*, pages 73–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[66] Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. Maximizing a Monotone Submodular Function subject to a Matroid Constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

[67] Alexander A Ageev and Maxim I Sviridenko. Pipage Rounding: a New Method of Constructing Algorithms with Proven Performance Guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

[68] Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent Randomized Rounding via Exchange Properties of Combinatorial Structures. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 575–584. IEEE, 2010.

[69] Yuval Filmus and Justin Ward. Monotone Submodular Maximization over a Matroid via Non-Oblivious Local Search. *SIAM Journal on Computing*, 43(2):514–542, 2014.

[70] Hamed Hassani, Mahdi Soltanolkotabi, and Amin Karbasi. Gradient Methods for Submodular Maximization. In *Advances in Neural Information Processing Systems*, pages 5841–5851, 2017.

[71] Aryan Mokhtari, Hamed Hassani, and Amin Karbasi. Conditional Gradient Method for Stochastic Submodular Maximization: Closing the Gap. In *International Conference on Artificial Intelligence and Statistics*, pages 1886–1895, 2018.

[72] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained Non-Monotone Submodular Maximization: Offline and Secretary Algorithms. In *International Workshop on Internet and Network Economics*, pages 246–257. Springer, 2010.

[73] TH Hubert Chan, Zhiyi Huang, Shaofeng H-C Jiang, Ning Kang, and Zhihao Gavin Tang. Online Submodular Maximization with Free Disposal: Randomization Beats $\frac{1}{4}$ for Partition Matroids. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1204–1223. SIAM, 2017.

[74] Matthew Streeter and Daniel Golovin. An Online Algorithm for Maximizing Submodular Functions. *Advances in Neural Information Processing Systems*, 21:1577–1584, 2008.

[75] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning Diverse Rankings with Multi-Armed Bandits. In *Proceedings of the 25th international conference on Machine learning*, pages 784–791, 2008.

[76] Lin Chen, Hamed Hassani, and Amin Karbasi. Online Continuous Submodular Maximization. In *International Conference on Artificial Intelligence and Statistics*, pages 1896–1905, 2018.

[77] Lin Chen, Christopher Harshaw, Hamed Hassani, and Amin Karbasi. Projection-Free Online Optimization with Stochastic Gradient: From Convexity to Submodularity. In *International Conference on Machine Learning*, pages 814–823, 2018.

[78] Matthew Streeter, Daniel Golovin, and Andreas Krause. Online Learning of Assignments. *Advances in Neural Information Processing Systems*, 22:1794–1802, 2009.

[79] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a Submodular Set Function subject to a Matroid Constraint. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 182–196. Springer, 2007.

[80] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-Armed Bandits in Metric Spaces. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 681–690, 2008.

[81] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration–Exploitation Tradeoff Using Bariance Estimates in Multi-Armed Bandits. *Theoretical Computer Science*, 410(19):1876–1902, 2009.

[82] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual Bandits with Linear Payoff Functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214. JMLR Workshop and Conference Proceedings, 2011.

[83] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. Taming the Monster: a Fast and Simple Algorithm for Contextual Bandits. In *International Conference on Machine Learning*, pages 1638–1646. PMLR, 2014.

[84] Miroslav Dudik, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang. Efficient Optimal Learning for Contextual Bandits. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 169–178, 2011.

[85] Mingrui Zhang, Lin Chen, Hamed Hassani, and Amin Karbasi. Online Continuous Submodular Maximization: From Full-Information to Bandit Feedback. In *NeurIPS*, 2019.

[86] Sanjeev Arora, Elad Hazan, and Satyen Kale. The Multiplicative Weights Update Method: A Meta-Algorithm and Applications. *Theory of Computing*, 8(1):121–164, 2012.

[87] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.

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

[89] Dario Rossi and Giuseppe Rossini. Caching Performance of Content Centric Networks under Multi-Path Routing (and More). *Relatório técnico, Telecom ParisTech*, pages 1–6, 2011.

[90] Ronald W Wolff. Poisson Arrivals See Time Averages. *Operations Research*, 30(2):223–231, 1982.

[91] Edith Cohen and Scott Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. *SIGCOMM Comput. Commun. Rev.*, 32(4):177–190, aug 2002.

[92] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, pages 1–12. ACM, 2009.

[93] Nikolaos Laoutaris, Sofia Syntila, and Ioannis Stavrakakis. Meta Algorithms for Hierarchical Web Caches. In *IEEE International Conference on Performance, Computing, and Communications, 2004*, pages 445–452, 2004.

[94] Eric Hall and Rebecca Willett. Dynamical Models and Tracking Regret in Online Convex Programming. In *International Conference on Machine Learning*, pages 579–587. PMLR, 2013.

[95] Eric C Hall and Rebecca M Willett. Online Convex Optimization in Dynamic Environments. *IEEE Journal of Selected Topics in Signal Processing*, 9(4):647–662, 2015.

[96] Omar Besbes, Yonatan Gur, and Assaf Zeevi. Non-stationary stochastic optimization. *Operations research*, 63(5):1227–1244, 2015.

[97] Zohar S Karnin and Oren Anava. Multi-armed bandits: Competing with optimal sequences. *Advances in Neural Information Processing Systems*, 29:199–207, 2016.

[98] N Bora Keskin and Assaf Zeevi. Chasing demand: Learning and earning in a changing environment. *Mathematics of Operations Research*, 42(2):277–307, 2017.

[99] Haipeng Luo, Chen-Yu Wei, Alekh Agarwal, and John Langford. Efficient contextual bandits in non-stationary worlds. In *Conference On Learning Theory*, pages 1739–1776. PMLR, 2018.

[100] Lilian Besson and Emilie Kaufmann. The generalized likelihood ratio test meets klucb: an improved algorithm for piece-wise non-stationary bandits. *Proceedings of Machine Learning Research vol XX*, 1:35, 2019.

[101] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal Regret Bounds for Reinforcement Learning. *Journal of Machine Learning Research*, 11(4), 2010.

[102] Yingjie Fei, Zhuoran Yang, Zhaoran Wang, and Qiaomin Xie. Dynamic Regret of Policy Optimization in Non-Stationary Environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[103] Weichao Mao, Kaiqing Zhang, Ruihao Zhu, David Simchi-Levi, and Tamer Basar. Near-Optimal Model-Free Reinforcement Learning in Non-Stationary Episodic MDPs. In *International Conference on Machine Learning*, pages 7447–7458. PMLR, 2021.

[104] Chávez, Gonzalo Navarro, Ricardo, and Marroquín. Searching in Metric Spaces. *ACM Comput. Surv.*, 33(3):273–321, September 2001.

[105] Stefan Berchtold, Christian Böhm, Bernhard Braunmüller, Daniel A Keim, and Hans-Peter Kriegel. Fast parallel similarity search in multimedia databases. *ACM SIGMOD Record*, 26(2):1–12, 1997.

[106] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. A Metric Cache for Similarity Search. In *Proceedings of the 2008 ACM workshop on Large-Scale distributed systems for information retrieval*, pages 43–50, 2008.

[107] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. Similarity Caching in Large-scale Image Retrieval. *Information Processing & Management*, 48(5):803–818, 2012. Large-Scale and Distributed Systems for Information Retrieval.

[108] Sandeep Pandey, Andrei Broder, Flavio Chierichetti, Vanja Josifovski, Ravi Kumar, and Sergei Vassilvitskii. Nearest-Neighbor Caching for Content-Match Applications. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 441–450, New York, NY, USA, 2009.

[109] Daniar Asanov et al. Algorithms and methods in recommender systems. *Berlin Institute of Technology, Berlin, Germany*, 2011.

[110] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri. Soft Cache Hits: Improving Performance Through Recommendation and Delivery of Related Content. *IEEE Journal on Selected Areas in Communications*, 36(6):1300–1313, 2018.

[111] Bin Yan, Derek R Lovley, and J Krushkal. Genome-wide similarity search for transcription factors and their binding sites in a metal-reducing prokaryote Geobacter sulfurreducens. *Biosystems*, 90(2):421–441, 2007.

[112] Auch, Klenk, and Göker. Standard operating procedure for calculating genome-to-genome distances based on high-scoring segment pairs. *Standards in genomic sciences*, 2(1):142, 2010.

[113] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[114] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 2014.

[115] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-Learning with Memory-Augmented Neural Networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1842–1850, New York, New York, USA, 20–22 Jun 2016. PMLR.

[116] Crankshaw, Wang, Gonzalez, and Michael J Franklin. Scalable training and serving of personalized models. In *NIPS 2015 Workshop on Machine Learning Systems*, 2015.

[117] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, Boston, MA, 2017. USENIX Association.

[118] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Cachier: Edge-Caching for Recognition Applications. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 276–286, 2017.

[119] Utsav Drolia, Katherine Guo, and Priya Narasimhan. Precog: Prefetching for Image Recognition Applications at the Edge. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '17, New York, NY, USA, 2017.

[120] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. FoggyCache: Cross-Device Approximate Computation Reuse. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, pages 19–34, New York, NY, USA, 2018.

[121] Peizhen Guo and Wenjun Hu. Potluck: Cross-Application Approximate Deduplication for Computation-Intensive Mobile Applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 271–284, New York, NY, USA, 2018.

[122] Srikumar Venugopal, Michele Gazzetti, Yiannis Gkoufas, and Kostas Katrinis. Shadow Puppets: Cloud-level Accurate AI Inference at the Speed and Economy of Edge. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA, July 2018.

[123] A. Kumar, A. Balasubramanian, S. Venkataraman, and A. Akella. Accelerating deep learning inference via freezing. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[124] A. Balasubramanian, A. Kumar, Y. Liu, H. Cao, S. Venkataraman, and A. Akella. Accelerating Deep Learning Inference via Learned Caches. *arXiv preprint arXiv:2101.07344*, 2021.

[125] Tareq Si Salem, Gabriele Castellano, Giovanni Neglia, Fabio Pianese, and Andrea Araldo. Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees. In *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, pages 1–8, 2021.

[126] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998.

[127] Giovanni Neglia, Michele Garetto, and Emilio Leonardi. Similarity Caching: Theory and Algorithms. In *IEEE/ACM Transactions on Networking*, pages 1–12, 2021.

[128] Thrasyvoulos Spyropoulos and Pavlos Sermpezis. Soft Cache Hits and the Impact of Alternative Content Recommendations on Mobile Edge Caching. In *Proceedings of the Eleventh ACM Workshop on Challenged Networks*, CHANTS '16, pages 51–56, New York, NY, USA, 2016. Association for Computing Machinery.

[129] D.G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.

[130] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[131] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *science*, 313(5786):504–507, 2006.

[132] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. Learning Compact Binary Descriptors With Unsupervised Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[133] Michael Zink, Ramesh Sitaraman, and Klara Nahrstedt. Scalable 360 video stream delivery: Challenges, solutions, and opportunities. *Proceedings of the IEEE*, 107(4):639–650, 2019.

[134] Similarity Caching Trace Repository. https://sim-cache.gitlabpages.inria.fr/similarity-caching-traces/.

[135] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, April 2004.

[136] J. Cortés and F. Bullo. Coordination and Geometric Optimization via Distributed Dynamical Systems. *SIAM Journal on Control and Optimization*, 44(5):1543–1574, 2005.

[137] Jorge Cortés, Sonia Martínez, and Francesco Bullo. Spatially-distributed coverage optimization and control with limited-range interactions. *ESAIM: COCV*, 11(4):691–719, 2005.

[138] Léon Bottou. On-line Learning and Stochastic Approximations. In David Saad, editor, *On-line Learning in Neural Networks*, pages 9–42. Cambridge University Press, New York, NY, USA, 1998.

[139] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. No-Regret Caching via Online Mirror Descent. In *ICC 2021 - IEEE International Conference on Communications*, pages 1–6, 2021.

[140] Alexandr Andoni and Piotr Indyk. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468, 2006.

[141] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021.

[142] Artem Babenko and Victor Lempitsky. The Inverted Multi-Index. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(6):1247–1260, 2015.

[143] Leonid Naidan, Boytsov and Eric Nyberg. Permutation search methods are efficient, yet faster search is possible. *arXiv preprint arXiv:1506.03163*, 2015.

[144] Yu A. Malkov and D. A. Yashunin. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

[145] A. Finamore, J. Roberts, M. Gallo, and D. Rossi. Accelerating Deep Learning Classification with Error-controlled Approximate-key Caching. In *IEEE Conference on Computer Communications (INFOCOM)*, 2022.

[146] T. Si Salem, G. Neglia, and D. Carra. AÇAI: Ascent Similarity Caching with Approximate Indexes. In *The 33rd International Teletraffic Congress (ITC 33)*, Avignon, France, August 2021.

[147] Faiss: A library for efficient similarity search. https://engineering.fb.com/data-infrastructure/faiss-a-library-for-efficient-similarity-search/.

[148] J. A. Bergstra and C. A. Middelburg. ITU-T Recommendation G.107 : The E-Model, a computational model for use in transmission planning. Technical report, 2003.

[149] H. Liu, Y. Li, Z. Duan, and C. Chen. A review on multi-objective optimization framework in wind energy forecasting techniques and applications. *Energy Conversion and Management*, 224:113324, 2020.

[150] Konstantinos Poularakis, George Iosifidis, Antonios Argyriou, and Leandros Tassiulas. Video delivery over heterogeneous cellular networks: Optimizing cost and performance. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1078–1086. IEEE, 2014.

[151] Damiano Carra, Giovanni Neglia, and Pietro Michiardi. Elastic Provisioning of Cloud Caches: A Cost-Aware TTL Approach. *IEEE/ACM Transactions on Networking*, 28(3):1283–1296, 2020.

[152] V.S. Borkar. *Stochastic Approximation: A Dynamical Systems Viewpoint*. Cambridge University Press, 2008.

[153] D.S. Berger, R.K. Sitaraman, and Mor Harchol-Balter. AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 483–498, Boston, MA, March 2017. USENIX Association.

[154] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. In Christian Beecks, Felix Borutta, Peer Kröger, and Thomas Seidl, editors, *Similarity Search and Applications*, pages 34–49, Cham, 2017.

[155] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. Viewport-adaptive navigable 360-degree video delivery. In *2017 IEEE international conference on communications (ICC)*, pages 1–7. IEEE, 2017.

[156] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 1–6, 2016.

[157] J. Park and Klara. Navigation Graph for Tiled Media Streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 447–455, 2019.

[158] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-Based Recommendations on Styles and Substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 43–52, New York, NY, USA, 2015.

[159] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative Deep Learning for Recommender Systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1235–1244, New York, NY, USA, 2015. Association for Computing Machinery.

[160] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. Collaborative Metric Learning. In *Proceedings of the 26th International Conference on World Wide Web*, pages 193–201, 2017.

[161] I. Shenbin, Anton, Elena, Valentin Malykh, and I. Nikolenko. RecVAE: A New Variational Autoencoder for Top-N Recommendations with Implicit Feedback. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, pages 528–536, NY, USA, 2020. Association for Computing Machinery.

[162] F. Maxwell Harper and Joseph A. Konstan. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[163] Peter N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 311–321, USA, 1993.

[164] Gonzalo Navarro. Searching in Metric Spaces by Spatial Approximation. *The VLDB Journal*, 11(1):28–46, aug 2002.

[165] M. Costantini and T. Spyropoulos. Impact of Popular Content Relational Structure on Joint Caching and Recommendation Policies. In *2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, pages 1–8, 2020.

[166] G. Paschos, E. Bastug, I. Land, G. Caire, and M. Debbah. Wireless Caching: Technical Misconceptions and Business Barriers. *IEEE Communications Magazine*, 54(8):16–22, 2016.

[167] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, 2017.

[168] J. Zhou, O. Simeone, X. Zhang, and W. Wang. Adaptive Offline and Online Similarity-Based Caching. *IEEE Networking Letters*, 2(4):175–179, 2020.

[169] Anirudh Sabnis, Tareq Si Salem, Giovanni Neglia, Michele Garetto, Emilio Leonardi, and Ramesh K Sitaraman. GRADES: Gradient Descent for Similarity Caching. In *IEEE Conference on Computer Communications (INFOCOM)*, 2021.

[170] Michele Garetto, Emilio Leonardi, and Giovanni Neglia. Content placement in networks of similarity caches. *Computer Networks*, 201:108570, 2021.

[171] 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.

[172] Giovanni Neglia, Emilio Leonardi, Guilherme Iecker Ricardo, and Thrasyvoulos Spyropoulos. A Swiss Army Knife for Online Caching in Small Cell Networks. *IEEE/ACM Transactions on Networking*, 29(6):2536–2547, 2021.

[173] JarosÅ,aw Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An Improved Approximation for k-median, and Positive Correlation in Budgeted Optimization. In *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 737–756, 2017.

[174] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing*, 12(4):559–592, 2014.

[175] Niklas Carlsson and Derek Eager. Worst-Case Bounds and Optimized Cache on Mth Request Cache Insertion Policies under Elastic Conditions. *ACM SIGMETRICS Performance Evaluation Review*, 46(3):37–38, jan 2019.

[176] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, USA, 2005.

[177] Andreas Krause and Daniel Golovin. Submodular Function Maximization. *Tractability*, 3:71–104, 2014.

[178] 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.

[179] G. L. Nemhauser and L. A. Wolsey. Best Algorithms for Approximating the Maximum of a Submodular Set Function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[180] Steven Diamond and Stephen Boyd. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *Journal of Machine Learning Research*, 17(1):2909–2913, jan 2016.

[181] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[182] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through Memorization: Nearest Neighbor Language Models. In *International Conference on Learning Representations (ICLR)*, 2020.

[183] Ion Stoica, Dawn Song, Raluca Ada Popa, David A. Patterson, Michael W. Mahoney, Randy H. Katz, Anthony D. Joseph, Michael Jordan, Joseph M. Hellerstein, Joseph Gonzalez, Ken Goldberg, Ali Ghodsi, David E. Culler, and Pieter Abbeel. A Berkeley View of Systems Challenges for AI. Technical Report UCB/EECS-2017-159, EECS Department, University of California, Berkeley, Oct 2017.

[184] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *preprint arXiv:1704.04861*, 2017.

[185] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proceedings of the IEEE*, 2020.

[186] Gautam Goel, Yiheng Lin, Haoyuan Sun, and Adam Wierman. Beyond Online Balanced Descent: An Optimal Algorithm for Smoothed Online Optimization. *Advances in Neural Information Processing Systems*, 2019.

[187] Shivnath Babu and Herodotos Herodotou. Massively Parallel Databases and MapReduce Systems. *Foundations and Trends® in Databases*, 2013.

[188] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, 2012.

[189] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and Open Problems in Federated Learning. *arXiv preprint arXiv:1912.04977*, 2021.

[190] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine*, 37(3):50–60, May 2020.

[191] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[192] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[193] Wentai Wu, Ligang He, Weiwei Lin, and Rui Mao. Accelerating Federated Learning over Reliability-Agnostic Clients in Mobile Edge Computing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[194] Giovanni Neglia, Gianmarco Calbi, Don Towsley, and Gayane Vardoyan. The Role of Network Topology for Distributed Machine Learning. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 2350–2358. IEEE, 2019.

[195] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. TensorFlow-Serving: Flexible, High-Performance ML Serving. *preprint arXiv:1712.06139*, 2017.

[196] David Chappell. Introducing Azure Machine Learning. *A guide for technical professionals, sponsored by microsoft corporation*, 2015.

[197] Vertex AI. https://cloud.google.com/vertex-ai.

[198] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 270–288. 2019.

[199] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. INFaaS: Managed and Model-less Inference Serving. *preprint arXiv:1905.13348*, 2019.

[200] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. InferLine: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 477–491, 2020.

[201] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. MLlib: Machine Learning in Apache Spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[202] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[203] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine Learning in Python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[204] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2016.

[205] N. Fernando, S. W. Loke, and W. Rahayu. Computing with Nearby Mobile Devices: A Work Sharing Algorithm for Mobile Edge-Clouds. *IEEE Transactions on Cloud Computing*, 7(2):329–343, 2019.

[206] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. *ACM SIGARCH Computer Architecture News*, 2017.

[207] S. Teerapittayanon, B. McDanel, and H. T. Kung. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016.

[208] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[209] Shuiguang Deng, Hailiang Zhao, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence. *IEEE Internet of Things Journal*, 2020.

[210] Samuel S. Ogden and Tian Guo. MODI: Mobile Deep Inference Made Efficient by Edge Computing. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[211] Yibo Jin, Lei Jiao, Zhuzhong Qian, Sheng Zhang, Ning Chen, Sanglu Lu, and Xiaoliang Wang. Provisioning Edge Inference as a Service via Online Learning. In *2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2020.

[212] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *IEEE/ACM Symposium on Edge Computing*, 2018.

[213] Xuanjia Qiu, Hongxing Li, Chuan Wu, Zongpeng Li, and Francis C.M. Lau. Cost-Minimizing Dynamic Migration of Content Distribution Services into Hybrid Clouds. *IEEE Transactions on Parallel and Distributed Systems*, 2015.

[214] J. Xu, L. Chen, and P. Zhou. Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 207–215, 2018.

[215] Shiqiang Wang, Rahul Urgaonkar, Ting He, Kevin Chan, Murtaza Zafer, and Kin K. Leung. Dynamic Service Placement for Mobile Micro-Clouds with Predicted Future Costs. *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[216] Ayoub Ben Ameur, Andrea Araldo, et al. On the Deployability of Augmented Reality Using Embedded Edge Devices. In *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, 2021.

[217] Minseok Choi, Joongheon Kim, and Jaekyun Moon. Wireless Video Caching and Dynamic Streming Under Differentiated Quality Requirements. *IEEE Journal on Selected Areas in Communications*, 36, 2018.

[218] Zakaria Ye, Francesco De Pellegrini, Rachid El-Azouzi, Lorenzo Maggi, and Tania Jimenez. Quality-Aware DASH Video Caching Schemes at Mobile Edge. In *2017 29th International Teletraffic Congress (ITC 29)*, volume 1, pages 205–213. IEEE, 2017.

[219] Cheng Zhan and Zhe Wen. Content Cache Placement for Scalable Video in Heterogeneous Wireless Network. *IEEE Communications Letters*, 21(12):2714–2717, 2017.

[220] Andrea Araldo, Fabio Martignon, and Dario Rossi. Representation selection problem: Optimizing video delivery through caching. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 323–331. IEEE, 2016.

[221] Zhihao Qu, Baoliu Ye, Bin Tang, Song Guo, Sanglu Lu, and Weihua Zhuang. Cooperative Caching for Multiple Bitrate Videos in Small Cell Edges. *IEEE Transactions on Mobile Computing*, 19(2):288–299, 2020.

[222] Konstantinos Poularakis, George Iosifidis, Antonios Argyriou, Iordanis Koutsopoulos, and Leandros Tassiulas. Caching and Operator Cooperation Policies for Layered Video Content Delivery. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[223] Michele Garetto, Emilio Leonardi, and Giovanni Neglia. Content Placement in Networks of Similarity Caches. *arXiv preprint arXiv:2102.04974*, 2021.

[224] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the State of Neural Network Pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[225] Geoffrey Hinton et al. Distilling the Knowledge in a Neural Network. *preprint arXiv:1503.02531*, 2015.

[226] Sujith Ravi. Custom On-Device ML Models with Learn2Compress, 2018.

[227] Zhou Fang, Tong Yu, Ole J. Mengshoel, and Rajesh K. Gupta. QoS-Aware Scheduling of Heterogeneous Servers for Inference in Deep Neural Networks. In *ACM Conference on Information and Knowledge Management (CIKM)*, volume Part F1318, pages 2067–2070, 2017.

[228] Yu Wang, Gu Yeon Wei, and David Brooks. Benchmarking TPU, GPU, and CPU platforms for deep learning. *arXiv preprint arXiv:1907.10701*, 2019.

[229] Yaron Fairstein, Ariel Kulik, Joseph Seffi Naor, Danny Raz, and Hadas Shachnai. A (1-1/e-ε)-Approximation for the Monotone Submodular Multiple Knapsack Problem. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[230] Alberto Ceselli, Marco Premoli, and Stefano Secci. Mobile Edge Cloud Network Design Optimization. *IEEE/ACM Transactions on Networking*, 25(3):1818–1831, 2017.

[231] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *preprint arXiv:2004.10934*, 2020.

[232] Yuxuan Cai et al. YOLObile: Real-Time Object Detection on Mobile Devices via Compression-Compilation Co-Design. *preprint arXiv:2009.05697*, 2020.

[233] F.P. Kelly, A. Maulloo, and D. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness, and Stability. *J. Oper. Res. Soc.*, 49(3):237–252, 1998.

[234] Jeonghoon Mo and Jean Walrand. Fair End-to-End Window-Based Congestion Control. *IEEE/ACM Transactions on networking*, 8(5):556–567, 2000.

[235] X. Lin, N. B. Shroff, and R. Srikant. A Tutorial on Cross-Layer Optimization in Wireless Networks. *IEEE J. Sel. Areas Commun.*, 24(8):1452–1463, 2006.

[236] Resource Allocation and Cross Layer Control in Wireless Networks. L. Georgiadis, M. J. Neely, and L. Tassiulas. *Found. Trends Netw.*, 1(1):1–143, 2006.

[237] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multiresource Allocation: Fairness-Efficiency Tradeoffs in a Unifying Framework. *IEEE/ACM Trans. on Networking*, 21(6):1785–1798, 2013.

[238] W. Wang, B. Li, and B. Liang. Dominant Resource Fairness in Cloud Computing Systems with Heterogeneous Servers. In *IEEE INFOCOM*, 2014.

[239] T. Bonald, and J. W. Roberts. Multi-Resource Fairness: Objectives, Algorithms and Performance. In *ACM Sigmetrics*, 2015.

[240] Pawan Kumar and Rakesh Kumar. Issues and Challenges of Load Balancing Techniques in Cloud Computing: A Survey. 51(6), feb 2019.

[241] Jiao Zhang, F. Richard Yu, Shuo Wang, Tao Huang, Zengyi Liu, and Yunjie Liu. Load Balancing in Data Center Networks: A Survey. *IEEE Communications Surveys & Tutorials*, 20(3):2324–2352, 2018.

[242] Francis Ysidro Edgeworth et al. Mathematical Psychics. *History of Economic Thought Books*, 1881.

[243] John F. Nash. The Bargaining Problem. *Econometrica*, 18(2):155–162, 1950.

[244] Ehud Kalai, Meir Smorodinsky, et al. Other Solutions to Nash's Bargaining Problem. *Econometrica*, 43(3):513–518, 1975.

[245] Leonidas Georgiadis, George Iosifidis, and Leandros Tassiulas. Exchange of services in networks: competition, cooperation, and fairness. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 43–56, 2015.

[246] Bozidar Radunovic and Jean-Yves Le Boudec. A Unified Framework for Max-Min and Min-Max Fairness With Applications. *IEEE/ACM Transactions on networking*, 15(5):1073–1083, 2007.

[247] Dritan Nace and Michal Pioro. Max-min Fairness and its Applications to Routing and Load-balancing in Communication Networks: a Tutorial. *IEEE Communications Surveys & Tutorials*, 10(4):5–17, 2008.

[248] Swati Gupta and Vijay Kamble. Individual Fairness in Hindsight. *J. Mach. Learn. Res.*, 22(144):1–35, 2021.

[249] Luofeng Liao, Yuan Gao, and Christian Kroer. Nonstationary Dual Averaging and Online Fair Allocation. *ArXiv e-prints*, February 2022.

[250] Devansh Jalota and Yinyu Ye. Online Learning in Fisher Markets with Unknown Agent Preferences. *arXiv preprint arXiv:2205.00825*, 2022.

[251] Sean R. Sinclair, Siddhartha Banerjee, and Christina Lee Yu. Sequential Fair Allocation: Achieving the Optimal Envy-Efficiency Tradeoff Curve. *SIGMETRICS Perform. Eval. Rev.*, 50(1):95–96, jun 2022.

[252] Naram Mhaisen, George Iosifidis, and Douglas Leith. Online Caching with no Regret: Optimistic Learning via Recommendations. *arXiv preprint arXiv:2204.09345*, 2022.

[253] Debjit Paria and Abhishek Sinha. LeadCache: Regret-Optimal Caching in Networks. *Advances in Neural Information Processing Systems*, 34:4435–4447, 2021.

[254] Archana Bura, Desik Rengarajan, Dileep Kalathil, Srinivas Shakkottai, and Jean-Francois Chamberland. Learning to Cache and Caching to Learn: Regret Analysis of Caching Algorithms. *IEEE/ACM Transactions on Networking*, 30(1):18–31, 2021.

[255] Yuanyuan Li and Stratis Ioannidis. Cache Networks of Counting Queues. *IEEE/ACM Transactions on Networking*, 29(6):2751–2764, 2021.

[256] Tareq Si Salem, Giovanni Neglia, and Damiano Carra. AÇAI: Ascent Similarity Caching with Approximate Indexes. In *2021 33rd International Teletraffic Congress (ITC-33)*, pages 1–9. IEEE, 2021.

[257] Shie Mannor, John N Tsitsiklis, and Jia Yuan Yu. Online Learning with Sample Path Constraints. *Journal of Machine Learning Research*, 10(3), 2009.

[258] Daron Anderson, George Iosifidis, and Douglas J Leith. Lazy Lagrangians with Predictions for Online Learning. *arXiv preprint arXiv:2201.02890*, 2022.

[259] Adrian Rivera, He Wang, and Huan Xu. The Online Saddle Point Problem and Online Convex Optimization with Knapsacks. *preprint arXiv:1806.08301*, 2018.

[260] Santiago R Balseiro, Haihao Lu, and Vahab Mirrokni. The Best of Many Worlds: Dual Mirror Descent for Online Allocation Problems. *Operations Research*, 2022.

[261] Yu-Hang Zhou, Chen Liang, Nan Li, Cheng Yang, Shenghuo Zhu, and Rong Jin. Robust online matching with user arrival distribution drift. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 459–466, 2019.

[262] John C Duchi, Alekh Agarwal, Mikael Johansson, and Michael I Jordan. Ergodic Mirror Descent. *SIAM Journal on Optimization*, 22(4):1549–1578, 2012.

[263] Holger Boche and Martin Schubert. A Generalization of Nash Bargaining and Proportional Fairness to Log-Convex Utility Sets With Power Constraints. *IEEE Transactions on Information Theory*, 57(6):3390–3404, 2011.

[264] George Iosifidis, Lin Gao, Jianwei Huang, and Leandros Tassiulas. Efficient and Fair Collaborative Mobile Internet Access. *IEEE/ACM Transactions on Networking*, 25(3):1386–1400, 2017.

[265] Wenjie Jiang, Rui Zhang-Shen, Jennifer Rexford, and Mung Chiang. Cooperative Content Distribution and Traffic Engineering in an ISP Network. *SIGMETRICS Performance Evaluation Review*, 37(1):239–250, jun 2009.

[266] Liang Wang, Gareth Tyson, Jussi Kangasharju, and Jon Crowcroft. Milking the Cache Cow With Fairness in Mind. *IEEE/ACM Transactions on Networking*, 25(5):2686–2700, 2017.

[267] Eitan Altman, Konstantin Avrachenkov, and Andrey Garnaev. Generalized α-Fair Resource Allocation in Wireless Networks. In *2008 47th IEEE Conference on Decision and Control*, pages 2414–2419. IEEE, 2008.

[268] Dimitris Bertsimas, Vivek F Farias, and Nikolaos Trichakis. The Price of Fairness. *Operations Research*, 59(1), 2011.

[269] Anthony B. Atkinson. On the Measurement of Inequality. *Journal of Economic Theory*, 2(3):244–263, 1970.

[270] Mohammad Sadegh Talebi and Alexandre Proutiere. Learning Proportionally Fair Allocations with Low Regret. *SIGMETRICS Performance Evaluation Review*, 46(1):50–52, jun 2018.

[271] Yasushi Kawase and Hanna Sumita. Online Max-min Fair Allocation. *ArXiv e-prints*, November 2021.

[272] Siddhartha Banerjee, Vasilis Gkatzelis, Artur Gorokh, and Billy Jin. Online Nash Social Welfare Maximization with Predictions. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–19. SIAM, 2022.

[273] MohammadHossein Bateni, Yiwei Chen, Dragos Florin Ciocan, and Vahab Mirrokni. Fair Resource Allocation in A Volatile Marketplace. *Operations Research*, 70(1):288–308, 2022.

[274] Semih Cayci, Swati Gupta, and Atilla Eryilmaz. Group-Fair Online Allocation in Continuous Time. *Advances in Neural Information Processing Systems*, 33:13750–13761, 2020.

[275] Gerdus Benade, Aleksandr M Kazachkov, Ariel D Procaccia, and Christos-Alexandros Psomas. How to Make Envy Vanish over Time. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 593–610, 2018.

[276] David Zeng and Alexandros Psomas. Fairness-Efficiency Tradeoffs in Dynamic Fair Division. In *Proceedings of the 21st ACM Conference on Economics and Computation*, pages 911–912, 2020.

[277] Sean R. Sinclair, Gauri Jain, Siddhartha Banerjee, and Christina Lee Yu. Sequential Fair Allocation of Limited Resources under Stochastic Demands. *ArXiv e-prints*, November 2020.

[278] Jackie Baek and Vivek Farias. Fair Exploration via Axiomatic Bargaining. In *NeurIPS*, 2021.

[279] H Brendan McMahan. A Survey of Algorithms and Analysis for Adaptive Online Learning. *The Journal of Machine Learning Research*, 18(1):3117–3166, 2017.

[280] Eyal Even-Dar, Robert Kleinberg, Shie Mannor, and Yishay Mansour. Online Learning with Global Cost Functions. In *22nd Annual Conference on Learning Theory, COLT*, 2009.

[281] David Blackwell. An Analog of the Minmax Theorem for Vector Payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, 1956.

[282] Alexander Rakhlin, Karthik Sridharan, and Ambuj Tewari. Online Learning: Beyond Regret. In Sham M. Kakade and Ulrike von Luxburg, editors, *Proceedings of the 24th Annual Conference on Learning Theory*, volume 19 of *Proceedings of Machine Learning Research*, pages 559–594, Budapest, Hungary, 09–11 Jun 2011. PMLR.

[283] Shipra Agrawal and Nikhil R. Devanur. Bandits with Concave Rewards and Convex Knapsacks. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation*, EC '14, pages 989–1006, New York, NY, USA, 2014. Association for Computing Machinery.

[284] Edmund Eisenberg and David Gale. Consensus of subjective probabilities: The pari-mutuel method. *The Annals of Mathematical Statistics*, 30(1):165–168, 1959.

[285] Dimitris Bertsimas, Vivek F Farias, and Nikolaos Trichakis. On The Efficiency-Fairness Trade-Off. *Management Science*, 58(12):2234–2250, 2012.

[286] Andrea Lodi, Philippe Olivier, Gilles Pesant, and Sriram Sankaranarayanan. Fairness over time in dynamic resource allocation with an application in Healthcare. *arXiv preprint arXiv:2101.03716*, 2021.

[287] Hossein Esfandiari, Nitish Korula, and Vahab Mirrokni. Online Allocation with Traffic Spikes: Mixing Adversarial and Stochastic Models. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 169–186, 2015.

[288] Alexander Rakhlin and Karthik Sridharan. Online Learning with Predictable Sequences. In *Conference on Learning Theory*, pages 993–1019. PMLR, 2013.

[289] Peter Auer, NicolÃ² Cesa-Bianchi, and Claudio Gentile. Adaptive and Self-Confident On-Line Learning Algorithms. *Journal of Computer and System Sciences*, 64(1):48–75, 2002.

[290] John C Harsanyi and Reinhard Selten. A generalized Nash solution for two-person bargaining games with incomplete information. *Management science*, 18(5-part-2):80–106, 1972.

[291] Yuezhou Liu, Yuanyuan Li, Qian Ma, Stratis Ioannidis, and Edmund Yeh. Fair Caching Networks. *Performance Evaluation*, 143:102138, 2020.

[292] Shai Shalev-Shwartz and Yoram Singer. *Online Learning: Theory, Algorithms, and Applications*. PhD thesis, Hebrew University, 2007.

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

[294] Rodrigo Paredes and Gonzalo Navarro. Optimal Incremental Sorting. In *2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 171–182. SIAM, 2006.

[295] Daniel Golovin, Andreas Krause, and Matthew Streeter. Online submodular maximization under a matroid constraint with application to learning assignments. *arXiv preprint arXiv:1407.1082*, 2014.

[296] Boris S Mordukhovich and Nguyen Mau Nam. Geometric Approach to Convex Subdifferential Calculus. *Optimization*, 66(6):839–873, 2017.

[297] R Tyrrell Rockafellar. *Convex Analysis*. Number 28. Princeton university press, 1970.

[298] Shai Shalev-Shwartz. *Online Learning: Theory, Algorithms, and Applications*. PhD thesis, The Hebrew University of Jerusalem, 2007.

[299] Michel X Goemans and David P Williamson. New $\frac{3}{4}$-Approximation Algorithms for the Maximum Satisfiability Problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994.

[300] Herbert S Wilf. Some Applications of the Inequality of Arithmetic and Geometric Means to Polynomial Equations. In *Proceedings of the American Mathematical Society*, volume 14, pages 263–265, 1963.

List of Figures

List of Tables