Probabilistic analysis for caching

Younes Ben Mazziane

UNIVERSITÉ CÔTE D'AZUR

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

THÈSE DE DOCTORAT

Analyse Probabiliste pour le Caching

Younes BEN MAZZIANE

Centre Inria d'Université Côte d'Azur, équipe NEO

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

Dirigée par : Sara ALOUF, Chargée de Recherche, Inria
Co-encadrée par : Giovanni NEGLIA, Directeur de Recherche, Inria
Soutenue le : 13 Mai 2024

Devant le jury, composé de :
Frédéric GIROIRE, Directeur de Recherche, CNRS
Emilio LEONARDI, Professeur, Politecnico di Torino
György DÁN, Professeur, KTH Royal Institute of Technology
Nicolas GAST, Chargé de Recherche, Inria

ANALYSE PROBABILISTE POUR LE CACHING

Probabilistic Analysis for Caching

Younes BEN MAZZIANE

Jury :

Président du jury
Frédéric GIROIRE, Directeur de Recherche, CNRS

Rapporteurs
Emilio LEONARDI, Professeur, Politecnico di Torino
György DÁN, Professeur, KTH Royal Institute of Technology

Examinateurs
Nicolas GAST, Chargé de Recherche, Inria

Directrice de thèse
Sara ALOUF, Chargée de Recherche, Inria

Co-encadrant de thèse
Giovanni NEGLIA, Directeur de Recherche, Inria

Université Côte d'Azur

Younes BEN MAZZIANE

Analyse Probabiliste pour le Caching

xiii+114 p.

Analyse Probabiliste pour le Caching

Résumé

Les caches sont de petites mémoires qui accélèrent la récupération des données. L'un des objectifs des politiques de mise en cache est de sélectionner le contenu du cache afin de minimiser le temps de réponse aux requêtes d'objets. Un problème plus général permet de répondre approximativement à la requête d'un objet par un objet similaire mis en cache. Ce concept, appelé "mise en cache par similarité", s'avère utile pour les systèmes de recommandation. L'objectif est de minimiser le temps de latence tout en fournissant des réponses satisfaisantes.

La compréhension théorique des algorithmes de gestion de la mémoire cache, sous des hypothèses spécifiques sur les requêtes, aide à choisir un algorithme approprié. Les politiques d'éviction du cache les plus répandues sont celles de l'utilisation la moins fréquente (LFU) et de l'utilisation la moins récente (LRU). LFU est efficace lorsque le processus requêtes est stationnaire, et LRU s'adapte aux changements dans les processus de requêtes. Les algorithmes d'apprentissage séquentiel, tels que l'algorithme aléatoire Follow-the-Perturbed Leader (FPL), appliqués à la mise en cache, bénéficient de garanties théoriques même dans le pire des cas.

LFU et FPL s'appuient sur le nombre de requêtes d'objets. Cependant, le comptage est un défi dans les scénarios à mémoire limitée. Pour y remédier, les politiques de mise en cache utilisent des schémas de comptage approximatifs, tels que la structure de données Count-Min Sketch avec mises à jour conservatrices (CMS-CU), afin d'équilibrer la précision des comptages et l'utilisation de la mémoire. Dans le cadre de la mise en cache par similarité, RND-LRU est une stratégie LRU modifiée. Malheureusement, il reste difficile de quantifier théoriquement à la fois la performance d'un cache LFU utilisant CMS-CU, celle d'un cache FPL avec un algorithme de comptage approximatif, ainsi que celle de RND-LRU.

Cette thèse explore trois algorithmes probabilistes : CMS-CU, FPL avec des estimations bruitées des nombres de requêtes d'objets (NFPL) et RND-LRU. Pour CMS-CU, nous proposons une approche novatrice pour trouver de nouvelles bornes supérieures sur l'espérance et le complémentaire de la fonction de répartition de l'erreur d'estimation sous un processus de requêtes i.i.d. De plus, nous démontrons que NFPL se comporte aussi bien que la politique de mise en cache statique, optimale et omnisciente, quelle que soit la séquence de requêtes (sous certaines conditions sur les comptages bruités). Enfin, nous introduisons une nouvelle politique de mise en cache qui est analytiquement résoluble. Nous montrons alors que cette politique approxime RND-LRU.

Mots-clés : comptage approximatif, algorithmes probabilistes, apprentissage séquentiel

Probabilistic Analysis for Caching

Abstract

Caches are small memories that speed up data retrieval. Caching policies may aim to choose cache content to minimize latency in responding to item requests. A more general problem permits an item's request to be approximately answered by a similar cached item. This concept, referred to as "similarity caching," proves valuable for content-based image retrieval and recommendation systems. The objective is to further minimize latency while delivering satisfactory answers.

Theoretical understanding of cache memory management algorithms under specific assumptions on the requests provides guidelines for choosing a suitable algorithm. The Least-Frequently-Used (LFU) and the Least-Recently-Used (LRU) are popular caching eviction policies. LFU is efficient when the requests process is stationary, while LRU adapts to changes in the patterns of the requests. Online learning algorithms, such as the randomized Follow-the-Perturbed Leader (FPL) algorithm, applied for caching, enjoy worst-case guarantees.

Both LFU and FPL rely on items' request count. However, counting is challenging in memory-constrained scenarios. To overcome this problem, caching policies operate with approximate counting schemes, such as the Count-Min Sketch with Conservative Updates (CMS-CU), to balance counts' accuracy and memory usage. In the similarity caching setting, RND-LRU is a modified LRU where a request is probabilistically answered by the most similar cached item. Unfortunately, a theoretical analysis of an LFU cache utilizing CMS-CU, an FPL cache with an approximate counting algorithm, and RND-LRU remains difficult.

This thesis investigates three randomized algorithms: CMS-CU, FPL with noisy items' request counts estimations (NFPL), and RND-LRU. For CMS-CU, we propose a novel approach to derive new upper bounds on the expected value and the complementary cumulative distribution function of the estimation error under a renewal request process. Additionally, we prove that NFPL behaves as well as the optimal omniscient static caching policy for any request sequence under specific conditions on the noisy counts. Finally, we introduce a new analytically tractable similarity caching policy and show that it can approximate RND-LRU.

Keywords: Approximate counting, online learning, randomized algorithms

Acknowledgements

I would like to express my sincere gratitude to my thesis advisors, Sara Alouf and Giovanni Neglia, for their unwavering patience, support, help, and guidance throughout this journey. Their insights and expertise have been invaluable, and our numerous discussions have greatly shaped the direction and content of this thesis.

I am also deeply thankful to Daniel Sadoc Menasche and Francescomaria Faticanti for their valuable contributions to this manuscript. Their input and feedback were instrumental in refining my research and ensuring the rigor and clarity of this work.

Additionally, I extend my heartfelt thanks to Frédéric Giroire, Emilio Leonardi, György Dán, and Nicolas Gast for graciously accepting to be part of my Ph.D. defense jury. I truly appreciate their commitment, as demonstrated by the time they allocated to reviewing my thesis, participating in my defense, and offering crucial comments and advice. Their expertise and perspectives have enriched my work and contributed significantly to its improvement.

I would also like to acknowledge the support and encouragement from my colleagues and friends, who provided both intellectual and emotional support during my Ph.D. journey. Their camaraderie and encouragement have been a source of motivation and strength.

Finally, I am eternally grateful to my family for their unconditional love and support. Their faith in me has been a constant source of inspiration and strength throughout this endeavor.

Contents

Appendix

CHAPTER 1

Introduction

Data retrieval systems often duplicate a subset of files from the main memory into cache memories. When requests involve cached items, responses are expedited compared to those for non-cached items retrieved directly from the main memory. Thus, the effectiveness of data retrieval systems relies on the policy overseeing the small cache memory. Caching policies typically lack foresight into future requests, leading them to store items in the cache based on previous requests. The percentage of data fulfilled by the cache serves as a performance metric for the caching policy. The choice of such a policy depends on the characteristics of the request process in the considered application.

Mapping an algorithm's input and parameters to a performance metric is crucial for determining its suitability for a specific application. Moreover, this mapping allows the comparison of algorithms for the same task according to a performance metric. One could identify this mapping through simulations. However, this solution is computationally expensive when dealing with vast potential values for the input and the parameters. The difficulty intensifies when the algorithm incorporates randomness, turning the performance metric into a random variable and demanding additional computational power to infer the mapping accurately. This computational challenge is particularly evident in randomized caching policies, where a large number of possible request sequences and parameter values exist. Theoretical analysis of algorithms can provide simple formulas capturing the performance metric of the algorithm as a function of the input and the parameters, alleviating the need for high computational cost simulations.

In this thesis, we provide a theoretical analysis of three randomized algorithms in caching under specific assumptions on the request sequence. The remainder of this introductory chapter is organized as follows: Section 1.2 presents mathematical models for the request sequence from the literature to capture the characteristics of real-world traffic. Following this, we review popular caching algorithms and their theoretical analysis under different traffic models in Section 1.3. Many caching policies base their decisions on the request count for items. We discuss in Section 1.4 different approaches to efficiently count items's appearances over a large data stream. Section 1.5 presents three randomized algorithms that we analyze in this thesis. Finally, Section 1.6 outlines the publications associated with this thesis.

1.1 Caching Applications

In CPUs, cache memories are small, high-speed memories employed to store portions of the main memory. Cache memories boosted the performance of CPUs due to a phenomenon known as temporal locality, where currently accessed items are likely to be accessed in the near future [Smi82].

As computing technology advanced and personal computers proliferation, caching enhanced the performance of local disk access [Smi85].

The invention of the internet in the 1990s marked a new era and led to the development of web caching to expedite content delivery. Web caching replicates popular items in small proxy servers or in the user's machine. This idea offers dual advantages: it reduces latency for users and mitigates network traffic between the proxy server and the original server [Wan99].

In wireless networks, mobile data usage increased, and consequently, it was necessary to deploy faster and more efficient content retrieval systems. Subsequently, caching techniques were employed at the edge of wireless networks, particularly within femtocell base stations [Liu+16].

Content-based image retrieval (CBIR) systems [Fal+08] benefit from deploying cache memories. In these systems, users submit image queries to retrieve visually similar images. The cache intercepts user requests, performs a local similarity search within stored items, and provides results if deemed satisfactory. Such a cache is termed a similarity cache and was later on proposed for contextual advertising [Pan+09] and recommendation systems [Ser+18].

Today, caching constitutes an integral component of nearly every computing system, spanning web browsers and databases to operating systems and distributed networks.

1.2 Traffic Model

In caching, a "hit" denotes the successful retrieval of a requested item from the cache. Conversely, a "miss" occurs when the requested item is not found in the cache, necessitating retrieval from the main memory or original server. The hit ratio (proportion of requests served by the cache) and the byte-hit ratio (fraction of bytes served) are metrics to measure the effectiveness of caching policies. Devising a caching policy that maximizes the hit ratio requires understanding the patterns of the request process. To this aim, researchers proposed mathematical models for the request process that simulate user behavior and item access patterns. We distinguish two categories of models.

Stochastic. A stochastic model assumes the request process is a realization from a specific stochastic process. The Independent Reference Model (IRM) is a basic model where item requests are independent, and the requested item is sampled from a fixed catalog using a categorical distribution [Fag77]. Numerous web caching policies are assessed under IRM assumption, often with request distributions following a generalized Zipf law [Cha+07]. Despite its apparent simplicity, IRM assumption proves acceptable as an approximation in scenarios where popularity variations occur gradually [Bre+99]. Renewal models [FRP16] generalizes IRM: requests for different items are independent, and the inter-arrival times for requests for the same item are identically and independently distributed (i.i.d.) random variables. Markovian models [Cas11] capture correlations in the request process. Renewal and Markovian models may fall short in capturing non-stationarity in the request process. In response, Traverso et al. [Tra+13] propose a novel traffic model named the Shot-Noise Model (SNM). Distinguishing itself from renewal processes, SNM envisions a potentially infinite catalog where each item has a distinct lifespan.

Adversarial. An adversarial model makes no prior statistical assumptions about the request process. This implies that requests may be thought to be generated by an adversary seeking to

maximize the cost, such as the number of misses, incurred by a specified caching policy. Analyzing caching policies within adversarial models provides insights into their worst-case performance and resilience against nonstationary request processes. In this context, the cost of the algorithm is compared to that of an optimal, either static or dynamic, policy with hindsight, i.e., with knowledge of future requests. We distinguish two types of analysis.

Competitive analysis. The performance metric is the competitive ratio, which is the ratio of the cost incurred by a caching policy and the cost of Belady's optimal eviction policy with hindsight [Bel66]. Notably, the Least-Recently-Used (LRU) and First-In-First-Out (FIFO) eviction policies achieve the optimal competitive ratio for any deterministic policy [ST85]. Moreover, randomized policies achieving the optimal competitive ratio for any randomized policy have been proposed [MS91; ACN00].

Regret analysis. The performance metric is regret, which is the difference between the cost of a caching policy and the cost of an optimal static policy with hindsight. In this setting, the aim is to design no-regret algorithms, i.e., policies whose regret grows sublinearly with the time horizon. Recently, many no-regret caching policies, inspired by the theory of Online Convex Optimization (OCO) [Zin03], have been proposed [Pas+19a; BBS20; SNI23]. These policies were motivated by the traffic in wireless networks' edge, where requests lack statistical regularity due to the smaller demand volume per edge cache and users' mobility between cells [Pas+18]. Andrew et al. [And+13] proves that no algorithm can have both sublinear regret and a constant competitive ratio.

Similarity caching is a generalization of the classical caching problem. Items are typically represented as vectors in a metric space, and the distance between their representative vectors, called embeddings, quantifies the degree of similarity. In similarity caching, there are two types of hits: exact and approximate. When evaluating the performance of a similarity caching policy, the hit ratio metric may be modified to take into account the quality of the hits. The design of a similarity caching policy takes into account the request characteristics and the representation of the items in the metric space. Recently, Neglia et al. [NGL21] studied the similarity caching problem under stochastic and adversarial traffic models.

1.3 Cache Management Algorithms

Caching systems may comprise a singular cache overseen by a caching policy dictating the rules for item admission or eviction. More complex systems incorporate multiple interconnected caches, allowing each cache to communicate with neighboring caches by forwarding requests. Examples of such intricate systems include hierarchical web and file system caches. Various caching policies tailored for networked caches have been proposed, often building upon strategies designed for managing individual caches [RKT10; Deh+17; JPD17].

This thesis focuses on algorithms designed for a singular cache. The literature on algorithms for managing an individual cache is extensive. Among the fundamental caching policies are the Least Recently Used (LRU) and the Least Frequently Used (LFU). We examine each of these policies and their variants.

LRU. This policy keeps the most recently requested items in the cache and removes the least recently used item upon a cache miss. LRU is widely used in practice because of its simplicity and good performance. Researchers studied LRU under adversarial and stochastic settings. Sleator and Tarjan [ST85] proved that LRU achieves the optimal competitive ratio. However, it is easy to prove that LRU has linear regret [Pas+19a]. The exact computation of LRU's hit ratio under IRM is computationally expensive [WK71], but Fagin [Fag77] proposed an efficient method to approximate it. Moreover, he proved that the aforementioned method is asymptotically accurate. This method is later called Che's approximation [CTW02], TTL approximation [JNT18], and extended to other LRU variants and under more generalized assumptions on the request process [LT15; GV17; JNT18].

LFU. This policy stores the items with the largest request counts. LFU achieves the optimal hit ratio under IRM [SKW00]. However, LFU suffers from two limitations. First, keeping statistics for all requested items may be expensive in terms of memory requirements. Second, LFU fails to exploit temporal locality. To address these limitations, many variations of LFU were proposed. For example, Window-LFU [KS02] keeps statistics over the last W requests. Tiny-LFU [EFM17] uses approximate counting data structures such as the Count-Min Sketch with Conservative Updates (CMS-CU) [EV02; CM05b] offering a trade-off between memory usage and accuracy in estimating items requests count. Under adversarial traffic, LFU has unbounded competitive ratio [CKZ01] and linear regret [Pas+19a].

Other caching policies combine ideas from LFU and LRU such as LRFU [Lee+01] and ARC [MMb]. Beyond recency and frequency, web caching policies take into account the size of the items and their retrieval costs [You91; Che98; BSH17]. Shuja et al. [Shu+21] survey machine learning techniques for devising caching policies.

No Regret Caching Policies. Recently, many papers have proposed caching policies based on algorithms from Online Convex Optimization (OCO) [Haz16]. These policies are studied under the adversarial model and evaluated via the static regret metric, which is the difference between the cost of the algorithm and the cost of the static optimal policy with knowledge of future requests. Prominent algorithms from OCO, such as Online-Gradient-Descent (OGD), Online-Mirror-Descent (OMD), Follow-the-Regulated-Leader (FRL), and Follow-the-Perturbed-Leader (FPL), enable the development of no-regret caching policies.

Paschos et al. [Pas+19a] were the first to derive a caching policy from OGD. Si Salem et al. [SNI23] proposed a caching policy based on OMD computationally less expensive than the OGD caching policy. In scenarios where a caching policy is endowed with oracle predictions regarding future requests, Mhaisen et al. [MIL23] formulate caching policies based on Follow-the-Regulated-Leader (FRL). These policies achieve sub-zero regret under perfect predictions and maintain a sublinear regret bound, even in the presence of arbitrarily inaccurate predictions. Policies based on OGD, OMD, or FRL are designed for continuous caching, such that it is assumed that each item is divided into a large number of chunks such that storing them can be approximated by continuous variables. Addressing this, Bhattacharjee et al.[BBS20] applied the Follow-the-Perturbed-Leader (FPL) algorithm for discrete caching. Mhaisen et al. [Mha+22a] modify the FPL caching algorithm to incorporate predictions with unknown quality while maintaining sublinear regret. Caching

policies based on FPL are renowned for their lower computational cost in comparison to OGD, OMD, and FRL [Mha+22a].

Similarity Caching Policies. Pandey et al.[Pan+09] introduced similarity caching policies as modified versions of LRU and LFU. One such adaptation is SIM-LRU, which replies to a given query with the nearest cached item if its dissimilarity is below a predefined threshold. Competitive analysis has been applied to evaluate SIM-LRU's performance [CKV09]. RND-LRU, a randomized version of SIM-LRU, CLS-LRU, SIM-LFU, and Q-cache are other similarity caching policies proposed in [Fal+08; Pan+09]. Neglia et al.[NGL21] formalized the similarity caching problem and introduced new policies with optimality guarantees under specific conditions. Recently, gradient-based similarity caching policies have also been put forward [Sab+21; SNC23].

1.4 Approximate counting

Numerous caching policies base their decisions on the previous requests' counts of items. A naive approach for monitoring the request count of items in a data stream utilizes a hash table to store key-value pairs. Each key represents the identifier of an item, and its associated value tracks the number of times the item has appeared thus far. This approach is memory expensive for high-speed, large data streams [Ben+17]. To address this issue, approximate counting algorithms provide a trade-off between memory usage and accuracy; they can be distinguished into two main types: counter-based and sketch-based.

Counter-Based Algorithms. Similarly to the exact counting mechanism, these algorithms maintain a hashtable where each entry is a key-value pair representing the item's identifier and an associated count. Popular algorithms include *Approximate Counting* [Mor78], *Lossy Counting* [MMa], *Frequent* [KSP03] and *Space Saving* [MAE05]. To address the limitations of the exact counting algorithm, these approaches maintain a constraint on the maximum number of entries. They periodically decrement counters larger than zero to prevent them from reaching their maximum value, and employ probabilistic counter increments to reduce memory accesses. The count estimation error for any item, employing *Frequent* and *Space Saving* algorithms with M as the maximal number of entries, is $\mathcal{O}\left(\frac{1}{M}\right)$, aligning with the optimal error for any deterministic counting algorithm with M entries [Ber+10].

Sketch-Based Algorithms. In contrast to counter-based techniques, these algorithms assign multiple counters to an individual item, with the peculiarity that counters are shared among multiple items. Sketch-based algorithms employ hash functions to map counters to items, eliminating the necessity to store the item's identifier. The selection of hash functions is done uniformly at random from a predefined family, introducing an element of randomness into these algorithms. Popular sketch algorithms include the Count-Sketch (CS) [CCF04] and the Count-Min Sketch (CMS) [CM05a]. The trade-off between memory and accuracy for CMS and CS is well understood [CCF04; CM05a; CM05b]. Many recent efficient counting algorithms are variants of CMS and CS [Yan+18b; Yan+18a; Li+20; Zha+21b].

1.5 Challenges and Contributions

1.5.1 Sketch algorithms for approximate counting

The Count-Min Sketch maintains a matrix of counters with w columns and d rows. An item is mapped to a counter from each row $r \in \{1, \dots d\}$ via a hash function h_r. Upon a request for an item n, the counters $(1, h_1(n)), \dots, (d, h_d(n))$ are incremented. Therefore, the values of the d counters of an item n represent upper bounds on n's request count. The Count-Min Sketch estimates n's request count with the minimum value among all n's counters. While CMS increments all the selected counters, an update rule known as the *Conservative Update* [EV02] or *Minimal Increment* [CM03], increments only the counters with the smallest value among the selected ones. Empirically, this update rule improves the accuracy of CMS. Nevertheless, a theoretical understanding of the advantages of the Conservative Update is missing in the literature. In CMS, a counter (r, l)'s value is equal to the aggregated request count of all items n such that $h_r(n) = l$. Conversely, in CMS with Conservative Updates (CMS-CU), requests for an item n, such that $h_r(n) = l$, only increment the counter (r, l) based on the state of the other selected counters. This correlation between counters' growth makes the analysis of CMS-CU challenging.

In this thesis, we prove new bounds on the complementary cumulative distribution function and the expected value of the estimation error for CMS-CU under an i.i.d. request process (IRM), providing a theoretical explanation for the advantages of the Conservative Update rule. In fact, under heterogeneous item probabilities, our formulas show that popular items' error estimation in CMS-CU can be considerably smaller than in CMS. Moreover, based on our bounds, we provide configuration rules for CMS-CU to detect the top C popular items with a specific precision. We conducted simulations to compare our theoretical bounds with empirical estimations. These contributions are based on our works [BAN22a; BAN22b], and are presented in Chapter 2.

1.5.2 Online learning for caching

Studying caching policies under adversarial models provides insight into their worst-case performance. Many simple caching policies achieve the optimal competitive ratio, including LRU [ST85]. However, classical caching policies such as LRU and LFU do not achieve sublinear regret.

In Section 1.3, we listed no-regret caching policies derived from popular algorithms in the theory of Online Convex Optimization (OCO). When applied to caching, OCO algorithms necessitate a memory proportional to the total number of items. FPL-based caching policies require this memory for counting the requests for each item. We explained in Section 1.4 the challenges in monitoring the exact request count for each item. We also highlighted that approximate counting algorithms tackle those challenges by compromising accuracy in estimating requests' counts. A natural question is whether an FPL caching policy, when coupled with an approximate counting scheme, maintains its sublinear regret property.

In this thesis, we prove that an FPL caching policy maintains a sublinear regret under specific conditions on the requests' count estimator. These conditions are for example verified when the requests' estimates are obtained by subsampling the request process. More specifically, we prove that the Follow-the-Perturbed-Leader with the limited knowledge of a substream generated from sampling each request from the original stream of length T with probability f has a sublinear regret

of $\mathcal{O}\left(\frac{1}{f}\sqrt{T}\right)$. Additionally, we run simulations to evaluate the performance of the Follow-the-Perturbed-Leader with noisy requests' estimates due to sampling, over synthetic and real-world traces. These contributions are based on [Ben+23] and are presented in Chapter 3.

1.5.3 LRU-based similarity caching policies

As indicated in Section 1.3, accurately computing the hit ratio of LRU under IRM is challenging. The difficulty arises from the exponential growth of the Markov chain's state space, which represents LRU dynamics, with the total number of items. The TTL approximation provides a linear-time estimation of LRU's hit ratio, proving to be asymptotically exact. LRU has correlated caching decisions across items due to the cache capacity constraint. Under IRM, the TTL approximation models LRU as a system where per-item caching decisions are independent and the cache capacity constraint holds only on average.

RND-LRU is a generalization of LRU for the similarity caching setting. Caching decisions for items in RND-LRU are strongly coupled, influenced by both the cache capacity constraint and the potential for a cached item to serve requests for similar items. This coupling prompts the question of whether the TTL approximation can be extended to RND-LRU.

In this thesis, we propose an extension for the TTL approximation, called the RND-TTL approximation, to estimate the hit ratio of RND-LRU under IRM. In particular, we approximate RND-LRU through another ideal caching policy, RND-TTL, where per-item caching decisions across items are independent, and the cache capacity constraint is satisfied in expectation.

This approximation models RND-LRU with a novel similarity caching model, that we call RND-TTL, where caching decisions across items are independent, and the cache capacity constraint is satisfied in expectation. We run simulations on synthetic and real-world traces to evaluate the accuracy of our approximation for evaluating RND-LRU's hit ratio. These contributions are based on [Ben+22; Ben+24] and presented in Chapter 4.

1.6 Publications

The contributions of this manuscript led to the following publications in peer-reviewed journals and conferences:

[BAN22a] *Ben Mazziane, Y., Alouf, S., & Neglia, G. (2022, May). A Formal Analysis of the Count-Min Sketch with Conservative Updates. In IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).*

[BAN22b] *Ben Mazziane, Y., Alouf, S.,& Neglia, G. (2022 November). Analyzing Count Min Sketch with Conservative Updates. Computer Networks, 217, 109315.*

[Ben+22] *Ben Mazziane, Y., Alouf, S., Neglia, G., & Menasche, D. S. (2022, December). Computing the Hit rate of Similarity Caching. In IEEE Global Communications Conference (GLOBECOM).*

[Ben+23] *Ben Mazziane, Y., Faticanti, F., Neglia, G., & Alouf, S. (2023 November). No-Regret Caching with Noisy Request Estimates. In IEEE Virtual Conference on Communications (VCC).*

[Ben+24] *Ben Mazziane, Y., Alouf, S., Neglia, G., & Menasche, D. S. (2024, March). TTL Model for an LRU-Based Similarity Caching policy. Computer Networks, 241, 110206.*

CHAPTER 2

Approximate Counting

LFU variants can benefit from approximate counting schemes to balance memory and accuracy. As discussed in Section 1.5.1, the conservative update improves the accuracy of the widely used Count-Min Sketch for approximate counting. Nonetheless, theoretical analysis of the performance of Count-Min Sketch with Conservative Updates (CMS-CU) is still missing because of its inherent difficulty. In this chapter, we propose a novel approach to study CMS-CU and derive new upper bounds on both the expected value and the complementary cumulative distribution function of the estimation error under an i.i.d. request process.

2.1 Introduction

Counting how many times a given item appears in a data stream is a basic step common to a variety of applications spanning different domains including network management. For example, routers and servers often routinely count the number of packets in each flow for troubleshooting, traffic monitoring [Ben+17], detection of denial of service attacks [LSK11], etc. Similarly, caching policies often rely on content popularity estimates [EFM17]. Counting is a deceptively simple operation: in many applications, the available memory does not permit to instantiate a counter for each possible item, because the number of items is huge (e.g., catalogs of cacheable objects in content delivery networks) or because counters are updated frequently and then require expensive fast memories (e.g., for high-rate inline packet flow processing). As a consequence, these applications rely on approximate counting techniques such as sketch-based algorithms [CH10], among which a popular one is the Count-Min Sketch (CMS) [CM05a]. Many recent sketch algorithms are variations of CMS [Yan+18b; Hsu+19; Zha+21b; Yan+21].

CMS achieves significant memory reduction by mapping different items to the same counters through hash functions. As different items may increment the same counter, CMS suffers from overestimation errors. When counters are only incremented, a slight modification to CMS operation, referred to as Conservative Update [EV02] or Minimal Increment [CM03], can reduce the estimation error. The Count-Min sketch with Conservative Updates (CMS-CU) is successfully employed for caching [EFM17], heavy flows detection [Wan+21], telemarketing call detection [BdN11], and natural language processing [GDC12].

Although conservative updates are a minor modification to CMS operation, they entangle the growth of the counters, making CMS-CU much more difficult to study than CMS. As CMS-CU reduces CMS estimation errors, it is still possible to maintain the upper bounds originally proposed for CMS [CM05a; CM05b]. This approach has been adopted in some papers, for example, to study

CMS-CU's trade-off between memory and accuracy [Wan+21; Ven+20], but it obviously fails to capture the specific advantages offered by CMS-CU.

To the best of our knowledge, only a few papers ventured to study CMS-CU [Bia+12; EF15; Che+21; FK23a; FK23b]. Bianchi et al. relied on a fluid approximation under the assumption that all counters are equally likely to be updated at each step [Bia+12]. This assumption may be satisfied only for a large number of counters and a large number of items with similar popularity. In [EF15], Einziger and Friedman modeled CMS-CU as a stack of Bloom filters [BM03] and derived bounds for the error's Complementary Cumulative Distribution Function (CCDF) when requests follow the Independent Reference Model (IRM) [Fag77]. Unfortunately, the CCDF computation in [EF15] requires an iterative procedure, whose time complexity grows quadratically with the target error value (and then in general with the stream length). Fusy and Kucherov [FK23a; FK23b] studied CMS-CU under IRM with uniform probabilities, revealing a phase transition in the conservative update strategy's accuracy. This transition depends on the ratio of distinct items to the number of counters. The analysis in [Bia+12; EF15; FK23a; FK23b] holds for families of strongly k-universal hash functions [MR95], only for sufficiently large values of k. Such families are however incompatible with memory-constrained applications that need CMS-CU since memory requirements and computation time grow with k [Sie04]. The authors of [Che+21] proposed a statistical estimator for CMS-CU's error, but it requires access to the counters associated with one hash function for a representative stream. Last, the bounds derived in [Bia+12; EF15; Che+21; FK23a; FK23b] are the same for all items regardless of their popularity, even though it has been observed in [Bia+12] that the most popular items are better estimated than less popular ones.

In this chapter, we propose a novel analysis of CMS-CU that leads to new upper bounds on both the CCDF and the expected value of the estimation error under an IRM request process. Our methodology diverges from related work as it quantifies the error on a per-item basis, which is particularly suitable for data streams with heterogeneous items' popularities. The analysis also overcomes the limitations of the previous studies as (i) it holds for pairwise independent hash functions, and (ii) it provides CCDF expressions with time complexity independent of the error's value. We show that our formulas can be successfully employed to derive both improved estimates for the precision of popular items' detection methods and improved configuration rules for CMS-CU. We compare our new bounds both qualitatively and quantitatively to those in [CM05a], [CM05b], [Bia+12], [EF15].

The rest of this chapter is organized as follows. In Section 2.2, we provide the background, review the state-of-the-art studies, and introduce the notation. The theoretical analysis is carried out in Section 2.3. Section 2.4 presents numerical experiments both on synthetic and real-world traces. Section 2.5 concludes the chapter.

2.2 Background, Notation, and Assumptions

2.2.1 Data Stream Model

A data stream is a sequence $S_t = (Z(s))_{s=1,\dots,t}$, where $Z(s)$ is an item from a universe $I = \{1, \dots, N\}$ [Mut05]. In general, we want to compute a function of the sequence, $\mathcal{F}(S_t)$, for example, the number of occurrences of a given item, the set of heavy-hitters (items whose number of requests exceeds a given threshold), or the top-k most frequent items. Streaming algorithms aim to compute the function of interest using a few passes through the data stream (only one for the

applications we consider) with a sublinear amount of memory in the universe's size N and the data stream size t. Even for the simple quantities mentioned above, an exact computation requires a linear amount of memory and then streaming algorithms need to settle for approximate results. The next section presents two popular streaming algorithms for approximate counting.

In the following, we use $[\![a, b]\!]$ to represent the set of integers between a and b and $[M]$ to represent the set of integers from 1 to $M \in \mathbb{N}$. Moreover, we do not append the sketch name to the symbols to lighten the notation. We believe there will be no ambiguity as each sketch is presented and analyzed in a separate section.

2.2.2 Count-Min Sketch (CMS)

A Count-Min sketch is a two-dimensional array with d rows, each with w counters. An item i is mapped to d counters, one per row, via d hash functions $\{h_r\}_{r\in[d]}$ chosen uniformly at random from a family of strongly 2-universal hash functions. We note that once selected, the hash functions do not change during the processing of the stream S_t. We model the association between items and counters as a bipartite undirected graph $G = (I, O, E)$, where O is the set of counters and $E \triangleq \{(i, h_r(i)) : i \in I, r \in [d]\}$ is the set of edges. We denote the open neighbourhood of item i in the graph as $N_G(i) \triangleq \{c : (i, c) \in E\}$ (we naturally have $|N_G(i)| = d$, for any item i). We denote the value at time t of the counter in row r corresponding to item i as $c_i^r(t)$, with $c_i^r(0) = 0$. When item i is requested at time t, the counters $\{h_r(i)\}_{r\in[d]}$ are incremented by 1. Namely,

$$c_i^r(t) = c_i^r(t-1) + 1, \ \forall r \in [d]. \tag{2.1}$$

Let $n_i(t)$ denote the number of occurrences of item i in the stream up to time t. Note that $c_i^r(t)$ is updated not only by new requests for item i, but also by requests for all items that are also mapped by h_r to the same counter $h_r(i)$, i.e., by all items in the set $\{j \in I : h_r(j) = h_r(i)\}$. These items are said to *collide* with i. It follows that $c_i^r(t) = \sum_{j:\ h_r(j)=h_r(i)} n_j(t)$. As such, $c_i^r(t)$ upper bounds $n_i(t)$. We denote the error resulting from using $c_i^r(t)$ for estimating $n_i(t)$ as $e_i^r(t)$, i.e., $e_i^r(t) \triangleq c_i^r(t) - n_i(t)$. Since all counters' values $\{c_i^r(t)\}_{r\in[d]}$ upper bound $n_i(t)$, their minimum also upper bounds $n_i(t)$. This minimum is the estimate of $n_i(t)$ provided by CMS and we denote it as $\hat{n}_i(t)$,

$$\hat{n}_i(t) \triangleq \min_{r\in[d]} c_i^r(t) \ . \tag{2.2}$$

The estimation error is then

$$e_i(t) \ \triangleq \hat{n}_i(t) - n_i(t) = \min_{r\in[d]} e_i^r(t) \ . \tag{2.3}$$

We also introduce $\delta_{i,j}^r(s)$ to represent the contribution of item $j \neq i$ to counter $h_r(i)$ at time s. We have:

$$\delta_{i,j}^r(s) \triangleq \mathbb{1}\big(Z(s) = j, \ h_r(i) = h_r(j)\big) \ , \tag{2.4}$$

$$e_i^r(t) = \sum_{s\in[t]} \sum_{j\in I\setminus\{i\}} \delta_{i,j}^r(s) \ . \tag{2.5}$$

All quantities we defined are random variables due to the initial random choice of the hash functions. From (2.5) and the definition of strongly 2-universal hash functions [MR95], one can immediately

conclude that $\mathbb{E}\left[e_i^r(t)\right] = \frac{\sum_{j\neq i} n_j(t)}{w} \leq \frac{t}{w}$. Applying (2.3), we obtain the following upper bound on the expected estimation error:

$$\mathbb{E}\left[e_i(t)\right] \leq \frac{t}{w} \quad \Leftrightarrow \quad \mathbb{E}\left[\frac{e_i(t)}{t}\right] \leq \frac{1}{w}\,. \tag{2.6}$$

Moreover, the random variables $\{e_i^r(t)\}_{r\in[d]}$ being i.i.d., an application of the Markov inequality leads to the following upper bound on the CCDF of $e_i(t)$:

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \left(\frac{1}{wx}\right)^d\,. \tag{2.7}$$

Cormode and Muthukrishnan proved this result in [CM05a, Theorem 1] for the specific value $x = \frac{\mathrm{e}}{w}$.

2.2.3 Count-Min Sketch with Conservative Updates (CMS-CU)

The conservative update [EV02] or minimal increment [CM03] is an optimization of CMS that consists in incrementing only the counters that attain the minimum value. The update procedure when item i is requested at time t becomes

$$c_i^r(t) = \max\left(c_i^r(t-1), \min_{f\in[d]} c_i^f(t-1) + 1\right),\ \forall r \in [d]\,. \tag{2.8}$$

The error $e_i^r(t)$ in each row r, the count estimate $\hat{n}_i(t)$, and the estimation error $e_i(t)$, all depend on $c_i^r(t)$ in the same way as in CMS. Equations (2.2) and (2.3) hold with CMS-CU. The quantities $\{\delta_{i,j}^r(s)\}_{s\in[t],j\neq i}$ are now defined as

$$\delta_{i,j}^r(s) \triangleq \mathbb{1}\big(Z(s) = j, h_r(i) = h_r(j), \hat{n}_j(s-1) = c_i^r(s-1)\big). \tag{2.9}$$

Equation (2.5) holds for CMS-CU. With respect to (2.4), (2.9) captures the additional condition that counter $h_r(i)$ is updated by a request for j at time s only if its current value $c_i^r(s-1)$ coincides with the current estimate $\hat{n}_j(s-1)$. Because of this additional condition, CMS-CU enjoys always a smaller error than CMS. Therefore, CMS upper bounds on the expectation (2.6) and on the CCDF (2.7) also hold for CMS-CU.

2.2.4 State of the art

When evaluating our analysis in Section 2.4 with synthetic and real traces, we will compare our results to the seminal paper [CM05a] and its follow-up by the same authors [CM05b] as well as to the more recent [Bia+12; EF15; Che+21].

Bianchi et al. consider in [Bia+12] a particular case where, at each step, all counters are equally likely to be updated. The corresponding error is called the error floor and denoted here as $\epsilon_f(t)$. Experimental observations suggest that the error floor bounds the expected estimation error for any stream process, i.e.

$$\mathbb{E}\left[e_i(t)\right] \lesssim \epsilon_f(t) \quad \Leftrightarrow \quad \mathbb{E}\left[\frac{e_i(t)}{t}\right] \lesssim \frac{\epsilon_f(t)}{t} \,. \tag{2.10}$$

The error floor (denoted here as $\epsilon_f(t)$) is approximated using the following formula

$$\epsilon_f(t) \approx \frac{\overline{g}}{w \cdot d} \cdot t \,, \tag{2.11}$$

$$\text{with} \quad \overline{g} = \lim_{t \to +\infty} \frac{1}{t} \sum_{s=1}^{t} g(s) \,, \tag{2.12}$$

$$g(s) = \mathbb{E}\left[\left|\{r : \; c^r_{Z(s)}(s-1) = \hat{n}_{Z(s)}(s-1)\}\right|\right] . \tag{2.13}$$

Equation (2.13) says that $g(s)$ is the expected number of increments made at time s. $\overline{g}$ can be approximated by $\frac{1}{t_0}\sum_{s=1}^{t_0} g(s)$, for t_0 large enough. Each value $g(s)$ can be estimated through Montecarlo simulations or solving numerically an opportune differential equation. The authors show that $\overline{g}$ depends on d but not on w.

Building on (2.10), we derive an approximate probabilistic bound on the error by a direct use of the Markov inequality:

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \lesssim \frac{\epsilon_f(t)}{xt} \,. \tag{2.14}$$

Einziger and Friedman modeled in [EF15] CMS-CU as a stack of Bloom filters. They proposed approximate bounds on the CCDF of the error $e_i(t)$ under the IRM model as follows:

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \lesssim \begin{cases} \mathrm{PFP}(A_{\lceil xt \rceil}), & \text{if } \mathbb{E}\left[n_i(t)\right] \geq 1, \\ \mathrm{FP}(A_{\lceil xt \rceil}), & \text{otherwise,} \end{cases} \tag{2.15}$$

where $A_k \triangleq \mathbb{E}\left[|\{j \in I : \; \hat{n}_j(t) \geq k\}|\right]$, $\mathrm{FP}(n)$ is the false positive probability of a regular Bloom filter after n insertions [BM03], and $\mathrm{PFP}(n)$ is its average over all the past insertions, i.e., $\mathrm{PFP}(n) = \frac{1}{n}\sum_{k=1}^{n} \mathrm{FP}(k)$. The false positive probability can be approximated as $\mathrm{FP}(n) \approx \left(1 - \mathrm{e}^{-n/w}\right)^d$, and A_k for $k \in \mathbb{N}$ can be recursively computed using the following formula,

$$A_k \approx D_k + \sum_{j=1}^{k-1} (D_j - D_{j+1}) \cdot \mathrm{PFP}(A_{k-j}), \tag{2.16}$$

where $D_k \triangleq |\{j \in I : \; \mathbb{E}\left[n_j(t)\right] \geq k\}|$. Note that the expected number of arrivals $\mathbb{E}\left[n_j(t)\right]$ is known under the IRM assumption.

Peiqing et al. [Che+21] proposed a method to estimate the error for CMS-CU by leveraging the knowledge of the sketch counters $(K^r_j(t))_{j\in[w]}$ associated to the hash function h_r. Under this assumption, they propose an estimator $(\hat{g}(\delta, t))$ for the $(1-\delta)$-quantile of the error $e_i(t)$, that is for $g_i(\delta, t) \triangleq \inf\{q \in [0, t] : \; \Pr\left(e_i(t) > q\right) = \delta\}$). Assuming the values $(K^r_j(t))_{j\in[w]}$ are sorted in decreasing order, the estimator $\hat{g}(\delta, t)$ has the following expression:

$$\hat{g}(\delta, t) = K^r_{\lceil \delta^{1/d} w \rceil}(t) \,. \tag{2.17}$$

We observe that in practice the knowledge of $(K^r_j(t))_{j\in[w]}$ requires to simulate the whole CMS-CU evolution with a computational cost proportional to the stream size t. On the contrary, the computational cost of our method does not depend on the stream size.

2.2.5 Our Assumptions

We will assume in our analysis that the request process follows the Independent Reference Model (IRM) [Fag77]; in other words, $\{Z(s)\}_{s\in[t]}$ are i.i.d. categorical random variables with $\Pr(Z(s)=i)=p_i$ for $i\in I$, and $\sum_{i\in I} p_i = 1$. We refer to p_i as the *popularity* of item i. Without loss of generality, we number items in I according to their popularity rank, hence $p_i \geq p_{i+1}$, for $i \in [N-1]$. Note that there are two sources of randomness in our setting: the hash functions' selection and the request process S_t. From now on, the expectation $\mathbb{E}[.]$ and the probability $\Pr(.)$ take both kinds of randomness into account.

2.3 Theoretical Analysis of CMS-CU

Under the IRM model, we first prove a tighter upper bound on the CCDF of $e_i(t)$ for CMS, then we upper bound both the CCDF and the expectation of $e_i(t)$ for CMS-CU.

2.3.1 CMS: CCDF of the Estimation Error

In this section we derive a bound for CMS error under the IRM assumption that is tighter than (2.7). As discussed in Section 2.2.3, our new bound applies as well to the CMS-CU error.

Proposition 2.1 (Upper bound on the CCDF of $e_i(t)/t$)**.** *The CCDF of the estimation error $e_i(t)$, when using CMS, verifies*

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \mathcal{A}(x)^d\,, \tag{2.18}$$

where

$$\mathcal{A}(x) \triangleq \min_{k\in[\![0,w-1]\!]} \mathcal{A}_k(x) \tag{2.19}$$

and

$$\mathcal{A}_k(x) \triangleq \min\left(\frac{1}{x(w-k)}\sum_{\substack{j>k\\ j\neq i}} p_j + \frac{k}{w}, 1\right). \tag{2.20}$$

Proof. See Appendix A.1 page 75. □

Proposition 2.1 extends known results in the literature. In particular, upper bounding the right-hand side of (2.18) by $(\mathcal{A}_0(x))^d$ yields (2.7), and then replacing $x = \mathrm{e}/w$, we obtain [CM05a, Theorem 1]. Proposition 2.1 also recovers [CM05b, Theorem 5.1], by considering the particular case where items are requested according to a Zipf distribution with parameter $\alpha > 1$ and upper bounding the right-hand side of (2.18) by $\mathcal{A}_{w/3}(x)^d$ and the tail $\sum_{j>w/3} p_j$ by $(w/3)^{1-\alpha}$. In our experimental evaluation in Section 2.4, we will use for comparison purposes a combination of [CM05a, Theorem 1] and [CM05b, Theorem 5.1], namely,

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \min\left(\mathcal{A}_0(x)^d, \mathcal{A}_{w/3}(x)^d\right). \tag{2.21}$$

To highlight the relevance of Proposition 2.1, we present an example where the improvement of (2.18) over (2.7) is evident.

Example 2.1. *We consider a setting with a small set of k popular items collecting a fraction α of the requests and $N-k$ unpopular ones. More specifically,*

$$p_i = \begin{cases} \frac{\alpha}{k}, & \text{if } i \leq k, \\ \frac{1-\alpha}{N-k}, & \text{otherwise}. \end{cases} \tag{2.22}$$

For $x = \frac{\beta}{w}$ for some $\beta \in (1-\alpha, 1]$, bounding the right-hand side of (2.18) *by $(\mathcal{A}_k(x))^d$, yields a bound asymptotically equivalent* to $\left(\frac{1-\alpha}{\beta}\right)^d$ as $w \to +\infty$, in sharp contrast with* (2.7) *which provides the trivial bound* 1.

2.3.2 CMS-CU: CCDF of the Estimation Error

We consider now CMS-CU and derive an upper bound on the CCDF of the estimation error $e_i(t)/t$. While the bound that we derived in Proposition 2.1 when CMS is used holds also with CMS-CU, our objective in this section is to derive a bound that makes use of the enhancement that CMS-CU brings over CMS. An important step to this end is to characterize the random variable $\delta_{i,j}^r(s)$ defined in (2.9). We establish first a preliminary result on the expectation of $\delta_{i,j}^r(s)$ that proved to be useful in the following.

Lemma 2.1 (Upper bound on $\mathbb{E}\left[\delta_{i,j}^r(s)\right]$)**.** *The expected contribution of item j to item i's count at row r at time s satisfies* (2.23)*.*

$$\exists \alpha_{i,j} > 0, \beta_{i,j} \geq 0: \quad \mathbb{E}\left[\delta_{i,j}^r(s)\right] \leq \frac{p_j}{w}\left(\gamma_{i,j} + \beta_{i,j} \mathrm{e}^{-\alpha_{i,j}(s-1)}\right), \tag{2.23}$$

with

$$\gamma_{i,j} \triangleq \begin{cases} 1, & \forall j \leq i, \\ \min\left(\mathcal{A}(p_i - p_j)^{d-1}, 1\right), & \forall j > i, \end{cases} \tag{2.24}$$

and $\mathcal{A}(x)$ given in (2.19)*.*

Proof. See Appendix A.2 page 75. □

It is interesting to observe the structure of the bound in (2.23). The term p_j/w is simply $\mathbb{E}\left[\delta_{i,j}^r(s)\right]$ with CMS as can readily be seen from (2.4). Therefore, the coefficient of p_j/w in (2.23) is an attenuation term that captures the effect of using the conservative update procedure. As $s \to \infty$, this attenuation term converges to $\gamma_{i,j}$. The larger the difference between p_i and p_j, the smaller the term $\gamma_{i,j}$. This is expected as, the larger the difference in popularity between any two items i and j, the likelier that $c_i^r(t) > \hat{n}_j(t)$, and then the lesser item j is able to interfere with item i's estimation.

*f(w) is asymptotically equivalent to g(w) when $w \to +\infty$ if $\lim_{w\to+\infty} \frac{f(w)}{g(w)} = 1$

Proposition 2.2 (Upper bound on the CCDF of $e_i(t)/t$)**.** *The CCDF of the estimation error $e_i(t)$, when using CMS-CU, is upper bounded as follows:*

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \mathcal{D}_i(x) + \mathcal{O}\left(\frac{1}{t}\right), \tag{2.25}$$

where

$$\mathcal{D}_i(x) \triangleq \min\left(\mathcal{A}(x)^d, \mathcal{B}_i(x), \mathcal{C}_i(x)\right), \tag{2.26}$$

$$\mathcal{B}_i(x) \triangleq \min_{k\in[\![0,w-1]\!]} \left(\frac{1}{x}\left(\frac{1}{w-k}\sum_{j>k} p_j\gamma_{i,j}\right) + \frac{k}{w}\right), \tag{2.27}$$

$$\mathcal{C}_i(x) \triangleq \min_{k\in[\![0,w-1]\!]} \left(\frac{1}{xw\left(1-\left(\frac{k}{w}\right)^d\right)}\left(\sum_{\substack{j>k\\ j\neq i}} p_j\gamma_{i,j}\right) + \left(\frac{k}{w}\right)^d\right), \tag{2.28}$$

and $\gamma_{i,j}$ is given in (2.24)*.*

Proof. See Appendix A.3 page 77. □

We use the popularity distribution (2.22) of Example 2.1 to illustrate the improvement brought by Proposition 2.2 with respect to the bounds (2.7), (2.14), and (2.15), that are respectively proposed/deduced from [CM05a], [Bia+12], and [EF15].

Example 2.2. *Consider the popularity distribution in* (2.22)*, $w, N \to +\infty$, and $w = o(N)$.*. In this setting, we show in A.5 that* (2.15) *results in the trivial bound* 1 *(this particular result holds for any popularity distribution).*

We define $\mathcal{C}_{i,k}(x)$ such that $\mathcal{C}_i(x) = \min_{k\in[\![0,w-1]\!]} \mathcal{C}_{i,k}(x)$, where $\mathcal{C}_i(x)$ is given in (2.28)*. From Proposition 2.2, if we upper bound $\mathcal{D}(x)$ by $\mathcal{C}_{i,k}(x)$ and $\gamma_{i,j}$ by $(\mathcal{A}_0(p_1 - p_{k+1}))^{d-1}$, we obtain, for any $i \leq k$,*

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \frac{1-\alpha}{x\left(1-\left(\frac{k}{w}\right)^d\right)w^d(p_1-p_{k+1})^{d-1}} + \left(\frac{k}{w}\right)^d + \mathcal{O}\left(\frac{1}{t}\right). \tag{2.29}$$

Choosing x such that $\frac{1}{x} = o\left(w^d\right)$ and $x \leq \frac{\overline{g}\cdot d}{w}$ ($\overline{g}$ is defined in (2.12)*), the bound in* (2.29) *is $\mathcal{O}(1/t)$, whereas* (2.7) *and* (2.14) *result in the trivial bound* 1*.*

In Section 2.4 we will provide evidence that the qualitative difference seen in Example 2.2 also exists for realistic values of the parameters and Zipf popularity distributions. For example, Figure 2.1 shows that our analysis correctly predicts that estimates for item with rank 100 are off by at most of few units, while state-of-the-art approaches estimate that the error should be at least 10 times larger.

*f(w)=o(g(w)) if $\lim_{w\to+\infty} \frac{f(w)}{g(w)} = 0$.

Remark 2.1. *Proposition 2.2 combines two results. The first, from our paper [BAN22a], is expressed as:*

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \min\left(\mathcal{A}(x)^d, \mathcal{B}_i(x)\right) + \mathcal{O}\left(\frac{1}{t}\right). \tag{2.30}$$

The second, from our paper [BAN22b], is given by:

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \min\left(\mathcal{A}(x)^d, \mathcal{C}_i(x)\right) + \mathcal{O}\left(\frac{1}{t}\right). \tag{2.31}$$

Remark 2.2. *The CCDF bound in Proposition 2.2 is equivalent to the CMS CCDF bound in Proposition 2.1 for the least popular items. Our analysis for these items does not capture the benefit of the conservative update procedure. Experimentally, we observed that the conservative update for the least popular items can be accommodated using* (2.10) *by modifying the functions* $\mathcal{C}_i(x)$ *in* (2.28) *as follows,*

$$\mathcal{C}_i(x) = \min_{k \in [\![0, w-1]\!]} \left(\frac{\epsilon_f(t)}{xt\left(1 - \left(\frac{k}{w}\right)^d\right)} \sum_{\substack{j>k \\ j\neq i}} p_j + \left(\frac{k}{w}\right)^d \right), \tag{2.32}$$

where $\epsilon_f(t)$ *is defined in* (2.11).

We highlight the usefulness of Proposition 2.2 in Section 2.3.4 where we estimate the precision achievable by CMS and CMS-CU methods applied to the heavy-hitters detection problem. We conclude our formal analysis of CMS-CU with the derivation of a bound on the expectation of the estimation error in the next section.

2.3.3 CMS-CU: Expected Estimation Error

The results obtained so far suggest two possible bounds on the expectation of $e_i(t)/t$. One bound can be derived using (2.3), (2.5) and Lemma 2.1. Another bound comes from writing the expectation as the sum of probability tails and using Proposition 2.2.

Proposition 2.3 (Upper bound on $\mathbb{E}\left[e_i(t)/t\right]$)**.** *The error experienced by item* i *is upper bounded as follows*

$$\mathbb{E}\left[\frac{e_i(t)}{t}\right] \leq \min\left(\frac{1}{w}\sum_{j\in\mathcal{I}\setminus\{i\}} p_j\gamma_{i,j}, \frac{1}{t}\sum_{n=0}^{t}\mathcal{D}_i\left(\frac{n}{t}\right)\right) + \mathcal{O}\left(\frac{1}{t}\right). \tag{2.33}$$

Proof. See Appendix A.4 page 78. □

While previous studies [Bia+12] and [EF15] bounded the error uniformly across items, Proposition 2.3 provides error bounds that depend on an item's popularity. In particular, our work is the first to support analytically the experimental evidence that the most popular items barely experience any error, as observed in [Bia+12] and also shown in Figure 2.3. In our earlier work [BAN22a], we use Lemma 2.1 to deduce a weaker bound than (2.33) given by,

$$\mathbb{E}\left[\frac{e_i(t)}{t}\right] \leq \frac{1}{w}\sum_{j\in\mathcal{I}\setminus\{i\}} p_j\gamma_{i,j} + \mathcal{O}\left(\frac{1}{t}\right). \tag{2.34}$$

In order to highlight the importance of Proposition 2.3, we compare (2.33), to (2.6) and (2.10) using the same popularity distribution as in Examples 2.1 and 2.2.

Example 2.3. *Recall the popularity distribution* (2.22)*. From Proposition 2.3, the expectation of the error for the k most popular items,i.e., any item i such that $i \leq k$, verifies,*

$$\mathbb{E}\left[\frac{e_i(t)}{t}\right] \leq \frac{1}{w}\sum_{j\in\mathcal{I}\setminus\{i\}} p_j\gamma_{i,j} + \mathcal{O}\left(\frac{1}{t}\right), \tag{2.35}$$

$$\leq \frac{1}{w}\sum_{\substack{j\leq k\\ j\neq i}} p_j + \frac{1}{w}\sum_{j>k} p_j\left(\mathcal{A}_0(p_1-p_{k+1})^{d-1} + \mathcal{O}\left(\frac{1}{t}\right)\right., \tag{2.36}$$

$$\leq \frac{\alpha}{w}\left(1-\frac{1}{k}\right) + \Theta\left(\frac{1}{w^d}\right) + \mathcal{O}\left(\frac{1}{t}\right), \tag{2.37}$$

Equation (2.36) *upper bounds $\gamma_{i,j}$ by $(\mathcal{A}_0(p_1-p_{k+1}))^{d-1}$ for $j > k$ and 1 otherwise. Equation* (2.37) *utilizes $\mathcal{A}_0(p_1-p_{k+1}) = \Theta\left(\frac{1}{w}\right)$.*

The ratio of (2.37) *and* (2.6) *is $\alpha(1-1/k)+o(1)$, which is smaller than 1 for large w and t. Comparing* (2.37) *and* (2.10)*, it is not possible in general to conclude which bound is tighter. In practice, we will consider the minimum of the two. We stress though that our analysis is formal whereas that in [Bia+12], leading to* (2.10)*, is based on experimental observations.*

Remark 2.3. *In practice, metrics like Average Absolute Error: $AAE = \frac{\sum_{i\in\mathcal{I}}|e_i(t)|}{|\mathcal{I}|}$, Average Relative Error: $ARE = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\frac{|e_i(t)|}{n_i(t)}$ and Weighted Average Absolute Error: $WAAE = \sum_{i\in\mathcal{I}}\frac{n_i(t)}{t}\cdot|e_i(t)|$ are used to evaluate the performance of a sketch, e.g., [Zho+18; Zha+21a; Hsu+19]. Approximating $n_i(t)$ by $p_i\cdot t$ allows one to evaluate the expected value of these metrics as follows:*

$$\mathbb{E}\left[AAE\right] = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\mathbb{E}\left[|e_i(t)|\right], \tag{2.38}$$

$$\mathbb{E}\left[ARE\right] \approx \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\frac{\mathbb{E}\left[|e_i(t)|\right]}{p_i t}, \tag{2.39}$$

$$\mathbb{E}\left[WAAE\right] \approx \sum_{i\in\mathcal{I}} p_i\cdot\mathbb{E}\left[|e_i(t)|\right], \tag{2.40}$$

then an upper bound can be derived using our result in Proposition 2.3.

2.3.4 Heavy-Hitters Application: Lower Bound on the Precision

Heavy-hitters are items whose request rate exceeds a given threshold ϕ. Detecting heavy-hitters in a stream can be done using a sketch (for instance CMS or CMS-CU), but sketches overestimate the number of requests and can then lead to "false positives," i.e., items with a rate smaller than ϕ can erroneously be classified as heavy-hitters. Let H be the set of heavy-hitters, $H = \{i:\ n_i(t) \geq \phi\cdot t\}$, and $\hat{H}$ be the set of items classified as heavy-hitters by the sketch, $\hat{H} = \{i:\ \hat{n}_i(t) \geq \phi\cdot t\}$. The

"precision" is one metric used for assessing the performance of the sketch [CH10], and is defined as follows: $P \triangleq |H|/|\hat{H}|$. For the sake of simplicity, we assume that $n_i(t) \approx p_i t$, for large enough t; this is reasonable because of the law of large numbers. Under this approximation, $|H|$ is constant and we can write the expected value of the precision as:

$$\mathbb{E}[P] \approx \frac{|H|}{|H| + \sum_{i>|H|} \Pr\left(\frac{e_i(t)}{t} \geq \phi - p_i\right)}. \tag{2.41}$$

Combining (2.41) with Proposition 2.2 we obtain a lower bound on the expected precision when CMS-CU is used. This lower bound will be illustrated in Section 2.4.5 and compared to experimental values.

Again we highlight qualitatively the advantage of our bound (2.25) (Proposition 2.2) with respect to (2.7), (2.14), and (2.15) through an example.

Example 2.4. *For the heavy-hitter problem we consider a popularity distribution, slightly more complex than* (2.22)*, with an additional group of medium popular items. More specifically,*

$$p_i = \begin{cases} \alpha_1/k_1, & \text{if } i \leq k_1, \\ \alpha_2/k_2, & \text{if } k_1 < i \leq k, \\ (1-\alpha)/(N-k), & \text{otherwise}, \end{cases} \tag{2.42}$$

where $\frac{\alpha_1}{k_1} > \frac{\alpha_2}{k_2}$, $\alpha = \alpha_1 + \alpha_2$, *and* $k = k_1 + k_2$.

We consider $\frac{\alpha_2}{k_2} < \phi < \frac{\alpha_1}{k_1}$, *that is, the heavy-hitters coincide with the* k_1 *most popular items. The precision is then* $\mathbb{E}[P] = k_1/(k_1 + S_1 + S_2)$, *where* S_1 *and* S_2 *are defined as follows:*

$$\begin{aligned} S_1 &\triangleq (N-k) \cdot \Pr\left(\frac{e_N(t)}{t} \geq \phi - p_N\right) \\ S_2 &\triangleq k_2 \cdot \Pr\left(\frac{e_k(t)}{t} \geq \phi - p_k\right). \end{aligned} \tag{2.43}$$

The different approaches estimate in different way the probabilities appearing in (2.43)*. We study the regime where both* N *and* w *diverge with* $w = o(N)$ *and* $N = o(w^d)$.

If we use the bound (2.15) *in* (2.43)*, the arguments in A.5 lead to conclude that* S_1 *is potentially unbounded (*N *diverges and* $\Pr(e_N(t)/t \geq \phi - p_N)$ *is upper-bounded by a positive constant). The conclusion is then trivial: the precision is lower bounded by* 0*. The bound* (2.14) *does not provide meaningful bounds for* S_1 *and the precision either.*

On the contrary, both (2.25) *and* (2.7) *guarantee that* S_1 *is arbitrarily small asymptotically. However, the two bounds may draw different conclusions for* S_2*. In fact,* (2.7) *concludes that* S_2 *can be made smaller than* ϵ *for* $\phi - p_k > \frac{1}{w}\sqrt[d]{\frac{k_2}{\epsilon}}$*. The bound* (2.25) *can be relaxed to the simpler form*

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \frac{k_2^{d-1}(1-\alpha)}{w^d x\left(1 - \left(\frac{k}{w}\right)^d\right)\alpha_2^{d-1}} + \left(\frac{k}{w}\right)^d + \mathcal{O}\left(\frac{1}{t}\right), \tag{2.44}$$

which is obtained similarly to (2.29)*, by upper bounding* $\mathcal{D}_i(x)$ *by* $\mathcal{C}_{i,k}(x)$ *in the right hand side of* (2.25) *and upper bounding* $\gamma_{i,j}$ *by* $(\mathcal{A}_0(p_i - p_j))^{d-1}$*. From* (2.44)*, we conclude that there exists*

Table 2.1: Labels and equations used in the comparison.

Metric	[CM05b] CM05	[Bia+12] BDLS12	[EF15] EF15	[Che+21] CWYJL21	[BAN22a] BAN22	This thesis Ours
CCDF popular i	(2.21)	(2.14)	(2.15)	(2.17)	(2.30)	(2.25)
non-popular i	(2.21)	(2.14)	(2.15)	(2.17)	(2.30)	(2.25), (2.32)
Expected error	$\frac{1-p_i}{w}$	(2.10)	–	–	(2.34)	(2.33)
Precision ((2.41))	(2.21)	(2.14)	(2.15)	–	(2.30)	(2.25)

a constant a' such that S_2 is guaranteed to be smaller than ϵ when $\phi - p_k > \frac{a' \cdot k_2}{w^d \epsilon}$. In conclusion, if $\phi - p_k$ belongs to the interval $\left(\frac{a' \cdot k_2}{w^d \epsilon}, \frac{1}{w} \sqrt[d]{\frac{k_2}{\epsilon}} \right]$, our bound predicts that the precision is at least $1 - \epsilon$, while the bound (2.7) *simply guarantees a precision inferior to $1 - \epsilon$.*

2.4 Experimental Evaluation and Numerical Analysis

2.4.1 Experimental Setting

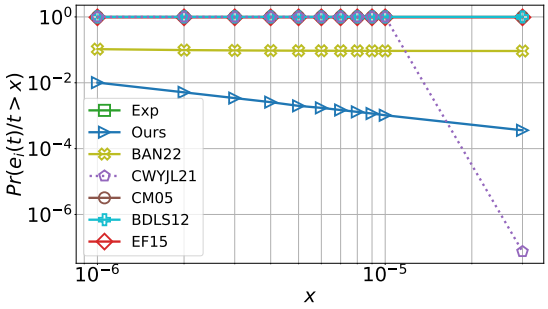
To support our analysis, we have undertaken three series of experiments in which we simulated requests for items over time and used CMS-CU to count the requests for each item.

In the first series of experiments, we generated 100 synthetic streams from a Zipf distribution with shape parameter $\alpha = 0.8$. Each stream contains 5 million requests for items in the set $I = [N]$ with $N = 10^6$. For each stream, we employed a sketch with a default configuration of width $w = 10^4$ and depth $d = 8$ and we selected different 'MurmurHash' hash functions [YN13] for each stream by choosing uniformly at random d different seeds. The experimental values reported for this setting are averaged over the 100 streams (or 100 independent trials). We also computed the 95% confidence intervals but did not report them as they are very narrow and would not be visible in the figures. The second series of experiments is identical to the first except for the shape parameter of the Zipf distribution which is $\alpha = 1.1$.

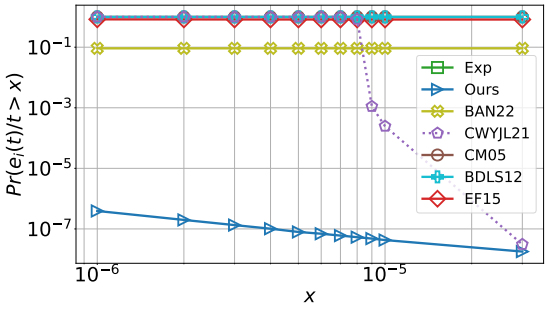
In the third series of experiments, we used a trace log of accesses to Wikipedia pages in all languages during September 2007 [UPS09]. The trace contains 10,628,125 requests. The number of distinct Wikipedia pages requested in this trace is 1,712,459. We extracted 10 non-overlapping chunks from this trace, each containing $N' = 10^6$ requests, and discarded the rest. The number of distinct items in each chunk is around $3 \cdot 10^5$. All theoretical values are computed assuming IRM and that item popularities coincide with empirical frequencies over the first chunk, i.e., $p_i \approx n_i(N')/N'$. The reported experimental values are instead computed on the remaining 9 chunks of the trace. The CMS-CU in this setting is configured by default with a width $w = 5000$ and a depth $d = 5$.

2.4.2 Numerical Evaluation

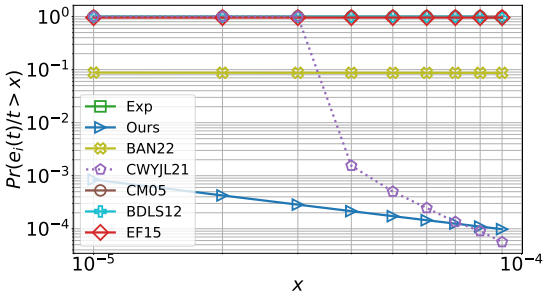
For each series of experiments, we report the results obtained for the following metrics: (i) the CCDF of the sketch estimation error for both a popular item and a non-popular one, (ii) the expected sketch estimation error for each item along with the Average Absolute Error (AAE), the Average Relative Error (ARE), the Weighted Average Absolute Error (WAAE) , and (iii) the precision



(a) synthetic, $\alpha = 0.8$



(b) synthetic, $\alpha = 1.1$



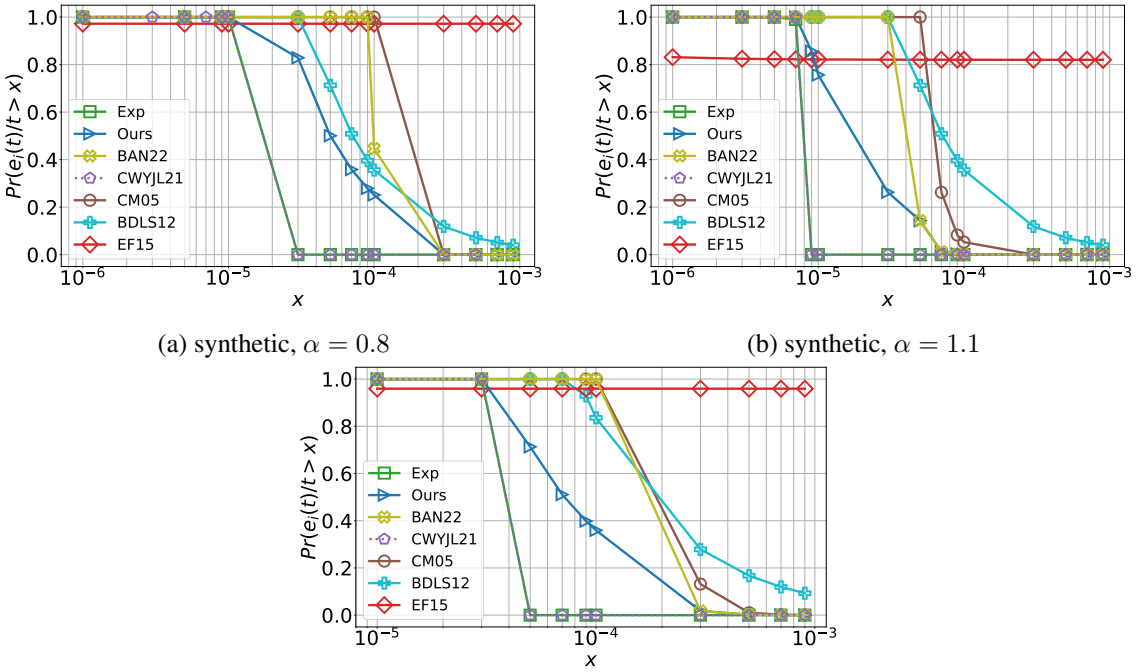
(c) Wikipedia

Figure 2.1: CCDF of error for a popular item.

in the heavy-hitters detection problem. We compare our results to those of [CM05a; CM05b], [Bia+12], [EF15], and our earlier work [BAN22a] which we refer to as 'CM05', 'BDLS12', 'EF15', and 'BAN22' respectively. We also carry out a limited comparison with the method in [Che+21]—referred to as 'CWYJL21'—, which differs from our approach and all other methods listed above in that, in order to estimate the error at time t, it requires access to a sample of the counters at time t for a representative data stream (see Section 4.2). For completeness, we list the relevant formulas used in Table 2.1. In the following sections, the estimation of the ground truth using simulations is referred to as 'Exp.'

2.4.3 The CCDF of the Sketch Estimation Error

The results are shown in Figures 2.1 and 2.2 for popular and non-popular items, respectively. The ground truth CCDF (Exp) is not visible in Figure 2.1 as it is 0. Note that all existing approaches, apart from our earlier work BAN22, provide a single bound for the CCDF that is valid for all items and is essentially tailored to the unpopular items for which errors are larger. In all three series of experiments, our bounds are better at capturing the CCDF of the error for popular items (ranks 100 and 50), for which the other state-of-the-art approaches provide only rough estimations; see Figure 2.1. The interval identified analytically in Example 2.2 (over which our bounds decrease whereas most of the state-of-the-art bounds are trivially 1) are visible in Figure 2.1. In particular, in Figure 2.1b our upper bound in the interval $[10^{-6}, 10^{-5}]$ decreases and is below 10^{-6} while CM05, BDLS12 and EF15 are equal to 1. The results for EF15 confirm our discussion in A.5.



(a) synthetic, $\alpha = 0.8$

(b) synthetic, $\alpha = 1.1$

(c) Wikipedia

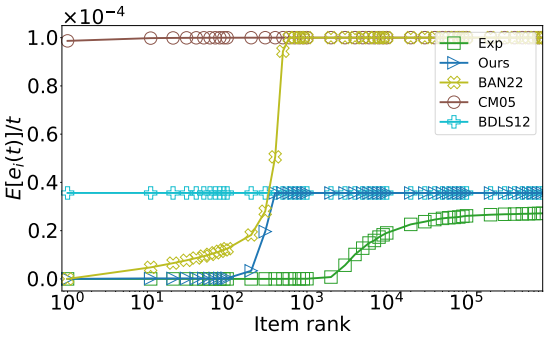
Figure 2.2: CCDF of error for a non-popular item.

The only exception is CWYJL21, which improves on our CCDF bound for large values of the error. CWYJL21 is better at capturing the CCDF of the estimation error for non-popular items (ranks 10^5 and 10^4), with our method being the second best one; see Figure 2.2. Note that CWYJL21 requires to simulate CMS-CU over a trace drawn from the same distribution, and then its computational cost grows at least linearly with the length of the stream t.

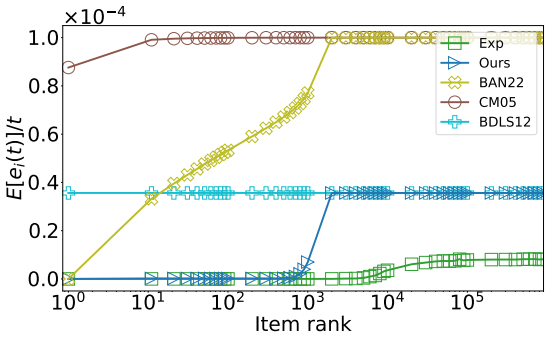
2.4.4 The Expected Sketch Estimation Error

The results for the synthetic and Wikipedia traces are depicted in Figure 2.3. We observe that our analysis correctly predicts that different items experience different errors. The bounds BAN22 and Ours improve over CM05. Our bound improves over BAN22 and BDLS12 in all three series of experiments, namely for the 300 most popular items when $\alpha = 0.8$, for the 1000 most popular items when $\alpha = 1.1$ and for the 400 most popular items for the Wikipedia trace. We notice that our bounds on the expectations are not always tight leaving room for improvement.

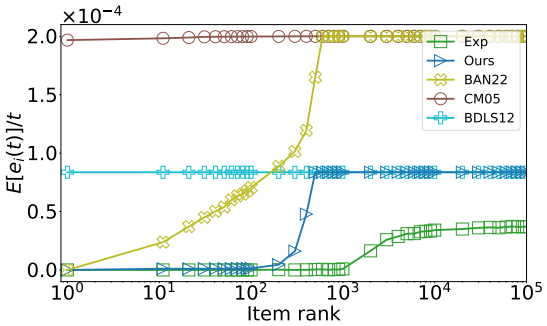
We also show in Table 2.2 other metrics—AAE, ARE and WAAE—commonly used to evaluate the performance of a sketch (see Remark 2.3). Our formulas are better than CM05 at predicting all three metrics but the improvement with respect to BDLS12 is only evident for WAAE. The reason is that our bounds improve BDLS12's ones only for the most popular items which constitute a small fraction of the whole catalogue. The difference is then marginal in terms of AAE and ARE which quantify average errors over the whole catalogue. On the contrary, WAAE is an average error per stream element and then errors on the most popular items are given a larger weight (i.e.,



(a) synthetic, $\alpha = 0.8$

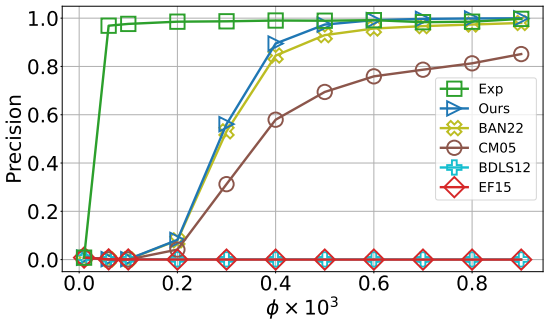


(b) synthetic, $\alpha = 1.1$

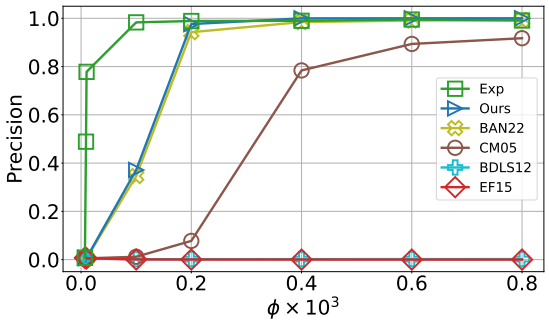


(c) Wikipedia

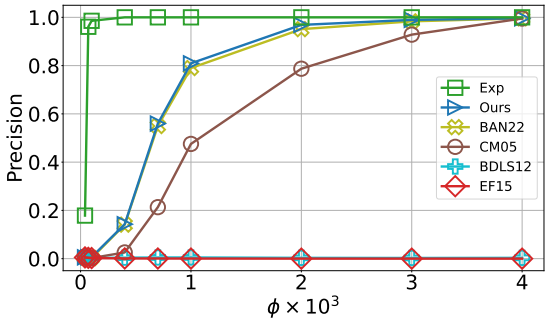
Figure 2.3: Estimation error for each item.



(a) synthetic, $\alpha = 0.8$



(b) synthetic, $\alpha = 1.1$



(c) Wikipedia

Figure 2.4: Precision as a function of the threshold ϕ.

Table 2.2: Average Absolute Error, Average Relative Error, Weighted Average Absolute Error.

	Synthetic: $\alpha = 0.8$			**Synthetic**: $\alpha = 1.1$			**Wikipedia**		
	AAE	**ARE**	**WAAE**	**AAE**	**ARE**	**WAAE**	**AAE**	**ARE**	**WAAE**
CM05	500	260	500	500	1531	500	200	176	200
BDLS12	178.1	93.4	178.1	178.1	545.1	178.1	83.6	73.5	83.6
Ours	178	93.4	152.2	177.9	545.1	54.3	83.5	73.5	44.9
Exp	132.7	70.7	26.5	39.7	124.3	6.8	36.2	32.3	14.8

proportional to their popularity). This metric shows then a clear difference between our approach and BDLS12.

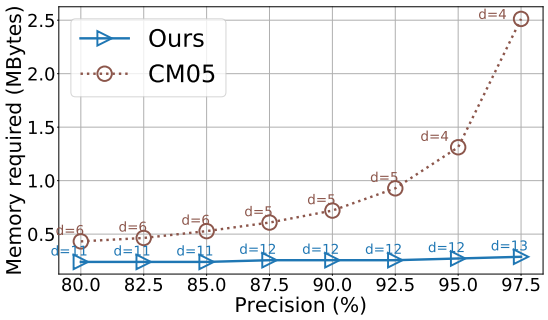
2.4.5 Precision in Detecting $\phi-$Heavy-Hitters

The results are presented in Figure 2.4. Our formulas outperform state-of-the-art methods in bounding the precision for both synthetic and real-world traces. The improvement is mainly due to the tightness of our CCDF bound for the most popular items. CM05 achieves high precision values only for large values of ϕ, as already qualitatively highlighted in Example 2.4. The poor lower bound on the precision achieved by EF15 was expected seeing its poor CCDF bound in Figures 2.1 and 2.2. It is interesting to observe that while the BDLS12 CCDF bound may be tighter than the CM05 CCDF bound for medium error values (like in Figures 2.1a and 2.2a for $x \in [3 \cdot 10^{-5}, 10^{-4}]$), its precision bound is much poorer. The reason is due to the sum of probability tails in the denominator of (2.41) being highly affected by the CCDF tail of non-popular items, and while the CM05 CCDF bound decreases exponentially fast, the BDLS12 CCDF bound decreases only inversely linearly (compare (2.14) with (2.7)).

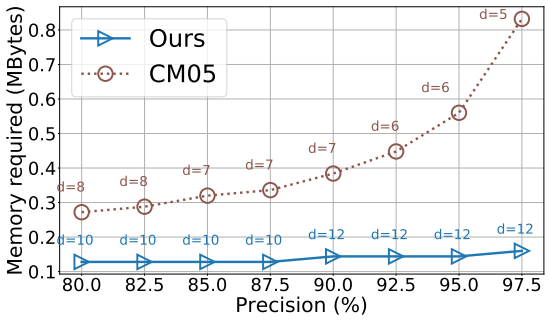
2.4.6 Configuring CMS-CU with QoS Guarantees

The bounds we derived can also be used to configure the width w and the depth d of CMS-CU in order to achieve the desired precision with the minimum amount of memory. If each counter uses 4 bytes, the memory cost of a CMS-CU is $M = 4wd$ bytes. We compared numerically the memory requirements determined by our approach and by CM05. We do not consider BDLS12 and EF15 in this section as their lower bound on the precision is poor, a configuration method relying on such bounds will thereby return prohibitively large memory requirements.

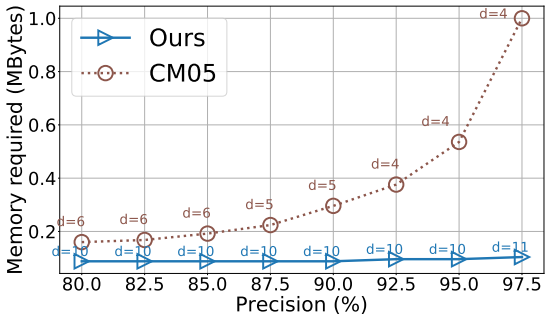
For target precision values in the range 0.8–0.975, we performed a search for the total number of counters (equal to $w \times d$) in the range $\left[\lceil \frac{2}{\phi} \rceil, N\right]$ (with a step of $\lceil 2/\phi \rceil$) and depth values between 2 and 15 to find the smallest memory which guarantees the target precision. Figure 2.5 shows the corresponding curves obtained using our approach and CM05. Our approach leads to configuring CMS-CU using a reduced amount of memory. For instance, we observe in Figure 2.5a a reduction factor of 4.82 for 95% precision target (1.312 MB with CM05 vs. 0.272 MB with Ours) and up to 8.72 for a precision target of 97.5% (2.512 MB with CM05 vs. 0.288 MB with Ours). The gain is of the same order for the other traces. In particular, when the shape parameter of the Zipf distribution is 1.1, the reduction factor is 3.89 and 5.2 for precision targets of 95% and 97.5%, respectively (see Figure 2.5b). In the case of the Wikipedia trace, the required memory for a precision target



(a) synthetic, $\alpha = 0.8$, $\phi = 5 \cdot 10^{-4}$



(b) synthetic, $\alpha = 1.1$, $\phi = 5 \cdot 10^{-4}$



(c) Wikipedia, $\phi = 10^{-3}$

Figure 2.5: Estimated memory requirement for a given precision.

of 97.5% is 0.104 MB with our approach as seen in Figure 2.5c, a reduction factor of 9.61 with respect to the 1 MB advocated by CM05 (reduction factor of 5.58 for a precision target of 95%).

Figure 2.5 suggests that large values of d are required to minimize the memory while achieving a desired precision target. At the same time, it may be undesirable to select a large number of hash functions due to computational constraints.

2.5 Conclusion

While it is a common belief that CMS-CU leads to smaller estimation errors for the most popular items [Bia+12], we are the first to provide quantitative support for such property, thanks to a per-item study of the estimation error. We showed that our analysis significantly improves existing bounds for the most popular items and leads, in comparison to the state of the art, to more accurate estimations for the precision in heavy-hitter detection problems as well as to improved configuration rules, which avoid to oversize the counting data structure. For less popular items, our bounds are not tighter than existing ones. In the future, we want then to focus on improving the bounds for the tail of the popularity distribution. Moreover, we plan on extending our analysis to other popular sketches based on CMS-CU like the one proposed in [Yan+18b].

CHAPTER 3

Online Learning for Caching

LFU-style caching policies encompass algorithms derived from the online learning literature. Notably, the classic Follow-the-Perturbed-Leader (FPL) algorithm, exhibits sublinear regret against an adversarial request process. Consequently, over time, FPL caching policies asymptotically perform as well as the optimal static caching policy with knowledge of future requests. However, a significant drawback lies in the assumption that the cache possesses knowledge of the exact requests' count, a presumption often unattainable in high-load or memory-constrained scenarios. In response to this limitation, we introduce in this chapter the Noisy-Follow-the-Perturbed-Leader (NFPL) algorithm. NFPL is a variant of the classic FPL, designed to accommodate scenarios where request estimates are noisy. We prove that NFPL exhibits sublinear regret under specific assumptions on the requests' estimator. We run simulations to evaluate the loss in performance due to noisy requests' estimates under synthetic and real-world request traces.

3.1 Introduction

Caching techniques are extensively employed in computer systems, serving various purposes such as accelerating CPU performance [TS98] and enhancing user experiences in content delivery networks (CDNs) [BOO07]. The primary objective of a caching system is to carefully choose files for storage in the cache to maximize the proportion of file requests that can be fulfilled locally. This approach effectively minimizes the dependence on remote server retrievals, which can be costly in terms of delay and network traffic. The presence of caching systems facilitates more efficient data delivery in network traffic and leads to enhanced overall system performance.

Caching policies have been thoroughly investigated under numerous assumptions concerning the statistical regularity of file request processes [Fag77; Tra+13]. However, real-world request sequences tend to deviate from these theoretical models, especially when aggregated over small geographic areas [Lec+16]. This deviation has inspired the exploration of online learning algorithms, beginning with the work of Paschos et al. [Pas+19a], which applied the Online Convex Optimization (OCO) framework [Zin03] to caching. These algorithms exhibit robustness to varying request process patterns, as they operate under the assumption that requests may be generated by an adversary.

In this context, the main metric of interest is the *regret*, which is the difference between the cost—e.g., the number of cache misses—incurred by a given online caching algorithm and the cost

of the optimal static cache allocation with hindsight, i.e., with knowledge of the future requests over a fixed time horizon. In this framework, the primary objective is to design *no-regret* algorithms, i.e., online policies whose regret grows sublinearly with the length of the time horizon [Pas+19a].

Several online caching policies have been proposed in the literature, drawing on well-known online algorithms such as Online Gradient Descent (OGD) [Pas+19a], Follow-the-Regularized-Leader (FRL) [MIL22] or Follow-the-Perturbed-Leader (FPL) [Mha+22b]. The latter is especially promising, as cache updates can be performed without the need for computationally intensive projection operations over the set of feasible cache states.

Caching policies, including no-regret ones, make admission and eviction decisions based on information from the request sequence. This can include factors such as the number of past requests for each file or a list of the most recently requested files. However, when dealing with a vast file catalog and/or a high request rate, a cache might have to depend on noisy information. For instance, limited availability of high-speed memory can necessitate the use of approximate counters based on hash functions [EFM17; BAN22b]. Alternatively, request sampling might be employed to decrease the frequency of counter updates [Li+16].

Surprisingly, much of the existing literature on no-regret caching policies overlooks these practical constraints. Typically, these studies operate under the assumption that caches have exact knowledge of the request sequences. A notable exception is the work presented in [LZ22]. However, it exclusively examines the scenario in which the cache is only aware of requests for files it already contains.

In this chapter, we bridge this gap by adapting the FPL algorithm—renowned for its computational efficiency and no-regret properties—to manage noisy request estimates. Our main contributions are the following:

1. We modify the FPL algorithm to handle noisy request estimates and prove that, under specific conditions on the estimator, the algorithm maintains sublinear regret. We refer to this extended version as Noisy-Follow-the-Perturbed-Leader (NFPL).

2. We propose two variants of the NFPL algorithm for the caching problem, namely, NFPL-Fix and NFPL-Var, where the requests estimator uses sampling. We prove that NFPL-Fix and NFPL-Var have sublinear regret.

3. We prove a new regret bound for the classic FPL caching policy that is independent of the catalog size.

4. We show through experimental analysis the advantage of the NFPL algorithm over classical caching policies. We also evaluate the impact of the sampling rate on the performance of NFPL-Fix and NFPL-Var.

This chapter is organized as follows. We describe the system assumptions and give background details in Section 3.2. The extension of FPL to deal with noisy requests and its analysis are described in Section 3.3. Experimental results are presented in Section 3.4. Finally, Section 3.5 concludes the chapter.

3.2 System Description and Background

3.2.1 Caching Problem: Model and Notation

We consider a single-cache system in which file requests for a catalog $\mathcal{I}$ with N files can either be served locally by a cache with finite capacity C or, in the case of a file miss, by a remote server.

Cache state. The local cache has a capacity $C \in \{1, \dots, N\}$ and stores files in their entirety. The cache state at time t is represented by the vector $\mathbf{x}_t = [x_{t,i}]_{i \in \mathcal{N}}$, which indicates the files missing in the cache; that is, $x_{t,i} = 1$ if and only if file i is not stored in the cache at time t. A feasible cache allocation is then represented by a vector in the set:

$$\mathcal{X} = \left\{ \mathbf{x} \in \{0,1\}^N \middle| \sum_{i=1}^{N} x_i = N - C \right\}. \tag{3.1}$$

Cache updates. Although caching policies are often assumed to update their state after each request, in high request rate regimes or when cache updates are computationally or communicationally expensive, the cache may update its state after receiving a batch of B requests [SNI23]. We study caching policies in this more general setting and consider a time-slotted operation. At each time slot $t = 1, \dots, T$, B requests are collected from the users and the cache state is updated. The request process is represented as a sequence of vectors $\mathbf{r}_t = (r_{t,i} \in \mathbb{N} : i \in \mathcal{I})\ \forall t$, where $r_{t,i}$ is the number of requests received for file i in the t-th batch. Then, each vector belongs to the set:

$$\mathcal{B} = \left\{ \mathbf{r} \in \mathbb{N}^N \middle| \sum_{i=1}^{N} r_i = B \right\}. \tag{3.2}$$

Cost. At each time slot t, the cache pays a cost equal to the number of misses, i.e., to the number of requests for files not in the cache. The cost can be computed as follows:

$$\langle \mathbf{r}_t, \mathbf{x}_t \rangle = \sum_{i=1}^{N} r_{t,i} x_{t,i}\ , \tag{3.3}$$

where $\langle \mathbf{r}, \mathbf{x} \rangle \triangleq \sum_{i=1}^{N} r_i x_i$ denotes the scalar product of the two vectors $\mathbf{r}$ and $\mathbf{x}$.

For the sake of conciseness, we introduce the following notation. For any vector $\mathbf{r}$, we denote by $M(\mathbf{r})$ an arbitrary element of $\arg\min_{\mathbf{x} \in \mathcal{X}} \langle \mathbf{r}, \mathbf{x} \rangle$. Furthermore, given a sequence of vectors $(\mathbf{r}_1, \dots, \mathbf{r}_t)$, we represent their aggregate sum as $\mathbf{r}_{1:t} \triangleq \sum_{s=1}^{t} \mathbf{r}_s$.

3.2.2 Caching and Online Learning

Caching can be framed as an online learning problem [Haz16], where an agent (the caching system) chooses an action $\mathbf{x}_t$ from the set $\mathcal{X}$ at each time slot t before an adversary reveals a request vector $\mathbf{r}_t$ from the set $\mathcal{B}$.

The cache state is determined by an online algorithm $\mathcal{A}$ that, at each time slot t, computes the cache state $\mathbf{x}_{t+1}$ for the next time slot given the current state $\mathbf{x}_t$ and the sequence of requests up to time t, that is $\{\mathbf{r_s}\}_{s=1}^{t}$.

N	catalog size
C	cache capacity
B	number of requests in each batch
$\mathcal{X}$	decision set
$\mathcal{B}$	set of request vectors
T	time horizon
$\mathbf{r}_t$	request vector at time step t
$\mathbf{x}_t$	decision vector at time step t
$\langle \mathbf{r}_t, \mathbf{x}_t \rangle$	cost at time step t
$\mathbf{r}_{1:t}$	sum of $\mathbf{r}_s$ for all values of s from 1 to t
$M(\mathbf{r})$	value of $\mathbf{x}$ in $\mathcal{X}$ that minimizes $\langle \mathbf{r}, \mathbf{x} \rangle$
$\mathcal{R}_T(\mathcal{A})$	regret algorithm $\mathcal{A}$
$\boldsymbol{\gamma}_t$	noise vector
$\hat{\mathbf{r}}_t$	noisy request estimates
$\hat{\mathcal{B}}$	$\hat{\mathbf{r}}_t$ state space

Table 3.1: Table of notation

The main performance metric used to evaluate an online deterministic algorithm $\mathcal{A}$ choosing action $\mathbf{x}_t$ at each time step t is the regret defined as:

$$\mathcal{R}_T(\mathcal{A}) = \sup_{\{\mathbf{r}_1,\ldots,\mathbf{r}_T\}} \left\{ \sum_{t=1}^{T} \langle \mathbf{r}_t, \mathbf{x}_t \rangle - \text{OPT}_T \right\}, \tag{3.4}$$

where $\text{OPT}_T = \langle \mathbf{r}_{1:T}, M(\mathbf{r}_{1:T}) \rangle$ is the cost incurred under the request sequence $\{\mathbf{r}_1, \ldots, \mathbf{r}_T\}$ by the optimal static allocation $\mathbf{x}^* = M(\mathbf{r}_{1:T})$. When the algorithm $\mathcal{A}$ is randomized, one can define the expected regret:

$$\mathcal{R}_T(\mathcal{A}) = \sup_{\{\mathbf{r}_1,\ldots,\mathbf{r}_t\}} \left\{ \mathbb{E}\left[\sum_{t=1}^{T} \langle \mathbf{r}_t, \mathbf{x}_t \rangle \right] - \text{OPT}_T \right\}, \tag{3.5}$$

where the expectation is taken over any random choice of the algorithm $\mathcal{A}$. The expected regret quantifies then the performance gap over a time horizon T between the algorithm $\mathcal{A}$ and the best static cache allocation with hindsight.

Given the supremum taken over all request sequences in both (3.4) and (3.5), it is evident that the regret metrics refrain from making any assumptions regarding the characteristics of the request sequence, such as any inherent statistical regularity. The request sequence may be thought to have been generated by an adversary seeking to degrade the performance of the caching system. In this setting, one aims for an algorithm with sublinear regret, $\mathcal{R}_T(\mathcal{A}) = o(T)$. These algorithms are commonly known as no-regret algorithms since their time-average cost approaches the optimal static policy's cost as T grows.

Various algorithms, such as Online Gradient Descent (OGD) and Follow-the-Regularized-Leader (FTRL), can attain $\mathcal{O}(\sqrt{T})$-regret for caching problems [Pas+19a; MIL22]. However, their cache update procedures require a computationally expensive projection of a tentative solution back onto the feasible set $\mathcal{X}$ (e.g., its cost is $O(N^2)$ for OGD [SNI23]).

Algorithm 1: Noisy-Follow-the-Perturbed-Leader with Uniform Noise (NFPL)

Input: Set of decisions $\mathcal{X}$; T; η
Output: Sequence of decisions: $\{\mathbf{x}_t\}_1^T$

$\hat{\textbf{costs}} \leftarrow 0$,
for *round* $t = 1, 2, ..., T$ **do**
  $\boldsymbol{\gamma}_t \sim \textbf{Unif}\left([0, \eta]^N, \mathbb{I}_{N\times N}\right)$
  $\mathbf{x}_t \leftarrow M(\hat{\textbf{costs}} + \boldsymbol{\gamma}_t)$
  Pay $\langle \mathbf{r}_t, \mathbf{x}_t \rangle$
  Observe $\hat{\mathbf{r}}_t$
  $\hat{\textbf{costs}} \leftarrow \hat{\textbf{costs}} + \hat{\mathbf{r}}_t$
end

In the next section, we present a lightweight caching algorithm with $\mathcal{O}(\sqrt{T})$-regret.

3.2.3 Follow-the-Perturbed-Leader (FPL)

Within the domain of online learning, the Follow-the-Perturbed-Leader (FPL) algorithm is a notable projection-free methodology known to achieve sublinear regret. This algorithm was initially introduced by Vempala et al. [KV05], and later studied within the caching framework by Bhattacharjee et al. [BBS20].

The FPL algorithm serves as a refined version of the traditional Follow-the-Leader (FTL) algorithm [LW94]. The latter greedily selects the state that would have minimized the past cumulative cost, i.e., $\mathbf{x}_{t+1}(\text{FTL}) = M(\mathbf{r}_{1:t})$.

While the FTL algorithm proves optimal when cost functions are sampled from a stationary distribution, it, unfortunately, yields linear regret in adversarial settings [De +14].

The FPL algorithm improves the performance of FTL by incorporating a noise vector $\boldsymbol{\gamma}_t$ at each time step t. This vector's components are independent and identically distributed (i.i.d.) random variables, pulled from a distinct distribution (such as the uniform and exponential distributions in [KV05], and the Gaussian distribution in [BBS20]). The update process unfolds similarly to FTL:

$$\mathbf{x}_t(\text{FPL}) = M(\mathbf{r}_{1:t-1} + \boldsymbol{\gamma}_t). \tag{3.6}$$

As shown in [BBS20], FPL provides optimal regret guarantees for the discrete caching problem. Moreover, the cache update, as specified in equation (3.6), involves storing the files that correspond to the largest elements of the vector $\mathbf{r}_{1:t-1} + \boldsymbol{\gamma}_t$. FPL cache update necessitates then a sorting operation. Notably, its computational complexity of $\mathcal{O}(N \log N)$ is less taxing than the projection step required by either the FRL or OGD algorithms, as highlighted in [BBS20].

3.3 Extending FPL

The traditional FPL algorithm needs to track the request count for each file in the catalog. As we discussed in the introduction, in scenarios with a large catalog and/or high request rate, the

cache may only have access to noisy estimates. For this reason, we introduce the Noisy-Follow-the-Perturbed-Leader (NFPL), a lightweight variant of FPL that employs noisy request estimates instead of exact request counts. In Section 3.3.1, we present the NFPL algorithm in detail along with its regret analysis. Subsequently, in Section 3.3.2, we study NFPL when noisy request estimates stem from sampling the request process as in [Li+16].

3.3.1 Noisy-Follow-the-Perturbed-Leader (NFPL)

The NFPL algorithm is described in Algorithm 1. NFPL follows in the footsteps of FPL with uniform noise but observes the estimated requests $\hat{\mathbf{r}}_t$ instead of the real requests $\mathbf{r}_t$. In particular, at each time slot t, the algorithm generates $\boldsymbol{\gamma}_t$ from a multivariate uniform distribution with uncorrelated components, constrained within the range $[0, \eta]^N$, and it updates the decision vector $\mathbf{x}_t$ with the minimizer of $\langle \mathbf{x}, \hat{\mathbf{r}}_{1:t-1} + \boldsymbol{\gamma}_t \rangle$ over $\mathbf{x} \in \mathcal{X}$. The cost paid at time slot t is equal to $\langle \mathbf{r}_t, x_t \rangle$. The total cost of the NFPL algorithm is

$$\text{NFPL}_T = \sum_{t=1}^{T} \langle \mathbf{r}_t, M(\hat{\mathbf{r}}_{1:t-1} + \boldsymbol{\gamma}_t) \rangle. \tag{3.7}$$

We remark that Algorithm 1 is not just confined to the caching scenario discussed in Section 3.2. Indeed, it is also applicable to any situation where the agent incurs costs represented by the equation $\langle \mathbf{r}_t, \mathbf{x}_t \rangle$.

Assumption 3.1. *$\hat{\mathbf{r}}_t$ is an unbiased estimator of $\mathbf{r}_t$, i.e., $\mathbb{E}[\hat{\mathbf{r}}_t] = \mathbf{r}_t$.*

Assumption 3.2. *Let $\hat{\mathcal{B}}$ be the state space of $\hat{\mathbf{r}}_t$. We assume the existence of the following constants:*

$$\hat{A} = \sup_{\hat{\mathbf{r}} \in \hat{\mathcal{B}}} \|\hat{\mathbf{r}}\|_1, \; \hat{R} = \sup_{\mathbf{x} \in \mathcal{X}, \mathbf{r} \in \hat{\mathcal{B}}} \langle \mathbf{r}, \mathbf{x} \rangle, \tag{3.8}$$

$$D = \sup_{\mathbf{x}, \mathbf{y} \in \mathcal{X}} \|\mathbf{x} - \mathbf{y}\|_1. \tag{3.9}$$

Theorem 3.1 (Regret bound NFPL)**.** *Under Assumptions 3.1 and 3.2, the NFPL algorithm with $\eta = \sqrt{\hat{R} \cdot \hat{A} \cdot T / D}$ enjoys sublinear regret:*

$$\mathcal{R}_T(\textit{NFPL}) \leq 2\sqrt{\hat{R} \cdot \hat{A} \cdot D \cdot T}. \tag{3.10}$$

Proof. It is convenient to define the following two auxiliary quantities

$$\hat{\text{NFPL}}_T = \sum_{t=1}^{T} \langle \hat{\mathbf{r}}_t, M(\hat{\mathbf{r}}_{1:t-1} + \boldsymbol{\gamma}_t) \rangle, \tag{3.11}$$

$$\hat{\text{OPT}}_T = \langle \hat{\mathbf{r}}_{1:T}, M(\hat{\mathbf{r}}_{1:T}) \rangle. \tag{3.12}$$

We compute the expectation—over $\{\hat{\mathbf{r}}_t, \boldsymbol{\gamma}_t\}_1^T$—of the difference between the total cost of NFPL and $\text{OPT}_T = \langle \mathbf{r}_{1:T}, M(\mathbf{r}_{1:T}) \rangle$ as follows

$$\begin{aligned} &\mathbb{E}\left[\text{NFPL}_T - \text{OPT}_T\right] \\ &= \mathbb{E}\left[\text{NFPL}_T - \hat{\text{NFPL}}_T\right] + \mathbb{E}\left[\hat{\text{NFPL}}_T - \hat{\text{OPT}}_T\right] \\ &\quad + \mathbb{E}\left[\hat{\text{OPT}}_T - \text{OPT}_T\right]. \end{aligned} \tag{3.13}$$

We have:

$$\mathbb{E}\left[\text{NFPL}_T - \hat{\text{NFPL}}_T\right] = 0, \tag{3.14}$$

$$\mathbb{E}\left[\hat{\text{OPT}}_T - \text{OPT}_T\right] \leq 0, \tag{3.15}$$

$$\mathbb{E}\left[\hat{\text{NFPL}}_T - \hat{\text{OPT}}_T\right] \leq 2\sqrt{\hat{R} \cdot \hat{A} \cdot D \cdot T}. \tag{3.16}$$

The random vectors $\hat{\mathbf{r}}_t$ and $M(\hat{\mathbf{r}}_{1:t-1} + \boldsymbol{\gamma}_t)$ are independent, hence $\mathbb{E}\left[\langle \hat{\mathbf{r}}_t, M(\hat{\mathbf{r}}_{1:t-1} + \boldsymbol{\gamma}_t)\rangle\right] = \langle \mathbf{r}_t, \mathbb{E}\left[M(\hat{\mathbf{r}}_{1:t-1} + \boldsymbol{\gamma}_t)\right]\rangle$ and by linearity of the expectation we deduce (3.14). We have that $\hat{\text{OPT}}_T \leq \langle \hat{\mathbf{r}}_{1:T}, M(\mathbf{r}_{1:T})\rangle$, we get then (3.15).

The quantity $\hat{\text{NFPL}}_T - \hat{\text{OPT}}_T$ can be seen as the difference between the cost of an FPL algorithm with uniform noise, that observes costs $\{\hat{\mathbf{r}}_t\}_1^T$, minus the cost incurred by the optimal static allocation $\mathbf{x}^* = M(\hat{\mathbf{r}}_{1:T})$. Therefore, applying [KV05, Theorem 1.1 a)] with $\epsilon = 1/\eta$ such that $\eta = \sqrt{\hat{R} \cdot \hat{A} \cdot T / D}$, we get

$$\mathbb{E}_{\{\boldsymbol{\gamma}_t\}_1^T}\left[\hat{\text{NFPL}}_T - \hat{\text{OPT}}_T\right] \leq 2\sqrt{\hat{R} \cdot \hat{A} \cdot D \cdot T}$$

for any $\{\hat{\mathbf{r}}_t\}_1^T$, and by taking the expectation over the randomness of $\{\hat{\mathbf{r}}_t\}_1^T$ in both sides of the last inequality, we find (3.16). Plugging (3.14), (3.15) and (3.16) in (3.13), we deduce that $\mathbb{E}\left[\text{NFPL}_T - \text{OPT}_T\right] \leq 2\sqrt{\hat{R} \cdot \hat{A} \cdot D \cdot T}$ for every $\{\mathbf{r}_t\}_1^T$, concluding the proof.

□

Remark 3.1. *NFPL regret bound in Theorem 3.1 can be written as $\alpha \cdot \beta$ where $\alpha = \hat{R} \cdot \hat{A}/(R \cdot A)$, $\beta = 2\sqrt{R \cdot A \cdot D \cdot T}$, $R = \sup_{\mathbf{x} \in \mathcal{X}, \mathbf{r} \in \bar{\mathcal{B}}} \langle \mathbf{r}, \mathbf{x}\rangle$ and $A = \sup_{\mathbf{r} \in \bar{\mathcal{B}}} \|\mathbf{r}\|_1$. Observe that β is FPL's classical regret bound when the algorithm knows the exact costs [KV05, Theorem 1.1 a)]. It is easy to verify that α is greater than or equal to 1 and can then be interpreted as the performance loss the algorithm incurs due to the noisy costs.*

3.3.2 NFPL for Caching

We apply NFPL to the caching problem (section 3.2.1), deriving $\hat{\mathbf{r}}_t$ from sampled requests. Two methods are explored: NFPL-Fix, sampling a fixed number of requests within each batch, and NFPL-Var, independently sampling each request within the batch with a fixed probability.

NFPL-Fix. The caching system samples $b \geq 1$ requests uniformly at random from a batch of B requests at each time slot. Let $\hat{\mathbf{d}}_t$ be the number of requests for each file in the sampled batch at time step t. NFPL-Fix is Algorithm 1 with noisy request estimates $\hat{\mathbf{r}}_t$ given by

$$\hat{\mathbf{r}}_t = \frac{B}{b} \cdot \hat{\mathbf{d}}_t. \tag{3.17}$$

Corollary 3.1 (Regret bound NFPL-Fix)**.** *NFPL-Fix with $\eta = B\sqrt{2T}/\sqrt{2C}$ has sublinear regret:*

$$\mathcal{R}_T(\textit{NFPL-Fix}) \leq 2\sqrt{2} \cdot B\sqrt{C \cdot T}. \tag{3.18}$$

Proof. Observe that with (3.17), we have that $\mathbb{E}\left[\hat{\mathbf{r}}_t\right] = \mathbf{r}_t$. Since $\|\hat{\mathbf{d}}_t\|_1 = b$, then $\hat{A} = B$ and $\hat{R} \leq B$. We have that $D \leq 2C$, hence by applying Theorem 3.1 in this setting, the regret bound readily follows concluding the proof. □

NFPL-Var. The caching system samples each request within the batch of requests with a probability $f > 0$ at each time slot. Let $\hat{\mathbf{s}}_t$ be the number of requests for each file in the sampled batch at time step t. NFPL-Var is Algorithm 1 with noisy request estimates $\hat{\mathbf{r}}_t$ expressed as

$$\hat{\mathbf{r}}_t = \frac{1}{f} \cdot \hat{\mathbf{s}}_t. \tag{3.19}$$

Corollary 3.2 (Regret bound NFPL-Var)**.** *NFPL-Var with* $\eta = B\sqrt{2T} / \left(f\sqrt{2C}\right)$ *has sublinear regret:*

$$\mathcal{R}_T(\textit{NFPL-Var}) \leq 2\sqrt{2} \cdot \frac{B}{f} \cdot \sqrt{C \cdot T}. \tag{3.20}$$

Proof. Observe that with (3.19), we have that $\mathbb{E}\left[\hat{\mathbf{r}}_t\right] = \mathbf{r}_t$. Moreover, we observe that the maximum of $\|\hat{\mathbf{r}}_t\|_1$ is attained when the sub-batch includes all the requests from the batch, i.e., $\|\hat{\mathbf{s}}_t\|_1 = B$. It follows that $\hat{A} = B/f$ and $\hat{R} \leq B/f$. We have that $D \leq 2C$, hence by applying Theorem 3.1 in this setting, the regret bound readily follows concluding the proof.

□

For $b = B$ and $f = 1$, request counts are exact, i.e., $\hat{\mathbf{r}}_t = \mathbf{r}_t$, and both NFPL-Fix and NFPL-Var coincide with the classic FPL. Using Corollary 3.1 or Corollary 3.2 we deduce the following corollary.

Corollary 3.3 (Regret bound FPL caching)**.** *FPL with* $\eta = B\sqrt{2T}/\sqrt{2C}$ *has sublinear regret:*

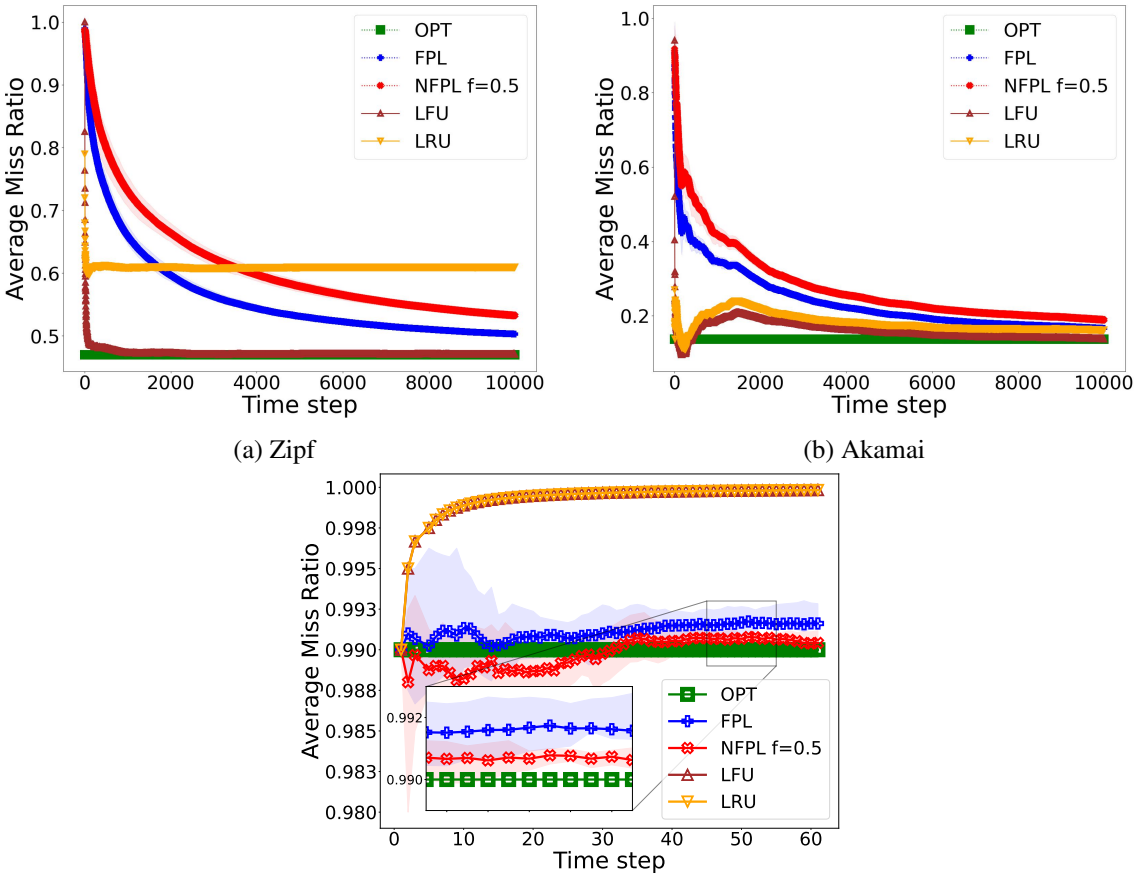
$$\mathcal{R}_T(\textit{FPL}) \leq 2\sqrt{2} \cdot B\sqrt{C \cdot T}. \tag{3.21}$$

The authors of [BBS20] proved regret guarantees for FPL applied to caching under perfect knowledge of the requests when $B = 1$. We report the result here for completeness.

Theorem 3.2. *[BBS20, Thm. 3] FPL applied to the caching problem with* $B = 1$*, learning rate* $\eta = \frac{1}{4\pi(\ln N)^{1/4}}\sqrt{\frac{T}{C}}$*, and noise vectors* $\{\boldsymbol{\gamma}_t\}_1^T$*, where* $\boldsymbol{\gamma}_t/\eta$ *is drawn from a standard multivariate normal distribution, has sublinear regret. More specifically*

$$\mathcal{R}_T(\textit{FPL}) \leq 1.51 \cdot (\ln N)^{1/4} \cdot \sqrt{C \cdot T}. \tag{3.22}$$

The comparison of Corollary 3.3 and Theorem 3.2 shows that our analysis is also of interest when requests are exactly known. First, our bound (3.21) is also valid when the requests are batched, which is of practical interest since updating the cache at each request might be computationally impractical. Second, our bound does not depend on the catalog size, as (3.22) does, and in particular it will not diverge as N goes to infinity.

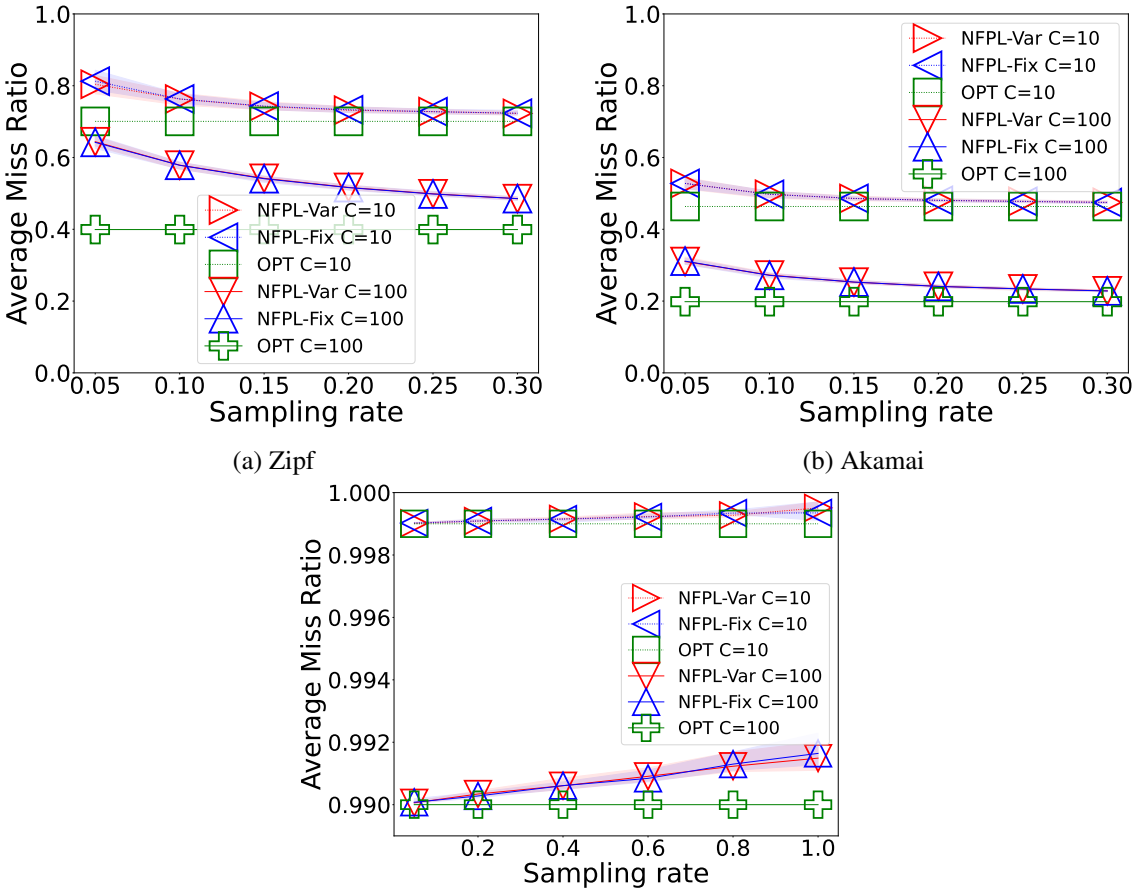


(a) Zipf

(b) Akamai

(c) Round-robin

Figure 3.1: Average miss ratio, $C = 100$.



(a) Zipf

(b) Akamai

(c) Round-robin

Figure 3.2: Average miss ratio vs. sampling probability.

3.4 Experiments

We conducted simulations of NFPL-Fix and NFPL-Var and other existing policies, using both synthetic and real-world traces. Details about the traces are presented in Section 3.4.1, while Section 3.4.2 discusses the caching baselines. We evaluate the effectiveness of our proposed algorithms, NFPL-Fix and NFPL-Var, from two perspectives. First, in Section 3.4.3, we compare the NFPL family of algorithms to traditional caching algorithms. Second, we compare NFPL-Fix and NFPL-Var and show the effect of sampling on their performance in Section 3.4.4.

3.4.1 Traces

Zipf trace. We generate a total of 5×10^6 requests from a catalog of $N = 10^4$ files following an i.i.d. Zipf distribution with exponent $\alpha = 1$. The Zipf distribution is a popular model for the request process in caching [Bre+99].

Akamai trace. The request trace, sourced from Akamai CDN as documented in [Neg+17], encompasses several days of file requests, amounting to a total of 2×10^7 requests for a catalog comprising $N = 10^3$ files.

Round-robin trace. We generate a total of 10^6 file requests from a catalog comprising $N = 10^4$ files in a round-robin fashion. The round-robin trace is commonly considered as an adversarial trace [BBS20].

3.4.2 Caching policies

We compare our methods (NFPL-Fix and NFPL-Var) with the optimal static cache allocation with hindsight (OPT), FPL with perfect knowledge of the requests (equivalent to NFPL-Var with $f = 1$), as well as two classic caching policies: Least-Frequently-Used (LFU) and Least-Recently-Used (LRU). Upon a miss, LFU and LRU evict from the cache the least popular file and the least recently requested file, respectively. FPL and NFPL policies are configured with T equal to the number of batches in the corresponding trace.

All the aforementioned caching policies are evaluated with the *average miss ratio* computed as follows

$$\frac{1}{Bt} \sum_{\tau=1}^{t} \langle \mathbf{r}_\tau, \mathbf{x}_\tau \rangle. \tag{3.23}$$

For NFPL-Fix and NFPL-Var, the average miss ratio is averaged over $M = 50$ runs, considering different noisy request estimates $\{\hat{\mathbf{r}}_t\}_1^T$ and noise vectors $\{\boldsymbol{\gamma}_t\}_1^T$. To account for the variability across the runs, we report the first and ninth deciles of the average miss ratio. In all experiments, the batch size B is set to 200.

3.4.3 NFPL vs. classical policies

We simulate NFPL-Var, with sampling probability $f = 0.5$, FPL, LRU, LFU, and OPT over all the presented traces. In Figure 4.3, we show the average miss ratio at each time step t.

In the Zipf trace, files popularity does not change over time and LFU rapidly discerns the most popular files and subsequently converges to OPT. However, due to the noise γ_t, FPL requires a longer duration to accurately determine the files to be stored. NFPL, on the other hand, grapples with two sources of noise: the inherent noise γ_t and the additional noise due to sampling. As a result, NFPL takes even longer to adjust. Nevertheless, both FPL and NFPL outperform LRU, whose missing ratio fails to converge to OPT.

In the Akamai trace, it is plausible to anticipate fluctuations in popularity over time, and requests' temporal correlations. Such patterns can be advantageous for LRU. In fact, LRU now performs almost on par with LFU. Notably, both FPL and NFPL appear to be converging to the performance of OPT.

Under the round-robin trace, optimality can be achieved with any static allocation of C distinct files. However, both LRU and LFU demonstrate equally disappointing performances. This is because at any time LRU stores the C most recently requested files, while LFU retains the C most frequently requested ones, but the next request is not for any of these cached files.

In contrast, both NFPL-Var and FPL showcase performances that are close to optimal. This reaffirms the resilience and adaptability of online learning policies across request processes as different as the three traces we considered. Intriguingly, NFPL-Var, which is inherently "noisier," outperforms FPL to some extent. This phenomenon can be explained: the noisier $\hat{\mathbf{r}}_{1:t} + \boldsymbol{\gamma}_t$, the more the cache tends to store a random set of files disregarding past requests. Such strategy is precisely up for the round-robin trace.

3.4.4 NFPL-Fix vs. NFPL-Var

We compare the performance of NFPL-Fix, NFPL-Var, and OPT on all the considered traces for two cache sizes: $C \in \{10, 200\}$ for the Zipf trace and $C \in \{10, 100\}$ for the Akamai and round-robin traces. Figure 3.2 illustrates the average miss ratio for all the aforementioned caching policies when varying sampling probabilities, i.e., f for NFPL-Var and b/B for NFPL-Fix.

Across the various traces we analyzed, the performance difference between NFPL-Fix and NFPL-Var is consistently minimal for all the sampling rates. This indicates that the selection of the sampling method may exert only a marginal impact on the performance of NFPL.

The influence of the sampling rate varies across the traces, aligning with the patterns previously noted in Figure 4.3. For the Zipf and Akamai traces, the performance of both NFPL-Fix and NFPL-Var tends towards that of OPT with increasing sampling rates. This is attributable to the relatively stationary nature of these traces, where the count of past requests serves as a good predictor for future requests; thus, more precise estimates bolster performance. In contrast, the round-robin trace benefits from noisier estimates, as it is preferable to overlook past requests in this scenario. As a result, the performance of NFPL-Fix and NFPL-Var deteriorates with a rising sampling rate.

3.5 Conlusion

In this chapter, we introduce the Noisy-Follow-the-Perturbed-Leader (NFPL) algorithm, a variant of the Follow-the-Perturbed-Leader (FPL) algorithm that incorporates noisy cost estimates, and provide conditions on the cost estimates estimator for which NFPL achieves sublinear regret. In the context of the caching problem, we propose two NFPL algorithms, NFPL-Fix and NFPL-Var, based on sampling, that achieve sublinear regret. By conducting experiments on both synthetic and real-world traces, we show the impact of request sampling on the performance of NFPL. In future work, we plan to investigate the regret of NFPL when the request estimator is based on approximate counting data structures such as the Count-Min Sketch [CM05a].

CHAPTER 4

Similarity Caching

In the preceding chapters, our analysis focused on frequency-based caching policies. However, caching strategies often integrate recency considerations alongside frequency. Among these, the Least Recently Used (LRU) policy stands out, valued for its simplicity and performance. LRU has been studied under various request models, with the Time-To-Live (TTL) approximation proving effective in estimating LRU's hit ratio under an i.i.d. request process. Nevertheless, LRU extensions in the context of similarity caching, such as RND-LRU, remain an area requiring attention. Similarity caching allows requests for an item to be fulfilled by a similar item. In this chapter, we delve into the extension of the TTL approximation to accommodate similarity caching. Specifically, we introduce a novel method for estimating the hit ratio of the similarity caching policy RND-LRU. Our proposed approach, named the RND-TTL approximation, introduces the RND-TTL cache model and tunes its parameters to estimate RND-LRU's hit ratio. The parameter tuning involves solving a fixed point system of equations for which we provide an algorithm for numerical resolution and sufficient conditions for its convergence. Our approach for approximating the hit ratio of RND-LRU is evaluated on synthetic and real-world traces.

4.1 Introduction

Many applications require to retrieve items similar to a given user's request. For example, in content-based image retrieval [Fal+08] systems, users can submit an image to obtain other visually similar images. A similarity cache may intercept the user's request, perform a local similarity search over the set of locally stored items, and then if the search result is evaluated satisfactorily, provide it to the user. The cache may thus speed up the reply and reduce the load on the server, at the cost of providing items *possibly less similar* than those provided by the server.

Originally proposed for content-based image retrieval [Fal+08] and contextual advertising [Pan+09], similarity caches are now a building block for a large variety of machine learning based inference systems for recommendations [Ser+18], image recognition [DGN17; Ven+18] and network traffic [Fin+22] classification. In these cases, the similarity cache stores past queries and the respective inference results to serve future similar requests. Motivated by the large number of applications, much effort has been devoted recently to formalize similarity caching [NGL21; GLN21] as well as to propose new caching policies [Zho+20; Sab+21; SNC23].

RND-LRU is a randomized similarity caching policy proposed in the seminal paper [Pan+09]. It is a variant of the least recently used (LRU) policy adapted to the similarity caching setting. We still lack an analytical evaluation of RND-LRU's performance. The aim of this chapter is to

fill this gap. Our objective is to compute the hit ratio, i.e., the fraction of requests satisfied by the RND-LRU cache.

Computing the hit ratio is a challenging task, even for the classic LRU policy under the Independent Reference Model (IRM), [Fag77]. Its computational cost is exponential in both the cache size and the number of items [WK71; DT90]. The so-called Che's or time-to-live (TTL) approximation is a highly efficient method for accurately estimating the hit ratio of LRU under IRM [CTW02; Cho+14]. The TTL approximation leverages the analysis of an opportune cache—which benefits from decoupling caching decisions across items—and utilizes its hit ratio as an estimate for the hit ratio of LRU. Many studies [Fag77; FRR12; LT15; JNT18] have provided theoretical support to the TTL approximation under different assumptions regarding the request process.

As items in a RND-LRU cache are strongly coupled, analyzing RND-LRU becomes even more challenging. In fact, in classic caching, an item in the cache serves only requests for itself, while in similarity caching, a cached item can serve requests for a set of similar items as long as neither these nor their most similar items are cached.

In this chapter, we extend the TTL approximation to RND-LRU, by introducing the RND-TTL approximation; the latter is based on a novel similarity caching model, that we call RND-TTL. This approximation involves tuning the parameters of the RND-TTL model to estimate the hit ratio of RND-LRU. We stress that there has been no prior analysis of the performance of RND-LRU. Our contributions can be summarized as follows:

1. We propose a novel similarity caching model named RND-TTL and we compute its hit ratio under IRM.

2. We derive constraints on the RND-TTL cache model's parameters to approximate RND-LRU's hit ratio.

3. The parameter tuning process for the RND-TTL model involves solving a system of fixed point equations; we present a parameterized iterative algorithm to solve this system and provide a practical method for selecting the algorithm's parameter.

4. We provide sufficient conditions for the iterative algorithm to converge.

5. We evaluate the accuracy of our RND-TTL approximation to estimate the hit ratio of RND-LRU on both synthetic and real-world traces.

The rest of the chapter is organized as follows: We present background material on similarity caching and the TTL approximation in Section 4.2 and introduce notation and assumptions in Section 4.3. We define the RND-TTL approximation in Section 4.4 and explain our iterative algorithm for tuning the parameters of the RND-TTL cache in Section 4.5. We evaluate the performance of our RND-TTL approximation in Section 4.6 on both synthetic and real word traces and summarize our findings in Section 4.7. We provide detailed proofs and supplementary material in the appendices (B.1–B.12).

4.2 Background

4.2.1 Similarity Caching

4.2.1.1 Similarity Search

In **similarity search** systems, users can query a remote server, storing a set of items $\mathcal{I}$, to send the k most similar items to a given item n, according to a specific definition of similarity. In practice, items are often represented by Euclidean vectors (called embeddings) [McA+15] so that the dissimilarity cost, $\mathrm{dis}(.,.) : \mathcal{I}^2 \to \mathbb{R}^+$, can be selected to be an opportune distance between the embeddings.

An instance of a similarity search system is the content-match system, which serves as a component in Internet advertising frameworks. Its purpose is to display contextual advertisements (ads) on a publisher's webpage upon user access [Pan+09]. Specifically, the task involves sending a set of k relevant ads to a page, taking into account both its content and the user profile. To evaluate the appropriateness of an ad for a particular page, a common approach involves representing both the page and the ad as vectors within the same high-dimensional metric space. The distance between these representative vectors acts as a measure of suitability, where a smaller distance signifies a higher suitability of the ad for the page. In this context, the content-match system conducts a similarity search to identify the k most relevant ads for the page.

Another example of a similarity search system is the Content-Based Image Retrieval (CBIR) system, which answers queries for an image by the k most similar images [Fal+08]. The similarity between images is measured via the distance between their representative vectors in a high-dimensional metric space.

4.2.1.2 Similarity Cache

In practical scenarios, meeting the time constraints for similarity search queries becomes challenging, especially when dealing with a large catalog size. Addressing this challenge, the seminal papers [Fal+08; Pan+09] advocate deploying a cache near users. This cache, known as a **similarity cache**, operates by maintaining a key-value pair for each item in a subset S of $\mathcal{I}$, where S has cardinality C. The key of an item n in S is its identifier, whereas the value of n is a list containing the $k' \geq k$ closest items to n (including n), and their corresponding embeddings, within the set $\mathcal{I}$. It follows that the similarity cache stores $W \leq C \cdot k'$ distinct items. A similarity caching policy may directly answer a similarity search query for an item n by selecting k items out of the W cached items based on a similarity measure between items' embeddings. The similarity caching policy may then provide answers potentially different from the actual k closest neighbors. For example, SIM-LRU [Pan+09] is a similarity caching policy that operates in two steps to answer a similarity search for an item n:

1) it locates the closest item to n in S, namely $\hat{n}$,

2) if $\hat{n}$ is found to be similar enough to n, SIM-LRU performs a k-nearest-neighbors search for n within $\hat{n}$'s value, that is, within the k' items that are closest to $\hat{n}$, and answers n's request with the resulting k items.

It follows that having a larger k' improves the quality of the approximate answer for the similarity search. Finally, a similarity cache reduces fetching costs at the expense of approximate answers, offering an efficient solution for handling time-sensitive similarity search queries.

4.2.1.3 Hit Ratio and Utility

Exact caching policies aim at maximizing the hit ratio given a fixed cache capacity. In similarity caching, however, hits include both exact and approximate ones. Therefore, a policy that maximizes the hit ratio might be settling for low-quality answers.

Neglia et al. [NGL21] introduced an objective for similarity caching policies. They assume the existence of a nonnegative approximation cost for serving requests for an item x with another item y, denoted $C_a(x, y)$. They also assume a fixed cost C_r for retrieving an item from the original server. The objective of the similarity caching policy is to minimize the total incurred cost over a time horizon.

An earlier formulation for an objective for similarity caching policies was proposed by Pandey et al. [Pan+09]. They assume that there exists a utility function that quantifies the satisfaction of the users by the answers provided via the similarity caching policy. The objective for similarity caching proposed by Pandey is to maximize the utility function under a constraint on the maximum tolerated delay. This constraint is application-dependent and can be expressed as a restriction on the minimal hit ratio.

Observe that adjusting the criterion of similarity between items in a similarity caching policy provides some flexibility. A looser criterion increases the portion of approximate hits with respect to exact hits thereby decreasing simultaneously the average response delay and the utility function. On the other hand, a stricter similarity criterion reduces the prevalence of approximate hits and as a result, the hit ratio decreases whereas the average response delay and the utility function both increase. Our contribution in this chapter is to quantify the hit ratio of the RND-LRU policy for a given criterion of similarity.

4.2.1.4 RND-LRU and SIM-LRU similarity caching policies

In exact caching, LRU manages a list of cached item keys based on access order. When the LRU cache receives a request for an item n, it checks its presence. If n is cached, the request is a *hit*, and the response is sent immediately to the user, moving n's key to the list's front. Otherwise, the request is a *miss*, and the request goes to the server. Upon obtaining n from the server, it is added to the cache, and its key is placed at the front of the list. The least recently used item (bottom of the list) is evicted, maintaining the fixed cache size.

RND-LRU [Pan+09] is a popular randomized LRU-based similarity caching policy. RND-LRU maintains LRU's procedure but adapts the hit definition for similarity caching. RND-LRU keeps an ordered list, L, of the items in the set S, defined in Section 4.2.1.2. Unlike LRU, even if n is not in L, RND-LRU can consider a request for n a hit. RND-LRU can answer n's similarity search request using its closest item in S, namely, $\hat{n} \triangleq \arg\min_{m \in L} \mathrm{dis}(n, m)$. More specifically, the request for n is probabilistically answered by sending k closest neighbors of n among $\hat{n}$'s k' closest neighbors. RND-LRU's randomness lies in parameters $\mathbf{q} = (q_m(n))_{n,m \in \mathcal{I}^2}$. For every pair of items n and m, $q_m(n)$ denotes the probability that a candidate item m is used to respond to a

query for n, given that $m = \hat{n}$. The function $q_m(n)$ decreases with the dissimilarity between m and n.

Algorithm 2: RND-LRU [Pan+09]

1: **Input:**
2: Sequence of requests $(r_1, \ldots, r_J)$ of lentgh J
3: Initial ordered list of cached items $L_0 = (l_{0,1}, \ldots, l_{0,C})$
4: Probabilities $(q_n(m))_{n,m \in \mathcal{I}^2}$
5: **Output:**
6: Ordered list of cached items at each time step $j \in \{1, \ldots, J\}$.
7: **Algorithm:**
8: **for** $j = 1$ to J **do**
9: $L_j = (l_{j,1}, \ldots, l_{j,C}) \leftarrow L_{j-1}$
10: Compute the closest item to r_j in L_{j-1} as $\hat{r}_j = \arg\min_{m \in L_{j-1}} \mathrm{dis}(r_j, m)$
11: Generate a uniform random number $\delta \in [0, 1]$
12: **if** $\delta \leq q_{\hat{r}_j}(r_j)$ **then**
13: **Case 1:** Hit, encompassing exact hit and approximate hit
14: $L_j \leftarrow$MoveToFront$(L_{j-1}, \hat{r}_j)$
15: **else**
16: **Case 2:** Miss
17: $L_j \leftarrow$InsertAtFront$(L_{j-1} \setminus l_{j-1,C}, r_j)$
18: **end if**
19: **end for**
20: **return** $L_1, \ldots, L_J$

The details of RND-LRU are presented in Algorithm 2. Upon receiving a request at time step t for item r_t, RND-LRU locates the closest item to r_t in the cache, denoted as $\hat{r}_t$ (line 10). A random sample δ is generated uniformly at random in the interval between 0 and 1 (line 11). If $\delta \leq q_{\hat{r}_t}(r_t)$ we have a hit (line 12), and the query for r_t is answered using the k closest neighbours of r_t among the k' neighbours of $\hat{r}_t$. Note that hits include approximate hits and exact hits ($\hat{r}_t = r_t$). After a hit, the cache is rearranged by moving $\hat{r}_t$'s key to the front of the list (line 14). Alternatively, if $\delta > q_{\hat{n}}(n)$ we have a miss: the request is forwarded to the original server to retrieve the list of k' closest items to r_t in $\mathcal{I}$, out of which the closest k items are provided to the user. RND-LRU evicts the least recently used key at the bottom of the list and its corresponding key-value pair in the cache. It then inserts the new key for r_t at the front of the list and the corresponding key-value pair into the cache (line 17).

SIM-LRU [Pan+09] is also an LRU-based similarity caching. It has a similarity threshold d and uses $\hat{n}$ to serve n's request only if $\mathrm{dis}(n, \hat{n}) \leq d$. SIM-LRU is a particular case of RND-LRU such that $q_m(n) = 1$ if $\mathrm{dis}(m, n) \leq d$ and $q_m(n) = 0$ otherwise. Note that LRU is equivalent to RND-LRU when $k' = k = 1$, $q_m(n) = 1$ if $m = n$ and $q_m(n) = 0$ otherwise.

Table 4.1: Table of notation.

Basic parameters

$\mathcal{I}$	set of items
$N = \lvert\mathcal{I}\rvert$	catalog size
C	cache capacity
λ_n	arrival rate of requests for item n
$\mathrm{dis}(\cdot,\cdot)$	function measuring the dissimilarity between items
d	similarity threshold

RND-LRU and **R-TTL**

$L_{\text{RND-LRU}}(t)$	ordered list of cached items in RND-LRU at time t
$\tilde{\Omega}$	state space of $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$
T_n	initial timer duration for item n in R-TTL
$\tilde{\boldsymbol{\pi}}$	limiting distribution of $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$
$\boldsymbol{\mu}$	limiting distribution of the set of cached items in R-TTL
$\tilde{\lambda}_n^i$	insertion rate of item n in RND-LRU
$\tilde{\lambda}_n^r$	refresh rate of item n in RND-LRU
H	hit ratio of RND-LRU
$q_n(m)$	probability to use candidate n to serve a request for m
$\mathcal{N}(n)$	neighbors of item n
$\mathcal{N}^c[n]$	neighbors of item n including n
$\mathcal{N}_m(n)$	items in $\mathcal{N}(n)$ strictly closer to n than m
$\mathcal{N}_m^c[n]$	items in $\mathcal{N}^c[n]$ strictly closer to n than m

RND-TTL

$S_{\text{RND-TTL}}(t)$	set of cached items in RND-TTL at time t
Ω	state space of $\{S_{\text{RND-TTL}}(t),\ t \geq 0\}$
$\boldsymbol{\pi}$	limiting distribution of $\{S_{\text{RND-TTL}}(t),\ t \geq 0\}$
p_n^i	insertion probability of item n given that n is not cached
λ_n^i	insertion rate of item n given that n is not cached
λ_n^r	timer refresh rate of item n given that n is cached
$X_n(t)$	1 if item n is in cache at time t and 0 otherwise
o_n	fraction of time item n spent in the cache
h_n	hit probability of item n
T_n	initial timer duration for item n

4.2.2 TTL Approximation for LRU Cache

4.2.2.1 TTL Cache

Time to Live (TTL) serves as a mechanism to limit the duration of data within a network. Various applications, such as Content Delivery Networks (CDNs) and the Domain Name System (DNS), leverage TTL to dictate the eviction time for cached items [CK03; CA13; ACN16; Mou+19]. In TTL caching policies, each cached item is associated with a timer, triggering eviction upon timer expiration. Analyzing the hit ratio is more straightforward in a TTL cache than in an LRU cache thanks to the decoupling of caching decisions across items in the former. The seminal work by Jung et al. [JBB03] introduces an analytical model for the hit ratio of a TTL cache, assuming that the inter-arrival times for each item are i.i.d. random variables, characterizing the request process as a renewal process. Subsequent research has explored adapting TTL choices to the request process [FRP16; Bas+18], and the analysis has been extended to encompass a network of TTL caches [Cho+14; CA13; ACN16; Ber+14]. A comprehensive overview of TTL caching policies is provided in [Has+23].

TTL caches are efficient modeling tools to analyze caching policies [GLM16]. In particular, a TTL caching policy with timers resets per hit was proposed as a model for LRU [Fag77; CTW02]. This TTL caching policy has enough storage for all items, and assigns a deterministic timer with value T_n to each item n whose expiration triggers eviction. Upon a request for a noncached item n, i.e., a miss, n is added to the cache with a timer duration of T_n. Conversely, when a request for a cached item n is received, i.e., a hit, the timer associated with n is reset to the original value T_n. In this chapter, except otherwise noted, we refer to TTL caches with resets per hit simply as TTL caches.

4.2.2.2 TTL Approximation

Fagin [Fag77] proposes an efficient method to estimate, under IRM, the hit ratio of the LRU caching policy. This method approximates the hit ratio of LRU, with the hit ratio of a discrete-time TTL cache* with a specific choice of TTL values. Specifically, the TTL value T_n for each item n is set to the characteristic time t_C [CTW02], which guarantees that the expected number of cached items in the TTL cache is equal to the cache capacity C of the LRU cache. This approximation is proven to be asymptotically accurate [Fag77]. Che et al. [CTW02] rediscovered Fagin's method under Poisson requests. Fagin's approximation is later extended to other caching policies and under more generalized assumptions on the request process [FRR12; LT15; JNT18; GLM16; GV17], earning the name **TTL approximation** in the literature.

In a discrete-time TTL cache, a hit for item n occurs whenever two consecutive requests for n are separated by strictly less than T_n requests. Under IRM, the request for any item n occurs with probability p_n independently of past requests, and then the hit probability for n in a discrete-time TTL cache is given by $h_n = 1 - (1 - p_n)^{T_n}$. Fagin's approximation is thus equivalent to setting, for every n, T_n to the characteristic time t_C which verifies:

$$\sum_{n \in \mathcal{I}} \left(1 - (1 - p_n)^{t_C}\right) = C. \tag{4.1}$$

*A discrete-time TTL policy is equivalent to the working set policy used by Fagin.

The above expression allows the computation of t_C, e.g., by using a bisection method. The hit ratio for LRU, H, is then approximated as:

$$H \approx \sum_{n \in \mathcal{I}} p_n \left(1 - (1 - p_n)^{t_C}\right). \tag{4.2}$$

Another variant of the TTL approximation assumes for every item n that the request process is Poisson with rate λ_n, i.e., the inter-arrival time for n is exponentially distributed with mean $1/\lambda_n$ [CTW02; FRR12]. This approach is similar to Fagin's method where for every n, T_n is set to the characteristic time t_C. However, the hit probability for n in the TTL cache becomes:

$$h_n = 1 - \mathrm{e}^{-\lambda_n t_C}, \tag{4.3}$$

and t_C verifies:

$$\sum_{n \in \mathcal{I}} \left(1 - \mathrm{e}^{-\lambda_n t_C}\right) = C. \tag{4.4}$$

The hit ratio of LRU is then approximated as

$$H \approx \left(\frac{1}{\sum_{i \in \mathcal{I}} \lambda_i}\right) \cdot \sum_{n \in \mathcal{I}} \lambda_n \left(1 - \mathrm{e}^{-\lambda_n t_C}\right). \tag{4.5}$$

The assumption of a Poisson request process for every item n is a particular case of IRM where the corresponding probability for n to be requested is $\lambda_n / \sum_{i \in \mathcal{I}} \lambda_i$. This specific IRM assumption leads to a formula for estimating the hit ratio of LRU (see (4.5)) different from the formula proposed by Fagin (see (4.2)). However, while the additional Poisson assumption is relevant for the hit probability of the TTL cache, the hit ratio of LRU is insensitive to this additional assumption. Indeed, both (4.2) and (4.5) are asymptotically accurate approximations to LRU under suitable conditions [Fag77; FRR12; JNT18].

4.3 Notation and Assumptions

Recall from Section 4.2.1.4 that the use of key-value pairs in SIM-LRU and RND-LRU essentially converts the search for the k closest items into a search for the closest item key in the cache. To lighten the presentation, we will simply say from now on that the similarity cache replies to a request for n with the closest *item* in the cache.

We list in Table 4.1 the main notation that we use. We assume equal size items and, as in [McA+15], assume that they can be represented by Euclidean vectors, such that an opportune distance between vectors informs on the dissimilarity cost, $\mathrm{dis}(.,.)$, between pairs of items. We maintain the same notation used in Section 4.2: $\mathcal{I}$ denotes the set of items with $|\mathcal{I}| = N$, $\hat{n}$ is the closest cached item to n, RND-LRU is parameterized by the vector $\mathbf{q} = (q_m(n))_{n,m \in \mathcal{I}^2}$, where $q_m(n)$ is the probability that a candidate item m is used to reply to a query for n given that $m = \hat{n}$, and SIM-LRU is parameterized by the similarity threshold d. We assume that $q_n(n) = 1$.

Under RND-LRU a request for item n could be served by an item m such that $q_m(n) > 0$. Therefore, it is convenient to define for n the set of such candidates items as $\mathcal{N}^c[n] \triangleq \{m \in \mathcal{I} : q_m(n) > 0\}$. We call the elements in $\mathcal{N}^c[n]$ *distinct* from n the neighbors of n and denote their set

as $\mathcal{N}(n) \triangleq \mathcal{N}^c[n] \setminus \{n\}$. For convenience, we define similarly the sets $\mathcal{N}_m(n)$ and $\mathcal{N}_m^c[n]$: these are the respective subsets of $\mathcal{N}(n)$ and $\mathcal{N}^c[n]$, designating items that are closer to n than m is. Namely, $\mathcal{N}_m(n) \triangleq \{l \in \mathcal{N}(n) : \ \mathrm{dis}(n,l) < \mathrm{dis}(n,m)\}$ and $\mathcal{N}_m^c[n] \triangleq \{l \in \mathcal{N}^c[n] : \ \mathrm{dis}(n,l) < \mathrm{dis}(n,m)\}$. We denote the ordered list of cached items at any time t in RND-LRU as $L_{\text{RND-LRU}}(t)$. As commonly used, $\mathbb{1}(A)$ stands for the indicator function that A is true.

Throughout the chapter, we assume that requests for items follow the Independent Reference Model (IRM). We also make use of the following assumptions.

Assumption 4.1. *Requests for items are mutually independent Poisson processes. The request rate for item n is λ_n and $\sum_{i\in\mathcal{I}} \lambda_i = 1$.*

Assumption 4.1 is a particular case of IRM where the probability of a request for item n coincides with its request rate. However, while Assumption 4.1 is relevant for the hit probability of the TTL-based similarity caching model that we introduce later, our approximation for the hit ratio of RND-LRU can be employed under the more general IRM assumption. For the sake of simplicity, we refer to both the rate of the request of item n and its probability of being requested as λ_n. Under Assumption 4.1, the ordered list of cached items in RND-LRU, namely $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$, is a continuous time Markov chain with finite state space denoted as $\tilde{\Omega}$.

Assumption 4.2. *$\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$ has a limiting distribution that we denote as $\tilde{\boldsymbol{\pi}} = (\tilde{\pi}_L)_{L\in\tilde{\Omega}}$.*

Assumption 4.2 eliminates cases where the hit ratio of RND-LRU depends on the initial list of cached items $L_{\text{RND-LRU}}(0)$. This assumption is verified when RND-LRU's Markov chain is irreducible. A sufficient condition for irreducibility is that $q_m(n) < 1$ for every $n \neq m$.

Assumption 4.3. *Items in $\mathcal{N}(n)$ can be strictly ordered according to their dissimilarity with respect to n, i.e., for any $m, l \in \mathcal{N}(n)$ and $m \neq l$, we have $\mathrm{dis}(n,m) \neq \mathrm{dis}(n,l)$.*

4.4 RND-TTL Approximation for Similarity Caching

Inspired by the TTL approximation that allows us to approximate the hit ratio of an LRU cache, we introduce in this section the RND-TTL cache model and the RND-TTL approximation method to estimate the hit ratio of RND-LRU. Firstly, in Section 4.4.1, we describe the RND-TTL cache model, highlighting its specific characteristics. Secondly, in Section 4.4.2, we explain how the RND-TTL model can capture the dynamics and behavior of RND-LRU. Thirdly, in Section 4.4.3, we present the RND-TTL approximation that imposes specific constraints on the RND-TTL caches' parameters to estimate the hit ratio of RND-LRU.

4.4.1 The RND-TTL Caching Model

Our objective in this section is to introduce a caching model that allows to extend the TTL approximation in the scope of estimating the hit ratio of RND-LRU. We present in the following a first extension (called R-TTL) of the TTL cache. While intuitive, this extension suffers from strong coupling between items, which led us to devise another extension (called RND-TTL) enabling us to estimate the hit ratio.

4.4.1.1 R-TTL: A TTL-Based Similarity Caching Policy

In contrast to the conventional Least Recently Used (LRU) caching strategy, RND-LRU, as detailed in Section 4.2.1.4, deviates solely in its characterization of hits or misses by permitting approximate hits. The Time-to-Live (TTL) cache, illustrated in Section 4.2.2.1, has been demonstrated to asymptotically capture the performance of LRU when its parameters are selected according to the TTL approximation in Section 4.2.2.2.

Building upon this understanding, we introduce a natural extension of the TTL cache tailored to emulate RND-LRU, denoted as R-TTL. This caching policy encompasses parameters that include the timers durations $(T_n)_{n\in\mathcal{I}}$, akin to those in a TTL cache, and $(q_m(n))_{n,m\in\mathcal{I}^2}$, where $q_m(n)$ refers to the probability that a request for item n is satisfied by item m under the condition that m is the closest to n in the cache. R-TTL maintains an analogous procedural framework as the TTL cache but embraces RND-LRU's definition of hits or misses.

We provide Algorithm 4 (in B.1) that can be used to simulate R-TTL. Observe how RND-LRU (see Algorithm 2) and R-TTL have in common the rules used to determine when and if items should be used to serve a given request. However, whereas an item in a RND-LRU cache can be evicted as a result of a request arrival that cannot be served, in R-TTL evictions occur after TTL reaches zero. In addition, a refresh in a RND-LRU cache corresponds to a "move to front" operation, whereas in R-TTL, it corresponds to a TTL reset. Note that R-TTL is versatile, as we can adjust the hit probability of an item n by controlling its timer duration T_n.

A natural extension of the TTL approximation, presented in Section 4.2.2.2, to RND-LRU, is to approximate the hit ratio of RND-LRU with the hit ratio of R-TTL, such that for every item n, the timer duration T_n is set to the characteristic time guaranteeing that the expected number of cached items in R-TTL is equal to the cache capacity C of RND-LRU. Unfortunately, while caching decisions are decoupled in a TTL cache, it is not the case for R-TTL because (i) an item might not be admitted in the cache if one of its neighbors is cached and (ii) an item's timer might be reset by requests for one of its neighbors. The coupling in the caching decisions of R-TTL makes computing its hit ratio or the characteristic time challenging even under IRM and hence also the application of the TTL approximation. For this reason, we propose another TTL cache model for RND-LRU, inspired by R-TTL, that decouples the caching decisions. We refer to this TTL cache model as RND-TTL.

4.4.1.2 RND-TTL Model for RND-LRU

The RND-TTL cache model is parameterized by the vector of TTLs $\boldsymbol{T} = (T_n)_{n\in\mathcal{I}}$ and by two additional vectors $\boldsymbol{\lambda^r} = (\lambda^r_n)_{n\in\mathcal{I}}$ and $\mathbf{p^i} = (p^i_n)_{n\in\mathcal{I}}$ as described next. The parameters $\boldsymbol{T}$ and $\boldsymbol{\lambda^r}$ dictate how long items remain in the cache, while $\mathbf{p^i}$ characterizes the cache insertion probability.

In the RND-TTL cache, each item is assigned a timer upon its insertion in the cache and is evicted from the cache when its timer expires. Item n's timer is initialized with the duration T_n and is reset to T_n, when n is cached, according to a Poisson process with rate λ^r_n (the superscript "r" refers to "reset" or "refresh").

Upon a request for an item n, either n is in the cache and is used to fulfill the request or it is not in the cache which gives rise to the two following possible scenarios:

The request for n results in a cache miss and consequently item n is inserted into the cache. This scenario occurs with probability p_n^i (the superscript "i" refers to "insertion").

The request is fulfilled by the nearest item to n in the cache*, which occurs with probability $1 - p_n^i$.

Note that the above model is inspired by the behavior of R-TTL described in the previous section while ensuring that the dynamics of items are decoupled from each other as in traditional TTL systems. Indeed, upon a request for a noncached item n, the insertion probability of n depends on the set of cached items in R-TTL while it is always equal to p_n^i in RND-TTL. Moreover, a request for an item m might reset item n's timer, when n is cached in R-TTL, while the reset process for item n's timer is Poisson with rate λ_n^r independently from other items' requests in RND-TTL. The parameters λ_n^r and p_n^i can be set according to the modeling purposes. We show later on that $\mathbf{T}$, $\boldsymbol{\lambda}^r$ and $\mathbf{p^i}$ can be set in such a way as to capture the behavior of RND-LRU, with the coupling between items reflected through a parametrization of these values.

4.4.1.3 Occupancies in RND-TTL

We are interested in computing the fraction of time o_n spent by item n in the RND-TTL cache in the stationary setting. Let $\{X_n(t),\ t \geq 0\}$ be the stochastic process taking value 1 when item n is in the cache and 0 otherwise. The occupancy o_n is formally written as follows.

$$o_n \triangleq \lim_{t \to +\infty} \frac{1}{t} \int_0^t \mathbb{1}(X_n(u) = 1)\, du\,. \tag{4.6}$$

Proposition 4.1 (Occupancy). *Under Assumption 4.1, the occupancy in the RND-TTL cache of item n is expressed as:*

$$o_n = \left(\frac{1}{\lambda_n^i} \cdot \frac{\lambda_n^r}{\mathrm{e}^{\lambda_n^r T_n} - 1} + 1 \right)^{-1}, \tag{4.7}$$

where

$$\lambda_n^i = \lambda_n \cdot p_n^i\,. \tag{4.8}$$

Proof. The result follows from a renewal argument, where $\mathbb{E}\left[T_n^{\mathrm{On}}\right]$ and $\mathbb{E}\left[T_n^{\mathrm{Off}}\right]$ are the mean time that an item resides on and off the cache, per cycle,

$$o_n = \frac{\mathbb{E}\left[T_n^{\mathrm{On}}\right]}{\mathbb{E}\left[T_n^{\mathrm{Off}}\right] + \mathbb{E}\left[T_n^{\mathrm{On}}\right]} = \left(\mathbb{E}\left[T_n^{\mathrm{Off}}\right] \cdot \frac{1}{\mathbb{E}\left[T_n^{\mathrm{On}}\right]} + 1 \right)^{-1}. \tag{4.9}$$

In the above expression, $\mathbb{E}\left[T_n^{\mathrm{On}}\right]$ is the mean duration of a busy period of an M/D/∞ queue with arrival rate and mean residence time given by λ_n^r and T_n, respectively,

$$\mathbb{E}\left[T_n^{\mathrm{On}}\right] = \frac{1}{\lambda_n^r}\left(e^{\lambda_n^r T_n} - 1\right). \tag{4.10}$$

*We assume that the cache statically stores a tombstone item whose distance to all items is infinite. Whenever a request arrives in an empty cache, the tombstone item is returned as the closest item in the cache.

$\mathbb{E}\left[T_n^{\text{Off}}\right]$ is the mean time to insert an item after it is removed,

$$\mathbb{E}\left[T_n^{\text{Off}}\right] = \frac{1}{\lambda_n^i} \,. \tag{4.11}$$

For additional details, we refer the reader to Appendix B.2 page 82.

□

We stress that under Assumption 4.1, $\{X_n(t),\ t \geq 0\}$ has limiting distribution given by the occupancy, namely,

$$\lim_{t\to+\infty} \Pr\left(X_n(t) = 1\right) = o_n,\ \lim_{t\to+\infty} \Pr\left(X_n(t) = 0\right) = 1 - o_n \,. \tag{4.12}$$

The above equation can be justified thanks to [Ros95, Thm. 3.4.4].

4.4.1.4 Distribution of Set of Cached Items

We denote the set of cached items in RND-TTL at time t as $S_{\text{RND-TTL}}(t)$. Formally,

$$S_{\text{RND-TTL}}(t) = \{n \in \mathcal{I} :\ X_n(t) = 1\} \,. \tag{4.13}$$

We denote the state space of the stochastic process $\{S_{\text{RND-TTL}}(t),\ t \geq 0\}$ as Ω. In TTL-based policies such as RND-TTL, $\Omega = 2^{\mathcal{I}}$, where $2^{\mathcal{I}}$ denotes the power set of $\mathcal{I}$. Observing that the caching decisions in the RND-TTL cache are independent across items and that $\{X_n(t), t \geq 0\}$ has a limiting distribution for any item n under Assumption 4.1, it follows that $\{S_{\text{RND-TTL}}(t),\ t \geq 0\}$ has a limiting distribution that we denote as $\boldsymbol{\pi} = (\pi_S)_{S\in\Omega}$. Using (4.12), for any set of cached items $S \in \Omega$, the corresponding limiting probability π_S can be computed as follows:

$$\pi_S = \prod_{n\in S} o_n \cdot \prod_{m\notin S} (1 - o_m) \,. \tag{4.14}$$

4.4.1.5 Item's Hit Probability

We now give an explicit expression for the hit probability for each item in the RND-TTL cache.

Proposition 4.2 (Item's hit probability). *Under Assumption 4.1, the hit probability h_n for item n in the RND-TTL cache is given by:*

$$h_n = o_n + (1 - o_n) \cdot (1 - p_n^i) \,, \tag{4.15}$$

where o_n is given in (4.7)*.*

Proof. The result follows from observing that in RND-TTL, whenever an item n is in the cache, an exact hit occurs upon a request for n. Conversely, when n is not in the cache, only an approximate hit may occur, with probability $1 - p_n^i$. For further details, we refer the reader to Appendices B.3 and B.4 page 84.

□

The RND-TTL cache can be seen as a generalization of the TTL cache as the latter can be obtained when two conditions are met: (i) $p_n^i = 1$ for each item n, and (ii) the timer refresh process of each item n coincides with its request process. The hit ratio of the TTL cache can be retrieved from (4.15) and (4.7) by letting $p_n^i = 1$ and $\lambda_n^r = \lambda_n^i = \lambda_n$. Equations (4.3), (4.7) and (4.15) are then all equivalent.

While we have described the RND-TTL cache model using parameters $\mathbf{T}$, $\boldsymbol{\lambda}^{\boldsymbol{r}}$ and $\mathbf{p^i}$, in what follows, it will be more convenient to retain as parameters $\mathbf{T}$, $\boldsymbol{\lambda}^{\boldsymbol{r}}$, and $\boldsymbol{\lambda}^{\boldsymbol{i}} = (\lambda_n^i)_{n\in\mathcal{I}}$ (see (4.8)).

4.4.2 Relation Between RND-LRU and RND-TTL

Using the RND-TTL cache to estimate the hit ratio of RND-LRU is analogous to using the TTL cache to approximate the hit ratio of LRU. Both RND-TTL and TTL enable the decoupling of caching decisions across items, with the goal of capturing the behavior of an item n in terms of its insertion and eviction from the cache, independently of other items. We revisit the concepts of timer expiration, insertion policy, and timer re-initialization in the TTL cache and the RND-TTL cache and establish their connection to the caching decisions of LRU and RND-LRU, respectively.

Timer expiration. In both RND-LRU and LRU, an item is evicted from the cache when it is no longer among the C recently used items. This behavior is captured and represented in RND-TTL and TTL caches by assigning a timer with a duration of T_n to each cached item n. An item is then evicted upon expiration of its timer.

Insertion in the cache. In LRU/TTL cache, a non cached item n is always inserted into the cache when it is requested. It follows that the insertion rate for n, when it is not cached, is equal to its request rate λ_n for both LRU and TTL. However, in RND-LRU, this is not the case as a non-cached item n can be served by a similar item already in the cache. As a result, when n is not in the cache, the **insertion rate** for item n in RND-LRU is smaller or equal to λ_n. In RND-TTL, the parameter λ_n^i serves as the insertion rate for item n when it is not cached, allowing RND-TTL to capture the insertion behavior of item n in the RND-LRU cache by tuning λ_n^i accordingly.

Timer re-initialization. In LRU, when a cached item n receives a request, it is refreshed by being moved to the front of the list. This behavior is captured in TTL by re-initializing the timer for item n. It follows that the **refresh rate** for n, when it is in the cache, is equal to λ_n for both LRU and TTL. However, in the case of RND-LRU, the refresh process is not solely based on its own request. Item n might also be refreshed when its neighboring items receive requests. As a result, in RND-LRU, when n is cached, the **refresh rate** of n is greater than or equal to λ_n. In RND-TTL, the parameter λ_n^r determines the rate at which n's timer is re-initialized when n is cached. By appropriately adjusting λ_n^r, RND-TTL can capture the refresh operation of an item in RND-LRU.

In the next section, we examine the insertion rate and refresh rate of an item in RND-LRU in detail, which allows us to derive guidelines on how to constrain the parameters $\boldsymbol{\lambda}^{\boldsymbol{i}}$, $\boldsymbol{\lambda}^{\boldsymbol{r}}$, and $\mathbf{T}$ for the RND-TTL approximation.

4.4.3 RND-TTL Approximation to RND-LRU

We propose an extension of the TTL approximation, named RND-TTL approximation, for estimating the hit ratio of RND-LRU under IRM. Recall that the TTL approximation uses the hit ratio of a TTL cache, with specific constraints on its parameters, as an approximation for LRU's hit ratio. We highlight that this approximation is asymptotically accurate [Fag77; FRR12; LT15; JNT18]. The RND-TTL approximation provides, as estimate for RND-LRU's hit ratio, the one of RND-TTL by constraining specifically the parameters of RND-TTL. The constraints on the timers of RND-TTL and the total occupancy are identical to those made by the TTL approximation regarding the TTL cache. In addition, the RND-TTL approximation introduces constraints related to the insertion and refresh rates, as detailed later on in this section. We next focus on expressing the insertion and refresh rates in RND-LRU.

Recall that $\tilde{\Omega}$ is the set of all possible ordered lists in the RND-LRU cache and Ω is the set of all possible sets of cached items in RND-TTL. We define the sets $\tilde{B}_n$ and B_n representing the lists and sets of cached items where none of the neighbors of item n is cached. Formally,

$$\tilde{B}_n \triangleq \left\{L \in \tilde{\Omega} :\ L \cap \mathcal{N}^c[n] = \emptyset\right\} \text{ and } B_n \triangleq \{S \in \Omega :\ S \cap \mathcal{N}^c[n] = \emptyset\}\,. \tag{4.16}$$

Additionally, $\forall m \in \mathcal{N}^c[n]$, we define the sets $\tilde{B}_{n,m}$ and $B_{n,m}$ representing the lists and sets of cached items where m is the closest neighbor of n in the cache. Specifically,

$$\tilde{B}_{n,m} \triangleq \left\{L \in \tilde{\Omega} :\ m \in L, L \cap \mathcal{N}_m^c[n] = \emptyset\right\}, \tag{4.17}$$

$$B_{n,m} \triangleq \{S \in \Omega :\ m \in S, S \cap \mathcal{N}_m^c[n] = \emptyset\}\,. \tag{4.18}$$

Proposition 4.3 (RND-LRU insertion rate). *Under Assumptions 4.1, 4.2 and 4.3, the insertion rate of item n in RND-LRU, $\tilde{\lambda}_n^i$, is expressed as:*

$$\tilde{\lambda}_n^i = \tilde{f}_{n,\mathbf{q}}^i\left(\tilde{\boldsymbol{\pi}}\right) \triangleq \lambda_n \left(\sum_{L \in \tilde{B}_n} \tilde{\pi}_L + \sum_{m \in \mathcal{N}(n)} (1 - q_m(n)) \sum_{K \in \tilde{B}_{n,m}} \tilde{\pi}_K\right), \tag{4.19}$$

where $\tilde{\boldsymbol{\pi}}$ is the limiting distribution of $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$.

Proof. The result follows from observing that in RND-LRU, when a request for an item n finds $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$ in state $\tilde{B}_n$, n is inserted in the cache with probability 1. On the other hand, if a request for n finds $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$ in state $\tilde{B}_{n,m}$ for any $m \in \mathcal{N}(n)$, n is inserted in the cache with probability $1 - q_m(n)$. For additional details, we refer the reader to Appendices B.3 and B.5 page 84.

□

Proposition 4.4 (RND-LRU refresh rate). *Under Assumptions 4.1, 4.2 and 4.3, the refresh rate of item n in RND-LRU, $\tilde{\lambda}_n^r$, is expressed as:*

$$\tilde{\lambda}_n^r = \tilde{f}_{n,\mathbf{q}}^r\left(\tilde{\boldsymbol{\pi}}\right) \triangleq \sum_{m \in \mathcal{N}^c[n]} q_n(m)\lambda_m \sum_{L \in \tilde{B}_{m,n}} \tilde{\pi}_L\,, \tag{4.20}$$

where $\tilde{\boldsymbol{\pi}}$ is the limiting distribution of $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$.

Proof. The result follows from observing that in RND-LRU, when a request for an item m finds $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$ in state $\tilde{B}_{m,n}$, cached item n is refreshed, i.e., moved to the front of the list, with probability $q_n(m)$. For additional details, we refer the reader to Appendices B.3 and B.6 page 85.

□

	RND-LRU	**RND-TTL**
Parameters	$\mathbf{q} = (q_m(n))_{n,m\in\mathcal{I}^2}$	$\boldsymbol{\lambda}^i, \boldsymbol{\lambda}^r,\ \mathbf{T}$
Content Occupancy	Obtained via simulations	o_n (see (4.7))
Total Occupancy	C, exact	$\sum_{n\in\mathcal{I}} o_n$, in expectation
Limiting Distribution	$\tilde{\boldsymbol{\pi}} = (\tilde{\pi}_L)_{L\in\tilde{\Omega}}$, obtained via simulations	$\boldsymbol{\pi} = (\pi_S)_{S\in\Omega}$ (see (4.14))
Hit Probability	Obtained via simulations	h_n (see (4.15))
Insertion Rate	$\tilde{f}^i_{n,\mathbf{q}}(\tilde{\boldsymbol{\pi}})$ (see (4.19))	$\lambda^i_n(1-o_n)$
Refresh Rate	$\tilde{f}^r_{n,\mathbf{q}}(\tilde{\boldsymbol{\pi}})$ (see (4.20))	$\lambda^r_n o_n$
Constraints on RND-TTL Parameters to Approximate RND-LRU		
Timers	$T_n = T, \forall n \in \mathcal{I}$	
Total Occupancy	Expected number of cached items $= C$ (see (4.22))	
Insertion Rate	$\lambda^i_n(1-o_n) = f^i_{n,\mathbf{q}}(\boldsymbol{\pi})$ (see (4.23))	
Refresh Rate	$\lambda^r_n o_n = f^r_{n,\mathbf{q}}(\boldsymbol{\pi})$ (see (4.24))	

Table 4.2: RND-TTL approximation

To recapitulate our derivations and ease the comparison between RND-TTL and RND-LRU, we depict in Table 4.2 the notation, the main metrics and their expressions (when available) for both policies.

Constraints on RND-TTL cache's parameters. Table 4.2 summarizes the RND-TTL approximation. First, it imposes on the RND-TTL cache's parameters the TTL approximation's constraints:

All items' timers share the same duration, which we denote as T, i.e.,

$$T_n = T,\ \forall n \in \mathcal{I}. \tag{4.21}$$

The expected number of cached items in the RND-TTL cache equals C,

$$\sum_{n\in\mathcal{I}} \left(\frac{1}{\lambda^i_n} \cdot \frac{\lambda^r_n}{\mathrm{e}^{\lambda^r_n T_n} - 1} + 1 \right)^{-1} = C. \tag{4.22}$$

Note that the term corresponding to item n in the sum in (4.22) is the limiting probability that n is cached (see (4.12) and Proposition 4.1).

Second, it constrains the (unconditional) insertion and refresh rates under RND-TTL, namely $\lambda^i_n(1-o_n)$ and $\lambda^r_n o_n$, to satisfy expressions similar to those verified by the same rates under RND-LRU, which are given by Propositions 4.3 and 4.4:

$$\lambda_n^i(1-o_n) = f_{n,\mathbf{q}}^i(\boldsymbol{\pi}) \triangleq \lambda_n \left(\sum_{S\in B_n} \pi_S + \sum_{m\in\mathcal{N}(n)} (1-q_m(n)) \sum_{K\in B_{n,m}} \pi_K \right), \tag{4.23}$$

$$\lambda_n^r o_n = f_{n,\mathbf{q}}^r(\boldsymbol{\pi}) \triangleq \sum_{m\in\mathcal{N}^c[n]} q_n(m)\lambda_m \sum_{S\in B_{m,n}} \pi_S\,, \tag{4.24}$$

where $\boldsymbol{\pi}$ is the limiting distribution for the set of cached items in the RND-TTL cache, computed in (4.14), and $f_{n,\mathbf{q}}^i(\boldsymbol{\pi})$ and $f_{n,\mathbf{q}}^r(\boldsymbol{\pi})$ have been defined to mimic $\tilde{f}_{n,\mathbf{q}}^i(\tilde{\boldsymbol{\pi}})$ and $\tilde{f}_{n,\mathbf{q}}^r(\tilde{\boldsymbol{\pi}})$, respectively given in (4.19) and (4.20). Substituting $\boldsymbol{\pi}$ from (4.14) into (4.23) and (4.24), we obtain the following expressions:

$$\lambda_n^i = \lambda_n \left(\prod_{m\in\mathcal{N}(n)} (1-o_m) + \sum_{m\in\mathcal{N}(n)} (1-q_m(n))\, o_m \prod_{l\in\mathcal{N}_m(n)} (1-o_l) \right), \tag{4.25}$$

$$\lambda_n^r = \lambda_n + \sum_{m\in\mathcal{N}(n)} q_n(m)\lambda_m \prod_{l\in\mathcal{N}_n^c[m]} (1-o_l)\,. \tag{4.26}$$

RND-LRU's hit ratio approximation. If we can compute values for $\boldsymbol{\lambda}^r$, $\boldsymbol{\lambda}^i$ and T verifying the constraints (4.21), (4.22), (4.25) and (4.26), then using Proposition 4.2, (4.8) and (4.25), we can estimate the hit ratio of RND-LRU as:

$$H \approx \sum_{n\in\mathcal{I}} \lambda_n \cdot h_n, \tag{4.27}$$

where

$$h_n = o_n + \sum_{m\in\mathcal{N}(n)} q_m(n) \cdot o_m \prod_{l\in\mathcal{N}_m^c[n]} (1-o_l). \tag{4.28}$$

Remark 4.1. *In Section 4.4.1.1, we introduced R-TTL as a natural extension of a TTL cache to model RND-LRU behavior. Recognizing the challenge of applying the TTL approximation to R-TTL, we proposed the analytically tractable RND-TTL model. Under Assumption 4.1 and assuming a limiting distribution $\boldsymbol{\mu}$ for the set of cached items in R-TTL, it is noteworthy that, by following steps similar to those used in proving Propositions 4.3 and 4.4, we can establish the insertion rate and refresh rate of an item n in R-TTL as $f_{n,\mathbf{q}}^i(\boldsymbol{\mu})$ and $f_{n,\mathbf{q}}^r(\boldsymbol{\mu})$, respectively. This implies that the constraints imposed by the RND-TTL approximation on the RND-TTL cache are properties verified for R-TTL.*

In the following section, we present an iterative algorithm that enables a numerical determination of the parameters $\boldsymbol{\lambda}^r$, $\boldsymbol{\lambda}^i$, and T based on the aforementioned description.

4.5 Algorithm for Finding Approximate Hit Probabilities

Let $\mathbf{o} = (o_n)_{n\in\mathcal{I}}$ be the vector representing the occupancies in the RND-TTL cache. By defining a function g as:

$$g(x_1,x_2,x_3) \triangleq \left(\frac{1}{x_2} \cdot \frac{x_1}{\mathrm{e}^{x_1x_3}-1} + 1 \right)^{-1}, \tag{4.29}$$

we can rewrite (4.7) as:

$$o_n = g(\lambda_n^r, \lambda_n^i, T)\ . \tag{4.30}$$

The RND-TTL approximation suggests to choose the parameters $\boldsymbol{\lambda}^{\boldsymbol{r}}$, $\boldsymbol{\lambda}^{\boldsymbol{i}}$ and T for the RND-TTL cache such that the following set of equations is verified:

$$\boldsymbol{\lambda}^{\boldsymbol{i}} = \mathbf{E}(\mathbf{o}) = (E_n(\mathbf{o}))_{n\in\mathcal{I}}, \tag{4.31}$$

$$\boldsymbol{\lambda}^{\boldsymbol{r}} = \mathbf{R}(\mathbf{o}) = (R_n(\mathbf{o}))_{n\in\mathcal{I}}, \tag{4.32}$$

$$\mathbf{o} = \mathbf{g}(\boldsymbol{\lambda}^{\boldsymbol{r}}, \boldsymbol{\lambda}^{\boldsymbol{i}}, T) = (g(\lambda_n^r, \lambda_n^i, T))_{n\in\mathcal{I}}, \tag{4.33}$$

$$T \in T_C(\boldsymbol{\lambda}^{\boldsymbol{r}}, \boldsymbol{\lambda}^{\boldsymbol{i}}) \iff \sum_{n\in\mathcal{I}} g(\lambda_n^r, \lambda_n^i, T) = C, \tag{4.34}$$

where $E_n(\mathbf{o})$ and $R_n(\mathbf{o})$ are functions respectively obtained from (4.25) and (4.26), $g(\lambda_n^r, \lambda_n^i, T)$ is obtained from (4.29), and $T_C(\boldsymbol{\lambda}^{\boldsymbol{r}}, \boldsymbol{\lambda}^{\boldsymbol{i}})$ is defined as the set of values of the shared timer value T guaranteeing that, for given $\boldsymbol{\lambda}^{\boldsymbol{r}}$ and $\boldsymbol{\lambda}^{\boldsymbol{i}}$, the expected number of cached items in RND-TTL equals C.

Combining (4.31)-(4.34), we obtain a system of $3N+1$ equations in $3N+1$ unknowns (recall that $|\mathcal{I}| = N$), from which we can obtain in particular the occupancies. Once the occupancies are known, we can compute the vector of hit probabilities, $\mathbf{h} = (h_n)_{n\in\mathcal{I}}$, according to (4.28) and estimate the hit ratio of RND-LRU according to (4.27).

In Section 4.5.1, we establish a sufficient condition for the existence of a solution for the system of equations (4.31)-(4.34). In Section 4.5.2, we propose an iterative algorithm, with a parameter $\beta \in [0, 1)$, for numerically finding a solution for the aforementioned system of equations. We also prove the convergence of this algorithm under specific conditions. Section 4.5.3 outlines a practical method for tuning the algorithm's parameter β.

4.5.1 Fixed Point Equations

We denote the set $T_C(\mathbf{R}(\mathbf{o}), \mathbf{E}(\mathbf{o}))$ (see (4.34)) as $T_C(\mathbf{o})$. We denote the capped simplex as Δ_C such that:

$$\Delta_C \triangleq \left\{ \mathbf{o} \in \mathbb{R}^N :\ 0 \le o_n \le 1,\ \sum_{n\in\mathcal{I}} o_n = C \right\}. \tag{4.35}$$

Lemma 4.1 ($T_C(\mathbf{o})$ **is a singleton).** *If the "no-coverage" condition:*

$$\textit{(no-coverage condition)} \qquad \forall \mathcal{M} \subset \mathcal{I} : |\mathcal{M}| \le C, \quad \left| \bigcup_{n\in\mathcal{M}} \mathcal{N}(n) \right| < N - C, \tag{4.36}$$

is satisfied, then $T_C(\mathbf{o})$ has a unique element for every $\mathbf{o} \in \Delta_C$. In other words:

$$\forall \mathbf{o} \in \Delta_C, \exists! T_0 \in \mathbb{R}^+ : \quad F(\mathbf{o}, T_0) = 0, \tag{4.37}$$

$$F(\mathbf{o}, T) \triangleq \sum_{n\in\mathcal{I}} g(R_n(\mathbf{o}), E_n(\mathbf{o}), T) - C, \tag{4.38}$$

where the symbol $\exists!$ refers to unique existence.

Proof. See Appendix B.7 page 86. □

A note on (4.36): the right-hand side of the inequality is the number of non-cached items. If the items in $\mathcal{M}$ are cached, then the left-hand side of the inequality refers to the number of items that are "covered" by the cache as these are neighbors of the cached items, so their requests could be served by the cache. Having (4.36) satisfied implies that, whichever C (or less) items are cached, there will always be at least one non-cached item that is not covered by the cache, hence the use of "no-coverage" to name the condition (4.36).

In practice, the catalog size is much larger than the cache capacity, and requests for many items miss the cache and are directed to the original server. In other words, the no-coverage condition is often satisfied in practical scenarios.

Remark 4.2. *If* (4.36) *is not met for some set $\mathcal{M}$, then Lemma 4.1 does not hold. But this also means that it suffices to store the items in $\mathcal{M}$ in the cache to enable the cache to cover at least $N - C$ items. (Possibly the entire catalog can be served by the cache if no pair of items in $\mathcal{M}$ are neighbors).*

From now on, we assume that the no-coverage condition in (4.36) is verified. Therefore, thanks to Lemma 4.1, $T_C(\mathbf{o})$ is a singleton and we can define a function t_C from Δ_C to $\mathbb{R}^+$, where for each $\mathbf{o}$, it associates the unique element in $T_C(\mathbf{o})$, i.e.,

$$t_C(\mathbf{o}) = T \iff F(\mathbf{o}, T) = 0. \tag{4.39}$$

We also introduce the function $\mathbf{G}$ from Δ_C to Δ_C defined as:

$$\forall \mathbf{o} \in \Delta_C,\ \mathbf{G}(\mathbf{o}) \triangleq \mathbf{g}\left(\mathbf{R}(\mathbf{o}), \mathbf{E}(\mathbf{o}), t_C(\mathbf{o})\right) \tag{4.40a}$$

$$= \left(g(R_1(\mathbf{o}), E_1(\mathbf{o}), t_C(\mathbf{o})), \cdots, g(R_N(\mathbf{o}), E_N(\mathbf{o}), t_C(\mathbf{o}))\right), \tag{4.40b}$$

where g is defined in (4.29). It follows that finding a solution for the system of equations (4.31)-(4.34) boils down to finding a fixed point of $\mathbf{G}$ within Δ_C.

For any sets A and B, we denote the set of functions from A to B that are continuously differentiable as $\mathcal{C}^1(A \to B)$.

Lemma 4.2 (**Differentiability of t_C**). *The function t_C is continuously differentiable within the set Δ_C, i.e., $t_C \in \mathcal{C}^1(\Delta_C \to \mathbb{R}^+)$. The gradient of t_C can be expressed as:*

$$\forall j \in \mathcal{I}, \quad \frac{\partial t_C}{\partial o_j}(\mathbf{o}) = -\frac{\partial F}{\partial o_j}(\mathbf{o}, t_C(\mathbf{o})) \cdot \left(\frac{\partial F}{\partial T}(\mathbf{o}, t_C(\mathbf{o}))\right)^{-1}, \tag{4.41}$$

where F has been defined in (4.38).

Proof. See Appendix B.8 page 87. □

Proposition 4.5 (**Fixed point existence**). *The function $\mathbf{G}$ is continuously differentiable within Δ_C and it has at least one fixed point in Δ_C.*

Algorithm 3: Fixed point method

Input: C, λ, $\mathrm{dis}(.,.)$, d, $(q_n(i))_{(n,i)\in I^2}$, β, stopping condition
Output: Estimation of $\mathbf{o}, \mathbf{h}, t_C$

Initialization:

Obtain $t_C(0)$ such that $\sum_{n\in I}\left(1-\mathrm{e}^{-\lambda_n \cdot t_C(0)}\right) = C$
$\mathbf{o}(0) \leftarrow 1-\mathrm{e}^{-\lambda \cdot t_C(0)}$
$\mathbf{h}(0) \leftarrow f^h(\mathbf{o}(0))$
$j \leftarrow 1$
while *Stopping condition not satisfied* **do**
  $\boldsymbol{\lambda^i}(j) \leftarrow \mathbf{E}(\mathbf{o}(j-1))$ (see (4.31))
  $\boldsymbol{\lambda^r}(j) \leftarrow \mathbf{R}(\mathbf{o}(j-1))$ (see (4.32))
  Obtain $t_C(j)$ such that : $\sum_{n\in I}(\mathbf{g}(\boldsymbol{\lambda^r}(j), \boldsymbol{\lambda^i}(j), t_C(j)))_n = C$ (see (4.34),(4.33))
  $\mathbf{o}(j) \leftarrow (1-\beta)\cdot \mathbf{g}(\boldsymbol{\lambda^r}(j), \boldsymbol{\lambda^i}(j), t_C(j)) + \beta \cdot \mathbf{o}(j-1)$
  $\mathbf{h}(j) = f^h(\mathbf{o}(j))$ (see (4.28))
  $j \leftarrow j+1$
end while
return $\mathbf{h}(j),\ \mathbf{o}(j),\ t_C(j)$

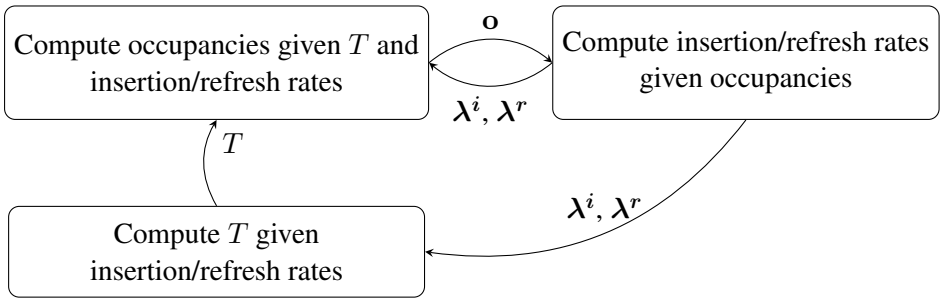


Figure 4.1: Essence of the fixed point algorithm.

Proof. It is evident that the functions $\mathbf{g}(\cdot,\cdot,\cdot)$, $\mathbf{R}(\cdot)$, and $\mathbf{E}(\cdot)$ are continuously differentiable within $(\mathbb{R}^+)^N \cdot (\mathbb{R}^+)^N \cdot \mathbb{R}^+$, Δ_C and Δ_C, respectively. Furthermore, according to Lemma 4.2, we have that $t_C \in \mathcal{C}^1(\Delta_C \to \mathbb{R}^+)$. As a result, we conclude that $\mathbf{G} \in \mathcal{C}^1(\Delta_C \to \Delta_C)$. Noting that Δ_C is a non empty compact convex set, Brouwer's fixed point theorem [Par99] implies the existence of a fixed point for $\mathbf{G}$.

□

Proposition 4.5 indicates that when the no-coverage condition (4.36) is satisfied, it is possible to find parameters for the RND-TTL cache model that verify the system of equations (4.31)-(4.34). Therefore, we can apply the RND-TTL approximation to estimate the hit ratio of RND-LRU.

4.5.2 Fixed Point Algorithm

We recall that solving the system of equations (4.31)-(4.34) reduces to solving a fixed point equation for the function $\mathbf{G}$ defined in (4.40a). A natural approach to finding a fixed point of $\mathbf{G}$ is through an iterative method. This is illustrated in Figure 4.1. Starting with an initial guess $\mathbf{o}(0)$, we perform iterations of the form $\mathbf{o}(j+1) = \beta\mathbf{o}(j) + (1-\beta)\mathbf{G}(\mathbf{o}(j))$, where $\beta \in [0,1)$ [Man53]. A detailed version of these iterations is presented in Algorithm 3. Initially, we guess the occupancies $\mathbf{o}$, using LRU occupancies as a starting point. Specifically, we set $\mathbf{o}(0) = 1 - \mathrm{e}^{-\lambda t_C(0)}$, where $t_C(0)$ satisfies (4.4) and $\sum_{n\in\mathcal{I}} o_n(0) = C$ (lines 1–2). Then, we compute $\boldsymbol{\lambda}^{\boldsymbol{i}}(1)$ and $\boldsymbol{\lambda}^{\boldsymbol{r}}(1)$ using (4.31) and (4.32), respectively (lines 5–7). Given $\boldsymbol{\lambda}^{\boldsymbol{i}}(1)$ and $\boldsymbol{\lambda}^{\boldsymbol{r}}(1)$, $t_C(1)$ is the unique solution of (4.34) (see (4.37)). This solution can be obtained using either the bisection or Newton's method. Next, we calculate the new estimate of the occupancies $\mathbf{o}(1)$ (line 9).

The same procedure is repeated for subsequent iterations until a stopping condition is met. This condition could be, for example, the difference between the occupancies computed at consecutive iterations becoming smaller than a given threshold, or reaching the maximum number of iterations ($j \leq n_{\text{iterations}}$).

Note that in Figure 4.1 the boxes on the left-hand side, together with their inputs (insertion and refresh rates), represent the conventional perspective on caching. This involves using fixed rates, and computing item occupancies to estimate hit probabilities. Under a TTL-based model, it also involves computing the characteristic time to approximate LRU, so that the sum of expected occupancies equals the cache capacity C. In contrast, the box on the right-hand side takes into consideration the unique nature of similarity caches. In similarity caches, the insertion and refresh rates are influenced by the currently cached items, and these rates are determined as a function of the occupancies.

For a given value of β, the iterations of Algorithm 3 are of the form: $\mathbf{o}(j+1) = \mathbf{G}_\beta(\mathbf{o}(j))$ such that:

$$\mathbf{G}_\beta(\mathbf{o}) \triangleq (1-\beta)\mathbf{G}(\mathbf{o}) + \beta\mathbf{o}\,. \tag{4.42}$$

The function $\mathbf{G}_\beta$ is continuously differentiable thanks to Proposition 4.5. We denote its Jacobian matrix as $\boldsymbol{J}_{\mathbf{G}_\beta}$. We further define for an operator norm $\|\cdot\|$ the constant μ_β as:

$$\mu_\beta \triangleq \sup_{\mathbf{o}\in\Delta_C} \|\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o})\|. \tag{4.43}$$

Proposition 4.6 (Fixed point uniqueness and convergence). *If*

$$\exists\beta \in [0,1) :\ \mu_\beta < 1, \tag{4.44}$$

then $\mathbf{G}$ has a unique fixed point in Δ_C and Algorithm 3 with parameter β converges to this unique fixed point that we denote as $\mathbf{o}^*$. Moreover, if $\mathbf{o}(j)$ is the estimation of $\mathbf{o}^*$ at iteration j in Algorithm 3, then we have:

$$\|\mathbf{o}(j) - \mathbf{o}^*\| \leq (\mu_\beta)^j \cdot \sup_{\mathbf{x},\mathbf{y}\in\Delta_C} \|\mathbf{x}-\mathbf{y}\|,\ \forall j \in \mathbb{N}, \tag{4.45}$$

where Δ_C is defined in (4.35)*.*

Proof. Under (4.36), Proposition 4.5 implies that $\mathbf{G}_\beta \in \mathcal{C}^1(\Delta_C \to \Delta_C)$, and we deduce that $\mathbf{G}_\beta$ is Lipshitz with constant μ_β [Wea18], i.e., $\|\mathbf{G}_\beta(\mathbf{x}) - \mathbf{G}_\beta(\mathbf{y})\| \leq \mu_\beta \|\mathbf{x} - \mathbf{y}\|$ for any $\mathbf{x},\ \mathbf{y}$. Leveraging Banach Fixed Point Theorem [MV97], we deduce that (i) $\mathbf{G}_\beta$ has a unique fixed point denoted as $\mathbf{o}^{*,\beta}$, (ii) Algorithm 3 with parameter β converges to $\mathbf{o}^{*,\beta}$, and (iii) the distance between $\mathbf{o}(j)$ and $\mathbf{o}^{*,\beta}$ satisfies

$$\|\mathbf{o}(j) - \mathbf{o}^{*,\beta}\| \leq (\mu_\beta)^j \cdot \sup_{\mathbf{x},\mathbf{y} \in \Delta_C} \|\mathbf{x} - \mathbf{y}\|. \tag{4.46}$$

Notice that for any β, the respective sets of fixed points of $\mathbf{G}$ and $\mathbf{G}_\beta$ coincide. Therefore, $\mathbf{o}^{*,\beta} = \mathbf{o}^*$ for any β, which concludes the proof.

□

In practice, one can compute the norm of the matrix $\boldsymbol{J}_{\mathbf{G}_\beta}$ for few vectors $\mathbf{o} \in \Delta_C$ to get an idea of the satisfaction of the sufficient condition in (4.44) and then on the convergence of Algorithm 3 with parameter β to a unique fixed point. In the next proposition, we give an explicit formula for $\boldsymbol{J}_{\mathbf{G}_\beta}$ to ease its computation.

We denote by $\mathrm{Diag}(\mathbf{x})$ an N-dimensional diagonal matrix, where the entries of the vector $\mathbf{x}$ are positioned along its diagonal, and by $\boldsymbol{I}_N$ the N-dimensional identity matrix.

Proposition 4.7 (Computation of $\boldsymbol{J}_{\mathbf{G}_\beta}$). *The Jacobian matrix $\boldsymbol{J}_{\mathbf{G}_\beta}$ has the following expression:*

$$\forall \mathbf{o} \in \Delta_C, \quad \boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o}) = (1-\beta)\boldsymbol{J}_{\mathbf{G}}(\mathbf{o}) + \beta\, \boldsymbol{I}_N, \tag{4.47}$$

where

$$\begin{aligned} \boldsymbol{J}_{\mathbf{G}}(\mathbf{o}) = \mathrm{Diag}\left(\boldsymbol{\partial_1 g}\right) \cdot \boldsymbol{J}_{\mathbf{R}}(\mathbf{o}) + \mathrm{Diag}\left(\boldsymbol{\partial_2 g}\right) \cdot \boldsymbol{J}_{\mathbf{E}}(\mathbf{o}) \\ - \frac{1}{\boldsymbol{\partial_3 g} \cdot \mathbf{1}} \boldsymbol{\partial_3 g}^\intercal \cdot \left(\boldsymbol{\partial_1 g} \cdot \boldsymbol{J}_{\mathbf{R}}(\mathbf{o}) + \boldsymbol{\partial_2 g} \cdot \boldsymbol{J}_{\mathbf{E}}(\mathbf{o})\right), \end{aligned} \tag{4.48}$$

with

$$\boldsymbol{\partial_j g} = \left(\frac{\partial g}{\partial x_j}(R_n(\mathbf{o}), E_n(\mathbf{o}), t_C(\mathbf{o})) \right)_{n \in \mathcal{I}}, \qquad \textit{for } j \in \{1,2,3\}, \tag{4.49}$$

$\boldsymbol{J}_{\mathbf{R}}$ and $\boldsymbol{J}_{\mathbf{E}}$ referring to the Jacobian matrices of the functions $\mathbf{R}$ and $\mathbf{E}$, respectively, $\mathbf{1}$ denoting the N-dimensional column vector with all components equal to 1, and g defined in (4.29)*.*

Proof. See Appendix B.9 page 87.

□

Time Complexity of Algorithm 3. Let D be the maximum number of neighbors for any item in $\mathcal{I}$ plus 1, more precisely,

$$D \triangleq \max_{n \in \mathcal{I}} |\mathcal{N}^c[n]|. \tag{4.50}$$

It is convenient to define $\mathcal{K}$ as

$$\mathcal{K} \triangleq \sum_{n \in \mathcal{I}} \sum_{m \in \mathcal{N}^c[n]} |\mathcal{N}^c[m]|. \tag{4.51}$$

We show in B.10 that the time complexity of one iteration of Algorithm 3 is $\mathcal{O}(\mathcal{K})$. On the other hand, the TTL approximation for computing the hit ratio of LRU consists of a single iteration whose time complexity is $\mathcal{O}(N)$, where $N = |\mathcal{I}|$. We note that $N \leq \mathcal{K} \leq N^3$.

We use Proposition 4.6 to bound the number of iterations of Algorithm 3. When the condition (4.44) is verified, Proposition 4.6 guarantees the convergence of Algorithm 3 to a unique solution and allows computing the number of iterations l in the algorithm to ensure that the vector of occupancies in the iteration number l, $\mathbf{o}(l)$, is within a distance ϵ from the fixed point $\mathbf{o}^*$, i.e., $\|\mathbf{o}(l) - \mathbf{o}^*\| \leq \epsilon$ for any norm $\|\cdot\|$ in $\mathbb{R}^N$. In particular, for norm 2, the diameter of Δ_C is $\sqrt{2C}$ and thanks to Proposition 4.6, we obtain that the number of iterations of Algorithm 3 is upper bounded by $\frac{\epsilon}{\ln(1/\mu_\beta)}\sqrt{2C}$, with μ_β defined in (4.43) and satisfying (4.44). We deduce that the time complexity of Algorithm 3 is $\mathcal{O}\left(\frac{\epsilon\sqrt{2C}}{\ln(1/\mu_\beta)} \cdot \mathcal{K}\right)$. Observing that $\mathcal{K} \leq ND^2$, the algorithm's time complexity is also $\mathcal{O}\left(\frac{\epsilon\sqrt{2C}}{\ln(1/\mu_\beta)} \cdot ND^2\right)$.

4.5.3 Choice of β

Our approach for selecting the value of β for Algorithm 3 is based on Proposition 4.6. Let $Y(\mathbf{o})$ be the set of values of β in $[0, 1)$ for which the spectral norm of $\boldsymbol{J}_{\mathbf{G}_\beta}$ is smaller than 1, i.e.,

$$Y(\mathbf{o}) \triangleq \left\{\beta \in [0,1) : \ \|\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o})\|_2 < 1\right\}. \tag{4.52}$$

Equation (4.44) in Proposition 4.6 is equivalent to the set $\bigcap_{\mathbf{o}\in\Delta_C} Y(\mathbf{o})$ being non-empty. In other words, choosing the parameter β of Algorithm 3 from the set $\bigcap_{\mathbf{o}\in\Delta_C} Y(\mathbf{o})$ guarantees the convergence of Algorithm 3 to the unique fixed point of $\mathbf{G}$.

We stress that the characterization of the sets $Y(\mathbf{o})$ and $\bigcap_{\mathbf{o}\in\Delta_C} Y(\mathbf{o})$ is difficult. For this reason, we proceed with a randomized approach. First, we randomly sample f vectors from Δ_C, $(\mathbf{o}_{(j)})_{1\leq j\leq f}$. Then, for each sampled vector $\mathbf{o}_{(j)}$, we compute a subset of values of β leading to $\|\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o}_{(j)})\|_2 < 1$. We denote the considered subset of $Y(\mathbf{o}_{(j)})$ as $\tilde{Y}(\mathbf{o}_{(j)})$, where $\tilde{Y}(\mathbf{o}_{(j)}) \subset Y(\mathbf{o}_{(j)})$. Finally, we take the intersection $\bigcap_{j=1}^{f} \tilde{Y}(\mathbf{o}_{(j)})$ as a set of candidate values for β. When we use a larger number of sampled vectors, f, the likelihood that the values of $\beta \in \bigcap_{j=1}^{f} \tilde{Y}(\mathbf{o}_{(j)})$ satisfy the condition in (4.44) increases. However, this also comes with the drawback of higher computational costs.

In the next proposition, we compute the aforementioned subset of $Y(\mathbf{o})$, $\tilde{Y}(\mathbf{o})$, based on the input vector $\mathbf{o}$. To this aim, we leverage the spectral radius of the Jacobian. Recall that the spectral radius of a matrix is defined as the maximum absolute value of its eigenvalues. We denote the spectral radius of matrix $\boldsymbol{M}$ by $\rho(\boldsymbol{M})$, and its spectral norm by $\|\boldsymbol{M}\|_2 = \sqrt{\rho(\boldsymbol{M}\boldsymbol{M}^\intercal)}$.

Proposition 4.8 (Properties of $Y(\mathbf{o})$). *Let γ be the squared spectral norm of the Jacobian matrix $\boldsymbol{J}_{\mathbf{G}}(\mathbf{o})$ and let η be the spectral radius of the matrix $\boldsymbol{J}_{\mathbf{G}}(\mathbf{o}) + \boldsymbol{J}_{\mathbf{G}}(\mathbf{o})^\intercal$,*

$$\gamma = \left(\|\boldsymbol{J}_{\mathbf{G}}(\mathbf{o})\|_2\right)^2 = \rho(\boldsymbol{J}_{\mathbf{G}}(\mathbf{o})\boldsymbol{J}_{\mathbf{G}}(\mathbf{o})^\intercal) \tag{4.53}$$

$$\eta = \rho(\boldsymbol{J}_{\mathbf{G}}(\mathbf{o}) + \boldsymbol{J}_{\mathbf{G}}(\mathbf{o})^\intercal). \tag{4.54}$$

If

$$\eta < \min\{2, \gamma + 1\} \tag{4.55}$$

then $Y(\mathbf{o})$ *satisfies*

$$Y(\mathbf{o}) \supset \tilde{Y}(\mathbf{o}) \tag{4.56}$$

where

$$\tilde{Y}(\mathbf{o}) = \left(\max\left\{0, \frac{\gamma - 1}{\gamma + 1 - \eta}\right\}, 1\right) . \tag{4.57}$$

Proof. See Appendix B.11 page 88. □

Remark 4.3. *If, for a given* $\mathbf{o}$*,* $\boldsymbol{J}_{\mathbf{G}}(\mathbf{o})$ *is antisymmetric, i.e,* $\boldsymbol{J}_{\mathbf{G}}(\mathbf{o}) = -\boldsymbol{J}_{\mathbf{G}}(\mathbf{o})^{\intercal}$*, then* $\eta = 0$ *and* (4.55) *is verified. It follows from Proposition 4.8 that in this case* $\tilde{Y}(\mathbf{o}) = \left(\max\left\{0, \frac{\gamma-1}{\gamma+1}\right\}, 1\right)$.

For each sample vector $\mathbf{o}_{(j)}$ and thus each Jacobian $\boldsymbol{J}_{\mathbf{G}}(\mathbf{o}_{(j)})$, we compute the associated constants γ_j and η_j according to (4.53)-(4.54). We verify next if (4.55) is satisfied for each pair (γ_j, η_j). If this condition holds for all pairs, then using Proposition 4.8 we can determine a subset of $\bigcap_{j=1}^{f} Y(\mathbf{o}_{(j)})$, namely $\bigcap_{j=1}^{f} \tilde{Y}(\mathbf{o}_{(j)})$, and select a value for β from within this subset. Note that as f increases, the more likely it is for the chosen β to meet the criteria of Proposition 4.6 and ensure the convergence of the proposed fixed point algorithm. Note also that if (4.55) does not hold for at least one of the considered pairs (γ_j, η_j), $j = 1, \ldots, f$, we cannot use Proposition 4.8 to analyze the convergence of the proposed fixed-point algorithm using the sufficient conditions established in Proposition 4.6. However, as we will indicate through numerical experiments, the algorithm converges in practice under much broader settings than those considered in Proposition 4.6.

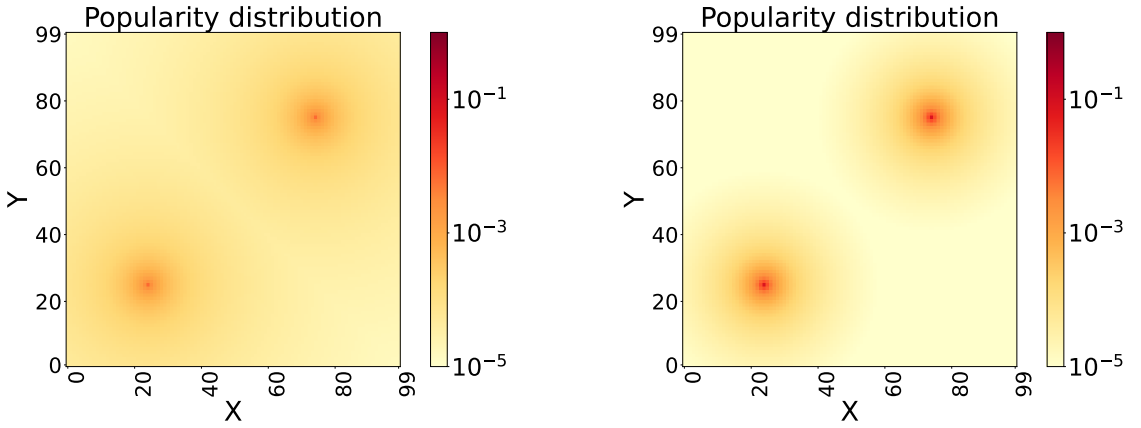
Remark 4.4. *Proposition 4.8 is a general result about fixed point algorithms of the form* (4.42)*. It establishes a closed-form expression for a subset of values for* β*, for which the condition of having the spectral norm of the Jacobian smaller than* 1 *is satisfied.*

4.6 Numerical Evaluation

We assess the accuracy of our proposed RND-TTL approximation method by conducting experiments on both synthetic and real-world traces. The traces are described in Section 4.6.1. To evaluate our method, we compare our approach for estimating the hit ratio of RND-LRU with other alternative solutions discussed in Section 4.6.2. Subsequently, we analyze the approximation accuracy in Section 4.6.3. Our RND-TTL approximation technique involves solving a set of equations using an iterative fixed-point method outlined in Algorithm 3. In Section 4.6.4, we evaluate the convergence of Algorithm 3.

4.6.1 Experimental Setting

We evaluate the efficiency of the proposed fixed point method (Algorithm 3) to predict the hit ratio on synthetic traces and on an Amazon trace [Sab+21].



(a) $\alpha = 1.4$ (b) $\alpha = 2.5$

Figure 4.2: Spatial popularity distribution.

Synthetic traces. For the synthetic traces, each item corresponds to two features, characterized by a point in a grid, $\mathcal{I} = [0..99]^2$ (e.g. Figure 4.2). The total number of items is $|\mathcal{I}| = 10^4$, and the dissimilarity function between items $\mathrm{dis}(\cdot, \cdot)$ is the Euclidean distance. Neighbors of item (x, y) at the same distance are ordered counterclockwise starting from the item to the right, i.e., from $(x + a, y)$ with $a > 0$. The synthetic traces are generated in an IRM fashion [Fag77], where the popularity distribution for an item $n = (x, y)$ is given by

$$p_{(x,y)} \sim \Big(\min \left\{ \mathrm{dis}(n, (24, 24)), \mathrm{dis}(n, (74, 74)) \right\} + 1 \Big)^{-\alpha}, \tag{4.58}$$

where α is a parameter controlling the skewness of the popularity distribution. We generate 50 synthetic streams for $\alpha = 1.4$ and $\alpha = 2.5$ having in each stream $r = 2 \cdot 10^5$ requests for items in $\mathcal{I}$. Figures 4.2a and 4.2b illustrate the popularity distribution in (4.58) for $\alpha \in \{1.4, 2.5\}$.

In addition to the popularity distribution (4.58), we consider also a Zipfian popularity distribution. The corresponding experimental results are presented in B.12.

Real world trace. For the Amazon trace, each item corresponds to an Amazon product. The request trace in [Sab+21] is generated by mapping every Amazon review for the item to an item request. Each item has been mapped to a Euclidean space of dimension 100 using the technique in [McA+15], where the Euclidean distance reflects dissimilarity between two items. Inspired by this methodology, we generate a corresponding IRM stream of requests matching the item popularity in the trace in [Sab+21].

4.6.2 Benchmarks and Alternative Approaches

In what follows, we compare hit ratio estimates provided by RND-LRU using Algorithm 3 with the hit ratio estimations for LRU and for the optimal static allocation. We also propose an alternative approach to estimate RND-LRU's hit ratio.

LRU. The hit ratio and the occupancy for an item n are computed using (4.3) and t_C is deduced using the cache capacity constraint given by (4.4).

Optimal Static Allocation. Under IRM, it is shown in [NGL21] that the maximal hit ratio for a similarity caching policy is achieved by a policy that permanently stores a set S^* of C items such that:

$$S^* \in \underset{S\subset\mathcal{I},|S|=C}{\arg\max} \sum_{n\in\bigcup_{j\in S}\mathcal{N}^c[j]} \lambda_n \,. \tag{4.59}$$

It follows that the hit ratio of a policy that stores S^* is an upper bound on the hit ratio of SIM-LRU and RND-LRU. The maximum hit ratio obtainable by a static allocation under similarity caching can be obtained by solving a maximum weighted coverage problem. We consider, as in SIM-LRU, that each item can be used to satisfy any request for items closer than d. The maximum weighted coverage problem takes as input a capacity C, a set of items $\mathcal{I}$, with $N = |\mathcal{I}|$, their corresponding weights $W = (w_n)_{n\in I}$ and a set of sets $R = \{R_1, \ldots, R_N\}$ such that $R_n \subset \mathcal{I}$. The objective is to find a set $\sigma^* \subset \{1, \ldots, N\}$ such that: $\sigma^* = \arg\max_{\sigma\subset\{1,\ldots,N\}:|\sigma|\leq C} \sum_{n\in\cup_{j\in\sigma}R_j} w_n$.

Finding the best static allocation is equivalent to solving a maximum-weighted coverage problem, with weights $w_n = \lambda_n$ for $n \in \mathcal{I}$, C the cache capacity, and R the set of neighbors for each item, i.e., $R = \{\mathcal{N}^c[n]\}_{n\in\mathcal{I}}$. The maximum weighted coverage problem is known to be NP-hard. In practice, a popular greedy algorithm guarantees a $(1 - 1/e)$ approximation ratio [NWF78; KMN99].

The greedy algorithm operates as follows: initially, it selects the set $R_{c_1} = R^0_{c_1}$ with the largest coverage, where c_1 is determined by $c_1 = \arg\max_{n\in\mathcal{I}} \sum_{m\in R_n} \lambda_m$. Subsequently, the algorithm considers sets $(R^1_n)_{n\in\mathcal{I}}$ defined as $R^1_n = R^0_n \setminus R^0_{c_1}$ in the next step, and it chooses the set R_{c_2} based on $c_2 = \arg\max_{n\in\mathcal{I}} \sum_{m\in R^1_n} \lambda_m$. The same procedure is repeated until C items are collected or all the items are chosen.

LRU with Aggregate Requests. Under SIM-LRU an item is refreshed by requests for all its neighbors. A naive approach to studying a SIM-LRU cache is then to consider that it operates as an LRU cache with request rates for each item equivalent to the sum of the request rates for all items in its neighborhood. One can then use the TTL approximation for LRU, leading to the following formulas:

$$h_n = 1 - \mathrm{e}^{-\sum_{i\in\mathcal{N}^c[n]} \lambda_i t_C}, \quad o_n = h_n. \tag{4.60}$$

We refer to the TTL approximation for LRU simply as LRU, the greedy algorithm as Greedy, and LRU with aggregate requests as LRU-agg.

4.6.3 RND-TTL approximation evaluation

We empirically compute the hit ratio of similarity cache mechanisms using SIM-LRU and RND-LRU on both synthetic and real-world traces described in Section 4.6.1. In the case of synthetic traces, SIM-LRU and RND-LRU are utilized with similarity threshold parameters $d = 1$ and $d = 2$, for request process skewness $\alpha = 2.5$ and $\alpha = 1.4$, respectively. Additionally, given two distinct

Table 4.3: Parameters of the experiments.

Variable	Synthetic traces	Amazon trace
$\mathcal{I}$	$[0..99]^2$	Products
$N = \lvert\mathcal{I}\rvert$	10^4	$\approx 10^4$
λ_n	(4.58)	Empirical
$\mathrm{dis}(\cdot,\cdot)$	Euclidean distance	Euclidean distance
d	1 and 2	300
Number of requests r	$2 \cdot 10^5$	$\approx 10^5$
Number of iterations Alg. 3	25 and 15	40
$q_n(m)$	$(\mathrm{dis}(n,m))^{-2}$	$(\mathrm{dis}(n,m))^{-0.2}$

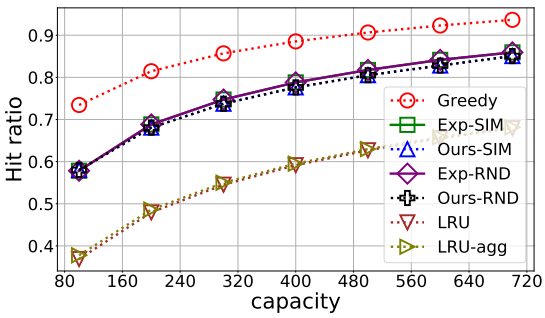
items n and m, we set RND-LRU parameters $q_n(m)$ to $(\mathrm{dis}(n,m))^{-2}$ when $\mathrm{dis}(n,m) \leq d$ and 0 otherwise. Note that when $d = 1$, RND-LRU reduces to SIM-LRU. Results for the hit ratio are averaged over the 50 request processes for $\alpha = 2.5$ and $\alpha = 1.4$. The 95% confidence intervals were smaller than $1.2 \cdot 10^{-3}$ in all the considered synthetic experiments for the hit ratio computation. For the Amazon trace, SIM-LRU and RND-LRU are employed with a similarity threshold $d = 300$. Furthermore, we set RND-LRU parameters to $q_n(m) = (\mathrm{dis}(n,m))^{-0.2}$ when $\mathrm{dis}(n,m) \leq d$ and 0 otherwise. In all experiments, we refer to the empirical hit ratios for SIM-LRU and RND-LRU as Exp-SIM and Exp-RND, respectively.

For all the theoretical computations of the hit ratio, the arrival rates $\boldsymbol{\lambda}$ for items are taken equal to the corresponding request probabilities. Our approach utilizes Algorithm 3 with parameter $\beta = 0.5$ and a stopping condition determined by a fixed number of iterations. Algorithm 3 is employed to estimate the approximate hit probabilities for all items, $\mathbf{h}$, and subsequently determines the overall cache hit ratio H. We refer to the latter estimate, for SIM-LRU and RND-LRU, as Ours-SIM and Ours-RND, respectively. Alternative methods that can possibly estimate the hit ratio were presented in Section 4.6.2. The numerical values used for all the experiments are summarized in Table 4.3.

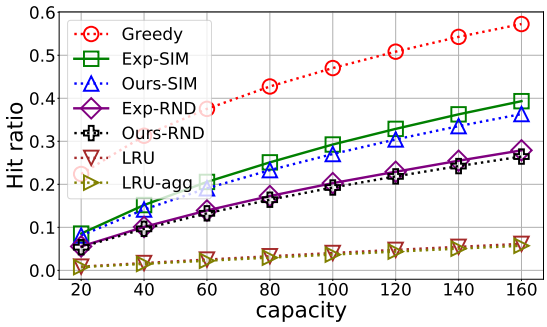
In Figure 4.3, we show the empirical hit ratio along with its estimates obtained through different approaches, for the two synthetic settings (Figures 4.3a-4.3b) and for the Amazon trace (Figure 4.3c). In the considered settings, as Greedy outperforms the other policies, its hit ratio would be an overestimation. LRU, in contrast, is underperforming and its hit ratio serves as an underestimation. LRU-agg, the naive approach to study SIM-LRU, underestimates the hit ratio.

Ours-SIM and Ours-RND clearly outperform all the alternative approaches presented in Section 4.6.2 in estimating the empirical hit ratio, while tending to underestimate it. As LRU does not take into account the similarity between items, the gap between LRU and Exp-SIM reveals the benefits of similarity caching over exact caching.

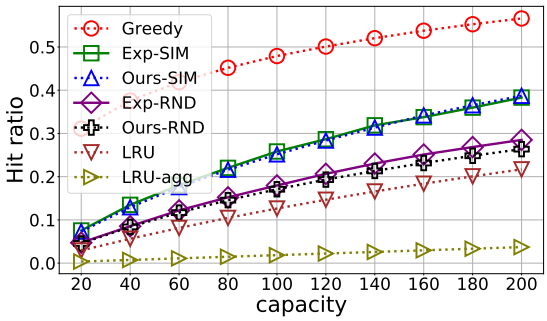
For the synthetic settings in Figures 4.3a and 4.3b, LRU and LRU-agg achieve similar hit ratios. When $\tilde{\boldsymbol{\lambda}} = (\tilde{\lambda}_n)_{n\in\mathcal{I}}$, where $\tilde{\lambda}_n = \sum_{m\in\mathcal{N}^c[n]} \lambda_m$, is proportional to $\boldsymbol{\lambda}$, LRU and LRU-agg achieve similar hit ratios. In the choice of the popularity distribution in Figures 4.3a and 4.3b (see (4.58)), a popular item and its neighbors share similar rates, i.e., $\lambda_n \approx \lambda_m$ for $m \in \mathcal{N}^c[n]$. It follows that in the settings of Figures 4.3a and 4.3b, the approximations $\tilde{\lambda}_n \approx 5 \cdot \lambda_n$ and $\tilde{\lambda}_n \approx 13 \cdot \lambda_n$ hold for



(a) Synthetic trace, $\alpha = 2.5$, $d = 1$, 25 iterations

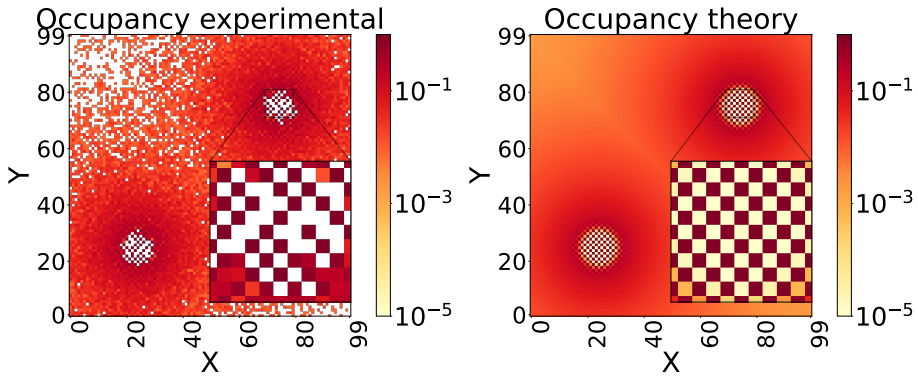


(b) Synthetic trace, $\alpha = 1.4$, $d = 2$, 15 iterations.



(c) Amazon trace, $d = 300$, 40 iterations.

Figure 4.3: Average hit ratio versus cache capacity, $\beta = 0.5$.



(a) $r = 2 \cdot 10^5$. (b) 25 iterations.

Figure 4.4: Synthetic trace occupancies: $C = 500$, $d = 1$, $\alpha = 2.5$.

the respective scenarios, especially for the popular items. This provides insight into the comparable hit ratios observed between LRU and LRU-agg in Figures 4.3a and 4.3b.

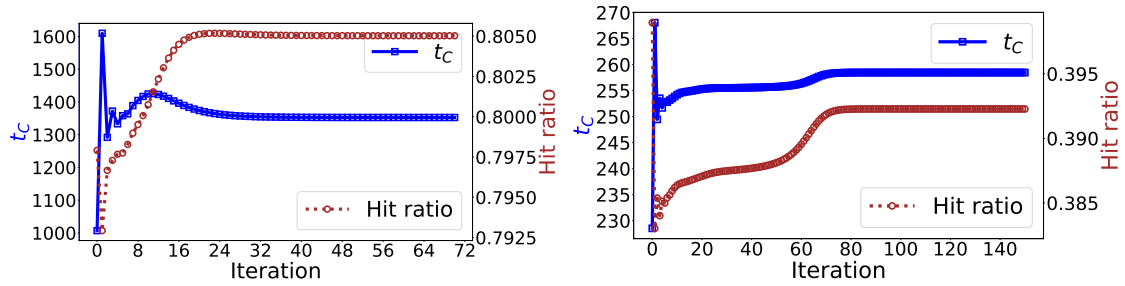
While our approach provides the best estimates, we can observe that it slightly underestimates the hit ratio. In order to understand this effect, we show in Figure 4.4 the empirically estimated occupancy vector and the one produced by Algorithm 3. The proposed algorithm broadly captures the empirical occupancy patterns, but with subtleties regarding symmetries. In particular, the zoom on Figure 4.4b shows that our approach produces a regular chess board pattern. Some items are predicted to stay almost all the time in the cache while their 4 neighbors are predicted to spend virtually no time in it. The corresponding empirical occupancy in Figure 4.4a shows a less symmetric pattern, implying that in this setup SIM-LRU is able to satisfy a group of requests using a smaller number of cache slots when compared against what is predicted by our approach. This, in turn, partially explains why our approach underestimates the hit ratio.

4.6.4 Convergence of Algorithm 3

The RND-TTL approximation selects the parameters $\boldsymbol{\lambda}^i$, $\boldsymbol{\lambda}^r$, and T for the RND-TTL cache in a way that ensures the occupancy vector, as described in (4.7), is a fixed point of the function $\mathbf{G}$ defined in (4.40a). Algorithm 3 employs an iterative procedure aimed at finding a fixed point of $\mathbf{G}$, thereby determining the appropriate values for the RND-TTL cache's parameters.

Figure 4.5 shows the evolution of characteristic time t_C and hit ratio H over different iterations. We observe that estimates of H and t_C by our algorithm converge in a few iterations (less than 70), under all considered scenarios. Note that $t_C(0)$, the value of t_C at iteration 0, is also the value of t_C for LRU (see (4.4)). In addition, across all experiments, t_C for Ours-SIM using Algorithm 3 converges to a value larger than $t_C(0)$. Indeed, under LRU, t_C is bounded by the time required for C distinct items to be requested. For SIM-LRU and RND-LRU, in contrast, after C distinct items are requested, an item previously in the cache can remain there, despite not serving any requests. This occurs due to approximate hits, explaining why t_C is larger for Ours-SIM than for LRU.

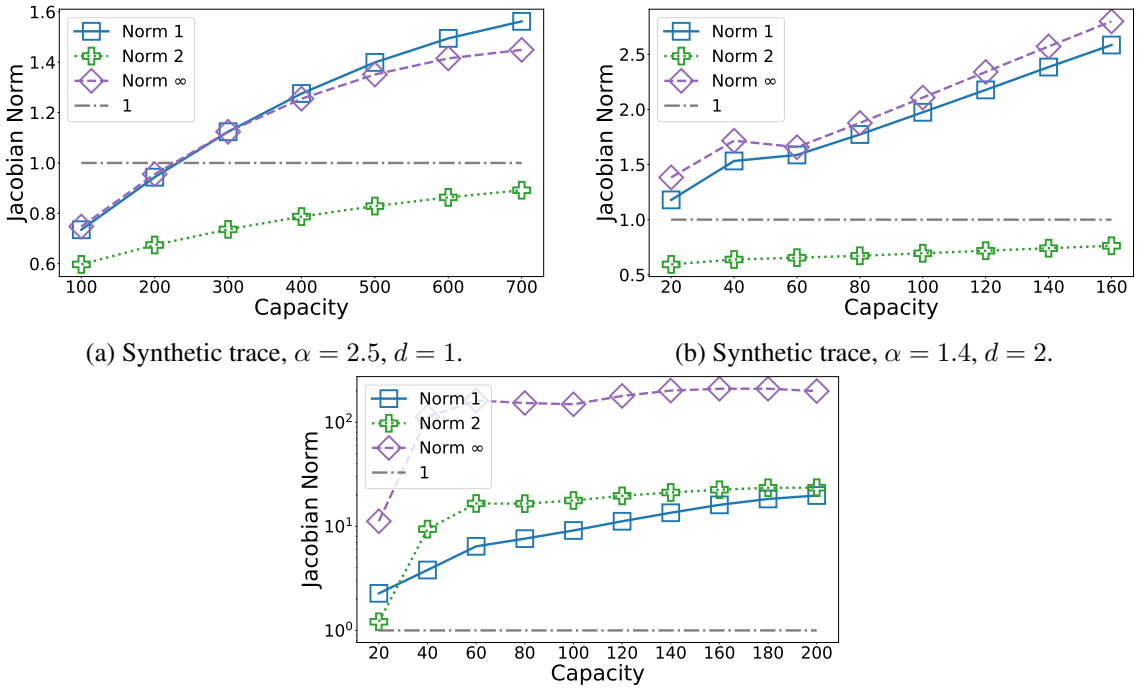
Proposition 4.6 provides a sufficient condition for the convergence of Algorithm 3 with parame-



(a) Synthetic trace, $\alpha = 2.5$, $d = 1$, $C = 500$

(b) Amazon trace, $d = 300$, $C = 200$

Figure 4.5: Characteristic time t_C and hit ratio in different iterations of Algorithm 3 for SIM-LRU.



(a) Synthetic trace, $\alpha = 2.5$, $d = 1$.

(b) Synthetic trace, $\alpha = 1.4$, $d = 2$.

(c) Amazon trace, $d = 300$.

Figure 4.6: Norm $\boldsymbol{J}_{\mathbf{G}_\beta}$ versus cache capacity, $\beta = 0.5$.

ter β towards a unique fixed point of $\mathbf{G}$. This condition requires an operator norm of the Jacobian matrix, $\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o})$, associated with the map $\mathbf{G}_\beta$, to be strictly less than 1 for any $\mathbf{o} \in \Delta_C$.

In Figure 4.6, we show the spectral norm and norms 1 and infinity of $\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o}(0))$, where $\mathbf{o}(0)$ is given in line 1 of Algorithm 3, for the two synthetic traces and the Amazon trace described in Section 4.6.1. We provide in B.13 details for the computation of $\boldsymbol{J}_{\mathbf{G}_\beta}$. In all the settings, β is set to 0.5. We observe in Figure 4.6 that the norm of the Jacobian matrix $\boldsymbol{J}_{\mathbf{G}_\beta}$ increases with the cache capacity. For the synthetic traces, the spectral norm of $\boldsymbol{J}_{\mathbf{G}_\beta}$ is smaller than 1 for all cache capacities, whereas in the Amazon trace, for all tested norms, $\|\boldsymbol{J}_{\mathbf{G}_\beta}\|$ exceeds 1. Nevertheless, Proposition 4.6 provides only sufficient conditions and our results (see e.g. Figure 4.5b) suggest that our algorithm converges also on the Amazon trace.

4.7 Conclusion

We proposed a method named the RND-TTL approximation for estimating the hit ratio of a popular similarity caching policy, RND-LRU, under IRM. This method tunes the parameters of RND-TTL, a novel similarity cache model we introduced, and uses its hit ratio as an estimation for RND-LRU's hit ratio. We are the first to propose an analytical method for estimating the hit ratio of RND-LRU. The RND-TTL approximation involves solving a system of fixed point equations via a parameterized iterative algorithm. We studied the convergence of this algorithm and proposed a practical way of choosing its parameter.

Our experimental benchmark shows that the RND-TTL approximation accurately estimates the hit ratio of RND-LRU under IRM, with a relative error below 5% across all tested configurations, as depicted in Figure 4.3. In future work, we envision investigating analytically the accuracy of our RND-TTL approximation, similarly to what was done in [FRR12; JNT18] for classic caching.

CHAPTER 5

Conclusion

In this thesis, we studied three randomized algorithms: the Count-Min Sketch with Conservative Updates (CMS-CU), a modified version of the Follow-the-Perturbed-Leader (FPL) algorithm for caching, and a variant of the Least-Recently-Used (LRU) policy tailored for similarity caching. We present summaries of the results for each algorithm and elaborate on potential future research directions.

5.1 CMS-CU

Summary. CMS-CU, a probabilistic hash-based data structure, estimates the frequency of items in a data stream. It serves as a building block in numerous recently proposed methods for frequency estimation [Yan+18b; Zha+21b]. Frequency-based caching policies employ CMS-CU to balance accuracy and memory usage [EFM17]. Despite the widespread adoption of CMS-CU, the advantages brought by the incorporation of Conservative Updates into the conventional Count-Min Sketch (CMS) remain insufficiently comprehended.

In Chapter 2, we proved new bounds on the estimation error for items' counts under an i.i.d. request process in CMS-CU. These bounds highlight that items with large request counts barely suffer any error when compared to CMS. Unfortunately, our bounds fail to capture the advantages of the Conservative Update for less popular items. Moreover, our bounds are exclusive for an i.i.d. request process.

Perspectives. CMS-CU selects d counters based on the d hash functions and the requested item at each time step. Simulations suggest that the error in CMS-CU is maximal when the counters selection is uniformly at random at each time step [Bia+12]. Bianchi et al. studied CMS-CU with this particular counter-selection via a fluid approximation. However, this approach solves numerically T differential equations to identify the error after T steps, which is computationally expensive.

In upcoming research, the objective is to investigate whether the error in CMS-CU is maximal in the uniformly at random counters selection setting. This instance of CMS-CU poses an intriguing probability problem, resembling a variation of the balls and bins load balancing problem [Ber+00]. Specifically, the scenario involves m bins and n balls. During each time step, a ball randomly selects d bins, and it is then placed in the least loaded bin among the chosen ones. This process shares similarities with CMS-CU with uniform counter selection, where the bins serve as counters. The key distinction lies in the balls-into-bins variant, where at each time step, exactly one counter is incremented, in contrast to potentially incrementing up to d counters in CMS-CU. Exploring

the connection between these problems might unveil efficient methods for computing the maximal error in CMS-CU.

5.2 FPL and Approximate Counting

Summary. Counting requests for items is crucial to achieving an optimal hit ratio under a stationary request process. Interestingly, even under an adversarial request process, this counting is a building block for a policy attaining a near-optimal regret of $\mathcal{O}(\sqrt{T})$, where T is the total number of requests. This policy is an instance of the Follow-the-Perturbed-Leader (FPL) [KV05], which has a near-optimal performance for a broad category of online problems, including the caching problem. In the stationary setting, the optimal strategy stores the items with the highest request counts. In the adversarial model, FPL adds independent and identically distributed (i.i.d.) noises, sourced from a distinct distribution, to each item's counts, storing those with the highest values.

In Chapter 3, we examined the possibility of substituting the exact items' request counting component within FPL with an approximate counting algorithm. We showed that FPL maintains a sublinear regret of $\mathcal{O}(\sqrt{T})$ when the estimates of the approximate counting algorithm are unbiased and satisfy additional criteria. In particular, when counting over a subset of the requests where each request is included with probability f, FPL achieves a regret of $\mathcal{O}(\frac{\sqrt{T}}{f})$.

Perspectives. One significant drawback of the Follow the Perturbed Leader algorithm for caching lies in its memory requirement of $\mathcal{O}\left(N \ln T\right)$, where N represents the total number of distinct items and T is the time horizon. Despite proving that counting over a subset of the requests sequence within FPL ensures sublinear regret of $\mathcal{O}(\sqrt{T})$, the associated memory cost remains at $\mathcal{O}\left(N \ln T\right)$.

In our future research, we plan to explore whether an online algorithm designed for the caching problem can attain a sublinear regret with sublinear memory in the total number of items. A parallel inquiry has recently been addressed in the context of the Expert problem. In this problem, a participant has predictions from M agents, and her objective is to choose a single agent at each time step, guided by the historical performance of these agents. Srinivas et al.[Sri+22] demonstrated that the regret for any online algorithm for the Expert problem, operating with a memory of $\mathcal{O}(S)$, is at least $\Omega\left(\sqrt{\frac{M \cdot T}{S}}\right)$. Meanwhile, Peng et al.[PZ23; PR23] proposed algorithms with sublinear memory in M, achieving sublinear regret.

The caching problem is a particular case of the Expert problem, with each agent representing a set of cached items. However, in this case, the number of experts is $\binom{N}{C}$, where C denotes the cache capacity. Despite this limitation, we are intrigued by the possibility of converting memory-efficient algorithms designed for the Expert problem into efficient algorithms for the caching problem, while preserving sublinear regret guarantees.

5.3 An LRU-Based Similarity Caching Policy

Summary. Beyond frequency, caching policies base their decisions on recency as well. The Least-Recently-Used (LRU) is a simple and efficient recency-based policy. RND-LRU is an adaptation of LRU for the similarity caching setting where an item's request can be approximately answered with

a similar cached item. RND-LRU probabilistically answers the request for an item with its most similar cached item. The analysis of RND-LRU is challenging because of the strong coupling in caching decisions across items.

In Chapter 4, we proposed an analytically tractable similarity caching model, called RND-TTL, where caching decisions across items are independent, to approximate the hit ratio of RND-LRU under an i.i.d. request process. RND-TTL captures the behavior of RND-LRU through a specific parametrization. Simulations highlight the accuracy of our method over multiple traces.

Perspectives. Our methodology shares similarities with the Time-to-Live (TTL) approximation, which efficiently estimates the hit ratio of LRU under an i.i.d. request process [Fag77]. Jiang et al. [JNT18] have previously bounded the estimation error of the TTL approximation. In our future work, we intend to explore the accuracy of our method in approximating the hit ratio of RND-LRU.

The natural method for precisely computing the hit ratio of RND-LRU is to calculate the limiting distribution of the Markov chain that represents the list of cached items. The state space of this Markov chain grows exponentially with the number of items and the cache capacity, making exact computation costly. Allmeier and Gast [AG22] investigate a more general problem by examining the dynamics of a Markov chain on a finite but extensive state space. In this case, the state space is a subset of $\mathcal{S}^n$, where $\mathcal{S}$ is a finite set. They demonstrate that, under specific conditions on the transition rates of the Markov chain, the probability of the chain being in a certain state can be approximated by the solution of an ordinary differential equation with an error of $\mathcal{O}(1/n)$. This approach is known in the literature as the *mean field approximation*. In future work, we plan to investigate the applicability of the aforementioned approximation to RND-LRU.

Appendix

APPENDIX A

Approximate Counting

A.1 Proof of Proposition 2.1 (page 14)

From (2.3) and the fact that the random variables $\{e_i^r(t)\}_{r\in[d]}$ are i.i.d. when using CMS, we have $\Pr\left(e_i(t)/t > x\right) = \left(\Pr\left(e_i^1(t)/t > x\right)\right)^d$. To prove (2.18) it is then sufficient to show that $\Pr\left(e_i^1(t)/t > x\right) \leq \mathcal{A}_k(x)$ for $k \in [\![0, w-1]\!]$. For a given $k \neq 0$ we consider the event, called $E_{i,k}$, of no hash collision in row 1 between item i and any of the k most popular items (other than i, if $i \leq k$). Formally,

$$E_{i,k} \Leftrightarrow h_1(i) = h_1(j),\ \forall j \leq k,\ j \neq i. \tag{A.1}$$

By first writing the law of total probabilities with respect to the partition $\{E_{i,k}, \overline{E_{i,k}}\}$, and then using the union bound to write $\Pr\left(\overline{E_{i,k}}\right) \leq k/w$ and the Markov inequality to upper bound $\Pr\left(e_i^1(t)/t \geq x \mid E_{i,k}\right)$, we obtain

$$\Pr\left(\frac{e_i^1(t)}{t} \geq x\right) \leq \Pr\left(\frac{e_i^1(t)}{t} \geq x \mid E_{i,k}\right) \cdot 1 + 1 \cdot \Pr\left(\overline{E_{i,k}}\right) \tag{A.2}$$

$$\leq \frac{\mathbb{E}\left[e_i^1(t) \mid E_{i,k}\right]}{xt} + \frac{k}{w} \tag{A.3}$$

$$\leq \frac{1}{x} \sum_{\substack{j>k \\ j\neq i}} p_j \Pr\left(h_1(i) = h_1(j) \mid E_{i,k}\right) + \frac{k}{w} \tag{A.4}$$

$$\leq \frac{1}{x(w-k)} \sum_{\substack{j>k \\ j\neq i}} p_j + \frac{k}{w} = \mathcal{A}_k(x) \tag{A.5}$$

where (A.4) follows from (2.4)-(2.5) and (A.5) uses $\Pr\left(\overline{E_{i,k}}\right) \leq k/w$. By observing that (A.5) holds also for $k = 0$, we have completed the proof.

A.2 Proof of Lemma 2.1 (page 15)

We will make use of two quantities to prove Lemma 2.1.

$$l_j^r \triangleq \sum_{e\in N_G(h_r(j))} p_e, \qquad g_j \triangleq \min_{r\in[d]} l_j^r. \tag{A.6}$$

For a given realization of G, l_j^r is an upper bound on the growth rate of counter $c_j^r(t)$ and g_j is an upper bound on the growth rate of $\hat{n}_j(t)$. To ease the writing, we use A, B, C, and D_j^r as shorthand for events "$h_r(i) = h_r(j)$", "$\hat{n}_j(s-1) = c_i^r(s-1)$", "$g_j \geq p_i$", and "$l_j^r \geq p_i$", respectively. Starting from (2.9) we write

$$\mathbb{E}\left[\delta_{i,j}^r\right] = p_j \Pr\left(A \cap B\right) \tag{A.7}$$

$$= p_j\Big(\Pr\left(A \cap B \cap C\right) + \Pr\left(A \cap B \cap \overline{C}\right)\Big) \tag{A.8}$$

$$\leq p_j\Big(\Pr\left(A \cap C\right) + \Pr\left(A \cap \overline{C}\right)\Pr\left(B \mid A, \overline{C}\right)\Big) \tag{A.9}$$

$$\leq p_j\Big(\Pr\left(A \cap \left(\cap_{e\in[d],e\neq r} D_j^e\right)\right) + \Pr\left(A\right)\Pr\left(B \mid A, \overline{C}\right)\Big) \tag{A.10}$$

$$\leq p_j \Pr\left(A\right)\Big(\Pr\left(D_j^1\right)^{d-1} + \Pr\left(B \mid A, \overline{C}\right)\Big). \tag{A.11}$$

We now move to deriving upper bounds on $\Pr\left(D_j^1\right)$ and $\Pr\left(B \mid A, \overline{C}\right)$. For $j \leq i$, we simply write $\Pr\left(D_j^1\right) \leq 1$. For $j > i$, we follow the steps in (A.2)-(A.3):

$$\Pr\left(D_j^1\right) = \Pr\left(l_j^1 - p_j \geq p_i - p_j\right) \tag{A.12}$$

$$\leq \Pr\left(l_j^1 - p_j \geq p_i - p_j \mid E_{j,k}\right) \cdot 1 + 1 \cdot \Pr\left(\overline{E_{j,k}}\right) \tag{A.13}$$

$$\leq \frac{\mathbb{E}\left[l_j^1 - p_j \mid E_{j,k}\right]}{p_i - p_j} + \frac{k}{w}. \tag{A.14}$$

We bound $\mathbb{E}\left[l_j^1 - p_j \mid E_{j,k}\right]$ similarly to what was done for $\mathbb{E}\left[e_j^1(t)/t \mid E_{j,k}\right]$ in (A.3)-(A.5):

$$\mathbb{E}\left[l_j^1 - p_j \mid E_{j,k}\right] = \sum_{i\in I\setminus\{j\}} p_i \cdot \Pr\left(i \in N_G(h_r(j)) \mid E_{j,k}\right) \tag{A.15}$$

$$= \sum_{i>k, i\neq j} p_i \cdot \Pr\left(h_r(i) = h_r(j) \mid E_{j,k}\right) \tag{A.16}$$

$$\leq \sum_{i>k, i\neq j} p_i \cdot \frac{1}{w-k} \tag{A.17}$$

We combine (A.14) and (A.17) to find $\Pr\left(D_j^1\right) \leq \mathcal{A}_k(p_i - p_j)$. We again observe that this holds also for $k = 0$, which implies that

$$\Pr\left(D_j^1\right) \leq \begin{cases} 1, & \forall j \leq i, \\ \min\left(\mathcal{A}(p_i - p_j), 1\right), & \forall j > i, \end{cases} \quad \Leftrightarrow \quad \Pr\left(D_j^1\right)^{d-1} \leq \gamma_{i,j} \tag{A.18}$$

where $\gamma_{i,j}$ is given in (2.24).

To bound $\Pr\left(B \mid A, \overline{C}\right)$, we start by making a change of variable. Let $z = s - 1$. We define the random variable $y_j(z)$ as,

$$y_j(z) \triangleq \sum_{\substack{e\in I \\ h_{r_0}(j)=h_{r_0}(e)}} n_e(z) : \; r_0 = \text{argmin}_{r\in[d]} l_j^r. \tag{A.19}$$

It follows that $\hat{n}_j(z) \leq y_j(z)$. We now use F, J and K as respective shorthand for events "$n_i(z) > y_j(z)$", "$n_i(z) > m(z)$", and "$y_j(z) < m(z)$", with $m(z) \triangleq (p_i + g_j)z/2$. Under the conditioning on A, the equality $c_i^r(z) = c_j^r(z)$ holds. Equation (2.2) implies then that $c_i^r(z) \geq \hat{n}_j(z)$. The event $\overline{B}$ conditioned on A boils down to $c_i^r(z) > \hat{n}_j(z)$. Since $c_i^r(z) \geq n_i(z)$ and $y_j(z) \geq \hat{n}_j(z)$, the event F implies the event $\overline{B}$ conditionally on A. We then write

$$\begin{aligned}\Pr\left(B \mid A, \overline{C}\right) = 1 - \Pr\left(\overline{B} \mid A, \overline{C}\right) &\leq 1 - \Pr\left(F \mid A, \overline{C}\right) \\ &\leq 1 - \Pr\left(J\right) \Pr\left(K \mid A, \overline{C}\right).\end{aligned}$$

The last step follows from the fact that $n_i(z)$ and $y_j(z)$ are negatively associated [JP83]. Following (A.19), for every fixed graph realization of G, $y_j(z)$ is the sum of negatively associated random variables [JP83] and has an expected value $g_j z$ that is less than $m(z)$ under the conditioning that the fixed graph G verifies $g_j < p_i$ (event $\overline{C}$). Thus using Chernoff bounds on events J and K we get

$$\Pr\left(B \mid A, \overline{C}\right) \leq \beta_{i,j} \mathrm{e}^{-\alpha_{i,j} z}. \tag{A.20}$$

Using $\Pr(A) \leq 1/w$, (A.18), and (A.20) in (A.11) we find (2.23) concluding the proof.

A.3 Proof of Proposition 2.2 (page 16)

Proving the bound by $\mathcal{C}_i(x)$. Let $E_{i,k}^r$ of no hash collision in row r between item i and any of the k most popular items (other than i, if $i \leq k$). We define $F_{i,k}$ as their union for all possible rows $r \in [d]$, i.e., $F_{i,k} = \bigcup_{r\in[d]} E_{i,k}^r$. By the law of total probabilities with respect to the partition $\{F_{i,k}, \overline{F_{i,k}}\}$, we write, for every k,

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \Pr\left(\frac{e_i(t)}{t} \geq x \mid F_{i,k}\right) \cdot 1 + 1 \cdot \Pr\left(\overline{F_{i,k}}\right) \tag{A.21}$$

$$\leq \frac{\mathbb{E}\left[e_i(t) \mid F_{i,k}\right]}{xt} + \left(\frac{k}{w}\right)^d. \tag{A.22}$$

The last step follows from the Markov inequality and the bound $\Pr\left(\overline{E_{i,k}^r}\right) \leq k/w$. To bound $\mathbb{E}\left[e_i(t) \mid F_{i,k}\right]$, we first observe that conditioning on $F_{i,k}$ implies there exists at least a row r_0 such that the event $E_{i,k}^{r_0}$ is true. Using (2.3) and (2.5) we write

$$\mathbb{E}\left[e_i(t) \mid F_{i,k}\right] \leq \mathbb{E}\left[e_i^{r_0}(t) \mid F_{i,k}\right] \leq \sum_{s\in[t]} \sum_{\substack{j>k \\ j\neq i}} \mathbb{E}\left[\delta_{i,j}^{r_0}(s) \mid F_{i,k}\right], \tag{A.23}$$

where we used the fact that $\mathbb{E}\left[\delta_{i,j}^{r_0}(s) \mid F_{i,k}\right] = 0$ for $j \leq k$. For $j > k$, the following holds

$$\mathbb{E}\left[\delta_{i,j}^{r_0}(s) \mid F_{i,k}\right] \leq \frac{\mathbb{E}\left[\delta_{i,j}^{r_0}(s)\right]}{1-\Pr\left(\overline{F_{i,k}}\right)} \leq \frac{\mathbb{E}\left[\delta_{i,j}^{r_0}(s)\right]}{1-\left(\frac{k}{w}\right)^d}. \tag{A.24}$$

Combining (A.22)-(A.24) with (2.23) leads to

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \frac{\sum_{j>k, j\neq i} p_j\gamma_{i,j}}{xw\left(1-\left(\frac{k}{w}\right)^d\right)} + \frac{1}{xwt\left(1-\left(\frac{k}{w}\right)^d\right)} \sum_{\substack{j>k \\ j\neq i}} \frac{p_j\beta_{i,j}}{1-\mathrm{e}^{-\alpha_{i,j}}} + \left(\frac{k}{w}\right)^d.$$

As this bound is valid for every $k \in [\![0, w-1]\!]$, we can write,

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \mathcal{C}_i(x) + \mathcal{O}\left(\frac{1}{t}\right). \tag{A.25}$$

Proving the bound by $\mathcal{B}_i(x)$. We rely on the same arguments used above and in the proofs of Proposition 2.1. We denote $E_{i,k}^1$ as $E_{i,k}$. We have,

$$\begin{aligned}
\Pr\left(e_i(t)/t \geq x\right) &\leq \Pr\left(e_i^1(t)/t \geq x\right) && \text{(A.26)}\\
&\leq \Pr\left(e_i^1(t)/t \geq x \mid E_{i,k}\right) \cdot 1 + 1 \cdot \Pr\left(\overline{E_{i,k}}\right) && \text{(A.27)}\\
&\leq \sum_{s\in[t]} \sum_{j>k} \mathbb{E}\left[\delta_{i,j}^1(s) \mid E_{i,k}\right] + \frac{k}{w} && \text{(A.28)}\\
&\leq \frac{1}{x(w-k)} \sum_{j>j} p_j\gamma_{i,j} + \frac{k}{w} + \mathcal{O}\left(\frac{1}{t}\right). && \text{(A.29)}
\end{aligned}$$

As this bound is valid for every $k \in [\![0, w-1]\!]$, we can write,

$$\Pr\left(\frac{e_i(t)}{t} \geq x\right) \leq \mathcal{B}_i(x) + \mathcal{O}\left(\frac{1}{t}\right). \tag{A.30}$$

The bounds that are valid with CMS are also valid with CMS-CU, thus by Proposition 2.1, the CCDF with CMS-CU is less than $\mathcal{A}(x)^d$. Therefore, we get the result of the proposition when we combine (A.30) and (A.25).

A.4 Proof of Proposition 2.3 (page 17)

We have $\mathbb{E}\left[e_i(t)\right] \leq \mathbb{E}\left[e_i^r(t)/t\right]$ thanks to (2.3) (that holds for CMS-CU). We use Lemma 2.1 and the linearity of the expectation to deduce that,

$$\begin{aligned}
\mathbb{E}\left[\frac{e_i^r(t)}{t}\right] &\leq \frac{1}{t} \sum_{s\in[t]} \sum_{j\in\mathcal{I}\setminus\{i\}} \frac{p_j}{w}\left(\gamma_{i,j} + \beta_{i,j}\mathrm{e}^{-\alpha_{i,j}(s-1)}\right) && \text{(A.31)}\\
&\leq \frac{1}{w} \sum_{j\in\mathcal{I}\setminus\{i\}} p_j\gamma_{i,j} + \frac{1}{wt} \sum_{j\in\mathcal{I}\setminus\{i\}} \frac{p_j\beta_{i,j}}{1-\mathrm{e}^{-\alpha_{i,j}}} && \text{(A.32)}\\
&\leq \frac{1}{w} \sum_{j\in\mathcal{I}\setminus\{i\}} p_j\gamma_{i,j} + \mathcal{O}\left(\frac{1}{t}\right). && \text{(A.33)}
\end{aligned}$$

We observe the following,

$$\mathbb{E}\left[\frac{e_i(t)}{t}\right] = \frac{1}{t}\sum_{n=0}^{t}\Pr\left(e_i(t) \geq n\right) = \frac{1}{t}\sum_{n=0}^{t}\Pr\left(\frac{e_i(t)}{t} \geq \frac{n}{t}\right) \tag{A.34}$$

We use Proposition 2.2 to deduce that,

$$\mathbb{E}\left[\frac{e_i(t)}{t}\right] \leq \frac{1}{t}\sum_{n=0}^{t}\mathcal{D}_i\left(\frac{n}{t}\right) + \mathcal{O}\left(\frac{1}{t}\right). \tag{A.35}$$

We combine (A.35)and (A.34) to find the proposition result.

A.5 Discussion on the bound (2.15)

When all items are requested at least once (which is true under the IRM model for a large enough stream process), the bound (2.15) proposed in [EF15] becomes trivial if $w(N) = o(N)$ and both w and N diverge. This happens because

$$\lim_{N\to+\infty}\mathrm{PFP}(A_k) = 1,\ \forall k \in \mathbb{N} \tag{A.36}$$

as we explain next. A_k is computed recursively using (2.16), with $A_1 = D_1 = N$ since all items are requested at least once. The first computation of (2.16) requires $\mathrm{PFP}(N)$. We have

$$\begin{aligned}
\mathrm{PFP}(N) &= \frac{1}{N}\sum_{i=1}^{N}\left(1-\mathrm{e}^{-\frac{i}{w(N)}}\right)^d \\
&= \frac{1}{N}\left(N + \sum_{j=1}^{d}\binom{d}{j}(-1)^j\sum_{i=1}^{N}\mathrm{e}^{-\frac{ij}{w(N)}}\right) \\
&= 1 + \sum_{j=1}^{d}\binom{d}{j}(-1)^j\mathrm{e}^{-\frac{j}{w(N)}}\frac{1-\mathrm{e}^{-\frac{jN}{w(N)}}}{N\left(1-\mathrm{e}^{-\frac{j}{w(N)}}\right)}.
\end{aligned}$$

The following then holds

$$\lim_{N\to+\infty} N\left(1-\mathrm{e}^{-\frac{j}{w(N)}}\right) = +\infty \quad\Rightarrow\quad \lim_{N\to+\infty}\mathrm{PFP}(N) = 1.$$

Applying (2.16) recursively yields (A.36), confirming that the bound (2.15) boils down to 1.

APPENDIX B

Similarity Caching

B.1 R-TTL

Algorithm 4: R-TTL

Input:
Sequence of requests $(r_1, \ldots, r_J)$
Sequence of time instants $(\tau_1, \ldots, \tau_J)$ when the requests occurred
Probabilities $(q_n(m))_{n,m\in\mathcal{I}^2}$ and timers duration $(T_n)_{n\in\mathcal{I}}$
Initial TTL vector $(u_{0,1}, \ldots, u_{0,|\mathcal{I}|})$ where $u_{0,n}$ is the initial value of TTL of item n and $0 \leq u_{0,n} \leq T_n$.
Output:
Set of cached items at times $\tau_1, \ldots, \tau_J$.
Algorithm:
$\tau_0 \leftarrow 0$
$S_0 \leftarrow \{i \in \mathcal{I} : \ u_{0,i} > 0\}$
for $j = 1$ to J **do**
  for $n = 1$ to $|\mathcal{I}|$ **do**
    $u_{j,n} \leftarrow \max\left(u_{j-1,n} - (\tau_j - \tau_{j-1}), 0\right)$
  end for
  $S_j \leftarrow \{i \in \mathcal{I} : \ u_{j,i} > 0\}$
  $\hat{r}_j \leftarrow \arg\min_{m\in S_j} \mathrm{dis}(r_j, m)$
  Generate a uniform random number $\delta \in [0, 1]$
  if $\delta \leq q_{\hat{r}_j}(r_j)$ **then**
    Case 1: Hit, encompassing exact/approximate hits
    $u_{j,\hat{r}_j} \leftarrow T_{\hat{r}_j}$
  else
    Case 2: Miss
    $S_j \leftarrow S_j \cup \{r_j\}$
    $u_{j,r_j} \leftarrow T_{r_j}$
  end if
end for
return $S_1, \ldots, S_J$

Algorithm 4 outlines a pseudo code for the similarity caching policy R-TTL. At each time

step j, the algorithm handles requests for items, updating TTLs and cache contents accordingly. The set S_0 represents the initially stored items, those with TTL values strictly greater than 0 (line 10). At any time step $j \in \{1, \dots, J\}$, and for any item i, the variable $u_{j,i}$, in line 13, monitors the value of the TTL of item i right before handling the request r_j. If the duration $\tau_j - \tau_{j-1} \geq u_{j-1,i}$, it indicates that item i's timer expired within the time interval $[\tau_{j-1}, \tau_j]$, resulting in $u_{j,i}$ being set to 0. Otherwise, i's timer at τ_j did not expire and its value is equal to $u_{j-1,i} - (\tau_j - \tau_{j-1})$.

The set S_j, in line 15, characterizes the items in the R-TTL cache immediately before handling the request for item r_j. R-TTL identifies the closest cached item to r_j, denoted as $\hat{r}_j$ (line 16). The request for r_j is approximately served by $\hat{r}_j$ with probability $q_{\hat{r}_j}(r_j)$, leading to the reset of $\hat{r}_j$'s timer (line 20). In the event of a miss, item r_j is added to the cache (line 23), and its timer is reset to T_{r_j} (line 24). Algorithm 4 returns a list of sets S_j for every $j \in \{1, \dots, J\}$, corresponding to the set of cached items in R-TTL immediately after handling the request of item r_j.

Note that in practice, we only need to keep track of TTL values greater than zero, corresponding to cached items. However, to simplify the presentation, Algorithm 4 assumes that TTLs are stored for all items. Finally, the algorithm assumes that the cache statically stores a tombstone item whose distance to all items is infinite. Whenever a request arrives in an empty cache, the tombstone item is returned as the closest item in the cache.

B.2 Proof of Proposition 4.1 (Occupancy, page 49)

To derive the occupancy of an item n, we first observe that the instants when item n is evicted from the cache are regeneration points of a renewal process [Ros14]. A renewal cycle consists of two consecutive time periods: a time period of duration T_n^{Off}, that starts immediately after item n is evicted from the cache and ends when it re-enters the cache, and a time period of duration T_n^{On}, that ends when item n is evicted again from the cache. From [Ros95, Thm. 3.6.1, Example 3.6(A)], the occupancy can be computed as:

$$o_n = \frac{\mathbb{E}\left[T_n^{\text{On}}\right]}{\mathbb{E}\left[T_n^{\text{Off}}\right] + \mathbb{E}\left[T_n^{\text{On}}\right]}. \tag{B.1}$$

We have that T_n^{On} verifies:

$$T_n^{\text{On}} = \sum_{j=1}^{F} Y_j + T_n, \tag{B.2}$$

where $(Y_j)_{j \in \{1,\dots,F\}}$ are exponentially distributed random variables with parameter λ_n^r such that $Y_j < T_n$ for $j = 1, \dots, F$, and F is a geometric random variable. Since we have:

$$\mathbb{E}\left[Y_j \mid Y_j < T_n\right] = \frac{1}{\lambda_n^r} - \frac{T_n}{\exp(\lambda_n^r T_n) - 1}, \tag{B.3}$$

$$\mathbb{E}\left[F\right] = \exp(\lambda_n^r T_n) - 1, \tag{B.4}$$

we conclude from Wald's identity and (B.2) that:

$$\mathbb{E}\left[T_n^{\text{On}}\right] = \frac{\mathrm{e}^{\lambda_n^r T_n} - 1}{\lambda_n^r}. \tag{B.5}$$

By combining (B.5) and (B.1) and observing that $\mathbb{E}\left[T_n^{\text{Off}}\right] = 1/\lambda_n^i$, we get our result.

B.3 Generalized Poisson Arrivals See Time Averages (PASTA) property

We derive in this appendix a generalization of the PASTA property that will be used in the proofs of Propositions 4.2–4.4. We will construct a counting process $\{Q(t),\ t \geq 0\}$ that is not Poisson but whose jumps coincide with one of many Poisson processes (to be defined). In our generalization, we prove that the arrivals of $\{Q(t),\ t \geq 0\}$ see time averages.

Let $\{M(t),\ t \geq 0\}$ be a stochastic process with finite state space $\mathcal{E}$. We assume that for every $S \in \mathcal{E}$, $\lim_{t\to+\infty} \Pr(M(t) = S)$ exists and we denote it as π^*_S. We also assume that $\lim_{T\to+\infty} \frac{1}{T}\int_0^T \mathbb{1}(M(u) = S)du$ exists and is equal to π^*_S. For every $S \in \mathcal{E}$, we define a Poisson process $\{P_S(t),\ t \geq 0\}$ with rate λ_S. For any S, the processes $\{P_S(t+u)-P_S(t),\ u \geq 0\}$ and $\{M(v), 0 \leq v \leq t\}$ are assumed to be independent. This assumption is known as the lack of anticipation assumption [Wol82; vR88]. We construct a stochastic process $\{Q(t),\ t \geq 0\}$ such that its jumps coincide with the jumps of $\{P_S(t),\ t \geq 0\}$ when $\{M(t),\ t \geq 0\}$ is in state S, for any $S \in \mathcal{E}$. If y_S is the number of jumps of $\{P_S(t),\ t \geq 0\}$ in $[0, t]$ and $t_{1,S}, \ldots t_{y_S,S}$ are the instants of those jumps, then $\{Q(t),\ t \geq 0\}$ can be formally written as,

$$Q(t) \triangleq \sum_{S\in\mathcal{E}} \sum_{j=1}^{y_S} \mathbb{1}\left(M(t_{j,S}) = S\right). \tag{B.6}$$

Theorem B.1 (Generalized PASTA)**.** *$Q(t)/t$ converges to $\sum_{S\in\mathcal{E}} \lambda_S \pi^*_S$, as t goes to $+\infty$, with probability* 1.

Proof. We re-write $\{Q(t),\ t \geq 0\}$ using the notation from [Wol82]:

$$Q(t) = \sum_{S\in\mathcal{E}} \int_0^t \mathbb{1}\left(M(u) = S\right)\, dP_S(u). \tag{B.7}$$

Next, we compute the limit of $Q(t)/t$ as follows,

$$\lim_{t\to+\infty} \frac{Q(t)}{t} = \lim_{t\to+\infty} \sum_{S\in\mathcal{E}} \left(\frac{P_S(t)}{t}\right) \cdot \left(\frac{1}{P_S(t)} \int_0^t \mathbb{1}\left(M(u) = S\right)\, dP_S(u)\right) \tag{B.8}$$

$$= \sum_{S\in\mathcal{E}} \lambda_S \cdot \lim_{t\to+\infty} \frac{1}{t} \int_0^t \mathbb{1}\left(M(u) = S\right) ds \tag{B.9}$$

$$= \sum_{S\in\mathcal{E}} \lambda_S \pi^*_S. \tag{B.10}$$

To obtain (B.9), we used the PASTA property [Wol82] for each term in the sum. Moreover, we used the fact that $\{P_S(t),\ t \geq 0\}$ is Poisson with rate λ_S and therefore $\lim_{t\to+\infty} P_S(t)/t = \lambda_S$. This concludes the proof.

□

A similar result is proven in [RS92, Sect. 3.3] for Markov-modulated Poisson processes.

Corollary B.1 (Generalized PASTA)**.** *For any partition* $(B_i)_{i\in\{1,\dots,l\}}$ *of* $\mathcal{E}$ *such that* $\lambda_S = \lambda_i$ *for any* S *in* B_i*, we have that*

$$\frac{Q(t)}{t} \xrightarrow{t\to+\infty} \sum_{i=1}^{l} \lambda_i \sum_{S\in B_i} \pi^*_S, \quad \textit{w.p. } 1. \tag{B.11}$$

Theorem B.1 and Corollary B.1 provide the average rate of the process $\{Q(t),\ t \geq 0\}$.

B.4 Proof of Proposition 4.2 (Item hit probability, page 50)

The proof uses the generalized PASTA property derived in B.3, and we will redefine the relevant processes to serve our purpose.

We redefine $M(t)$ as the set of cached items in the RND-TTL cache at time t, $S_{\text{RND-TTL}}(t)$. It follows that $\mathcal{E}$ is equal to Ω, the state space of $\{S_{\text{RND-TTL}}(t),\ t \geq 0\}$. We partition Ω into $B_1 = \{S \in \Omega :\ n \in S\}$ and $B_2 = \{S \in \Omega :\ n \notin S\}$. For any S in B_1, we redefine $\{P_S(t),\ t \geq 0\}$ as the request process of n, that is Poisson with rate λ_n by Assumption 4.1. For any S in B_2, we redefine $\{P_S(t),\ t \geq 0\}$ as the request process for n thinned with probability $1 - p^i_n$.

With $\{M(t),\ t \geq 0\}$ and $\{\{P_S(t),\ t \geq 0\}, S \in \Omega\}$ redefined, Corollary B.1 computes the rate of $\{Q(t),\ t \geq 0\}$, defined in (B.6), as $\lambda_n \cdot o_n + \lambda_n(1 - p^i_n)(1 - o_n)$, where $o_n = \sum_{S\in B_1} \pi_S$ and $1 - o_n = \sum_{S\in B_2} \pi_S$. Finally, we only need to show that $Q(t)$ is the number of hits for item n until time t to deduce that the hit probability is equal to $\lambda_n \cdot o_n + \lambda_n(1 - p^i_n)(1 - o_n)$.

In RND-TTL, whenever an item n is in the cache, an exact hit occurs upon a request for n. In other words, if $\{S_{\text{RND-TTL}}(t),\ t \geq 0\}$ is in state B_1, the number of hits for n grows as its request process (and those added hits are all exact). Conversely, when n is not in the cache, only an approximate hit may occur, with probability $1 - p^i_n$. In other words, if $\{S_{\text{RND-TTL}}(t),\ t \geq 0\}$ is in state B_2, the number of hits for n grows as its request process thinned with probability $1 - p^i_n$ (and those added hits are all approximate). It is clear then that the number of hits for n until time t is $Q(t)$, which concludes the proof.

B.5 Proof of Proposition 4.3 (RND-LRU insertion rate, page 52)

The proof is similar to that for Proposition 4.2 in B.4.

We redefine $M(t)$ as the ordered list of cached items in the RND-LRU cache at time t, $L_{\text{RND-LRU}}(t)$. It follows that $\mathcal{E} = \tilde{\Omega}$, the state space of $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$. For a given item n, we partition $\tilde{\Omega}$ into $\tilde{B}_n = \{L \in \tilde{\Omega} :\ L \cap \mathcal{N}^c[n] = \emptyset\}$, $\tilde{B}_{n,m} = \{L \in \tilde{\Omega} :\ m \in L, L \cap \mathcal{N}^c_m[n] = \emptyset\}$ for every $m \in \mathcal{N}(n)$, and $\{L \in \tilde{\Omega} :\ n \in L\}$. For any L in state $\tilde{B}_n$, we redefine $\{P_L(t),\ t \geq 0\}$ as the request process for n that is Poisson with rate λ_n by Assumption 4.1. For any L in state $\tilde{B}_{n,m}$ with $m \in \mathcal{N}(n)$, we redefine $\{P_L(t),\ t \geq 0\}$ as the request process for n thinned with probability $1 - q_m(n)$. For any L such that $n \in L$, we redefine $(P_L(t))$ to be always equal to 0 with probability 1.

With $\{M(t),\ t \geq 0\}$ and $\{\{P_L(t),\ t \geq 0\}, L \in \tilde{\Omega}\}$ redefined, Corollary B.1 computes the

rate of $\{Q(t),\ t \geq 0\}$, defined in (B.6) as

$$\lambda_n \left(\sum_{L \in \tilde{B}_n} \tilde{\pi}_L + \sum_{m \in \mathcal{N}(n)} (1 - q_m(n)) \sum_{K \in \tilde{B}_{n,m}} \tilde{\pi}_K \right). \tag{B.12}$$

Therefore we only need to show that $Q(t)$ is the number of insertions for item n until time t to prove the formula for the insertion rate.

In RND-LRU, when neither n nor any of its neighbors are in the cache, a request for n will correspond to a miss, and thereby n is inserted in the cache. In other words, when $L_{\text{RND-LRU}}(t)$ is in state $\tilde{B}_n$, the number of insertions of n grows as its request process. However, when n is not in the cache but a neighbor is, n may be inserted with some probability. Specifically, if m is the closest neighbor of n in the cache, n will be inserted upon a request with probability $1 - q_m(n)$. That is, when $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$ is in state $\tilde{B}_{n,m}$, the number of insertions of n grows as its request process thinned with probability $1 - q_m(n)$. We deduce that $Q(t)$ is the number of insertions of n until time t, which concludes the proof.

B.6 Proof of Proposition 4.4 (RND-LRU refresh rate, page 52)

As in B.5, we redefine $M(t)$ as the ordered list of cached items in RND-LRU at time t, $L_{\text{RND-LRU}}(t)$. For a given item n, for every $m \in \mathcal{N}^c[n]$, we define the set $\tilde{B}_{m,n} = \{L \in \tilde{\Omega} :\ n \in L, L \cap \mathcal{N}_n^c[m] = \emptyset\}$.

For any L in $\tilde{\Omega}$, we redefine $\{P_L(t),\ t \geq 0\}$ as the aggregation of thinned request processes for items in the subset $\{m \in \mathcal{N}^c[n] :\ L \in \tilde{B}_{m,n}\}$. For every m in the aforementioned subset, the thinning probability of m's request process is $q_n(m)$. Under Assumption 4.1, $\{P_L(t),\ t \geq 0\}$ is Poisson and its rate is given by,

$$\lambda_L = \sum_{m \in \mathcal{N}^c[n]} q_n(m) \lambda_m \mathbb{1}(L \in \tilde{B}_{m,n}). \tag{B.13}$$

With $\{M(t),\ t \geq 0\}$ and $\{\{P_L(t),\ t \geq 0\}, L \in \tilde{\Omega}\}$ redefined, Theorem B.1 computes the rate of $\{Q(t),\ t \geq 0\}$, defined in B.3 as,

$$\lim_{t \to +\infty} \frac{Q(t)}{t} = \sum_{L \in \tilde{\Omega}} \sum_{m \in \mathcal{N}^c[n]} q_n(m) \lambda_m \mathbb{1}(L \in \tilde{B}_{m,n})\, \tilde{\pi}_L \tag{B.14a}$$

$$= \sum_{m \in \mathcal{N}^c[n]} q_n(m) \lambda_m \sum_{L \in \tilde{\Omega}} \mathbb{1}(L \in B_{m,n}) \tilde{\pi}_L \tag{B.14b}$$

$$= \sum_{m \in \mathcal{N}^c[n]} q_n(m) \lambda_m \sum_{L \in \tilde{B}_{m,n}} \tilde{\pi}_L. \tag{B.14c}$$

It suffices then to show that $\{Q(t),\ t \geq 0\}$ is the number of times n's timer is refreshed until time t to deduce that the refresh rate $\tilde{\lambda}_n^r$ is given by (B.14c).

In RND-LRU, whenever an item n is in the cache, a hit on n occurs upon a request for m, with probability $q_n(m)$, if n is the item in cache closest to m, and as a result n's timer is refreshed.

This holds for any neighbor m of item n (including n). In other words, for any neighbor m such that $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$ is in state $\tilde{B}_{m,n}$, m's request process thinned with probability $q_n(m)$ contributes to the growth of the number of timer refreshes for n. No other process contributes to the refresh counting process, thereby, when $\{L_{\text{RND-LRU}}(t),\ t \geq 0\}$ is in state L, the number of times n's timer is refreshed grows as the Poisson process $P_L(t)$ with rate λ_L given in (B.13). We conclude that $\{Q(t),\ t \geq 0\}$ is the number of times n's timer is refreshed until time t, finishing the proof.

B.7 Proof of Lemma 4.1 ($T_C(\mathrm{o})$ is a singleton, page 55)

Let $\mathbf{o} \in \Delta_C$. We first observe that $F(\mathbf{o}, \cdot)$ is an increasing and continuous function in $\mathbb{R}^+$ since it is the sum of increasing and continuous functions in $\mathbb{R}^+$ (see (4.38)). As $F(\mathbf{o}, 0) = -C < 0$, proving the existence of a root T_0 of $F(\mathbf{o}, \cdot)$ boils down to proving that $\lim_{T\to+\infty} F(\mathbf{o}, T) > 0$, thanks to the intermediate value theorem. We prove next that $\lim_{T\to+\infty} F(\mathbf{o}, T)$ is indeed strictly positive.

For a given $\mathbf{o} \in \Delta_C$, we consider the set $\mathcal{M}_\mathbf{o}$ having the items with occupancy equal to 1. In other words, $\mathcal{M}_\mathbf{o} = \{n \in \mathcal{I} :\ o_n = 1\}$ (note that we may have $\mathcal{M}_\mathbf{o} = \emptyset$). We can write

$$\mathbf{o} \in \Delta_C \overset{(4.35)}{\Longrightarrow} |\mathcal{M}_\mathbf{o}| \leq C \tag{B.15}$$

$$\overset{(4.36)}{\Longrightarrow} \left| \bigcup_{m\in\mathcal{M}_\mathbf{o}} \mathcal{N}(m) \right| < N - C \tag{B.16}$$

$$\overset{\substack{\text{same}\\ \text{set}}}{\Longrightarrow} \left| \left\{ n \in \mathcal{I} :\ \prod_{m\in\mathcal{N}(n)} (1 - o_m) = 0 \right\} \right| < N - C \tag{B.17}$$

$$\Longrightarrow \left| \left\{ n \in \mathcal{I} :\ \prod_{m\in\mathcal{N}(n)} (1 - o_m) > 0 \right\} \right| > C \tag{B.18}$$

$$\overset{\substack{(4.25)\\(4.31)}}{\Longrightarrow} |\{n \in \mathcal{I} :\ E_n(\mathbf{o}) > 0\}| > C \tag{B.19}$$

$$\overset{(4.29)}{\Longrightarrow} \left| \left\{ n \in \mathcal{I} :\ \lim_{T\to+\infty} g\left(R_n(\mathbf{o}), E_n(\mathbf{o}), T\right) = 1 \right\} \right| > C \tag{B.20}$$

$$\overset{(4.38)}{\Longrightarrow} \lim_{T\to+\infty} F(\mathbf{o}, T) > 0\,. \tag{B.21}$$

The implication (B.15) is due to the definition of Δ_C (4.35), whereas (B.16) follows from the no-coverage condition (4.36). Observing that any item n in the set of all neighbors $\bigcup_{m\in\mathcal{M}_\mathbf{o}} \mathcal{N}(m)$ has at least one neighbor with occupancy 1 (the one from the set $\mathcal{M}_\mathbf{o}$), we can rewrite (B.16) as (B.17). The inequality in (B.18) follows by considering the complementary set. From the definition of the function $\mathbf{E}$ (see (4.25) and (4.31)), it comes that the set in (B.18) is included in the set in (B.19), which justifies writing (B.19). The implication (B.20) follows since $\lim_{T\to+\infty} g\left(R_n(\mathbf{o}), E_n(\mathbf{o}), T\right) = 1$ whenever $E_n(\mathbf{o}) > 0$ (see (4.29)). Using the definition of $F(\mathbf{o}, T)$ in (4.38), it is straightforward to write (B.21). We deduce then the existence of a root of $F(\mathbf{o}, \cdot)$, namely T_0.

The last step of the proof is to show the uniqueness of the root T_0. Given that under the no-coverage condition, there are at least $C + 1$ functions $g\left(R_n(\mathbf{o}), E_n(\mathbf{o}), \cdot\right)$ that are strictly

increasing (see (4.29) and (B.20)). Consequently, $F(\mathbf{o}, \cdot)$ is strictly increasing and T_0 is unique, which concludes the proof.

B.8 Proof of Lemma 4.2 (Differentiability of t_C, page 56)

We follow the steps of the proof of [Oli12][Th.1]. By proving the existence of the partial derivatives of t_C, we show that t_C is differentiable. Let $\mathbf{o} \in \Delta_C$ and for $j \in \mathcal{I}$ we define $\mathbf{e}_j$ as the N-dimensional vector with all components 0 except the jth one which is 1. We want to show the existence of

$$\lim_{\epsilon \to 0} \frac{t_C(\mathbf{o} + \epsilon \mathbf{e}_j) - t_C(\mathbf{o})}{\epsilon} \,. \tag{B.22}$$

Let $A = (\mathbf{o}, t_C(\mathbf{o}))$ and $B_j = (\mathbf{o} + \epsilon \mathbf{e}_j, t_C(\mathbf{o} + \epsilon \mathbf{e}_j))$. Since F is the sum of continuously differentiable functions (see (4.38)), then F is continuously differentiable over $\Delta_C \times \mathbb{R}^+$. By the mean value theorem, there exists some δ between 0 and 1 such that

$$F(B_j) - F(A) = \nabla F((1-\delta)A + \delta B_j) \cdot (B_j - A). \tag{B.23}$$

To ease the writing we define $L_j = (1-\delta)A + \delta B_j$. From the definition of t_C in (4.39), we have that $F(A) = F(B_j) = 0$. We expand then the right-hand side of (B.23) to write

$$\begin{aligned} &\epsilon \frac{\partial F}{\partial o_j}(L_j) + \big(t_C(\mathbf{o} + \epsilon \mathbf{e}_j) - t_C(\mathbf{o})\big) \frac{\partial F}{\partial T}(L_j) = 0 \\ &\qquad \Leftrightarrow \quad \frac{t_C(\mathbf{o} + \epsilon \mathbf{e}_j) - t_C(\mathbf{o})}{\epsilon} = -\frac{\partial F}{\partial o_j}(L_j) \left(\frac{\partial F}{\partial T}(L_j) \right)^{-1}, \end{aligned} \tag{B.24}$$

where we used the fact that $\frac{\partial F}{\partial T} > 0$ to write the last equality. (Recall from B.7 that $F(\mathbf{o}, \cdot)$ is continuous and strictly increasing.) When $\epsilon \to 0$, L_j converges to A and (B.24) boils down to (4.41), completing the proof.

B.9 Proof of Proposition 4.7 (page 59)

The Jacobian matrix of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is a rectangular matrix with m rows and n columns. The element in the ith row and jth column represents the partial derivative of the ith component of the function f with respect to the jth variable.

Let $\mathbf{o} \in \Delta_C$ and $P_i(\mathbf{o}) = (R_i(\mathbf{o}), E_i(\mathbf{o}), t_C(\mathbf{o}))$. We use the chain rule on $\mathbf{G}$ to compute the partial derivative of its ith component, G_i, with respect to the jth variable.

$$\frac{\partial G_i}{\partial o_j}(\mathbf{o}) = \frac{\partial g}{\partial x_1}(P_i(\mathbf{o})) \frac{\partial R_i}{\partial o_j}(\mathbf{o}) + \frac{\partial g}{\partial x_2}(P_i(\mathbf{o})) \frac{\partial E_i}{\partial o_j}(\mathbf{o}) + \frac{\partial g}{\partial x_3}(P_i(\mathbf{o})) \frac{\partial t_C}{\partial o_j}(\mathbf{o}). \tag{B.25}$$

Observe that

$$\left(\frac{\partial g}{\partial x_1}(P_i(\mathbf{o})) \frac{\partial R_i}{\partial o_j}(\mathbf{o}) \right)_{\substack{1 \le i \le N \\ 1 \le j \le N}} = \mathrm{Diag}\left(\boldsymbol{\partial_1 g}\right) \cdot \boldsymbol{J_R}(\mathbf{o}), \tag{B.26}$$

and

$$\left(\frac{\partial g}{\partial x_2}(P_i(\mathbf{o}))\frac{\partial E_i}{\partial o_j}(\mathbf{o})\right)_{\substack{1\leq i\leq N\\1\leq j\leq N}} = \mathrm{Diag}\left(\boldsymbol{\partial_2 g}\right)\cdot \boldsymbol{J_E}(\mathbf{o}). \tag{B.27}$$

To prove (4.47), it suffices to prove that

$$\left(\frac{\partial g}{\partial x_3}(P_i(\mathbf{o}))\frac{\partial t_C}{\partial o_j}(\mathbf{o})\right)_{\substack{1\leq i\leq N\\1\leq j\leq N}} = -\frac{\boldsymbol{\partial_3 g}^\intercal}{\boldsymbol{\partial_3 g}\cdot\mathbf{1}}\cdot\left(\boldsymbol{\partial_1 g}\cdot\boldsymbol{J_R}(\mathbf{o})+\boldsymbol{\partial_2 g}\cdot\boldsymbol{J_E}(\mathbf{o})\right). \tag{B.28}$$

To this aim, we compute the partial derivatives of t_C. These are given in Lemma 4.2 as follows:

$$\frac{\partial t_C}{\partial o_j}(\mathbf{o}) = -\frac{\partial F}{\partial o_j}(\mathbf{o}, t_C(\mathbf{o}))\cdot\left(\frac{\partial F}{\partial T}(\mathbf{o}, t_C(\mathbf{o}))\right)^{-1}. \tag{B.29}$$

From the definition of F (see (4.38)) and the chain rule, we get

$$\frac{\partial F}{\partial o_j}(\mathbf{o}, t_C(\mathbf{o})) = \sum_{i\in\mathcal{I}}\frac{\partial g}{\partial x_1}(P_i(\mathbf{o}))\frac{\partial R_i}{\partial o_j}(\mathbf{o}) + \sum_{i\in\mathcal{I}}\frac{\partial g}{\partial x_2}(P_i(\mathbf{o}))\frac{\partial E_i}{\partial o_j}(\mathbf{o}), \tag{B.30}$$

$$\frac{\partial F}{\partial T}(\mathbf{o}, t_C(\mathbf{o})) = \sum_{i\in\mathcal{I}}\frac{\partial g}{\partial x_3}(P_i(\mathbf{o})) = \boldsymbol{\partial_3 g}\cdot\mathbf{1}\,. \tag{B.31}$$

Substituting (B.30) and (B.31) into (B.29) we deduce (B.28). This concludes the proof.

B.10 Time Complexity of Single Iteration in Algorithm 3

In each iteration of Algorithm 3, we compute the functions $\mathbf{E}(\mathbf{o})$, $\mathbf{R}(\mathbf{o})$, $t_C(\mathbf{o})$ and finally $\mathbf{g}\left(\mathbf{E}(\mathbf{o}), \mathbf{R}(\mathbf{o}), t_C(\mathbf{o})\right)$ for a given $\mathbf{o}\in\Delta_C$. It is easy to deduce from (4.31), (4.32) and (4.33) that the time complexity for computing the functions $\mathbf{E}$, $\mathbf{R}$ and $\mathbf{g}$ is $\mathcal{O}(\mathcal{K})$, $\mathcal{O}(\mathcal{K})$ and $\mathcal{O}(N)$, respectively. The computation of $t_C(\mathbf{o})$ can be done either through bisection or Newton's method thanks to Lemma 4.2. If we consider that the number of iterations in the computation of $t_C(\mathbf{o})$ is constant, then the time complexity for computing $t_C(\mathbf{o})$ is $\mathcal{O}(\mathcal{K})$. Finally, we deduce that the time complexity for one iteration of Algorithm 3 is $\mathcal{O}(\mathcal{K})$.

B.11 Proof of Proposition 4.8 (Properties of $Y(\mathbf{o})$, page 60)

Let $\beta\in[0,1]$ and $a=\max\left\{0, \frac{\gamma-1}{\gamma+1-\eta}\right\}$. For a given $\mathbf{o}\in\Delta_C$, in order to show that $(a,1)\subset Y(\mathbf{o})$, we first find an upper bound on the squared spectral norm of $\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o})$. We next observe that should (4.55) hold, then the upper bound that we found would be smaller than 1 if and only if β lies within the interval $(a,1)$, concluding thereby that $(a,1)\subset Y(\mathbf{o})$.

We derive now an upper bound of the square of the spectral norm of $\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o})$. In our derivations, we denote the spectral radius of a matrix $\boldsymbol{M}$ as $\rho(\boldsymbol{M})$ and use $\boldsymbol{I}_N$ for the N-dimensional identity

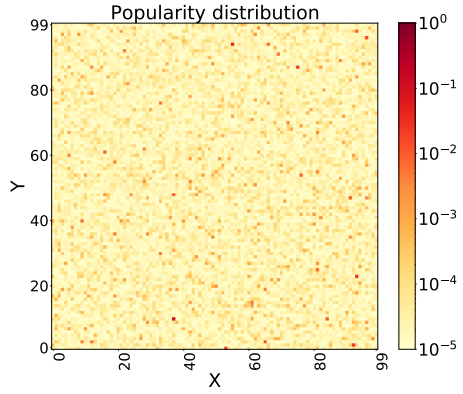


Figure B.1: Spatial popularity distribution: Zipf with exponent $z = 1.0$.

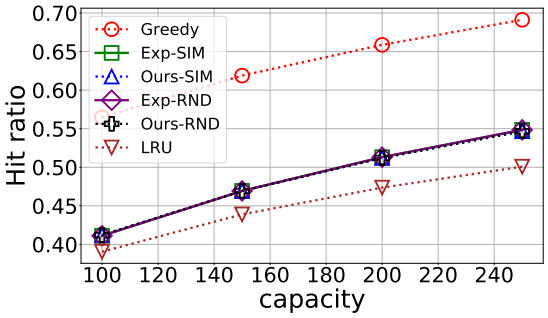
matrix. Letting $\boldsymbol{A} = \boldsymbol{J}_{\mathbf{G}}(\mathbf{o})$, we can start from (4.47) to write

$$\begin{aligned}
\|\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o})\|_2^2 &= \left(\|(1-\beta)\boldsymbol{A} + \beta \boldsymbol{I}_N\|_2\right)^2 && \text{(B.32a)}\\
&= \rho\left((1-\beta)^2\boldsymbol{A}\boldsymbol{A}^\intercal + \beta(1-\beta)(\boldsymbol{A}+\boldsymbol{A}^\intercal) + \beta^2\boldsymbol{I}_N\right) && \text{(B.32b)}\\
&\leq \rho\left((1-\beta)^2\boldsymbol{A}\boldsymbol{A}^\intercal + \beta(1-\beta)(\boldsymbol{A}+\boldsymbol{A}^\intercal)\right) + \beta^2 && \text{(B.32c)}\\
&= \|(1-\beta)^2\boldsymbol{A}\boldsymbol{A}^\intercal + \beta(1-\beta)(\boldsymbol{A}+\boldsymbol{A}^\intercal)\|_2 + \beta^2 && \text{(B.32d)}\\
&\leq \|(1-\beta)^2\boldsymbol{A}\boldsymbol{A}^\intercal\|_2 + \|\beta(1-\beta)(\boldsymbol{A}+\boldsymbol{A}^\intercal)\|_2 + \beta^2 && \text{(B.32e)}\\
&= (1-\beta)^2\gamma + \beta(1-\beta)\eta + \beta^2 && \text{(B.32f)}\\
&= (\gamma + 1 - \eta)\beta^2 - (2\gamma - \eta)\beta + \gamma. && \text{(B.32g)}
\end{aligned}$$

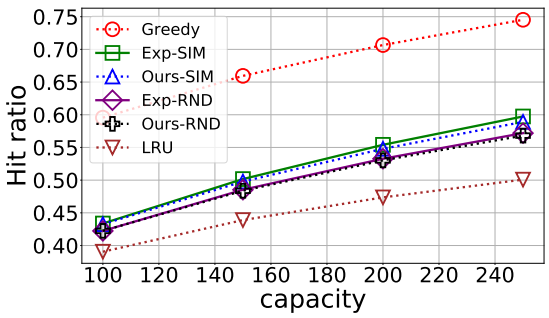
Equation (B.32b) follows from the definition of the spectral norm, i.e., $\|\boldsymbol{M}\|_2 = \sqrt{\rho(\boldsymbol{M}\boldsymbol{M}^\intercal)}$. We can write (B.32c) by observing that $\rho(\boldsymbol{M} + \beta^2\boldsymbol{I}) \leq \rho(\boldsymbol{M}) + \beta^2$ for any matrix $\boldsymbol{M}$. Given that the spectral norm of a symmetric matrix coincides with its spectral radius, (B.32d) follows. We can write (B.32e) thanks to the triangular inequality, and (B.32f) using again that $\|\boldsymbol{M}\|_2 = \rho(\boldsymbol{M})$ when $\boldsymbol{M}$ is symmetric and then the definitions of γ and η in (4.53) and (4.54), respectively. Equation (B.32g) readily follows. The upper bound expressed in (B.32g) is less than 1 when β is in the interval $\left(\max\left\{0, \frac{\gamma-1}{\gamma+1-\eta}\right\}, 1\right)$ under the condition that $\eta < \min\{2, \gamma+1\}$.

B.12 Additional Experiments

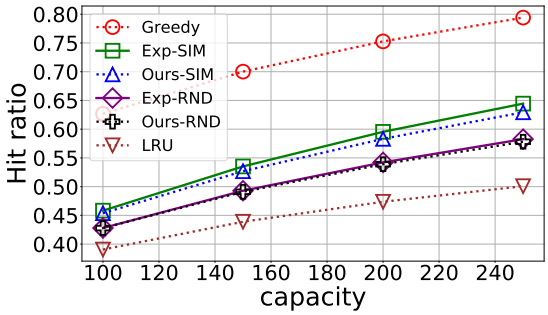
Zipf trace on a grid. Each item corresponds to two features, characterized by a point in a grid, $\mathcal{I} = [0..99]^2$. The total number of items is $|\mathcal{I}| = 10^4$, and the dissimilarity function between items $\mathrm{dis}(\cdot,\cdot)$ is the Euclidean distance. Neighbors of item (x, y) at the same distance are ordered counterclockwise starting from the item to the right, i.e., from $(x + a, y)$ with $a > 0$. Traces are generated in an IRM fashion where the popularity distribution for an item in $\mathcal{I}$ is Zipf. We generate 50 synthetic streams using Zipf exponent $z = 1.0$ and having in each stream $r = 2 \cdot 10^5$ requests for items in $\mathcal{I}$. Figure B.1 illustrates the spatial popularity distribution for $z = 1.0$.



(a) $d = 1$, 25 iterations



(b) $d = 1.5$, 20 iterations



(c) $d = 2.0$, 15 iterations

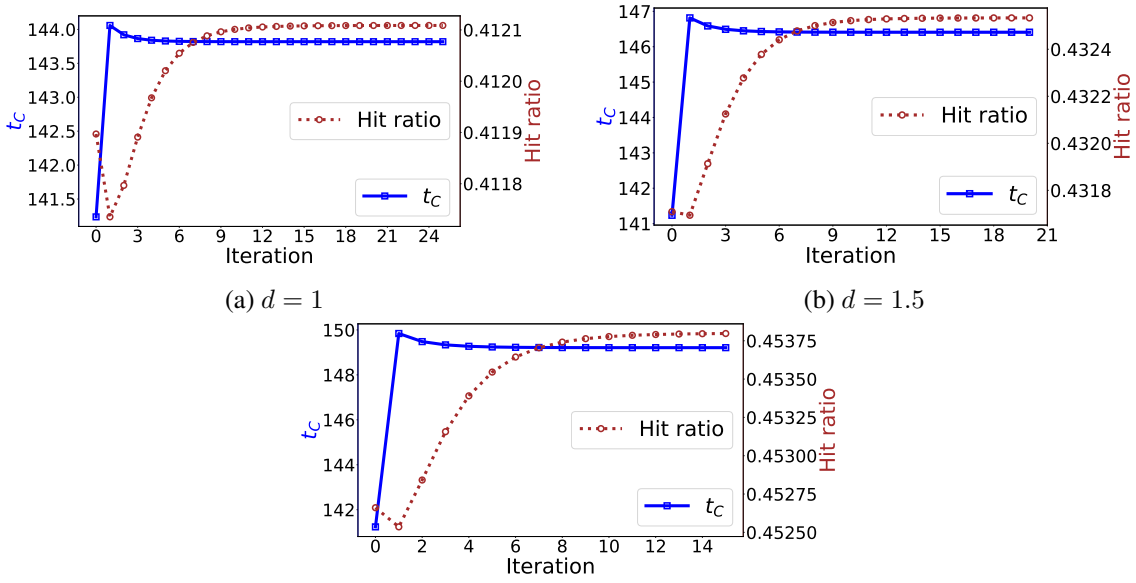
Figure B.2: Hit ratio versus cache capacity: $r = 2 \cdot 10^5$, Zipf with exponent $z = 1.0$, $\beta = 0.5$.

Hit ratio computation. We empirically compute the hit ratio of similarity cache mechanisms using SIM-LRU and RND-LRU on the Zipf trace with parameter $z = 1$. We consider three similarity threshold values $d = 1$, $d = 1.5$ and $d = 2$. Additionally, given two distinct items n and m, we set RND-LRU parameters $q_n(m)$ to $(\mathrm{dis}(n, m))^{-2}$ if $\mathrm{dis}(n, m) \leq d$ and 0 otherwise. Note that when $d = 1$, RND-LRU reduces to SIM-LRU. Results for the hit ratio are averaged over the 50 request processes. We refer to the empirical results for SIM-LRU and RND-LRU as Exp-SIM and Exp-RND, respectively. Our approach utilizes Algorithm 3 with parameter $\beta = 0.5$ and a stopping condition determined by a fixed number of iterations. Algorithm 3 enables us to estimate the approximate hit probabilities for all items, $\mathbf{h}$, and subsequently determine the overall cache hit ratio H. We refer to our results, for SIM-LRU and RND-LRU, as Ours-SIM and Ours-RND, respectively. Possible alternative methods to estimate the hit ratio are presented in Section 4.6.2, like LRU and Greedy.

Figure B.2 shows the empirical hit ratio along with its estimates obtained through different approaches. The depicted results confirm the accuracy of our approach in approximating RND-LRU's hit ratio.

Figure B.3 illustrates the values of the characteristic time t_C and the hit ratio H over different iterations of Algorithm 3. The findings shown in Figure B.3 validate that Algorithm 3 converges within a few iterations.

Table B.1 provides details on the average runtime per iteration in Algorithm 3 for $d = 1.0$, $d = 1.5$ and $d = 2.0$ when the number of iterations is respectively 25, 20 and 15.



(a) $d = 1$ (b) $d = 1.5$

(c) $d = 2.0$

Figure B.3: Characteristic time t_C and hit ratio in different iterations of Algorithm 3 for SIM-LRU: Zipf with exponent $z = 1.0$, $\beta = 0.5$.

Table B.1: Average runtime per iteration in Algorithm 3: $C = 100$, Zipf with $z = 1.0$, $\beta = 0.5$.

Similarity threshold d	Number of neighbors $\lvert\mathcal{N}^c[n]\rvert$	Average runtime per iteration
1.0	5	0.7 seconds
1.5	9	2.5 seconds
2.0	13	5.2 seconds

B.13 Implementation Details

When computing $\|\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o})\|$, we follow a specific procedure. First, we use the formula in (4.48) to compute the Jacobian matrix $\boldsymbol{J}_{\mathbf{G}}$. To compute the Jacobian matrices $\boldsymbol{J}_{\mathbf{E}}$ and $\boldsymbol{J}_{\mathbf{R}}$, we utilize a function from the torch.autograd Pytorch's library [Pas+19b]. However, we do not use this function in the computation of the vectors $\boldsymbol{\partial_1 g}$, $\boldsymbol{\partial_2 g}$ and $\boldsymbol{\partial_3 g}$ to avoid potential errors that may arise from floating point precision. Instead, we implement these vectors separately, ensuring accurate results. The vectors $\partial \mathbf{g}_1$, $\partial \mathbf{g}_2$, and $\partial \mathbf{g}_3$ are implemented separately without relying on the aforementioned PyTorch function. This is done to ensure accurate results by avoiding potential errors that may arise from floating point precision.

Bibliography

Bibliography

[ACN00] Dimitris Achlioptas, Marek Chrobak, and John Noga. "Competitive analysis of randomized paging algorithms". In: *Theoretical Computer Science* 234 (2000). DOI: https://doi.org/10.1016/S0304-3975(98)00116-9 (cit. on p. 3).

[ACN16] Sara Alouf, Nicaise Choungmo Fofack, and Nedko Nedkov. "Performance models for hierarchy of caches: Application to modern DNS caches". In: *Performance Evaluation* 97 (Mar. 2016), pp. 57–82. DOI: 10.1016/j.peva.2016.01.001. URL: https://inria.hal.science/hal-01258189 (cit. on p. 45).

[AG22] Sebastian Allmeier and Nicolas Gast. "Mean Field and Refined Mean Field Approximations for Heterogeneous Systems: It Works!" In: *ACM Meas. Anal. Comput. Syst.* 6 (2022). DOI: 10.1145/3508033 (cit. on p. 71).

[And+13] 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". In: *26th Annual Conference on Learning Theory*. Vol. 30. Proceedings of Machine Learning Research. PMLR, 2013. URL: https://proceedings.mlr.press/v30/Andrew13.html (cit. on p. 3).

[BAN22a] Younes Ben Mazziane, Sara Alouf, and Giovanni Neglia. "A Formal Analysis of the Count-Min Sketch with Conservative Updates". In: *IEEE INFOCOM - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2022. DOI: 10.1109/INFOCOMWKSHPS54753.2022.9798146 (cit. on pp. 6, 7, 17, 20, 21).

[BAN22b] Younes Ben Mazziane, Sara Alouf, and Giovanni Neglia. "Analyzing Count Min Sketch with Conservative Updates". In: *Computer Networks* 217 (2022), p. 109315. DOI: https://doi.org/10.1016/j.comnet.2022.109315 (cit. on pp. 6, 7, 17, 28).

[Bas+18] Soumya Basu, Aditya Sundarrajan, Javad Ghaderi, Sanjay Shakkottai, and Ramesh Sitaraman. "Adaptive TTL-based caching for content delivery". In: *IEEE/ACM transactions on networking* 26.3 (2018), pp. 1063–1077 (cit. on p. 45).

[BBS20] Rajarshi Bhattacharjee, Subhankar Banerjee, and Abhishek Sinha. "Fundamental Limits on the Regret of Online Network-Caching". In: *ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. 2020. DOI: 10.1145/3392143 (cit. on pp. 3, 4, 31, 34, 37).

[BdN11] Giuseppe Bianchi, Nico d'Heureuse, and Saverio Niccolini. "On-demand time-decaying Bloom filters for telemarketer detection". In: *ACM SIGCOMM Comput. Communi. Rev.* 41 (2011). DOI: 10.1145/2043165.2043167 (cit. on p. 9).

[Bel66] L. A. Belady. "A study of replacement algorithms for a virtual-storage computer". In: *IBM Systems Journal* 5 (1966). DOI: 10.1147/sj.52.0078 (cit. on p. 3).

[Ben+17] Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. "Optimal elephant flow detection". In: *IEEE INFOCOM - IEEE Conference on Computer Communications*. 2017. DOI: 10.1109/INFOCOM.2017.8057216 (cit. on pp. 5, 9).

[Ben+22] Younes Ben Mazziane, Sara Alouf, Giovanni Neglia, and Daniel Sadoc Menasche. "Computing the Hit Rate of Similarity Caching". In: *IEEE GLOBECOM - IEEE Global Communications Conference*. 2022. DOI: 10.1109/GLOBECOM48099.2022.10000890 (cit. on p. 7).

[Ben+23] Younes Ben Mazziane, Francescomaria Faticanti, Giovanni Neglia, and Sara Alouf. "No-Regret Caching with Noisy Request Estimates". In: *IEEE VCC - IEEE Virtual Conference on Communications*. 2023 (cit. on p. 7).

[Ben+24] Younes Ben Mazziane, Sara Alouf, Giovanni Neglia, and Daniel Sadoc Menasche. "TTL model for an LRU-based similarity caching policy". In: *Computer Networks* (2024), p. 110206. DOI: https://doi.org/10.1016/j.comnet.2024.110206 (cit. on pp. 7, 8).

[Ber+00] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. "Balanced allocations: The heavily loaded case". In: *thirty-second annual ACM symposium on Theory of computing*. 2000 (cit. on p. 69).

[Ber+10] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. "Space-optimal heavy hitters with strong error bounds". In: *ACM Trans. Database Syst.* 35 (2010). DOI: 10.1145/1862919.1862923 (cit. on p. 5).

[Ber+14] Daniel S Berger, Philipp Gland, Sahil Singla, and Florin Ciucu. "Exact analysis of TTL cache networks". In: *Performance Evaluation* 79 (2014), pp. 2–23 (cit. on p. 45).

[Bia+12] Giuseppe Bianchi, Ken Duffy, Douglas Leith, and Vsevolod Shneer. "Modeling conservative updates in multi-hash approximate count sketches". In: *24th International Teletraffic Congress (ITC 24)*. 2012 (cit. on pp. 10, 12, 16–18, 20, 21, 26, 69).

[BM03] Andrei Broder and Michael Mitzenmacher. "Network Applications of Bloom Filters: A Survey". In: *Internet Mathematics* 1 (2003) (cit. on pp. 10, 13).

[BOO07] Tolga Bektas, Osman Oguz, and Iradj Ouveysi. "Designing cost-effective content distribution networks". In: *Computers & Operations Research* 34.8 (2007), pp. 2436–2449 (cit. on p. 27).

[Bre+99] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. "Web caching and Zipf-like distributions: evidence and implications". In: *IEEE INFOCOM - IEEE Conference on Computer Communications*. 1999. DOI: 10.1109/INFCOM.1999.749260 (cit. on pp. 2, 36).

[BSH17] Daniel S. Berger, Ramesh 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)*. 2017. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/berger (cit. on p. 4).

[CA13] Nicaise Choungmo Fofack and Sara Alouf. “Modeling modern DNS caches”. In: *VALUETOOLS-7th International Conference on Performance Evaluation Methodologies and Tools*. 2013, pp. 184–193 (cit. on p. 45).

[Cas11] Giuliano Casale. “Building accurate workload models using Markovian arrival processes”. In: *ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. 2011. DOI: 10.1145/1993744.1993783 (cit. on p. 2).

[CCF04] Moses Charikar, Kevin Chen, and Martin Farach-Colton. “Finding frequent items in data streams”. In: *Theoretical Computer Science* 312 (2004). DOI: https://doi.org/10.1016/S0304-3975(03)00400-6 (cit. on p. 5).

[CH10] Graham Cormode and Marios Hadjieleftheriou. “Methods for finding frequent items in data streams”. In: *The VLDB Journal* 19 (2010) (cit. on pp. 9, 19).

[Cha+07] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. “I tube, you tube, everybody tubes: analyzing the world’s largest user generated content video system”. In: *7th ACM SIGCOMM Conference on Internet Measurement*. 2007. DOI: 10.1145/1298306.1298309 (cit. on p. 2).

[Che+21] Peiqing Chen, Yuhan Wu, Tong Yang, Junchen Jiang, and Zaoxing Liu. “Precise error estimation for sketch-based flow measurement”. In: *21st ACM Internet Measurement Conf.* 2021, pp. 113–121 (cit. on pp. 10, 12, 13, 20, 21).

[Che98] Ludmila Cherkasova. *Improving WWW proxies performance with greedy-dual-size-frequency caching policy*. Hewlett-Packard Laboratories Palo Alto, CA, USA, 1998 (cit. on p. 4).

[Cho+14] Nicaise Choungmo Fofack, Philippe Nain, Giovanni Neglia, and Don Towsley. “Performance evaluation of hierarchical TTL-based cache networks”. In: *Computer Networks* 65 (2014), pp. 212–231 (cit. on pp. 40, 45).

[CK03] Edith Cohen and Haim Kaplan. “Proactive caching of DNS records: Addressing a performance bottleneck”. In: *Computer Networks* 41.6 (2003), pp. 707–726 (cit. on p. 45).

[CKV09] Flavio Chierichetti, Ravi Kumar, and Sergei Vassilvitskii. “Similarity caching”. In: *Twenty-Eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. 2009. DOI: 10.1145/1559795.1559815 (cit. on p. 5).

[CKZ01] Edith Cohen, Haim Kaplan, and Uri Zwick. “Competitive Analysis of the LRFU Paging Algorithm”. In: *Algorithms and Data Structures*. 2001 (cit. on p. 4).

[CM03] Saar Cohen and Yossi Matias. “Spectral bloom filters”. In: *ACM SIGMOD International Conference on Management of Data*. 2003. DOI: 10.1145/872757.872787 (cit. on pp. 6, 9, 12).

[CM05a] Graham Cormode and S. Muthukrishnan. “An improved data stream summary: the count-min sketch and its applications”. In: *Journal of Algorithms* 55 (2005). DOI: https://doi.org/10.1016/j.jalgor.2003.12.001 (cit. on pp. 5, 9, 10, 12, 14, 16, 21, 38).

[CM05b] Graham Cormode and S. Muthukrishnan. “Summarizing and Mining Skewed Data Streams”. In: *SIAM International Conference on Data Mining (SDM)*. 2005. DOI: 10.1137/1.9781611972757.5 (cit. on pp. 4, 5, 9, 10, 12, 14, 20, 21).

[CTW02] Hao Che, Ye Tung, and Zhijun Wang. “Hierarchical Web caching systems: modeling, design and experimental results”. In: *IEEE Journal on Selected Areas in Communications* 20 (2002). DOI: 10.1109/JSAC.2002.801752 (cit. on pp. 4, 40, 45, 46).

[De +14] Steven De Rooij et al. “Follow the leader if you can, hedge if you must”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1281–1316 (cit. on p. 31).

[Deh+17] Mostafa Dehghan, Bo Jiang, Anand Seetharam, Ting He, Theodoros Salonidis, Jim Kurose, Don Towsley, and Ramesh Sitaraman. “On the Complexity of Optimal Request Routing and Content Caching in Heterogeneous Cache Networks”. In: *IEEE/ACM Transactions on Networking* 25 (2017). DOI: 10.1109/TNET.2016.2636843 (cit. on p. 3).

[DGN17] U. Drolia, K. Guo, and P. Narasimhan. “Precog: Prefetching for image recognition applications at the edge”. In: *ACM/IEEE Symposium on Edge Computing*. 2017 (cit. on p. 39).

[DT90] Asit Dan and Don Towsley. “An approximate analysis of the LRU and FIFO buffer replacement schemes”. In: *ACM SIGMETRICS conference on Measurement and modeling of computer systems*. 1990, pp. 143–152 (cit. on p. 40).

[EF15] Gil Einziger and Roy Friedman. “A formal analysis of conservative update based approximate counting”. In: *International Conference on Computing, Networking and Communications (ICNC)*. 2015. DOI: 10.1109/ICCNC.2015.7069350 (cit. on pp. 10, 12, 13, 16, 17, 20, 21, 79).

[EFM17] Gil Einziger, Roy Friedman, and Ben Manes. “TinyLFU: A Highly Efficient Cache Admission Policy”. In: *ACM Trans. Storage* 13 (2017). DOI: 10.1145/3149371 (cit. on pp. 4, 9, 28, 69).

[EV02] Cristian Estan and George Varghese. “New directions in traffic measurement and accounting”. In: *SIGCOMM Comput. Commun. Rev.* 32 (2002). DOI: 10.1145/964725.633056 (cit. on pp. 4, 6, 9, 12).

[Fag77] Ronald Fagin. “Asymptotic miss ratios over independent references”. In: *Journal of Computer and System Sciences* 14 (1977). DOI: https://doi.org/10.1016/S0022-0000(77)80014-7 (cit. on pp. 2, 4, 10, 14, 27, 40, 45, 46, 52, 62, 71).

[Fal+08] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. “A metric cache for similarity search”. In: *ACM Workshop on Large-Scale Distributed Systems for Information Retrieval*. 2008. DOI: 10.1145/1458469.1458473 (cit. on pp. 2, 5, 39, 41).

[Fin+22] A. Finamore, J. Roberts, M. Gallo, and D. Rossi. “Accelerating Deep Learning Classification with Error-controlled Approximate-key Caching”. In: *IEEE INFOCOM - IEEE Conference on Computer Communications*. 2022 (cit. on p. 39).

[FK23a] Éric Fusy and Gregory Kucherov. “Count-Min Sketch with Variable Number of Hash Functions: An Experimental Study”. In: *String Processing and Information Retrieval: 30th International Symposium, SPIRE*. 2023. DOI: 10.1007/978-3-031-43980-3_17 (cit. on p. 10).

[FK23b] Éric Fusy and Gregory Kucherov. “Phase Transition in Count Approximation by Count-Min Sketch with Conservative Updates”. In: *Algorithms and Complexity: 13th International Conference, CIAC 2023*. 2023. DOI: 10.1007/978-3-031-30448-4_17 (cit. on p. 10).

[FRP16] Andrés Ferragut, Ismael Rodriguez, and Fernando Paganini. “Optimizing TTL Caches under Heavy-Tailed Demands”. In: *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*. 2016. DOI: 10.1145/2896377.2901459 (cit. on pp. 2, 45).

[FRR12] Christine Fricker, Philippe Robert, and James Roberts. “A versatile and accurate approximation for LRU cache performance”. In: *2012 24th International Teletraffic Congress (ITC 24)*. IEEE. Krakow, Poland, 2012, pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=2414276.2414286 (cit. on pp. 40, 45, 46, 52, 68).

[GDC12] Amit Goyal, Hal Daumé III, and Graham Cormode. “Sketch algorithms for estimating point queries in NLP”. In: *Conf. on empirical methods in natural language processing and computational natural language learning*. 2012 (cit. on p. 9).

[GLM16] Michele Garetto, Emilio Leonardi, and Valentina Martina. “A unified approach to the performance analysis of caching systems”. In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 1.3 (2016), pp. 1–28 (cit. on p. 45).

[GLN21] M. Garetto, E. Leonardi, and G. Neglia. “Content placement in networks of similarity caches”. In: *Computer Networks* 201 (2021), p. 108570 (cit. on p. 39).

[GV17] Nicolas Gast and Benny Van Houdt. “TTL approximations of the cache replacement algorithms LRU (m) and h-LRU”. In: *Performance Evaluation* 117 (2017), pp. 33–57 (cit. on pp. 4, 45).

[Has+23] Gerhard Hasslinger, Mahshid Okhovatzadeh, Konstantinos Ntougias, Frank Hasslinger, and Oliver Hohlfeld. “An overview of analysis methods and evaluation results for caching strategies”. In: *Computer Networks* 228 (2023), p. 109583 (cit. on p. 45).

[Haz16] Elad Hazan. “Introduction to Online Convex Optimization”. In: *Foundations and Trends® in Optimization* 2 (2016). DOI: 10.1561/2400000013 (cit. on pp. 4, 29).

[Hsu+19] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. “Learning-Based Frequency Estimation Algorithms”. In: *ICLR*. 2019, pp. 1–20 (cit. on pp. 9, 18).

[JBB03] J. Jung, A.W. Berger, and Hari Balakrishnan. “Modeling TTL-based Internet caches”. In: *IEEE INFOCOM - IEEE Conference on Computer Communications*. 2003. DOI: 10.1109/INFCOM.2003.1208693 (cit. on p. 45).

[JNT18] Bo Jiang, Philippe Nain, and Don Towsley. "On the Convergence of the TTL Approximation for an LRU Cache under Independent Stationary Request Processes". In: *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3 (2018). DOI: 10.1145/3239164 (cit. on pp. 4, 40, 45, 46, 52, 68, 71).

[JP83] Kumar Joag-Dev and Frank Proschan. "Negative Association of Random Variables with Applications". In: *The Annals of Statistics* 11 (1983). DOI: 10.1214/aos/1176346079 (cit. on p. 77).

[JPD17] Slađana Jošilo, Valentino Pacifici, and György Dán. "Distributed algorithms for content placement in hierarchical cache networks". In: *Computer Networks* 125 (2017). DOI: 10.1016/j.comnet.2017.05.029 (cit. on p. 3).

[KMN99] Samir Khuller, Anna Moss, and Joseph Seffi Naor. "The budgeted maximum coverage problem". In: *Information processing letters* 70.1 (1999), pp. 39–45 (cit. on p. 63).

[KS02] G. Karakostas and D.N. Serpanos. "Exploitation of different types of locality for Web caches". In: *ISCC - Seventh International Symposium on Computers and Communications*. 2002. DOI: 10.1109/ISCC.2002.1021680 (cit. on p. 4).

[KSP03] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. "A simple algorithm for finding frequent elements in streams and bags". In: *ACM Trans. Database Syst.* 28 (2003). DOI: 10.1145/762471.762473 (cit. on p. 5).

[KV05] Adam Kalai and Santosh Vempala. "Efficient algorithms for online decision problems". In: *Journal of Computer and System Sciences* 71.3 (2005), pp. 291–307 (cit. on pp. 31, 33, 70).

[Lec+16] Mathieu Leconte et al. "Placing dynamic content in caches with small population". In: *IEEE INFOCOM - IEEE Conference on Computer Communications*. 2016 (cit. on p. 27).

[Lee+01] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, S.H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. "LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies". In: *IEEE Transactions on Computers* 50 (2001). DOI: 10.1109/TC.2001.970573 (cit. on p. 4).

[Li+16] Sheng Li et al. "Full-stack architecting to achieve a billion-requests-per-second throughput on a single key-value store server platform". In: *ACM Transactions on Computer Systems (TOCS)* 34.2 (2016), pp. 1–30 (cit. on pp. 28, 32).

[Li+20] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. "WavingSketch: An Unbiased and Generic Sketch for Finding Top-k Items in Data Streams". In: *26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020. DOI: 10.1145/3394486.3403208 (cit. on p. 5).

[Liu+16] Dong Liu, Binqiang Chen, Chenyang Yang, and Andreas F. Molisch. "Caching at the wireless edge: design aspects, challenges, and future directions". In: *IEEE Communications Magazine* 54 (2016). DOI: 10.1109/MCOM.2016.7565183 (cit. on p. 2).

[LSK11] Haiqin Liu, Yan Sun, and Min Sik Kim. “Fine-grained DDoS detection scheme based on bidirectional count sketch”. In: *20th International Conference on Computer Communications and Networks (ICCCN)*. 2011 (cit. on p. 9).

[LT15] Emilio Leonardi and Giovanni Luca Torrisi. “Least recently used caches under the shot noise model”. In: *IEEE INFOCOM - IEEE Conference on Computer Communications*. 2015 (cit. on pp. 4, 40, 45, 52).

[LW94] Nick Littlestone and Manfred K Warmuth. “The weighted majority algorithm”. In: *Information and computation* 108.2 (1994), pp. 212–261 (cit. on p. 31).

[LZ22] Qingsong Liu and Yaoyu Zhang. “Learning to Caching Under the Partial-feedback Regime”. In: *2022 18th International Conference on Network and Service Management (CNSM)*. 2022, pp. 154–162 (cit. on p. 28).

[MAE05] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. “Efficient computation of frequent and top-k elements in data streams”. In: *10th International Conference on Database Theory*. 2005. DOI: 10.1007/978-3-540-30570-5_27 (cit. on p. 5).

[Man53] W. Robert Mann. “Mean value methods in iteration”. In: *American Mathematical Society* 4 (1953) (cit. on p. 58).

[McA+15] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. “Image-based recommendations on styles and substitutes”. In: *38th international ACM SIGIR conference on research and development in information retrieval*. 2015 (cit. on pp. 41, 46, 62).

[Mha+22a] Naram Mhaisen, Abhishek Sinha, Georgios Paschos, and George Iosifidis. “Optimistic No-regret Algorithms for Discrete Caching”. In: *ACM Meas. Anal. Comput. Syst.* 6 (2022). DOI: 10.1145/3570608 (cit. on pp. 4, 5).

[Mha+22b] Naram Mhaisen, Abhishek Sinha, Georgios Paschos, and George Iosifidis. “Optimistic No-regret Algorithms for Discrete Caching”. In: *ACM Meas. Anal. Comput. Syst.* 6 (2022). DOI: 10.1145/3570608 (cit. on p. 28).

[MIL22] Naram Mhaisen, George Iosifidis, and Douglas Leith. “Online Caching with Optimistic Learning”. In: *IFIP Networking Conference*. 2022 (cit. on pp. 28, 30).

[MIL23] Naram Mhaisen, George Iosifidis, and Douglas Leith. “Online Caching with no Regret: Optimistic Learning via Recommendations”. In: *IEEE Transactions on Mobile Computing* (2023). DOI: 10.1109/TMC.2023.3317943 (cit. on p. 4).

[MMa] Gurmeet Singh Manku and Rajeev Motwani. “Approximate Frequency Counts over Data Streams”. In: *28th International Conference on Very Large Databases*. DOI: doi.org/10.1016/B978-155860869-6/50038-X (cit. on p. 5).

[MMb] Nimrod Megiddo and Dharmendra S. Modha. “ARC: A Self-Tuning, Low Overhead Replacement Cache”. In: *2nd USENIX Conference on File and Storage Technologies (FAST 03)*. URL: https://www.usenix.org/conference/fast-03/arc-self-tuning-low-overhead-replacement-cache (cit. on p. 4).

[Mor78] Robert Morris. “Counting large numbers of events in small registers”. In: *Commun. ACM* 21 (1978). DOI: 10.1145/359619.359627 (cit. on p. 5).

[Mou+19] Giovane CM Moura, John Heidemann, Ricardo de O Schmidt, and Wes Hardaker. “Cache me if you can: Effects of DNS Time-to-Live”. In: *ACM Internet Measurement Conference*. 2019 (cit. on p. 45).

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995 (cit. on pp. 10, 11).

[MS91] Lyle A McGeoch and Daniel D Sleator. “A strongly competitive randomized paging algorithm”. In: *Algorithmica* 6 (1991). DOI: doi.org/10.1007/BF01759073 (cit. on p. 3).

[Mut05] Shanmugavelayutham Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005 (cit. on p. 10).

[MV97] Reinhold Meise and Dietmar Vogt. *Introduction to functional analysis*. Clarendon Press, 1997 (cit. on p. 59).

[Neg+17] Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. “Access-Time-Aware Cache Algorithms”. In: *ACM Trans. Model. Perform. Eval. Comput. Syst.* 2 (2017). DOI: 10.1145/3149001 (cit. on p. 37).

[NGL21] Giovanni Neglia, Michele Garetto, and Emilio Leonardi. “Similarity Caching: Theory and Algorithms”. In: *IEEE/ACM Transactions on Networking* (2021). DOI: 10.1109/TNET.2021.3126368 (cit. on pp. 3, 5, 39, 42, 63).

[NWF78] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. “An analysis of approximations for maximizing submodular set functions—I”. In: *Mathematical programming* 14.1 (1978), pp. 265–294 (cit. on p. 63).

[Oli12] Oswaldo Rio Branco de Oliveira. “The implicit and the inverse function theorems: easy proofs”. In: *arXiv preprint arXiv:1212.2066* (2012) (cit. on p. 87).

[Pan+09] Sandeep Pandey, Andrei Broder, Flavio Chierichetti, Vanja Josifovski, Ravi Kumar, and Sergei Vassilvitskii. “Nearest-neighbor caching for content-match applications”. In: *18th International Conference on World Wide Web*. 2009. DOI: 10.1145/1526709.1526769 (cit. on pp. 2, 5, 39, 41–43).

[Par99] Sehie Park. “Ninety years of the Brouwer fixed point theorem”. In: *Vietnam J. Math* 27.3 (1999), pp. 187–222 (cit. on p. 57).

[Pas+18] Georgios S. Paschos, George Iosifidis, Meixia Tao, Don Towsley, and Giuseppe Caire. “The Role of Caching in Future Communication Systems and Networks”. In: *IEEE Journal on Selected Areas in Communications* 36 (2018). DOI: 10.1109/JSAC.2018.2844939 (cit. on p. 3).

[Pas+19a] Georgios S. Paschos, Apostolos Destounis, Luigi Vigneri, and George Iosifidis. “Learning to Cache With No Regrets”. In: *IEEE INFOCOM - IEEE Conference on Computer Communications*. 2019. DOI: 10.1109/INFOCOM.2019.8737446 (cit. on pp. 3, 4, 27, 28, 30).

[Pas+19b] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf (cit. on p. 91).

[PR23] Binghui Peng and Aviad Rubinstein. "Near Optimal Memory-Regret Tradeoff for Online Learning". In: *arXiv preprint arXiv:2303.01673* (2023) (cit. on p. 70).

[PZ23] Binghui Peng and Fred Zhang. "Online prediction in sub-linear space". In: *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2023 (cit. on p. 70).

[RKT10] Elisha J. Rosensweig, Jim Kurose, and Don Towsley. "Approximate Models for General Cache Networks". In: *IEEE INFOCOM - IEEE Conference on Computer Communications*. 2010. DOI: 10.1109/INFCOM.2010.5461936 (cit. on p. 3).

[Ros14] Sheldon M. Ross. *Introduction to probability models*. Academic press, 2014. DOI: https://doi.org/10.1016/C2012-0-03564-8 (cit. on p. 82).

[Ros95] Sheldon M Ross. *Stochastic processes*. John Wiley & Sons, 1995 (cit. on pp. 50, 82).

[RS92] Walter A Rosenkrantz and Rahul Simha. "Some theorems on conditional Pasta: A stochastic integral approach". In: *Operations Research Letters* 11 (1992). DOI: https://doi.org/10.1016/0167-6377(92)90082-E (cit. on p. 83).

[Sab+21] Anirudh Sabnis, Tareq Si Salem, Giovanni Neglia, Michele Garetto, Emilio Leonardi, and Ramesh K. Sitaraman. "GRADES: Gradient Descent for Similarity Caching". In: *IEEE INFOCOM - IEEE Conference on Computer Communications*. 2021. DOI: 10.1109/INFOCOM42981.2021.9488757 (cit. on pp. 5, 39, 61, 62).

[Ser+18] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri. "Soft Cache Hits: Improving Performance Through Recommendation and Delivery of Related Content". In: *IEEE Journal on Selected Areas in Communications* 36 (2018). DOI: 10.1109/JSAC.2018.2844983 (cit. on pp. 2, 39).

[Shu+21] Junaid Shuja, Kashif Bilal, Waleed Alasmary, Hassan Sinky, and Eisa Alanazi. "Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey". In: *Journal of Network and Computer Applications* 181 (2021). DOI: https://doi.org/10.1016/j.jnca.2021.103005 (cit. on p. 4).

[Sie04] Alan Siegel. "On Universal Classes of Extremely Random Constant-Time Hash Functions". In: *SIAM Journal on Computing* 33 (2004). DOI: 10.1137/S0097539701386216 (cit. on p. 10).

[SKW00] D.N. Serpanos, G. Karakostas, and W.H. Wolf. "Effective caching of Web objects using Zipf's law". In: *IEEE International Conference on Multimedia and Expo.* 2000. DOI: 10.1109/ICME.2000.871464 (cit. on p. 4).

[Smi82] Alan Jay Smith. “Cache Memories”. In: *ACM Comput. Surv.* 14 (1982). DOI: 10.1145/356887.356892 (cit. on p. 1).

[Smi85] Alan J. Smith. “Disk cache—miss ratio analysis and design considerations”. In: *ACM Trans. Comput. Syst.* 3 (1985). DOI: 10.1145/3959.3961 (cit. on p. 2).

[SNC23] Tareq Si Salem, Giovanni Neglia, and Damiano Carra. “Ascent Similarity Caching With Approximate Indexes”. In: *IEEE/ACM Transactions on Networking* 31 (2023). DOI: 10.1109/TNET.2022.3217012 (cit. on pp. 5, 39).

[SNI23] Tareq Si Salem, Giovanni Neglia, and Stratis Ioannidis. “No-regret Caching via Online Mirror Descent”. In: *ACM Trans. Model. Perform. Eval. Comput. Syst.* 8 (2023). DOI: 10.1145/3605209 (cit. on pp. 3, 4, 29, 30).

[Sri+22] Vaidehi Srinivas et al. “Memory Bounds for the Experts Problem”. In: *54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. 2022. DOI: 10.1145/3519935.3520069 (cit. on p. 70).

[ST85] Daniel D. Sleator and Robert E. Tarjan. “Amortized efficiency of list update and paging rules”. In: *Commun. ACM* 28 (1985). DOI: 10.1145/2786.2793 (cit. on pp. 3, 4, 6).

[Tra+13] Stefano Traverso, Mohamed Ahmed, Michele Garetto, Paolo Giaccone, Emilio Leonardi, and Saverio Niccolini. “Temporal locality in today’s content caching: why it matters and how to model it”. In: *SIGCOMM Comput. Commun. Rev.* 43 (2013). DOI: 10.1145/2541468.2541470 (cit. on pp. 2, 27).

[TS98] John Tse and Alan Jay Smith. “CPU cache prefetching: Timing evaluation of hardware implementations”. In: *IEEE Transactions on Computers* 47.5 (1998), pp. 509–526 (cit. on p. 27).

[UPS09] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. “Wikipedia Workload Analysis for Decentralized Hosting”. In: *Elsevier Computer Networks* 53.11 (July 2009), pp. 1830–1845 (cit. on p. 20).

[Ven+18] S. Venugopal, M. Gazzetti, Y. Gkoufas, and K. Katrinis. “Shadow puppets: Cloud-level accurate AI inference at the speed and economy of edge”. In: *USENIX HotEdge*. 2018 (cit. on p. 39).

[Ven+20] Federica Ventruto, Marco Pulimeno, Massimo Cafaro, and Italo Epicoco. “On Frequency Estimation and Detection of Heavy Hitters in Data Streams”. In: *Future Internet* 12.9 (2020), p. 158 (cit. on p. 10).

[vR88] Erik A van Doorn and G.J.K Regterschot. “Conditional PASTA”. In: *Operations Research Letters* 7 (1988). DOI: doi.org/10.1016/0167-6377(88)90036-3 (cit. on p. 83).

[Wan+21] Rui Wang, Hongchao Du, Zhaoyan Shen, and Zhiping Jia. “DAP-Sketch: An accurate and effective network measurement sketch with Deterministic Admission Policy”. In: *Computer Networks* 194 (2021), p. 108155 (cit. on pp. 9, 10).

[Wan99] Jia Wang. “A survey of web caching schemes for the Internet”. In: *SIGCOMM Comput. Commun. Rev.* 29 (1999). DOI: 10.1145/505696.505701 (cit. on p. 2).

[Wea18] Nik Weaver. *Lipschitz algebras*. World Scientific, 2018 (cit. on p. 59).

[WK71] Thomas J. Watson and W.F. King. *Analysis of Paging Algorithms*. IBM-Report. 1971. URL: https://books.google.fr/books?id=KTvaPgAACAAJ (cit. on pp. 4, 40).

[Wol82] Ronald W. Wolff. “Poisson Arrivals See Time Averages”. In: *Operations Research* 30 (1982). URL: http://www.jstor.org/stable/170165 (cit. on p. 83).

[Yan+18a] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. “HeavyGuardian: Separate and Guard Hot Items in Data Streams”. In: *24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018. DOI: 10.1145/3219819.3219978 (cit. on p. 5).

[Yan+18b] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. “Elastic sketch: adaptive and fast network-wide measurements”. In: *Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM ’18. 2018. DOI: 10.1145/3230543.3230544 (cit. on pp. 5, 9, 26, 69).

[Yan+21] Kaicheng Yang, Yuanpeng Li, Zirui Liu, Tong Yang, Yu Zhou, Jintao He, Tong Zhao, Zhengyi Jia, Yongqiang Yang, et al. “SketchINT: Empowering INT with TowerSketch for Per-flow Per-switch Measurement”. In: *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE. 2021, pp. 1–12 (cit. on p. 9).

[YN13] Fumito Yamaguchi and Hiroaki Nishi. “Hardware-based hash functions for network applications”. In: *2013 19th IEEE International Conference on Networks (ICON)*. 2013, pp. 1–6. DOI: 10.1109/ICON.2013.6781990 (cit. on p. 20).

[You91] Neal Young. “On-line caching as cache size varies”. In: *Second Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’91. 1991 (cit. on p. 4).

[Zha+21a] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. “CocoSketch: high-performance sketch-based measurement over arbitrary partial key query”. In: *ACM SIGCOMM*. 2021 (cit. on p. 18).

[Zha+21b] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, et al. “{LightGuardian}: A {Full-Visibility}, Lightweight, In-band Telemetry System Using Sketchlets”. In: *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 2021 (cit. on pp. 5, 9, 69).

[Zho+18] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. “Cold filter: A meta-framework for faster and more accurate stream processing”. In: *International Conference on Management of Data*. 2018 (cit. on p. 18).

[Zho+20] J. Zhou, O. Simeone, X. Zhang, and W. Wang. “Adaptive offline and online similarity-based caching”. In: *IEEE Networking Letters* 2.4 (2020), pp. 175–179 (cit. on p. 39).

[Zin03] Martin Zinkevich. “Online convex programming and generalized infinitesimal gradient ascent”. In: *Twentieth International Conference on International Conference on Machine Learning*. ICML’03. 2003 (cit. on pp. 3, 27).

List of Figures

List of Tables

List of Algorithms

Analyse Probabiliste pour le Caching

Younes BEN MAZZIANE

Résumé

Les caches sont de petites mémoires qui accélèrent la récupération des données. L'un des objectifs des politiques de mise en cache est de sélectionner le contenu du cache afin de minimiser le temps de réponse aux requêtes d'objets. Un problème plus général permet de répondre approximativement à la requête d'un objet par un objet similaire mis en cache. Ce concept, appelé "mise en cache par similarité", s'avère utile pour les systèmes de recommandation. L'objectif est de minimiser le temps de latence tout en fournissant des réponses satisfaisantes.
La compréhension théorique des algorithmes de gestion de la mémoire cache, sous des hypothèses spécifiques sur les requêtes, aide à choisir un algorithme approprié. Les politiques d'éviction du cache les plus répandues sont celles de l'utilisation la moins fréquente (LFU) et de l'utilisation la moins récente (LRU). LFU est efficace lorsque le processus requêtes est stationnaire, et LRU s'adapte aux changements dans les processus de requêtes. Les algorithmes d'apprentissage séquentiel, tels que l'algorithme aléatoire Follow-the-Perturbed Leader (FPL), appliqués à la mise en cache, bénéficient de garanties théoriques même dans le pire des cas.
LFU et FPL s'appuient sur le nombre de requêtes d'objets. Cependant, le comptage est un défi dans les scénarios à mémoire limitée. Pour y remédier, les politiques de mise en cache utilisent des schémas de comptage approximatifs, tels que la structure de données Count-Min Sketch avec mises à jour conservatrices (CMS-CU), afin d'équilibrer la précision des comptages et l'utilisation de la mémoire. Dans le cadre de la mise en cache par similarité, RND-LRU est une stratégie LRU modifiée. Malheureusement, il reste difficile de quantifier théoriquement à la fois la performance d'un cache LFU utilisant CMS-CU, celle d'un cache FPL avec un algorithme de comptage approximatif, ainsi que celle de RND-LRU.
Cette thèse explore trois algorithmes probabilistes : CMS-CU, FPL avec des estimations bruitées des nombres de requêtes d'objets (NFPL) et RND-LRU. Pour CMS-CU, nous proposons une approche novatrice pour trouver de nouvelles bornes supérieures sur l'espérance et le complémentaire de la fonction de répartition de l'erreur d'estimation sous un processus de requêtes i.i.d. De plus, nous démontrons que NFPL se comporte aussi bien que la politique de mise en cache statique, optimale et omnisciente, quelle que soit la séquence de requêtes (sous certaines conditions sur les comptages bruités). Enfin, nous introduisons une nouvelle politique de mise en cache qui est analytiquement résoluble. Nous montrons alors que cette politique approxime RND-LRU.

Mots-clés : comptage approximatif, algorithmes probabilistes, apprentissage séquentiel