TTL model for an LRU-based similarity caching policy

Younes Ben Mazziane, Sara Alouf, Giovanni Neglia, Daniel S. Menasche

TTL Model for an LRU-Based Similarity Caching Policy

Younes Ben Mazziane[a], Sara Alouf[a], Giovanni Neglia[a], Daniel S. Menasche[b]

[a] *Université Côte d'Azur, Inria, Sophia Antipolis, France*
[b] *Federal University of Rio de Janeiro, UFRJ, Rio de Janeiro, Brazil*

Abstract

Similarity caching allows requests for an item to be served by a similar item. Applications include recommendation systems, multimedia retrieval, and machine learning. Recently, many similarity caching policies have been proposed, like SIM-LRU and its generalization RND-LRU, but the performance analysis of their hit ratio is still wanting. In this paper, we show how to extend the popular time-to-live approximation in classic caching to similarity caching. In particular, we propose a method to estimate the hit ratio of the similarity caching policy RND-LRU. Our method, the RND-TTL approximation, introduces the RND-TTL cache model and then tunes its parameters in such a way as to mimic the behavior of RND-LRU. 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 both synthetic and real-world traces.

Keywords: Similarity caching, Time-to-live approximation, Performance evaluation

1. Introduction

Many applications require to retrieve items similar to a given user's request. For example, in content-based image retrieval [1] 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 [1] and contextual advertising [2], similarity caches are now a building block for a large variety of machine learning based inference systems for recommendations [3], image recognition [4, 5] and network traffic [6] 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 [7, 8] as well as to propose new caching policies [9, 10, 11].

RND-LRU is a randomized similarity caching policy proposed in the seminal paper [2]. 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 paper 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), [12]. Its computational cost is exponential in both the cache size and the number of items [13, 14]. 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 [15, 16]. 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 [12, 17, 18, 19] 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 paper, 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:

- We propose a novel similarity caching model named RND-TTL and we compute its hit ratio under IRM.
- We derive constraints on the RND-TTL cache model's parameters to approximate RND-LRU's hit ratio.
- 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.
- We provide sufficient conditions for the iterative algorithm to converge.
- We evaluate the accuracy of our RND-TTL approximation to estimate the hit ratio of RND-LRU on both synthetic and real-world traces.

This paper revisits and extends our previous work [20]. In particular, we have reframed the analysis of the RND-LRU cache after introducing the RND-TTL approximation. Also, the convergence results of the iterative algorithm are new.

The rest of the paper is organized as follows: We present background material on similarity caching and the TTL approximation in Section 2 and introduce notation and assumptions in Section 3. We define the RND-TTL approximation in Section 4 and explain our iterative algorithm for tuning the parameters of the RND-TTL cache in Section 5. We evaluate the performance of our RND-TTL approximation in Section 6 on both synthetic and real word traces and summarize our findings in Section 7. We provide detailed proofs and supplementary material in the appendices (Appendix A–Appendix L).

2. Background

2.1. Similarity Caching

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) [21] so that the dissimilarity cost, $\text{dis}(.,.) : \mathcal{I}^2 \to \mathbb{R}^+$, 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 [2]. 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 [1]. The similarity between images is measured via the distance between their representative vectors in a high-dimensional metric space.

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 [1, 2] 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 [2] 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.

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. [7] 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. [2]. 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 paper is to quantify the hit ratio of the RND-LRU policy for a given criterion of similarity.

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 [2] 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 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 1: RND-LRU [2]

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 1. 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

Table 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

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 [2] is also an LRU-based similarity caching. It has a similarity threshold d and uses $\hat{n}$ to serve n's request only if $\text{dis}(n, \hat{n}) \leq d$. SIM-LRU is a particular case of RND-LRU such that $q_m(n) = 1$ if $\text{dis}(m, n) \leq d$ 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.

2.2. TTL Approximation for LRU Cache

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 [22, 23, 24, 25]. 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. [26] 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 [27, 28], and the analysis has been extended to encompass a network of TTL caches [16, 23, 24, 29]. A comprehensive overview of TTL caching policies is provided in [30].

TTL caches are efficient modeling tools to analyze caching policies [31]. In particular, a TTL caching policy with timers resets per hit was proposed as a model for LRU [12, 15]. 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 paper, except otherwise noted, we refer to TTL caches with resets per hit simply as TTL caches.

2.2.2. TTL Approximation

Fagin [12] 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[1] 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 [15], 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 [12]. Che et al. [15] 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 [17, 18, 19, 31, 32], 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

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

setting, for every n, T_n to the characteristic time t_C which verifies:

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

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:

$$H \approx \sum_{n\in\mathcal{I}} p_n \left(1-(1-p_n)^{t_C}\right). \tag{2}$$

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 $1/\lambda_n$ [15, 17]. 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:

$$h_n = 1-\mathrm{e}^{-\lambda_n t_C}, \tag{3}$$

and t_C verifies:

$$\sum_{n\in\mathcal{I}} \left(1-\mathrm{e}^{-\lambda_n t_C}\right) = C. \tag{4}$$

The hit ratio of LRU is then approximated as

$$H \approx \left(\frac{1}{\sum_{i\in\mathcal{I}}\lambda_i}\right) \cdot \sum_{n\in\mathcal{I}} \lambda_n \left(1-\mathrm{e}^{-\lambda_n t_C}\right). \tag{5}$$

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 (5)) different from the formula proposed by Fagin (see (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 (2) and (5) are asymptotically accurate approximations to LRU under suitable conditions [12, 17, 19].

3. Notation and Assumptions

Recall from Section 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 1 the main notation that we use. We assume equal size items and, as in [21], assume that they can be represented by Euclidean vectors, such that an opportune distance between vectors informs on the dissimilarity cost, dis(.,.), between pairs of items. We maintain the same notation used in Section 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, $\mathcal{N}_m(n) \triangleq \{l \in \mathcal{N}(n) : \ \text{dis}(n,l) < \text{dis}(n,m)\}$ and $\mathcal{N}_m^c[n] \triangleq \{l \in \mathcal{N}^c[n] : \ \text{dis}(n,l) < \text{dis}(n,m)\}$. 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 paper, we assume that requests for items follow the Independent Reference Model (IRM). We also make use of the following assumptions.

Assumption 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 1 is a particular case of IRM where the probability of a request for item n coincides with its request rate. However, while Assumption 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 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 2. *$\{L_{RND\text{-}LRU}(t),\ t \geq 0\}$ has a limiting distribution that we denote as $\tilde{\boldsymbol{\pi}} = (\tilde{\pi}_L)_{L \in \tilde{\Omega}}$.*

Assumption 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 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* $\text{dis}(n, m) \neq \text{dis}(n, l)$.

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 4.1, we describe the RND-TTL cache model, highlighting its specific characteristics. Secondly, in Section 4.2, we explain how the RND-TTL model can capture the dynamics and behavior of RND-LRU. Thirdly, in Section 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.

4.1. 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.

4.1.1. 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 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 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 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 3 (in Appendix A) that can be used to simulate R-TTL. Observe how RND-LRU (see Algorithm 1) 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 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.

4.1.2. 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 $\boldsymbol{\lambda^r} = (\lambda_n^r)_{n\in\mathcal{I}}$ and $\mathbf{p^i} = (p_n^i)_{n\in\mathcal{I}}$ 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_n^r (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[2], which occurs with probability $1 - p_n^i$.

[2]We assume that the cache statically stores a tombstone item whose distance to all items

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 $\mathbf{T}$, $\boldsymbol{\lambda}^{\boldsymbol{r}}$ and $\mathbf{p}^{\mathbf{i}}$ 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.

4.1.3. 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.

$$o_n \triangleq \lim_{t \to +\infty} \frac{1}{t} \int_0^t \mathbb{1}(X_n(u) = 1)\, du\ . \tag{6}$$

Proposition 1 (Occupancy). *Under Assumption 1, the occupancy in the RND-TTL cache of item n is expressed as:*

$$o_n = \left(\frac{1}{\lambda_n^i} \cdot \frac{\lambda_n^r}{\mathrm{e}^{\lambda_n^r T_n} - 1} + 1 \right)^{-1}, \tag{7}$$

where

$$\lambda_n^i = \lambda_n \cdot p_n^i\ . \tag{8}$$

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,*

$$o_n = \frac{\mathbb{E}\left[T_n^{\mathrm{On}}\right]}{\mathbb{E}\left[T_n^{\mathrm{Off}}\right] + \mathbb{E}\left[T_n^{\mathrm{On}}\right]} = \left(\mathbb{E}\left[T_n^{\mathrm{Off}}\right] \cdot \frac{1}{\mathbb{E}\left[T_n^{\mathrm{On}}\right]} + 1 \right)^{-1}. \tag{9}$$

is infinite. Whenever a request arrives in an empty cache, the tombstone item is returned as the closest item in the cache.

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

$$\mathbb{E}\left[T_n^{\mathrm{On}}\right] = \frac{1}{\lambda_n^r}\left(e^{\lambda_n^r T_n} - 1\right). \tag{10}$$

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

$$\mathbb{E}\left[T_n^{\mathrm{Off}}\right] = \frac{1}{\lambda_n^i}\ . \tag{11}$$

For additional details, we refer the reader to Appendix B. ■

We stress that under Assumption 1, $\{X_n(t),\ t \geq 0\}$ has limiting distribution given by the occupancy, namely,

$$\lim_{t\to+\infty} \Pr\left(X_n(t) = 1\right) = o_n,\ \lim_{t\to+\infty} \Pr\left(X_n(t) = 0\right) = 1 - o_n\ . \tag{12}$$

The above equation can be justified thanks to [33, Thm. 3.4.4].

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,

$$S_{\text{RND-TTL}}(t) = \{n \in \mathcal{I} :\ X_n(t) = 1\}\,. \tag{13}$$

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 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 (12), for any set of cached items $S \in \Omega$, the corresponding limiting probability π_S can be computed as follows:

$$\pi_S = \prod_{n\in S} o_n \cdot \prod_{m\notin S} (1 - o_m)\ . \tag{14}$$

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 2 (**Item's hit probability**). *Under Assumption 1, the hit probability h_n for item n in the RND-TTL cache is given by:*

$$h_n = o_n + (1 - o_n) \cdot (1 - p_n^i) \,, \tag{15}$$

where o_n is given in (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 Appendix C and Appendix D.* ■

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 (15) and (7) by letting $p_n^i = 1$ and $\lambda_n^r = \lambda_n^i = \lambda_n$. Equations (3), (7) and (15) are then all equivalent.

While we have described the RND-TTL cache model using parameters $\mathbf{T}$, $\boldsymbol{\lambda}^{\mathbf{r}}$ and $\mathbf{p}^{\mathbf{i}}$, in what follows, it will be more convenient to retain as parameters $\mathbf{T}$, $\boldsymbol{\lambda}^{\mathbf{r}}$, and $\boldsymbol{\lambda}^{\mathbf{i}} = (\lambda_n^i)_{n \in \mathcal{I}}$ (see (8)).

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 $\boldsymbol{\lambda^i}, \boldsymbol{\lambda^r}$, and $\mathbf{T}$ for the RND-TTL approximation.

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 [12, 17, 18, 19]. 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,

$$\tilde{B}_n \triangleq \left\{L \in \tilde{\Omega} :\ L \cap \mathcal{N}^c[n] = \emptyset\right\} \text{ and } B_n \triangleq \{S \in \Omega :\ S \cap \mathcal{N}^c[n] = \emptyset\}. \quad (16)$$

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,

$$\tilde{B}_{n,m} \triangleq \left\{L \in \tilde{\Omega} :\ m \in L, L \cap \mathcal{N}_m^c[n] = \emptyset\right\}, \quad (17)$$

$$B_{n,m} \triangleq \{S \in \Omega :\ m \in S, S \cap \mathcal{N}_m^c[n] = \emptyset\}. \quad (18)$$

Proposition 3 (**RND-LRU insertion rate**). *Under Assumptions 1, 2 and 3, the insertion rate of item n in RND-LRU, $\tilde{\lambda}_n^i$, is expressed as:*

$$\tilde{\lambda}_n^i = \tilde{f}_{n,\mathbf{q}}^i(\tilde{\boldsymbol{\pi}}) \triangleq \lambda_n \left(\sum_{L \in \tilde{B}_n} \tilde{\pi}_L + \sum_{m \in \mathcal{N}(n)} (1 - q_m(n)) \sum_{K \in \tilde{B}_{n,m}} \tilde{\pi}_K \right), \quad (19)$$

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

Proof. *The result follows from observing that in RND-LRU, when a request for an item n finds $\{L_{RND\text{-}LRU}(t),\ t \geq 0\}$ in state $\tilde{B}_n$, n is inserted in the cache with probability* 1. *On the other hand, if a request for n finds $\{L_{RND\text{-}LRU}(t),\ t \geq 0\}$ 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 Appendix C and Appendix E.* ■

Proposition 4 (**RND-LRU refresh rate).** *Under Assumptions 1, 2 and 3, the refresh rate of item n in RND-LRU, $\tilde{\lambda}_n^r$, is expressed as:*

$$\tilde{\lambda}_n^r = \tilde{f}_{n,\mathbf{q}}^r\left(\tilde{\boldsymbol{\pi}}\right) \triangleq \sum_{m \in \mathcal{N}^c[n]} q_n(m)\lambda_m \sum_{L \in \tilde{B}_{m,n}} \tilde{\pi}_L \ , \tag{20}$$

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

Proof. *The result follows from observing that in RND-LRU, when a request for an item m finds $\{L_{RND\text{-}LRU}(t),\ t \geq 0\}$ 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 Appendix C and Appendix F.* ■

To recapitulate our derivations and ease the comparison between RND-TTL and RND-LRU, we depict in Figure 1a the notation, the main metrics and their expressions (when available) for both policies.

Constraints on RND-TTL cache's parameters. The RND-TTL approximation is summarized in Figure 1. First, it imposes on the RND-TTL cache's parameters the TTL approximation's constraints:

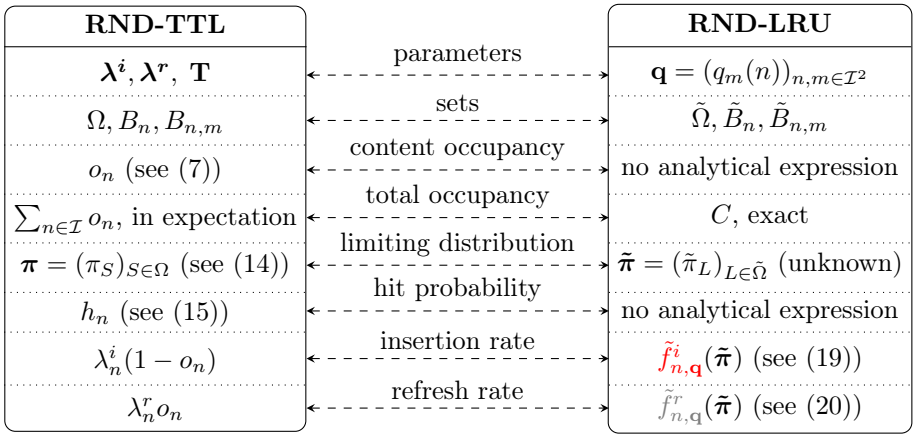
- All items' timers share the same duration, which we denote as T, i.e.,

$$T_n = T,\ \forall n \in \mathcal{I}. \tag{21}$$

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

$$\sum_{n \in \mathcal{I}} \left(\frac{1}{\lambda_n^i} \cdot \frac{\lambda_n^r}{\mathrm{e}^{\lambda_n^r T_n} - 1} + 1 \right)^{-1} = C. \tag{22}$$

  Note that the term corresponding to item n in the sum in (22) is the limiting probability that n is cached (see (12) and Proposition 1).



(a) Mapping notation and metrics in RND-TTL and RND-LRU.

Timers	$T_n = T,\ \forall n \in \mathcal{I}$
Total occupancy	Expected number of cached items $= C$ (see (22))
Insertion rate	$\lambda_n^i(1-o_n) = f^i_{n,\mathbf{q}}(\boldsymbol{\pi})$ (see (23))
Refresh rate	$\lambda_n^r o_n = f^r_{n,\mathbf{q}}(\boldsymbol{\pi})$ (see (24))

(b) Constraints on RND-TTL parameters to approximate RND-LRU.

Figure 1: RND-TTL approximation to RND-LRU. The metrics with unknown analytical expression including our key metric of interest, the hit probability of RND-LRU, are approximated using the RND-TTL model.

Second, it constrains the (unconditional) insertion and refresh rates under RND-TTL, namely $\lambda_n^i(1-o_n)$ and $\lambda_n^r o_n$, to satisfy expressions similar to those verified by the same rates under RND-LRU, which are given by Propositions 3 and 4:

$$\lambda_n^i(1-o_n) = f^i_{n,\mathbf{q}}(\boldsymbol{\pi}) \triangleq \lambda_n \left(\sum_{S\in B_n} \pi_S + \sum_{m\in\mathcal{N}(n)} (1-q_m(n)) \sum_{K\in B_{n,m}} \pi_K \right), \quad (23)$$

$$\lambda_n^r o_n = f^r_{n,\mathbf{q}}(\boldsymbol{\pi}) \triangleq \sum_{m\in\mathcal{N}^c[n]} q_n(m)\lambda_m \sum_{S\in B_{m,n}} \pi_S\ , \quad (24)$$

where $\boldsymbol{\pi}$ is the limiting distribution for the set of cached items in the RND-TTL cache, computed in (14), and $f^i_{n,\mathbf{q}}(\boldsymbol{\pi})$ and $f^r_{n,\mathbf{q}}(\boldsymbol{\pi})$ have been defined to mimic $\tilde{f}^i_{n,\mathbf{q}}(\tilde{\boldsymbol{\pi}})$ and $\tilde{f}^r_{n,\mathbf{q}}(\tilde{\boldsymbol{\pi}})$, respectively given in (19) and (20). Substituting $\boldsymbol{\pi}$

from (14) into (23) and (24), we obtain the following expressions:

$$\lambda_n^i = \lambda_n \left(\prod_{m \in \mathcal{N}(n)} (1 - o_m) + \sum_{m \in \mathcal{N}(n)} (1 - q_m(n))\, o_m \prod_{l \in \mathcal{N}_m(n)} (1 - o_l) \right), \tag{25}$$

$$\lambda_n^r = \lambda_n + \sum_{m \in \mathcal{N}(n)} q_n(m) \lambda_m \prod_{l \in \mathcal{N}_n^c[m]} (1 - o_l)\,. \tag{26}$$

RND-LRU's hit ratio approximation. If we can compute values for $\boldsymbol{\lambda^r}$, $\boldsymbol{\lambda^i}$ and T verifying the constraints (21), (22), (25) and (26), then using Proposition 2, (8) and (25), we can estimate the hit ratio of RND-LRU as:

$$H \approx \sum_{n \in \mathcal{I}} \lambda_n \cdot h_n, \tag{27}$$

where

$$h_n = o_n + \sum_{m \in \mathcal{N}(n)} q_m(n) \cdot o_m \prod_{l \in \mathcal{N}_m^c[n]} (1 - o_l). \tag{28}$$

Remark 1. *In Section 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 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 3 and 4, we can establish the insertion rate and refresh rate of an item n in R-TTL as $f^i_{n,\mathbf{q}}(\boldsymbol{\mu})$ and $f^r_{n,\mathbf{q}}(\boldsymbol{\mu})$, 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.

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:

$$g(x_1, x_2, x_3) \triangleq \left(\frac{1}{x_2} \cdot \frac{x_1}{\mathrm{e}^{x_1 x_3} - 1} + 1 \right)^{-1}, \tag{29}$$

we can rewrite (7) as:

$$o_n = g(\lambda_n^r, \lambda_n^i, T) \ . \tag{30}$$

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

$$\boldsymbol{\lambda^i} = \mathbf{E}(\mathbf{o}) = (E_n(\mathbf{o}))_{n\in\mathcal{I}}, \tag{31}$$

$$\boldsymbol{\lambda^r} = \mathbf{R}(\mathbf{o}) = (R_n(\mathbf{o}))_{n\in\mathcal{I}}, \tag{32}$$

$$\mathbf{o} = \mathbf{g}(\boldsymbol{\lambda^r}, \boldsymbol{\lambda^i}, T) = (g(\lambda_n^r, \lambda_n^i, T))_{n\in\mathcal{I}}, \tag{33}$$

$$T \in T_C(\boldsymbol{\lambda^r}, \boldsymbol{\lambda^i}) \iff \sum_{n\in\mathcal{I}} g(\lambda_n^r, \lambda_n^i, T) = C, \tag{34}$$

where $E_n(\mathbf{o})$ and $R_n(\mathbf{o})$ are functions respectively obtained from (25) and (26), $g(\lambda_n^r, \lambda_n^i, T)$ is obtained from (29), and $T_C(\boldsymbol{\lambda^r}, \boldsymbol{\lambda^i})$ is defined as the set of values of the shared timer value T guaranteeing that, for given $\boldsymbol{\lambda^r}$ and $\boldsymbol{\lambda^i}$, the expected number of cached items in RND-TTL equals C.

Combining (31)-(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 (28) and estimate the hit ratio of RND-LRU according to (27).

In Section 5.1, we establish a sufficient condition for the existence of a solution for the system of equations (31)-(34). In Section 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 5.3 outlines a practical method for tuning the algorithm's parameter β.

5.1. Fixed Point Equations

We denote the set $T_C(\mathbf{R}(\mathbf{o}), \mathbf{E}(\mathbf{o}))$ (see (34)) as $T_C(\mathbf{o})$. We denote the capped simplex as Δ_C such that:

$$\Delta_C \triangleq \left\{ \mathbf{o} \in \mathbb{R}^N : \ 0 \leq o_n \leq 1, \ \sum_{n\in\mathcal{I}} o_n = C \right\}. \tag{35}$$

Lemma 1 ($T_C(\mathbf{o})$ **is a singleton).** *If the "no-coverage" condition:*

$$\textit{(no-coverage condition)} \qquad \forall \mathcal{M} \subset \mathcal{I} : |\mathcal{M}| \leq C, \quad \left| \bigcup_{n \in \mathcal{M}} \mathcal{N}(n) \right| < N - C, \tag{36}$$

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

$$\forall \mathbf{o} \in \Delta_C, \exists! T_0 \in \mathbb{R}^+ : \quad F(\mathbf{o}, T_0) = 0, \tag{37}$$

$$F(\mathbf{o}, T) \triangleq \sum_{n \in \mathcal{I}} g(R_n(\mathbf{o}), E_n(\mathbf{o}), T) - C, \tag{38}$$

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

Proof. *See Appendix G.* ■

A note on (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 (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 (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 2. *If (36) is not met for some set $\mathcal{M}$, then Lemma 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 (36) is verified. Therefore, thanks to Lemma 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.,

$$t_C(\mathbf{o}) = T \iff F(\mathbf{o}, T) = 0. \tag{39}$$

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

$$\forall \mathbf{o} \in \Delta_C,\ \mathbf{G}(\mathbf{o}) \triangleq \mathbf{g}\left(\mathbf{R}(\mathbf{o}), \mathbf{E}(\mathbf{o}), t_C(\mathbf{o})\right) \tag{40a}$$

$$= \left(g(R_1(\mathbf{o}), E_1(\mathbf{o}), t_C(\mathbf{o})), \cdots, g(R_N(\mathbf{o}), E_N(\mathbf{o}), t_C(\mathbf{o}))\right), \tag{40b}$$

where g is defined in (29). It follows that finding a solution for the system of equations (31)-(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 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:*

$$\forall j \in \mathcal{I}, \quad \frac{\partial t_C}{\partial o_j}(\mathbf{o}) = -\frac{\partial F}{\partial o_j}(\mathbf{o}, t_C(\mathbf{o})) \cdot \left(\frac{\partial F}{\partial T}(\mathbf{o}, t_C(\mathbf{o}))\right)^{-1}, \tag{41}$$

where F has been defined in (38).

Proof. *See Appendix H.* ■

Proposition 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.*

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 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 [34] implies the existence of a fixed point for $\mathbf{G}$.* ■

Proposition 5 indicates that when the no-coverage condition (36) is satisfied, it is possible to find parameters for the RND-TTL cache model that verify the system of equations (31)-(34). Therefore, we can apply the RND-TTL approximation to estimate the hit ratio of RND-LRU.

Algorithm 2: Fixed point method

Input: C, λ, dis(.,.), d, $(q_n(i))_{(n,i)\in I^2}$, β, stopping condition

Output: Estimation of $\mathbf{o}, \mathbf{h}, t_C$

Initialization:

1: Obtain $t_C(0)$ such that $\sum_{n\in I}\left(1-\mathrm{e}^{-\lambda_n \cdot t_C(0)}\right) = C$

2: $\mathbf{o}(0) \leftarrow 1-\mathrm{e}^{-\lambda \cdot t_C(0)}$

3: $\mathbf{h}(0) \leftarrow f^h(\mathbf{o}(0))$

4: $j \leftarrow 1$

5: **while** *Stopping condition not satisfied* **do**

6: $\quad \boldsymbol{\lambda^i}(j) \leftarrow \mathbf{E}(\mathbf{o}(j-1))$ (see (31))

7: $\quad \boldsymbol{\lambda^r}(j) \leftarrow \mathbf{R}(\mathbf{o}(j-1))$ (see (32))

8: $\quad$ 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 (34),(33))

9: $\quad \mathbf{o}(j) \leftarrow (1-\beta)\cdot \mathbf{g}(\boldsymbol{\lambda^r}(j), \boldsymbol{\lambda^i}(j), t_C(j)) + \beta \cdot \mathbf{o}(j-1)$

10: $\quad \mathbf{h}(j) = f^h(\mathbf{o}(j))$ (see (28))

11: $\quad j \leftarrow j+1$

12: **end while**

13: **return** $\mathbf{h}(j)$, $\mathbf{o}(j)$, $t_C(j)$

5.2. Fixed Point Algorithm

We recall that solving the system of equations (31)-(34) reduces to solving a fixed point equation for the function $\mathbf{G}$ defined in (40a). A natural approach to finding a fixed point of $\mathbf{G}$ is through an iterative method. This is illustrated in Figure 2. 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 $\beta \in [0,1)$ [35]. A detailed version of these iterations is presented in Algorithm 2. 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) and $\sum_{n\in\mathcal{I}} o_n(0) = C$ (lines 1–2). Then, we compute $\boldsymbol{\lambda^i}(1)$ and $\boldsymbol{\lambda^r}(1)$ using (31) and (32), respectively (lines 5–7). Given $\boldsymbol{\lambda^i}(1)$ and $\boldsymbol{\lambda^r}(1)$, $t_C(1)$ is the unique solution of (34) (see (37)). This

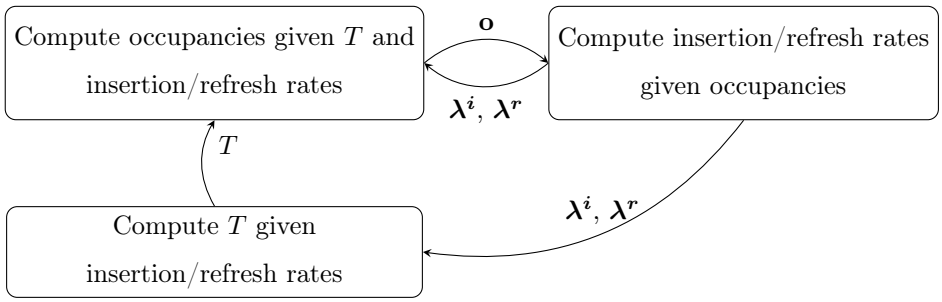


Figure 2: Essence of the fixed point algorithm.

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 2 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 2 are of the form: $\mathbf{o}(j{+}1) = \mathbf{G}_\beta(\mathbf{o}(j))$ such that:

$$\mathbf{G}_\beta(\mathbf{o}) \triangleq (1-\beta)\mathbf{G}(\mathbf{o}) + \beta\mathbf{o} \ . \tag{42}$$

The function $\mathbf{G}_\beta$ is continuously differentiable thanks to Proposition 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:

$$\mu_\beta \triangleq \sup_{\mathbf{o}\in\Delta_C} \left\| \boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o}) \right\|. \tag{43}$$

Proposition 6 (Fixed point uniqueness and convergence). *If*

$$\exists \beta \in [0,1) : \ \mu_\beta < 1, \tag{44}$$

then $\mathbf{G}$ *has a unique fixed point in* Δ_C *and Algorithm 2 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 2, then we have:*

$$\|\mathbf{o}(j) - \mathbf{o}^*\| \leq (\mu_\beta)^j \cdot \sup_{\mathbf{x},\mathbf{y}\in\Delta_C} \|\mathbf{x}-\mathbf{y}\|, \ \forall j \in \mathbb{N}, \tag{45}$$

where Δ_C *is defined in* (35)*.*

Proof. *Under* (36)*, Proposition 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 *[36], 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 [37], we deduce that* (i) $\mathbf{G}_\beta$ *has a unique fixed point denoted as* $\mathbf{o}^{*,\beta}$*,* (ii) *Algorithm 2 with parameter* β *converges to* $\mathbf{o}^{*,\beta}$*, and* (iii) *the distance between* $\mathbf{o}(j)$ *and* $\mathbf{o}^{*,\beta}$ *satisfies*

$$\left\|\mathbf{o}(j) - \mathbf{o}^{*,\beta}\right\| \leq (\mu_\beta)^j \cdot \sup_{\mathbf{x},\mathbf{y}\in\Delta_C} \|\mathbf{x}-\mathbf{y}\|. \tag{46}$$

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 (44) and then on the convergence of Algorithm 2 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 7 (Computation of $\boldsymbol{J}_{\mathbf{G}_\beta}$). *The Jacobian matrix $\boldsymbol{J}_{\mathbf{G}_\beta}$ has the following expression:*

$$\forall \mathbf{o} \in \Delta_C, \quad \boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o}) = (1-\beta)\boldsymbol{J}_{\mathbf{G}}(\mathbf{o}) + \beta\, \boldsymbol{I}_N, \tag{47}$$

where

$$\begin{aligned}\boldsymbol{J}_{\mathbf{G}}(\mathbf{o}) = \operatorname{Diag}\left(\boldsymbol{\partial_1 g}\right) \cdot \boldsymbol{J}_{\mathbf{R}}(\mathbf{o}) + \operatorname{Diag}\left(\boldsymbol{\partial_2 g}\right) \cdot \boldsymbol{J}_{\mathbf{E}}(\mathbf{o}) \\ - \frac{1}{\boldsymbol{\partial_3 g} \cdot \mathbf{1}} \boldsymbol{\partial_3 g}^{\intercal} \cdot \left(\boldsymbol{\partial_1 g} \cdot \boldsymbol{J}_{\mathbf{R}}(\mathbf{o}) + \boldsymbol{\partial_2 g} \cdot \boldsymbol{J}_{\mathbf{E}}(\mathbf{o})\right),\end{aligned} \tag{48}$$

with

$$\boldsymbol{\partial_j g} = \left(\frac{\partial g}{\partial x_j}(R_n(\mathbf{o}), E_n(\mathbf{o}), t_C(\mathbf{o}))\right)_{n \in \mathcal{I}}, \qquad \text{for } j \in \{1,2,3\}, \tag{49}$$

$\boldsymbol{J}_{\mathbf{R}}$ 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 (29).

Proof. *See Appendix I.* ■

Time Complexity of Algorithm 2. Let D be the maximum number of neighbors for any item in $\mathcal{I}$ plus 1, more precisely,

$$D \triangleq \max_{n \in \mathcal{I}} |\mathcal{N}^c[n]|. \tag{50}$$

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

$$\mathcal{K} \triangleq \sum_{n \in \mathcal{I}} \sum_{m \in \mathcal{N}^c[n]} |\mathcal{N}^c[m]|. \tag{51}$$

We show in Appendix J that the time complexity of one iteration of Algorithm 2 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 6 to bound the number of iterations of Algorithm 2. When the condition (44) is verified, Proposition 6 guarantees the convergence of Algorithm 2 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 6, we obtain that the number of iterations of Algorithm 2 is upper bounded by $\frac{\epsilon}{\ln(1/\mu_\beta)}\sqrt{2C}$, with μ_β defined in (43) and satisfying (44). We deduce that the time complexity of Algorithm 2 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 $\mathcal{O}\left(\frac{\epsilon\sqrt{2C}}{\ln(1/\mu_\beta)} \cdot ND^2\right)$.

5.3. Choice of β

Our approach for selecting the value of β for Algorithm 2 is based on Proposition 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.,

$$Y(\mathbf{o}) \triangleq \left\{\beta \in [0,1) : \ \left\|\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o})\right\|_2 < 1\right\} . \tag{52}$$

Equation (44) in Proposition 6 is equivalent to the set $\bigcap_{\mathbf{o}\in\Delta_C} Y(\mathbf{o})$ being nonempty. In other words, choosing the parameter β of Algorithm 2 from the set $\bigcap_{\mathbf{o}\in\Delta_C} Y(\mathbf{o})$ guarantees the convergence of Algorithm 2 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 $\left\|\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o}_{(j)})\right\|_2 < 1$. 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 (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 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 $\boldsymbol{J}_{\mathbf{G}}(\mathbf{o}) + \boldsymbol{J}_{\mathbf{G}}(\mathbf{o})^\intercal$,*

$$\gamma = \left(\|\boldsymbol{J}_{\mathbf{G}}(\mathbf{o})\|_2\right)^2 = \rho(\boldsymbol{J}_{\mathbf{G}}(\mathbf{o})\boldsymbol{J}_{\mathbf{G}}(\mathbf{o})^\intercal) \tag{53}$$

$$\eta = \rho(\boldsymbol{J}_{\mathbf{G}}(\mathbf{o}) + \boldsymbol{J}_{\mathbf{G}}(\mathbf{o})^\intercal). \tag{54}$$

If

$$\eta < \min\{2, \gamma + 1\} \tag{55}$$

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

$$Y(\mathbf{o}) \supset \tilde{Y}(\mathbf{o}) \tag{56}$$

where

$$\tilde{Y}(\mathbf{o}) = \left(\max\left\{0, \frac{\gamma - 1}{\gamma + 1 - \eta}\right\}, 1\right) \ . \tag{57}$$

Proof. *See Appendix K.* ■

Remark 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 (55) is verified. It follows from Proposition 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 (53)-(54). We verify next if (55) is satisfied for each pair (γ_j, η_j). If this condition holds for all pairs, then using Proposition 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 6 and ensure the convergence of the proposed fixed point algorithm. Note also that if (55) does not hold for at least one of the considered pairs (γ_j, η_j), $j = 1, \ldots, f$, we cannot use Proposition 8 to analyze the convergence of the proposed fixed-point algorithm using the sufficient conditions established in Proposition 6. However,

as we will indicate through numerical experiments, the algorithm converges in practice under much broader settings than those considered in Proposition 6.

Remark 4. *Proposition 8 is a general result about fixed point algorithms of the form* (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.*

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 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 6.2. Subsequently, we analyze the approximation accuracy in Section 6.3. Our RND-TTL approximation technique involves solving a set of equations using an iterative fixed-point method outlined in Algorithm 2. In Section 6.4, we evaluate the convergence of Algorithm 2.

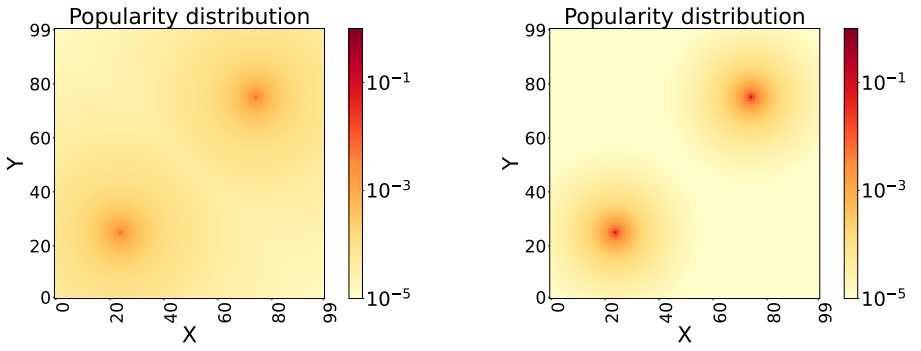
6.1. Experimental Setting

We evaluate the efficiency of the proposed fixed point method (Algorithm 2) to predict the hit ratio on synthetic traces and on an Amazon trace [10].

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 3). The total number of items is $|\mathcal{I}| = 10^4$, and the dissimilarity function between items $\text{dis}(\cdot, \cdot)$ 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 [12], where the popularity distribution for an item $n = (x, y)$ is given by

$$p_{(x,y)} \sim \Big(\min \big\{ \text{dis}(n, (24, 24)), \text{dis}(n, (74, 74)) \big\} + 1 \Big)^{-\alpha}, \tag{58}$$

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



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

Figure 3: Spatial popularity distribution.

$r = 2 \cdot 10^5$ requests for items in $\mathcal{I}$. Figures 3a and 3b illustrate the popularity distribution in (58) for $\alpha \in \{1.4, 2.5\}$.

In addition to the popularity distribution (58), we consider also a Zipfian popularity distribution. The corresponding experimental results are presented in Appendix L.

Real world trace. For the Amazon trace, each item corresponds to an Amazon product. The request trace in [10] 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 [21], 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 [10].

6.2. Benchmarks and Alternative Approaches

In what follows, we compare hit ratio estimates provided by RND-LRU using Algorithm 2 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 (3) and t_C is deduced using the cache capacity constraint given by (4).

Optimal Static Allocation. Under IRM, it is shown in [7] 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:

$$S^* \in \underset{S\subset\mathcal{I},|S|=C}{\arg\max} \sum_{n\in\bigcup_{j\in S}\mathcal{N}^c[j]} \lambda_n \ . \tag{59}$$

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 [38, 39].

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:

$$h_n = 1 - \mathrm{e}^{-\sum_{i \in \mathcal{N}^c[n]} \lambda_i t_C}, \quad o_n = h_n. \tag{60}$$

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

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 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 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 2 with parameter $\beta = 0.5$ and a stopping condition determined by a fixed number of iterations. Algorithm 2 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 6.2. The numerical values used for all the experiments are summarized in Table 2.

In Figure 4, we show the empirical hit ratio along with its estimates obtained through different approaches, for the two synthetic settings (Figures 4a-4b)

Table 2: Parameters of the experiments.

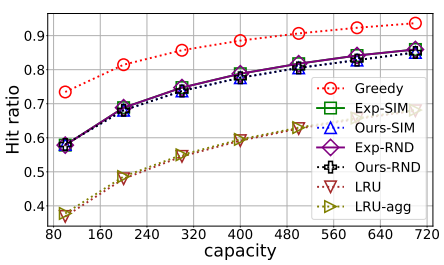
Variable	Synthetic traces	Amazon trace
$\mathcal{I}$	$[0..99]^2$	Products
$N = \lvert\mathcal{I}\rvert$	10^4	$\approx 10^4$
λ_n	(58)	Empirical
$\mathrm{dis}(\cdot,\cdot)$	Euclidean distance	Euclidean distance
d	1 and 2	300
Number of requests r	$2 \cdot 10^5$	$\approx 10^5$
Number of iterations Alg. 2	25 and 15	40
$q_n(m)$	$(\mathrm{dis}(n,m))^{-2}$	$(\mathrm{dis}(n,m))^{-0.2}$

and for the Amazon trace (Figure 4c). 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.

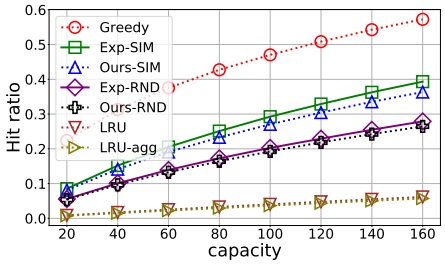
Ours-SIM and Ours-RND clearly outperform all the alternative approaches presented in Section 6.2 in estimating the empirical hit ratio, while tending to underestimate it. As LRU does not take into account the similarity between items, the gap between LRU and Exp-SIM reveals the benefits of similarity caching over exact caching.

For the synthetic settings in Figures 4a and 4b, 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 4a and 4b (see (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 4a and 4b, the approximations $\tilde{\lambda}_n \approx 5 \cdot \lambda_n$ and $\tilde{\lambda}_n \approx 13 \cdot \lambda_n$ hold for the respective scenarios, especially for the popular items. This provides insight into the comparable hit ratios observed between LRU and LRU-agg in Figures 4a and 4b.

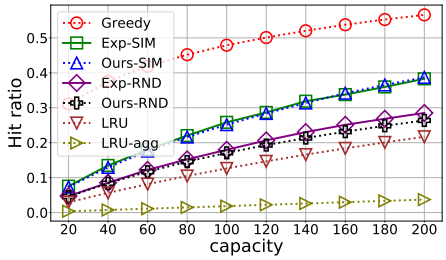
While our approach provides the best estimates, we can observe that it



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



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



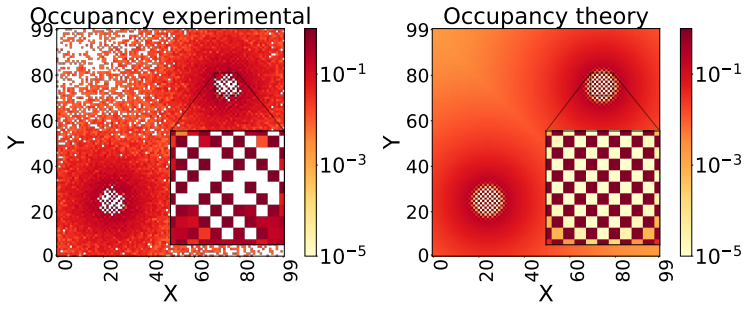
(c) Amazon trace, $d = 300$, 40 iterations.

Figure 4: Average hit ratio versus cache capacity, $\beta = 0.5$.

slightly underestimates the hit ratio. In order to understand this effect, we show in Figure 5 the empirically estimated occupancy vector and the one produced by Algorithm 2. The proposed algorithm broadly captures the empirical occupancy patterns, but with subtleties regarding symmetries. In particular, the zoom on Figure 5b 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 5a 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.

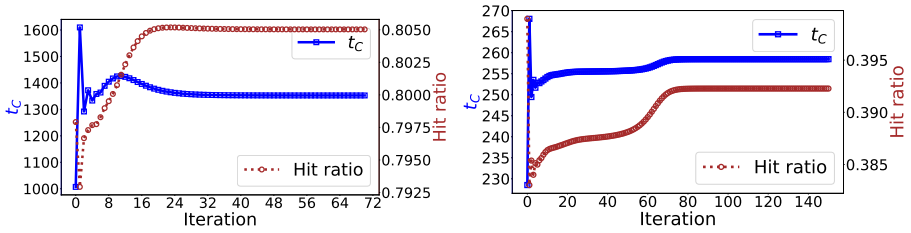
6.4. Convergence of Algorithm 2

The RND-TTL approximation selects the parameters $\boldsymbol{\lambda}^{\boldsymbol{i}}$, $\boldsymbol{\lambda}^{\boldsymbol{r}}$, and T for the RND-TTL cache in a way that ensures the occupancy vector, as described in (7), is a fixed point of the function $\mathbf{G}$ defined in (40a). Algorithm 2 employs



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

Figure 5: Synthetic trace occupancies: $C = 500$, $d = 1$, $\alpha = 2.5$.

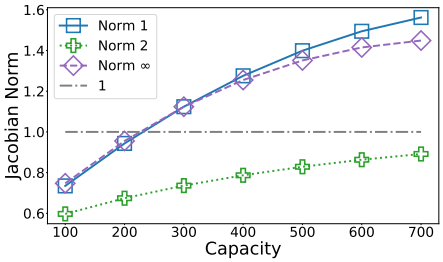


(a) Synthetic trace, $\alpha = 2.5$, $d = 1$, $C = 500$ (b) Amazon trace, $d = 300$, $C = 200$

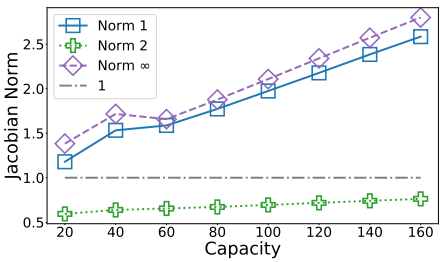
Figure 6: Characteristic time t_C and hit ratio in different iterations of Algorithm 2 for SIM-LRU.

an iterative procedure aimed at finding a fixed point of $\mathbf{G}$, thereby determining the appropriate values for the RND-TTL cache's parameters.

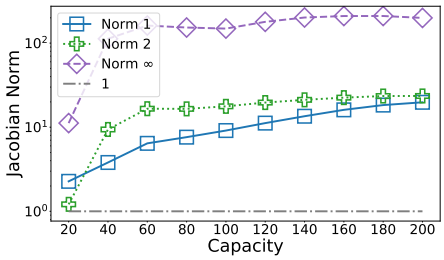
Figure 6 shows the evolution of characteristic time t_C and hit ratio H over different iterations. We observe that estimates of H and t_C by our algorithm converge in a few iterations (less than 70), under all considered scenarios. Note that $t_C(0)$, the value of t_C at iteration 0, is also the value of t_C for LRU (see (4)). In addition, across all experiments, t_C for Ours-SIM using Algorithm 2 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



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



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



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

Figure 7: Norm $\boldsymbol{J}_{\mathbf{G}_\beta}$ versus cache capacity, $\beta = 0.5$.

to approximate hits, explaining why t_C is larger for Ours-SIM than for LRU.

Proposition 6 provides a sufficient condition for the convergence of Algorithm 2 with parameter 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 7, 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 2, for the two synthetic traces and the Amazon trace described in Section 6.1. We provide in Appendix M details for the computation of $\boldsymbol{J}_{\mathbf{G}_\beta}$. In all the settings, β is set to 0.5. We observe in Figure 7 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, $\left\|\boldsymbol{J}_{\mathbf{G}_\beta}\right\|$ exceeds 1. Nevertheless, Proposition 6 provides only sufficient conditions and our results (see e.g. Figure 6b) suggest that our algorithm converges also on the Amazon trace.

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. In future work, we envision investigating analytically the accuracy of our RND-TTL approximation, similarly to what was done in [17, 19] for classic caching.

Acknowledgement

This project was funded in part by CAPES, CNPq, FAPERJ Grant JCNE:E-26/201.376/2021, in part by the French Government through the "Plan de Relance" and "Programme d'investissements d'avenir," and by Inria under the exploratory action MAMMALS.

References

[1] F. Falchi, C. Lucchese, S. Orlando, R. Perego, F. Rabitti, A Metric Cache for Similarity Search, in: Proc. of 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval, 2008, pp. 43–50.

[2] S. Pandey, A. Broder, F. Chierichetti, V. Josifovski, R. Kumar, S. Vassilvitskii, Nearest-neighbor caching for content-match applications, in: Proceedings of the 18th International Conference on World Wide Web, 2009, pp. 441–450.

[3] P. Sermpezis, T. Giannakas, T. Spyropoulos, L. Vigneri, Soft cache hits: Improving performance through recommendation and delivery of related content, IEEE Journal on Selected Areas in Communications 36 (2018) 1300–1313. doi:10.1109/JSAC.2018.2844983.

[4] U. Drolia, K. Guo, P. Narasimhan, Precog: Prefetching for image recognition applications at the edge, in: Proc. of ACM/IEEE Symposium on Edge Computing, 2017, pp. 1–13.

[5] S. Venugopal, M. Gazzetti, Y. Gkoufas, K. Katrinis, Shadow puppets: Cloud-level accurate AI inference at the speed and economy of edge, in: USENIX HotEdge , 2018.

[6] A. Finamore, J. Roberts, M. Gallo, D. Rossi, Accelerating deep learning classification with error-controlled approximate-key caching, in: IEEE Conference on Computer Communications (INFOCOM), 2022.

[7] G. Neglia, M. Garetto, E. Leonardi, Similarity Caching: Theory and Algorithms, IEEE/ACM Transactions on Networking (2021) 1–12. doi:10.1109/TNET.2021.3126368.

[8] M. Garetto, E. Leonardi, G. Neglia, Content placement in networks of similarity caches, Computer Networks 201 (2021) 108570.

[9] J. Zhou, O. Simeone, X. Zhang, W. Wang, Adaptive offline and online similarity-based caching, IEEE Networking Letters 2 (2020) 175–179.

[10] A. Sabnis, T. Si Salem, G. Neglia, M. Garetto, E. Leonardi, R. K. Sitaraman, Grades: Gradient descent for similarity caching, in: IEEE Conference on Computer Communications (INFOCOM), IEEE, 2021.

[11] T. Si Salem, G. Neglia, D. Carra, AÇAI: Ascent Similarity Caching with Approximate Indexes, in: 2021 33th International Teletraffic Congress (ITC-33), IEEE, 2021, pp. 1–9.

[12] R. Fagin, Asymptotic miss ratios over independent references, Journal of Computer and System Sciences 14 (1977) 222–250.

[13] W. King, Analysis of paging algorithms, in: Proc. IFIP 1971 Congress, Ljubljana, North-Holland, 1972, pp. 485–490.

[14] A. Dan, D. Towsley, An approximate analysis of the LRU and FIFO buffer replacement schemes, in: Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems, 1990, pp. 143–152.

[15] H. Che, Y. Tung, Z. Wang, Hierarchical web caching systems: Modeling, design and experimental results, IEEE journal on Selected Areas in Communications 20 (2002) 1305–1314.

[16] N. Choungmo Fofack, P. Nain, G. Neglia, D. Towsley, Performance evaluation of hierarchical TTL-based cache networks, Computer Networks 65 (2014) 212–231.

[17] C. Fricker, P. Robert, J. Roberts, A versatile and accurate approximation for LRU cache performance, in: 2012 24th International Teletraffic Congress (ITC 24), IEEE, 2012, pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=2414276.2414286.

[18] E. Leonardi, G. L. Torrisi, Least recently used caches under the shot noise model, in: IEEE Conference on Computer Communications (INFOCOM), IEEE, 2015, pp. 2281–2289.

[19] B. Jiang, P. Nain, D. Towsley, On the convergence of the TTL approximation for an LRU cache under independent stationary request processes, ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) 3 (2018) 1–31.

[20] Y. Ben Mazziane, S. Alouf, G. Neglia, D. S. Menasche, Computing the hit rate of similarity caching, in: GLOBECOM 2022-2022 IEEE Global Communications Conference, IEEE, 2022, pp. 141–146.

[21] J. McAuley, C. Targett, Q. Shi, A. Van Den Hengel, Image-based recommendations on styles and substitutes, in: Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval, 2015, pp. 43–52.

[22] E. Cohen, H. Kaplan, Proactive caching of DNS records: Addressing a performance bottleneck, Computer Networks 41 (2003) 707–726.

[23] N. Choungmo Fofack, S. Alouf, Modeling modern DNS caches, in: VALUETOOLS-7th International Conference on Performance Evaluation Methodologies and Tools, 2013, pp. 184–193.

[24] S. Alouf, N. Choungmo Fofack, N. Nedkov, Performance models for hierarchy of caches: Application to modern DNS caches, Performance Evaluation 97 (2016) 57–82. URL: `https://inria.hal.science/hal-01258189`. doi:10.1016/j.peva.2016.01.001.

[25] G. C. Moura, J. Heidemann, R. d. O. Schmidt, W. Hardaker, Cache me if you can: Effects of DNS Time-to-Live, in: Proceedings of the Internet Measurement Conference, 2019, pp. 101–115.

[26] J. Jung, A. Berger, H. Balakrishnan, Modeling TTL-based Internet caches, in: IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428), volume 1, 2003, pp. 417–426 vol.1. doi:10.1109/INFCOM.2003.1208693.

[27] A. Ferragut, I. Rodríguez, F. Paganini, Optimizing TTL caches under heavy-tailed demands, ACM SIGMETRICS Performance Evaluation Review 44 (2016) 101–112.

[28] S. Basu, A. Sundarrajan, J. Ghaderi, S. Shakkottai, R. Sitaraman, Adaptive TTL-based caching for content delivery, IEEE/ACM transactions on networking 26 (2018) 1063–1077.

[29] D. S. Berger, P. Gland, S. Singla, F. Ciucu, Exact analysis of TTL cache networks, Performance Evaluation 79 (2014) 2–23.

[30] G. Hasslinger, M. Okhovatzadeh, K. Ntougias, F. Hasslinger, O. Hohlfeld, An overview of analysis methods and evaluation results for caching strategies, Computer Networks 228 (2023) 109583.

[31] M. Garetto, E. Leonardi, V. Martina, A unified approach to the performance analysis of caching systems, ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) 1 (2016) 1–28.

[32] N. Gast, B. Van Houdt, TTL approximations of the cache replacement algorithms LRU (m) and h-LRU, Performance Evaluation 117 (2017) 33–57.

[33] S. M. Ross, Stochastic processes, John Wiley & Sons, 1995.

[34] S. Park, Ninety years of the Brouwer fixed point theorem, Vietnam J. Math 27 (1999) 187–222.

[35] W. R. Mann, Mean value methods in iteration, Proceedings of the American Mathematical Society 4 (1953) 506–510.

[36] N. Weaver, Lipschitz algebras, World Scientific, 2018.

[37] R. Meise, D. Vogt, Introduction to functional analysis, Clarendon Press, 1997.

[38] G. L. Nemhauser, L. A. Wolsey, M. L. Fisher, An analysis of approximations for maximizing submodular set functions—I, Mathematical programming 14 (1978) 265–294.

[39] S. Khuller, A. Moss, J. S. Naor, The budgeted maximum coverage problem, Information processing letters 70 (1999) 39–45.

[40] S. M. Ross, Introduction to probability models, Academic press, 2014.

[41] R. W. Wolff, Poisson arrivals see time averages, Operations research 30 (1982) 223–231.

[42] E. A. Van Doorn, G. Regterschot, Conditional PASTA, Operations Research Letters 7 (1988) 229–232.

[43] W. A. Rosenkrantz, R. Simha, Some theorems on conditional PASTA: A stochastic integral approach, Operations Research Letters 11 (1992) 173–177.

[44] O. R. B. de Oliveira, The implicit and the inverse function theorems: easy proofs, arXiv preprint arXiv:1212.2066 (2012).

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.

Appendix A. R-TTL

Algorithm 3 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 $j \in \{1, \ldots, J\}$, 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

Algorithm 3: 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 $S_1, \ldots, S_J$

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 3 returns a list of sets S_j for every $j \in \{1, \ldots, J\}$, 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 3 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.

Appendix B. Proof of Proposition 1 (Occupancy)

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 [40]. 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 [33, 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 $(Y_j)_{j \in \{1,\ldots,F\}}$ are exponentially distributed random variables with parameter λ_n^r such that $Y_j < T_n$ for $j = 1, \ldots, F$, 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:

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

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

Appendix C. 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 2–4. 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 $S \in \mathcal{E}$, $\lim_{t\to+\infty} \Pr\left(M(t) = S\right)$ 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 [41, 42]. 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,

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

Theorem 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 [41]:*

$$Q(t) = \sum_{S\in\mathcal{E}} \int_0^t \mathbb{1}\left(M(u) = S\right)\, dP_S(u). \tag{C.2}$$

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

$$\lim_{t\to+\infty}\frac{Q(t)}{t} = \lim_{t\to+\infty}\sum_{S\in\mathcal{E}}\left(\frac{P_S(t)}{t}\right)\cdot\left(\frac{1}{P_S(t)}\int_0^t \mathbb{1}\left(M(u)=S\right)\,dP_S(u)\right) \quad \text{(C.3)}$$

$$= \sum_{S\in\mathcal{E}}\lambda_S\cdot\lim_{t\to+\infty}\frac{1}{t}\int_0^t \mathbb{1}\left(M(u)=S\right)ds \quad \text{(C.4)}$$

$$= \sum_{S\in\mathcal{E}}\lambda_S\pi_S^*. \quad \text{(C.5)}$$

To obtain (C.4)*, we used the PASTA property [41] 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 [43, Sect. 3.3] for Markov-modulated Poisson processes.

Corollary 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*

$$\frac{Q(t)}{t}\xrightarrow{t\to+\infty}\sum_{i=1}^{l}\lambda_i\sum_{S\in B_i}\pi_S^*, \quad w.p.\ 1. \quad \text{(C.6)}$$

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

Appendix D. Proof of Proposition 2 (Item hit probability)

The proof uses the generalized PASTA property derived in Appendix C, 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 1. For any S in B_2, we redefine $\{P_S(t),\ t\geq 0\}$ as the request process for n thinned with probability $1-p_n^i$.

With $\{M(t),\ t\geq 0\}$ and $\{\{P_S(t),\ t\geq 0\},S\in\Omega\}$ redefined, Corollary 1 computes the rate of $\{Q(t),\ t\geq 0\}$, defined in (C.1), as $\lambda_n\cdot o_n+\lambda_n(1-p_n^i)(1-o_n)$,

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 $\lambda_n \cdot o_n + \lambda_n(1 - p_n^i)(1 - o_n)$.

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 $1 - p_n^i$. 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 $1 - p_n^i$ (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.

Appendix E. Proof of Proposition 3 (RND-LRU insertion rate)

The proof is similar to that for Proposition 2 in Appendix D.

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 $\tilde{B}_n = \{L \in \tilde{\Omega} :\ L \cap \mathcal{N}^c[n] = \emptyset\}$, $\tilde{B}_{n,m} = \{L \in \tilde{\Omega} :\ m \in L, L \cap \mathcal{N}_m^c[n] = \emptyset\}$ 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 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 1 computes the rate of $\{Q(t),\ t \geq 0\}$, defined in (C.1) as

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

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.

Appendix F. Proof of Proposition 4 (RND-LRU refresh rate)

As in Appendix E, 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 1, $\{P_L(t),\ t \geq 0\}$ is Poisson and its rate is given by,

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

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

$$\lim_{t \to +\infty} \frac{Q(t)}{t} = \sum_{L \in \tilde{\Omega}} \sum_{m \in \mathcal{N}^c[n]} q_n(m)\lambda_m \mathbb{1}(L \in \tilde{B}_{m,n})\, \tilde{\pi}_L \tag{F.2a}$$

$$= \sum_{m \in \mathcal{N}^c[n]} q_n(m)\lambda_m \sum_{L \in \tilde{\Omega}} \mathbb{1}(L \in B_{m,n})\tilde{\pi}_L \tag{F.2b}$$

$$= \sum_{m \in \mathcal{N}^c[n]} q_n(m)\lambda_m \sum_{L \in \tilde{B}_{m,n}} \tilde{\pi}_L. \tag{F.2c}$$

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 (F.2c).

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 (F.1). We conclude that $\{Q(t),\ t \geq 0\}$ is the number of times n's timer is refreshed until time t, finishing the proof.

Appendix G. Proof of Lemma 1 ($\boldsymbol{T_C(\mathbf{o})}$ is a singleton)

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 (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

$$\mathbf{o} \in \Delta_C \overset{(35)}{\Longrightarrow} |\mathcal{M}_{\mathbf{o}}| \leq C \tag{G.1}$$

$$\overset{(36)}{\Longrightarrow} \left| \bigcup_{m \in \mathcal{M}_{\mathbf{o}}} \mathcal{N}(m) \right| < N - C \tag{G.2}$$

$$\overset{\substack{\text{same} \\ \text{set}}}{\Longrightarrow} \left| \left\{ n \in \mathcal{I} : \prod_{m \in \mathcal{N}(n)} (1 - o_m) = 0 \right\} \right| < N - C \tag{G.3}$$

$$\Longrightarrow \left| \left\{ n \in \mathcal{I} : \prod_{m \in \mathcal{N}(n)} (1 - o_m) > 0 \right\} \right| > C \tag{G.4}$$

$$\overset{\substack{(25) \\ (31)}}{\Longrightarrow} |\{ n \in \mathcal{I} : \; E_n(\mathbf{o}) > 0 \}| > C \tag{G.5}$$

$$\overset{(29)}{\Longrightarrow} \left| \left\{ n \in \mathcal{I} : \lim_{T \to +\infty} g\left(R_n(\mathbf{o}), E_n(\mathbf{o}), T\right) = 1 \right\} \right| > C \tag{G.6}$$

$$\overset{(38)}{\Longrightarrow} \lim_{T \to +\infty} F(\mathbf{o}, T) > 0 \; . \tag{G.7}$$

The implication (G.1) is due to the definition of Δ_C (35), whereas (G.2) follows from the no-coverage condition (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 (G.2) as (G.3). The inequality in (G.4) follows by considering the complementary set. From the definition of the function $\mathbf{E}$ (see (25) and (31)), it comes that the set in (G.4) is included in the set in (G.5), which justifies writing (G.5). The implication (G.6) 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 (29)). Using the definition of $F(\mathbf{o}, T)$ in (38), it is straightforward to write (G.7). 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 (29) and (G.6)). Consequently, $F(\mathbf{o}, \cdot)$ is strictly increasing and T_0 is unique, which concludes the proof.

Appendix H. Proof of Lemma 2 (Differentiability of t_C)

We follow the steps of the proof of [44][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

$$\lim_{\epsilon \to 0} \frac{t_C(\mathbf{o} + \epsilon \mathbf{e}_j) - t_C(\mathbf{o})}{\epsilon} \ . \tag{H.1}$$

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 (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

$$F(B_j) - F(A) = \nabla F((1-\delta)A + \delta B_j) \cdot (B_j - A). \tag{H.2}$$

To ease the writing we define $L_j = (1-\delta)A + \delta B_j$. From the definition of t_C in (39), we have that $F(A) = F(B_j) = 0$. We expand then the right-hand side of (H.2) to write

$$\begin{aligned} &\epsilon \frac{\partial F}{\partial o_j}(L_j) + \big(t_C(\mathbf{o} + \epsilon \mathbf{e}_j) - t_C(\mathbf{o})\big) \frac{\partial F}{\partial T}(L_j) = 0 \\ &\qquad \Leftrightarrow \quad \frac{t_C(\mathbf{o} + \epsilon \mathbf{e}_j) - t_C(\mathbf{o})}{\epsilon} = -\frac{\partial F}{\partial o_j}(L_j) \left(\frac{\partial F}{\partial T}(L_j) \right)^{-1}, \end{aligned} \tag{H.3}$$

where we used the fact that $\frac{\partial F}{\partial T} > 0$ to write the last equality. (Recall from Appendix G that $F(\mathbf{o}, \cdot)$ is continuous and strictly increasing.) When $\epsilon \to 0$, L_j converges to A and (H.3) boils down to (41), completing the proof.

Appendix I. Proof of Proposition 7 (Computation of $J_{\mathbf{G}_\beta}$)

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.

$$\frac{\partial G_i}{\partial o_j}(\mathbf{o}) = \frac{\partial g}{\partial x_1}(P_i(\mathbf{o}))\frac{\partial R_i}{\partial o_j}(\mathbf{o}) + \frac{\partial g}{\partial x_2}(P_i(\mathbf{o}))\frac{\partial E_i}{\partial o_j}(\mathbf{o}) + \frac{\partial g}{\partial x_3}(P_i(\mathbf{o}))\frac{\partial t_C}{\partial o_j}(\mathbf{o}). \tag{I.1}$$

Observe that

$$\left(\frac{\partial g}{\partial x_1}(P_i(\mathbf{o}))\frac{\partial R_i}{\partial o_j}(\mathbf{o})\right)_{\substack{1\le i\le N\\ 1\le j\le N}} = \mathrm{Diag}\left(\boldsymbol{\partial_1 g}\right)\cdot \boldsymbol{J_R}(\mathbf{o}), \tag{I.2}$$

and

$$\left(\frac{\partial g}{\partial x_2}(P_i(\mathbf{o}))\frac{\partial E_i}{\partial o_j}(\mathbf{o})\right)_{\substack{1\le i\le N\\ 1\le j\le N}} = \mathrm{Diag}\left(\boldsymbol{\partial_2 g}\right)\cdot \boldsymbol{J_E}(\mathbf{o}). \tag{I.3}$$

To prove (47), it suffices to prove that

$$\left(\frac{\partial g}{\partial x_3}(P_i(\mathbf{o}))\frac{\partial t_C}{\partial o_j}(\mathbf{o})\right)_{\substack{1\le i\le N\\ 1\le j\le N}} = -\frac{\boldsymbol{\partial_3 g}^{\intercal}}{\boldsymbol{\partial_3 g}\cdot \mathbf{1}}\cdot\left(\boldsymbol{\partial_1 g}\cdot \boldsymbol{J_R}(\mathbf{o}) + \boldsymbol{\partial_2 g}\cdot \boldsymbol{J_E}(\mathbf{o})\right). \tag{I.4}$$

To this aim, we compute the partial derivatives of t_C. These are given in Lemma 2 as follows:

$$\frac{\partial t_C}{\partial o_j}(\mathbf{o}) = -\frac{\partial F}{\partial o_j}(\mathbf{o}, t_C(\mathbf{o}))\cdot\left(\frac{\partial F}{\partial T}(\mathbf{o}, t_C(\mathbf{o}))\right)^{-1}. \tag{I.5}$$

From the definition of F (see (38)) and the chain rule, we get

$$\frac{\partial F}{\partial o_j}(\mathbf{o}, t_C(\mathbf{o})) = \sum_{i\in\mathcal{I}}\frac{\partial g}{\partial x_1}(P_i(\mathbf{o}))\frac{\partial R_i}{\partial o_j}(\mathbf{o}) + \sum_{i\in\mathcal{I}}\frac{\partial g}{\partial x_2}(P_i(\mathbf{o}))\frac{\partial E_i}{\partial o_j}(\mathbf{o}), \tag{I.6}$$

$$\frac{\partial F}{\partial T}(\mathbf{o}, t_C(\mathbf{o})) = \sum_{i\in\mathcal{I}}\frac{\partial g}{\partial x_3}(P_i(\mathbf{o})) = \boldsymbol{\partial_3 g}\cdot\mathbf{1}\ . \tag{I.7}$$

Substituting (I.6) and (I.7) into (I.5) we deduce (I.4). This concludes the proof.

Appendix J. Time Complexity of Single Iteration in Algorithm 2

In each iteration of Algorithm 2, 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 (31), (32) and (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 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 2 is $\mathcal{O}(\mathcal{K})$.

Appendix K. Proof of Proposition 8 (Properties of $Y(\mathbf{o})$)

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 (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 matrix. Letting $\boldsymbol{A} = \boldsymbol{J}_{\mathbf{G}}(\mathbf{o})$, we can start from (47) to write

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

Equation (K.1b) follows from the definition of the spectral norm, i.e., $\|\boldsymbol{M}\|_2 = \sqrt{\rho(\boldsymbol{M}\boldsymbol{M}^\intercal)}$. We can write (K.1c) 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, (K.1d) follows. We can write (K.1e) thanks to

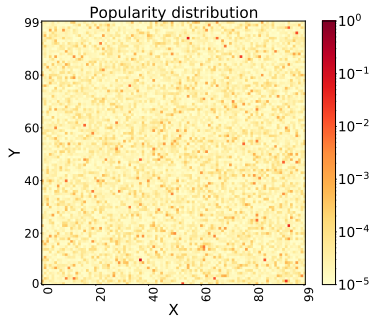


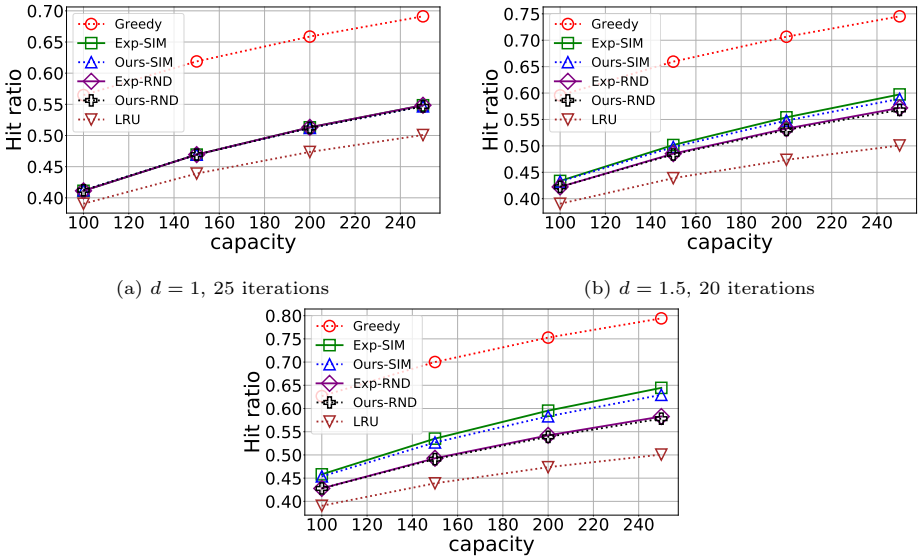
Figure 8: Spatial popularity distribution: Zipf with exponent $z = 1.0$.

the triangular inequality, and (K.1f) using again that $\|\boldsymbol{M}\|_2 = \rho(\boldsymbol{M})$ when $\boldsymbol{M}$ is symmetric and then the definitions of γ and η in (53) and (54), respectively. Equation (K.1g) readily follows. The upper bound expressed in (K.1g) 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 $\eta < \min\{2, \gamma+1\}$.

Appendix L. 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 $\text{dis}(\cdot, \cdot)$ 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 8 illustrates the spatial popularity distribution for $z = 1.0$.

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 $(\text{dis}(n, m))^{-2}$ if $\text{dis}(n, m) \leq d$ and 0 otherwise. Note



(a) $d = 1$, 25 iterations

(b) $d = 1.5$, 20 iterations

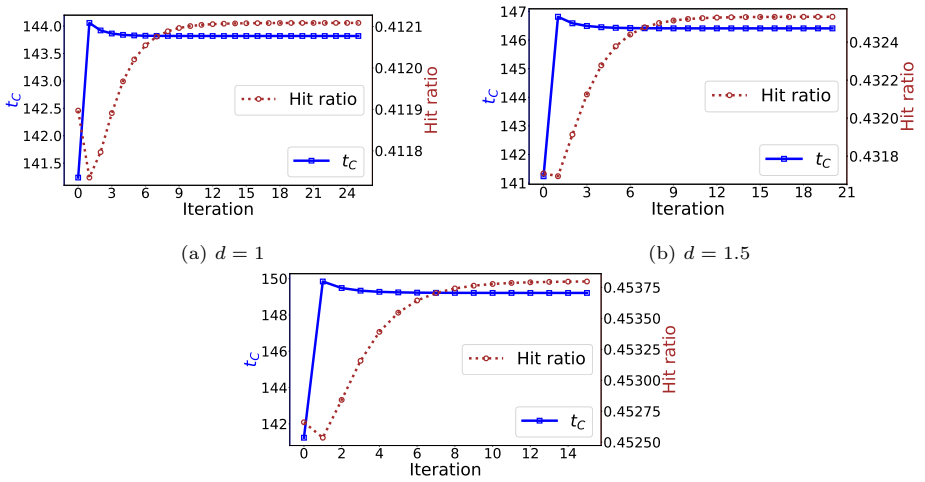
(c) $d = 2.0$, 15 iterations

Figure 9: Hit ratio versus cache capacity: $r = 2 \cdot 10^5$, Zipf with exponent $z = 1.0$, $\beta = 0.5$.

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 2 with parameter $\beta = 0.5$ and a stopping condition determined by a fixed number of iterations. Algorithm 2 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 6.2, like LRU and Greedy.

Figure 9 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 10 illustrates the values of the characteristic time t_C and the hit ratio H over different iterations of Algorithm 2. The findings shown in Figure 10 validate that Algorithm 2 converges within a few iterations.



(a) $d = 1$

(b) $d = 1.5$

(c) $d = 2.0$

Figure 10: Characteristic time t_C and hit ratio in different iterations of Algorithm 2 for SIM-LRU: Zipf with exponent $z = 1.0$, $\beta = 0.5$.

Table 3: Average runtime per iteration in Algorithm 2: $C = 100$, 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

Table 3 provides details on the average runtime per iteration in Algorithm 2 for $d = 1.0$, $d = 1.5$ and $d = 2.0$ when the number of iterations is respectively 25, 20 and 15.

Appendix M. Implementation Details

When computing $\left\|\boldsymbol{J}_{\mathbf{G}_\beta}(\mathbf{o})\right\|$, we follow a specific procedure. First, we use the formula in (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 [45]. However, we do not use this function in the computation of the

vectors $\boldsymbol{\partial_1 \mathbf{g}}$, $\boldsymbol{\partial_2 \mathbf{g}}$ and $\boldsymbol{\partial_3 \mathbf{g}}$ to avoid potential errors that may arise from floating point precision. Instead, we implement these vectors separately, ensuring accurate results.