Content placement in networks of similarity caches

Content Placement in Networks of Similarity Caches

Michele Garetto[a,*], Emilio Leonardi[b], Giovanni Neglia[c]

[a] *Università degli Studi di Torino, C.so Svizzera 185, Torino, Italy*
[b] *Politecnico di Torino, C.so Duca degli Abruzzi 24, Torino, Italy*
[c] *Inria - Université Côte d'Azur, 2004 route des Lucioles, Sophia Antipolis, France*

Abstract

Similarity caching systems have recently attracted the attention of the scientific community, as they can be profitably used in many application contexts, like multimedia retrieval, advertising, object recognition, recommender systems and online content-match applications. In such systems, a user request for an object o, which is not in the cache, can be (partially) satisfied by a similar stored object o', at the cost of a loss of user utility. In this paper we make a first step into the novel area of similarity caching networks, where requests can be forwarded along a path of caches to get the best efficiency-accuracy tradeoff. The offline problem of content placement can be easily shown to be NP-hard, while different polynomial algorithms can be devised to approach the optimal solution in discrete cases. As the content space grows large, we propose a continuous problem formulation whose solution exhibits a simple structure in a class of tree topologies. We verify our findings using synthetic and realistic request traces.

Keywords: Cache networks, Similarity search, Content distribution

1. Introduction

Similarity caching is an extension to traditional (exact) caching, whereby a request for an object can be satisfied by providing a similar cached item, under a

*Corresponding author

Email addresses: michele.garetto@unito.it (Michele Garetto), emilio.leonardi@polito.it (Emilio Leonardi), giovanni.neglia@inria.fr (Giovanni Neglia)

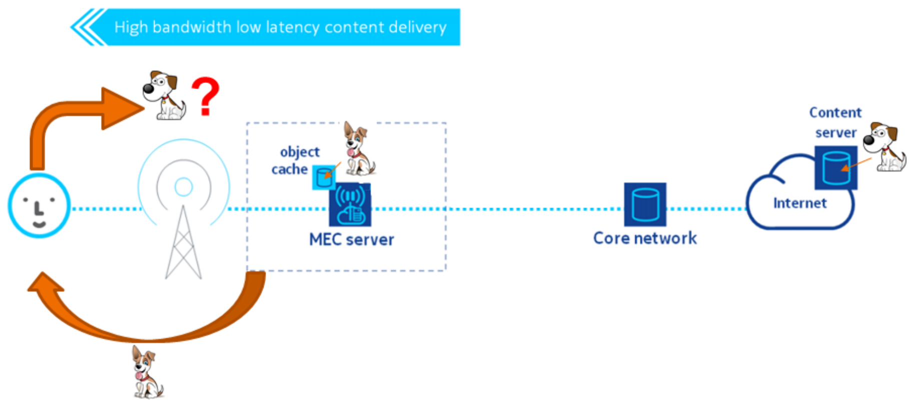

Figure 1: Application of similarity caching to Mobile Edge Computing.

dissimilarity cost. In some cases, user requests are themselves queries for objects similar to a given one (similarity searching [1]). Caching at network edges can drastically reduce the latency experienced by users, as well as backbone traffic and server provisioning.

Similarity searching and caching have several applications in multimedia retrieval [2], contextual advertising [3], object recognition [4, 5, 6, 7], caching of videos with different qualities/resolutions [8, 9, 10], recommender systems [3, 11], online prediction serving systems [12, 13, 14]. Figure 1 shows an application scenario of similarity caching in the context of Mobile Edge Computing [15]. Mobile users accessing cloud services from an ultra-low latency, high bandwidth cellular wireless networks (e.g., 5G) can strongly benefit from the availability of a cache installed directly at the radio network controller: in such a way, the need for a particular object, for example by an Augmented Reality application, can be satisfied locally with minimum latency by a sufficiently similar object, without having to traverse a possible long path towards the object repository. References [4, 5, 6, 7] consider this specific scenario for object recognition applications. In this case, the request is an image, for which the MEC server provides some labels extracted from similar images found in a local database. Note that several caches can be deployed along the path from the user to the cloud (e.g., at micro regional data centers) forming a tree-like similarity caching network (in the case of many geographically spread users).

Despite its interesting applications, theoretical understanding of similarity caching and development of related algorithms and policies are still at their early stages.

1.1. Paper contribution

Our contributions can be summarized as follows:

1. while the content placement problem in networks of similarity caches is NP-hard, we show that it can be formulated as the maximization of a sub-modular function over a matroid; therefore a polynomial Greedy algorithm can be defined with $1/2$ approximation ratio;

2. we propose the randomized LocalSwap algorithm that does not enjoy worst-case guarantees as Greedy, but asymptotically converges to a locally optimal solution;

3. we characterize the structure of the optimal similarity-caching placement problem in special cases; in particular, we show that, under mild assumptions, when the cache network has a regular tree structure and requests arrive only at the leaves the optimal solution in the large catalog regime has a relatively simple structure;

4. we show that the above structure is lost in general networks, analyzing a simple tandem network where requests arrive at both caches;

5. we propose an online, λ-unaware policy called NetDuel, that extends Duel [16] to the networked setting;

6. we illustrate our findings considering both synthetic and real request processes for Amazon items.

1.2. Paper outline

We discuss related work in Section 2. In Section 3 we present the main system assumptions and we formulate the problem. In Sections 4 and 5 we analyse respectively the discrete content case and the continuous content case, presenting algorithms and theoretical performance results. In Section 6 we introduce NetDuel, an efficient on-line caching policy. In Section 7 we report simulation results, obtained both in synthetic scenarios and more realistic scenarios based on Amazon traces. We conclude in Section 8, pointing out directions of future research.

2. Related Work

Despite the multiple applications of similarity caching, our theoretical understanding of the general problem is still limited even in the single-cache scenario, and similarity caching policies have mostly been proposed in an ad-hoc way without taking advantage of the body of work built in the last decades for exact caching (e.g., [17, 18, 19, 20]).

For example the seminal papers [2, 3], which introduced the concept of similarity caching, proposed only simple modifications to the Least Recently Used policy (LRU) and evaluated them empirically. Similarly, references [12, 4, 5, 6, 21, 7] focused more on the specific application system (machine learning prediction serving and object recognition), without specific contributions in terms of cache management policies (e.g., they apply minor changes to exact caching policies like LRU or LFU).

An adversarial setting was studied in [22] by competitive analysis. The authors of [23] have proposed a similarity caching policy (for a single cache) tailored for the case when cached objects may be embedded in $\mathbb{R}^d$ with a distance that captures dissimilarity costs. The work most closely related to this paper is [16], where we have analyzed a single similarity cache in the offline, adversarial, and stochastic settings, proposing also some dynamic online policies to manage the cache.

We mention that many researchers have studied networks of exact caches (e.g., [24, 25, 17, 19, 26, 27]), however their results cannot be applied to the similarity caching setting, which is a fundamentally different problem (in exact caching there is no notion of distance between objects).

Networks of caches for videos with different qualities have been studied in [8, 9, 10], but references [9, 10] consider a single layer of caches deployed at the edge of the network (the request is served by one of these caches or forwarded to the authoritative server), while we study more complex architectures like trees. The authors of [8] consider a general architecture, but, while they correctly model user's QoE dependence on video quality, they ignore the cost of retrieving the videos from farther caches. Moreover, video placement is based on heuristic policies with no performance guarantees.

Similarity caches for content recommendation have been considered in [11, 28]. The authors have studied how to statically place contents in edge caches of a cellular network, given their popularity and the utility for a user interested in content o to receive a similar content o'. In contrast to us, they focus on the cellular scenario with spatial cache overlaps (also known as "femtocaching" [29]).

The recent letter [30] has considered a network of similarity caches, where requests can be forwarded along a path of caches towards a repository storing all objects, at the cost of increasing delays and resource consumption. The authors of [30] have proposed a heuristic based on the gradient descent/ascent algorithm to jointly decide request routing and caching, similarly to what was done in [19] for exact caches but without the corresponding theoretical guarantees. The proposed algorithm requires memory proportional to the size of the catalog, and appears to be computationally feasible only on small-scale systems. Table 1 offers a schematic summary of previous work, considering as criteria the application context, the network architecture, and the catalog type (discrete/continuous).

In our work, similarly to [30], we focus mainly on the offline setting, i.e., the problem of statically placing objects in the caches so as to minimize the expected

Works	Application	Architecture	Catalog
[2, 3, 23, 22]	generic	single cache	discrete
[16]	generic	single cache	discrete/continuous
[12, 4, 5, 6, 21, 7]	machine learning	single cache	discrete
[9, 10]	video	single layer	discrete
[8]	video	network of caches	discrete
[30]	generic	network of caches	discrete
this paper	generic	network of caches	discrete/continuous

Table 1: Schematic summary of previous work on similarity caching.

cost under known content request rates and routing. In contrast to [30], we first propose algorithms with guaranteed performance, and then we move to the continuous limit of the large requests/catalog space, where we investigate the *structure* of the optimal solution.

In the recent publication [14], one of the authors has proposed the idea of inference delivery networks, an Internet-wide architecture for fast delivery of machine learning predictions. Inference delivery networks can be seen as a particular network of similarity caches. Beside the focus on a specific application, reference [14] considers an adversarial request process for a finite number of possible objects (machine learning models in their case), while we focus on more common stochastic request process and consider both finite and infinite catalogs of objects.

In summary, our paper advances the state of art by providing a first analysis of networks of similarity caches in the same spirit of works devoted to networks of exact caches. Specifically, we focus on the offline setting and characterize the structure of the optimal solution in the large catalog regime.

3. Main assumptions and problem formulation

Let $\mathcal{X}$ be the (finite or infinite) set of objects that can be requested by the users. We assume that all objects have equal size and cache i can store up to k_i objects.

We consider a network of caches with requests potentially arriving at every node. Some nodes can act as content repositories, where (a subset of) requests can be satisfied exactly or with a small approximation cost. Specifically, we assume that each request has at least one repository acting as 'authoritative server' for it, meaning that the approximation cost at the content repository is either zero or it is negligible as compared to the fixed cost to reach the repository (see next). Let $\mathcal{K}$ be the set of all nodes in the network (including caches and repositories).

A request r is a pair (o, i) where o is the requested object and i is the node where the request first enters the network. Every request is issued according to a Poisson process with rate λ_r.

At each cache, for any two objects x and y in $\mathcal{X}$ there is a non-negative (potentially infinite) cost $C_a(x, y)$ to locally approximate x with y. We consider $C_a(x, x) = 0$. We assume that caches can efficiently compute, upon arrival of a request for x, the closest stored object y. This is typically done resorting to locality sensitive hashing (LSH) [3].

Moreover, there is an additional retrieval cost $h(i, j)$ to reach node j from cache i, which is assumed to increase as more and more hops need to be traversed by the request. Costs $h(i, j)$ represent the additional penalty (in terms of network delay) incurred by requests, in addition to the approximation cost C_a. If a request from i cannot be forwarded to cache j, then $h(i, j) = +\infty$.

We call an approximizer α a pair (o', j), where object o' has been placed at cache j. If a request $r = (o, i)$ is served by object o' at node j, it will incur a total cost $C(r, \alpha) = C_a(o, o') + h(i, j)$, that depends on how dissimilar o is from o' and how far node i is from node j. For approximizers located at a content repository j, we take $C(r, \alpha) = h(i, j)$, neglecting the local approximation cost.

We assume that each cache knows how to route each request to a corresponding repository. Nevertheless, deciding if a request should be served locally or should be forwarded along the path to the repository is still a challenging problem to solve in a distributed way: while a relatively good approximizer can be found at a cache i, a better one may be located at an upstream cache j, justifying the additional cost $h(i,j)$. This is in sharp contrast to what happens in exact caching network, where the forwarding operation is straightforward (a request is forwarded upon a miss).

In our initial investigation, we will suppose that optimal forwarding strategy is available at all caches, i.e., that each cache knows whether to solve a request locally or forward it towards the repository. This assumption is reasonable in two possible scenarios: i) when caches exchange meta-data information about their stored objects (this is acceptable when content is static or quasi-static); ii) when the dominant component of the delay is content download, so that, prior to download, small request messages can go all the way up to the repository and back, dynamically finding the best approximizer along the path. We leave to future work the challenging case in which optimal forwarding is not available at the nodes.

A consequence of our assumptions is that each request r will be served minimizing the total cost, i.e., given $\mathcal{S}$ the initial set of approximizers at content repositories, and $\mathcal{A}$ the set of approximizers at the caches, we have

$$C(r,\mathcal{A}) = \min_{\alpha \in \mathcal{A} \cup \mathcal{S}} C(r,\alpha). \tag{1}$$

In what follows we will consider two main instances for $\mathcal{X}$ and $C_a()$. In the first instance, $\mathcal{X}$ is a finite set of objects and thus the approximation cost can be characterized by an $|\mathcal{X}| \times |\mathcal{X}|$ matrix of non-negative values. This case could well describe the (dis)similarity of contents (e.g. videos) in a finite catalog. In the second instance, $\mathcal{X}$ is a subset of $\mathbb{R}^p$ and $C_a(x,y) = f(d(x,y))$, where $f : \mathbb{R}^+ \to \mathbb{R}^+$ is a non-decreasing non-negative function and $d(x,y)$ is a metric in $\mathbb{R}^p$ (e.g. the Euclidean one). This case is more suitable to describe objects characterized by continuous features, as in machine learning applications. For example, consider a query to retrieve similar images, as one can issue to `images.google.com`. The set of images the user may query Google for is essentially unbounded, and in any case it is larger than the catalog of images Google has indexed.

In the continuous case, we assume a spatial density of requests arriving at each cache defined by a Borel-measurable function $\lambda_{x,i} : \mathcal{X} \times \mathcal{K} \to \mathbb{R}_+$, i.e., for every Borel set $\mathcal{B} \subseteq \mathcal{X}$, and every cache $i \in \mathcal{K}$, the rate with which requests for objects in $\mathcal{B}$ arrive at node i is given by $\int_{\mathcal{B}} \lambda_{x,i} \, \mathrm{d}x$. We will refer to the above two instances as *discrete* and *continuous*, respectively.

Under the above assumptions, our goal is to find the optimal static allocation $\mathcal{A}$ that minimizes the expected cost $\mathcal{C}(\mathcal{A})$ per time unit (or per request, if we normalize the aggregate request arrival rate to 1):

$$\mathcal{C}(\mathcal{A}) \triangleq \begin{cases} \sum_r \lambda_r C(r, \mathcal{A}), & \text{discrete case} \\ \sum_{i \in \mathcal{K}} \int_{\mathcal{X}} \lambda_{x,i} C((x,i), \mathcal{A}) \, \mathrm{d}x, & \text{continuous case} \end{cases} \tag{2}$$

i.e.,

$$\begin{aligned} \underset{\mathcal{A}}{\text{minimize}} \quad & \mathcal{C}(\mathcal{A}) \\ \text{subject to} \quad & \sum_{o:(o,i) \in \mathcal{A}} 1 \leq k_i, \quad \forall i \in \mathcal{K} \end{aligned} \tag{3}$$

Having mathematically formalized the problem, in the next section we take an algorithmic perspective to characterize and approximate its optimal solution.

4. Algorithms for the Discrete case

In this section, we restrict ourselves to the discrete scenario, as this allows us to make rigorous statements about NP-hardness and algorithms' complexity.

4.1. NP-Hardness and Submodularity

Proposition 4.1. *The static off-line similarity caching problem in a network* (3) *is NP-hard.*

This is an immediate consequence of the fact that, as shown in [16, Thm. III.1], the static off-line similarity caching problem is already NP-hard for a single cache. Nevertheless, we will show in Sec. 5 that, when the cache network has a regular tree structure, a simple characterization of the optimal solution can be determined in the large catalog regime, by exploiting a continuous approximation.

Given the initial set $\mathcal{S}$ of objects allocated at content repositories, we want to pick an additional set $\mathcal{A}$ of objects and place them at the caches. Let $\mathcal{I}$ denote the set of possible allocations that satisfy cardinality constraints at each cache (corresponding to the constraints in (3)). Let $G(\mathcal{A})$ quantify the *caching gain* [31, 19] from allocation $\mathcal{A}$ in comparison to the case when each request needs to be served by its content repository, i.e.,

$$G(\mathcal{A}) = \mathcal{C}(\emptyset) - \mathcal{C}(\mathcal{A}). \tag{4}$$

Problem (3) is equivalent to the following maximization problem

$$\underset{\mathcal{A}\in\mathcal{I}}{\text{maximize}} \quad G(\mathcal{A}). \tag{5}$$

Proposition 4.2. *The static off-line similarity caching problem in a network is a submodular maximization problem with matroid constraints.*

The result does not rely on any specific assumption on $C(r, \alpha)$ but for the cost being non-negative. In particular, we can define $C(r, \alpha)$ to embed requests' routing constraints. For example, given a request $r = (o, i)$, we can enforce the request to be satisfied by the repository of content o or by one of the caches on the routing path between node i and the repository (we denote it as $P_{i,o}$). This constraint can be imposed by selecting $C((o, i), (o', j)) = \infty$ for each $j \notin P_{i,o}$. The proof is quite standard and we report it in Appendix A for completeness.

In the next subsections we introduce two different algorithms to deal in practice with the off-line similarity caching problem.

4.2. Greedy *algorithm and its complexity*

As Problem (5) is the maximization of a monotone non-negative submodular function with matroid constraints, the Greedy algorithm has 1/2 guaranteed approximation ratio, i.e., $G(\mathcal{A}_{\text{Greedy}}) \geq \frac{1}{2}\max_{\mathcal{A}\in\mathcal{I}} G(\mathcal{A})$ [32]. We mention that there exists also a randomized algorithm that combines a continuous greedy process and pipage rounding to achieve a $1 - 1/e$ approximation ratio *in expectation* [33].

The Greedy algorithm proceeds from an empty allocation $\mathcal{A} = \emptyset$ and progressively adds to the current allocation an approximizer in $\text{argmax}_\alpha G(\mathcal{A} \cup \{\alpha\}) - G(\mathcal{A}) = \text{argmax}_\alpha \sum_r \lambda_r (C(r, \mathcal{A}) - C(r, \mathcal{A} \cup \{\alpha\}))$ up to select $\sum_i k_i = K$ objects, where K is the total cache capacity in the network (by respecting local constraints at individual caches). The detailed pseudocode is reported in Appendix B.

Let O, O_R, and N denote the number of objects in the catalog, the number of objects that can be requested, and the number of caches in the network. When choosing the i-th approximizer the greedy algorithms needs in general to evaluate $ON - i + 1$ possible approximizers, and how they reduce the cost for the set of requests with cardinality at most $O_R N$. The time-complexity of the algorithm is then bounded by $\sum_{i=1}^{K} O_R N(ON - i + 1) = O_R N(ONK - K(K-1)/2)$. A smart implementation can avoid to evaluate the gain of all possible approximizers at each step, but despite the optimizations, the Greedy algorithm would be too complex for catalogue sizes O beyond a few thousands of objects. Moreover, the set of possible requested objects O_R may be much larger than O.

4.3. LocalSwap *algorithm and its complexity*

We now present a different algorithm, called LocalSwap, which is based on the simple idea to systematically move to states with a smaller expected cost (2). LocalSwap can be used both in an off-line and on-line scenario. It works as follows. At the beginning the state of caches is populated by random contents. Then, in the on-line scenario the algorithm adapts the cache state upon every request. In the off-line scenario, instead, a sequence of emulated requests is generated (satisfying the same statistical properties of the original arrival process), and applied to drive cache state changes. Let $\mathcal{A}_t$ be the allocation obtained by the algorithm at iteration t. Upon an (emulated) request r for o, LocalSwap computes the maximum decrement in the expected cost that can be obtained by replacing one of the objects currently stored at some cache along the forwarding path with o, i.e., $\Delta\mathcal{C} \triangleq \min_{\alpha \in \mathcal{A}_t} \mathcal{C}(\mathcal{A}_t \cup \{r\} \setminus \{\alpha\}) - \mathcal{C}(\mathcal{A}_t)$.

- if $\Delta\mathcal{C} < 0$, then cache i_e replaces content y_e with content o, where $(y_e, i_e) \in \arg\min_{(y,i)\in\mathcal{A}_t} \mathcal{C}(\mathcal{A}_t \cup \{(o,i)\} \setminus \{(y,i)\})$;
- if $\Delta\mathcal{C} \geq 0$, the cache allocation is not updated.

The detailed pseudocode is reported in Appendix C.

LocalSwap does not provide worst case guarantees as Greedy, but it asymptotically reaches a locally optimal cache configuration, defined as a configuration whose cost (2) is lower than the cost of all configurations that can be obtained by replacing just one content in one cache. On the contrary, Greedy does not necessarily reach a local optimal state (as we show below in Sect. 4.4).

Proposition 4.3. *For long enough request sequence* LocalSwap *converges with probability 1 to a locally optimal cache configuration.*

LocalSwap generalizes a similar algorithm proposed in [16] for a single cache (called there "greedy") with similar theoretical guarantees. Under the assumption that requests are optimally forwarded, the proof of Proposition 4.3 is essentially the same of [16, Thm. V.3], so we omit it. By clever data structure design, the computational cost of each iteration can be kept $\mathcal{O}(NO_R)$.

Remark 1. Note that by cascading Greedy and LocalSwap it is possible to achieve a locally optimal cache configuration whose approximation ratio is guaranteed to be at least 1/2 (i.e., $G(\mathcal{A}_{\text{Greedy+LocalSwap}}) \geq \frac{1}{2}\max_{\mathcal{A}\in\mathcal{I}} G(\mathcal{A})$).

4.4. Greedy *and* LocalSwap *in a toy example*

This example shows that 1) Greedy does not converge necessarily to a locally optimal cache configuration, and 2) there are both settings where Greedy finds the optimal cache configuration while LocalSwap may not, and settings where LocalSwap finds the optimal cache configuration while Greedy does not.

Consider a scenario with 5 contents x_i for $1 \leq i \leq 5$. Let us assume that $C_a(x_2, x_3) = C_a(x_3, x_4) = 0$, $C_a(x_1, x_2) = C_a(x_4, x_5) = \epsilon > 0$[1], while $C_a(x_i, x_j) = \infty$ otherwise. We want to solve the content placement problem for a single cache with $k = 2$ and $\lambda_{x_3} > \lambda_{x_2} = \lambda_{x_4} > \lambda_{x_1} = \lambda_{x_5}$. The cost to retrieve the objects from the remote server is $h_s > 2\epsilon$. The optimal placement configuration is: $\{x_2, x_4\}$. GREEDY will reach one of the following equivalent sub-optimal configurations $\{x_3, x\}$, with $x \in \{x_1, x_5\}$. LOCALSWAP, on the contrary, will reach the optimal configuration $\{x_2, x_4\}$ (because it is the unique locally optimal configuration). We observe that the configurations reached by GREEDY are not locally optimal: for example if GREEDY selects $\{x_3, x_1\}$, it is convenient to replace x_3 with x_4.

If we consider two caches 1 and 2 in tandem, each of size $k = 1$ with requests arriving only to the first cache and retrieval cost equal to $h(1, 2)$ if the object is retrieved from cache 2, and $h(1, 2) + h_s$ if it is retrieved by the server. The optimal configurations will maintain a similar structure for $h(1, 2)$ small enough. In particular the optimal configurations will be: $\{(x_4, 1), (x_2, 2)\}$ and $\{(x_2, 1), (x_4, 2)\}$. GREEDY will still reach a state $\{(x_3, 1), (x, 2)\}$ with $x \in \{x_1, x_5\}$, while LOCALSWAP will reach an optimal state. For $h(1, 2)$ large enough the optimal states become $\{(x_3, 1), (x, 2)\}$ with $x \in \{x_1, x_5\}$ and both previous algorithms will succeed in reaching an optimal solution. At the same time there are settings for which the configurations $\{(x_3, 1), (x_1, 2)\}$ and $\{(x_3, 1), (x_5, 2)\}$ correspond to global minima, the configurations $\{(x_4, 1), (x_2, 2)\}$ and $\{(x_2, 1), (x_4, 2)\}$ correspond to local minima, and GREEDY finds one of the first configurations, while LOCALSWAP may reach one of the second configurations. For example this is the case for $h_s = 1$, $h(1, 2) = \epsilon = 4/9$, $\lambda_1 = \lambda_5 = 1$, and $\lambda_2 = \lambda_4 = 4/3$ and any $\lambda_3 > \lambda_2$.

[1] All costs are assumed to be symmetric.

5. The Continuous case

When O_R is much larger than O, or O is itself very large, it makes sense to study the request space as continuous. Such continuous representation permits us to formulate a simplified optimization problem whose solution well approximates the optimal cost achieved in *discrete* scenarios with large catalog size.

If the number of objects in the catalog is finite, one could in principle devise a GREEDY algorithm also for this case, working exactly as in the *discrete* case. Indeed the problem (3) can be easily shown to be still submodular even when requests lies over a continuous space. However, one now has to evaluate, for each possible candidate approximizer α to add to the current allocation, complex integrals over the infinite query space. It is not simple to define in general the complexity of such operations but it is evident that previous algorithmic approaches becomes rapidly unfeasible for large set of requests and/or large catalog.

Hereinafter, we will assume that both the request space and the catalog space are continuous.

5.1. Preliminary: continuous formulation for a single cache

As a necessary background, we summarize here some results obtained in [16] for the case of a single cache with capacity k_1. Let $\mathcal{B}_r(y_0)$ be the closed ball of radius r around y_0, i.e., the set of points y such that $d(y, y_0) \leq r$. The authors of [16] proved:

Proposition 5.1. *Under a homogeneous request process with intensity λ over a bounded set $\mathcal{X}$, any cache state $\mathcal{A} = \{y_1, \ldots, y_{k_1}\}$, such that, for some r, the balls $\mathcal{B}_r(y_h)$ for $h = 1, \ldots, k_1$ are a tessellation of $\mathcal{X}$ (i.e., $\cup_h \mathcal{B}_r(y_h) = \mathcal{X}$ and $|\mathcal{B}_r(y_i) \cap \mathcal{B}_d(y_j)| = 0$ for each i and j), is optimal.*

Such regular tessellation exists, in all dimensions, under the norm-1 distance, and corresponds to the case in which balls are squares (assuming that k_1 such squares cover exactly the domain $\mathcal{X}$).

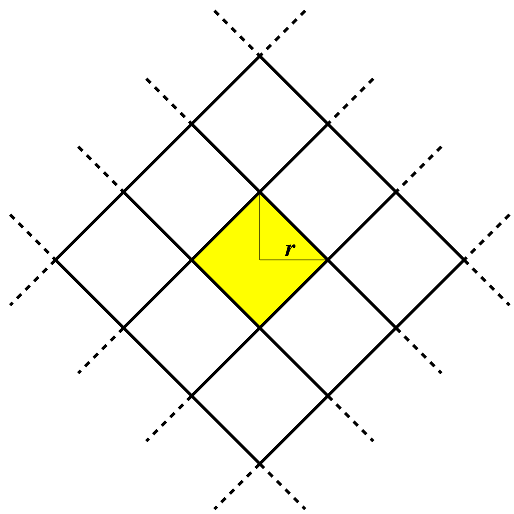

Figure 2: Perfect tessellation with square cells in a two-dimensional domain, under the norm-1 distance.

It is then immediate to analytically compute the optimal cost for this case. For example, in a two-dimensional domain (see Fig. 2), requests arriving in a particular ball produce an approximation cost:

$$c(r) = 4\int_0^r \int_0^{r-x} (x+y)^\gamma \lambda \, \mathrm{d}y \, \mathrm{d}x = 4\lambda \frac{r^{\gamma+2}}{\gamma+2} \tag{6}$$

and the total cost is just $\mathcal{C}(\mathcal{A}) = k_1 c(r)$.

Equation (6) provides a simple close-form expression of the approximation cost, but it relies on the assumption that the request space is continuous. To assess the extent of the approximation, we compare it to the cost achieved in the case of a discrete request space, where requests (and catalog objects) are constrained to lie on the points of a $L \times L$ square with unitary step and wrap-around conditions.

For some special values of L, namely $L = 1 + 2l(l+1)$, where $l \in \mathcal{N}$, there exists a regular tessellation of the grid with L squares, each comprising L points. Figure 3 provides an example of such regular tessellation in the case $l = 2$, $L = 13$. When $k_1 = L$, the discrete versions of Proposition 5.1 allows us to conclude that storing in the cache the central object of each square is

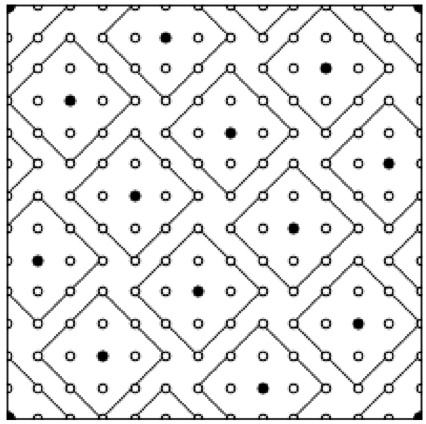

Figure 3: Example of perfect tessellation of a square grid with wrap-around conditions, in the case $l = 2$, $L = 13$. Black dots correspond to a minimum cost cache configuration under homogeneous request process.

optimal, achieving the per-request approximation cost:

$$c_{grid}(r) = \sum_{i=1}^{l} \frac{4i^{\gamma+1}}{L} \tag{7}$$

which can be understood by noticing that there are $4i$ points at hop distance i from the central object. The optimal cost as described by Equation (7) for these special discrete cases can be compared to the continuous approximation (6), where we need to set $r = \sqrt{L/2}$, $\lambda = 1/L$.

Figure 4 shows the result of this comparison as function of $k_1 = L$, for different values of γ. We observe that the continuous approximation is very good provided that the number of objects falling in each square is not too small (say larger than a few tens).

If the request rate is not space-homogeneous, one can apply the results above over small regions $\mathcal{X}_i$ of $\mathcal{X}$ where λ_x can be approximated by a constant value $\lambda_{\mathcal{X}_i}$. Intuitively, the approximation becomes better and better the more λ_x varies smoothly over each Voronoi cell of region i. This in particular occurs when λ_x is smooth over the entire domain, and the cache size increases.

Under this approximation, let $k_{i,1}$ be the number of cache slots devoted

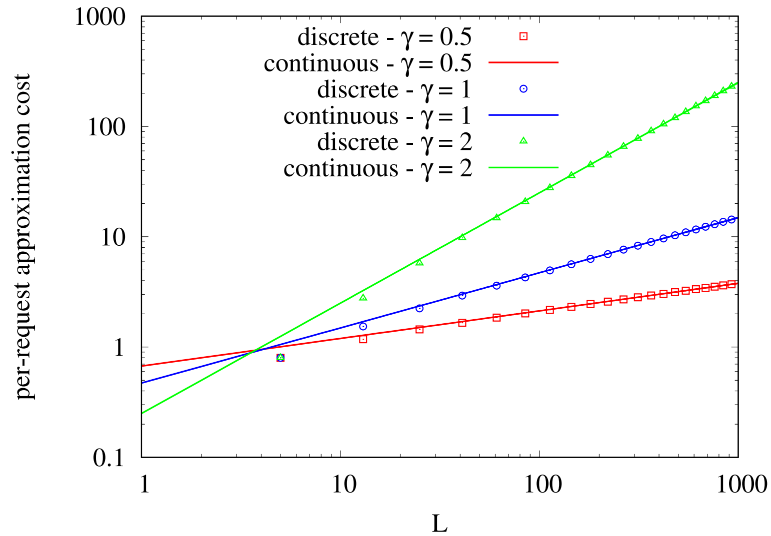

Figure 4: Per-request approximation cost as function of $k_1 = L$, for different values of γ, under uniform request process. Comparison between continuous request space (lines) and discrete request space (marks).

to region i (with the constraint that $\sum_i k_{i,1} = k_1$). Then, using standard constrained optimization methods, it is possible to determine the optimal value of $k_{i,1}$ as function of the local request rate $\lambda_{\mathcal{X}_i}$. Without loss of generality, we can assume that domain $\mathcal{X}$ is partitioned into M regions of unitary area, on which the request rate is approximately assumed to be constant and equal to λ_i, $1 \leq i \leq M$.

Then, focusing for simplicity on the two dimensional case when $d(x, y)$ is the norm-1, and $C_a(x, y) = d(x, y)^\gamma$, each cache slot is used to approximate requests falling in a square of area $1/k_{i,1}$ and radius $r_i = \sqrt{1/(2k_{i,1})}$. Following (6), the approximation cost c_i within a square belonging to region i can be easily computed as:

$$c_i(r_i) = 4\lambda_i \frac{r_i^{\gamma+2}}{\gamma + 2} = \zeta \lambda_i k_{i,1}^{-\frac{\gamma+2}{2}} \tag{8}$$

where $\zeta \triangleq 2^{(2-\gamma)/2}/(\gamma+2)$. Hence the total approximation cost in the whole domain, which depends on the vector $\boldsymbol{k}$ of cache slots $k_{i,1}$'s, is $\mathcal{C}(\boldsymbol{k}) = \sum_{i=1}^{M} k_{i,1} c_{i,1}(k_{i,1})$.

We select the values $\boldsymbol{k}$ that minimize the expected cost:

$$\begin{aligned} \underset{k_{1,1},\ldots,k_{M,1}}{\text{minimize}} \quad & \zeta \sum_{i=1}^{M} \lambda_i k_{i,1}^{-\gamma/2} \\ \text{subject to} \quad & \sum_{i=1}^{M} k_{i,1} = k_1 \end{aligned} \tag{9}$$

Employing the standard Lagrange method, one obtains that $\lambda_i k_{i,1}^{-(\gamma+2)/2}$ equals some unique constant for any region i, which means that $k_{i,1}$ has to be proportional to $\lambda_i^{2/(\gamma+2)}$. After some algebra we get:

$$\min \mathcal{C}(\boldsymbol{k}) = \zeta k_1^{-\gamma/2} \left(\sum_{i=1}^{M} \lambda_i^{\frac{2}{\gamma+2}} \right)^{\frac{\gamma+2}{2}}. \tag{10}$$

In the limit of large M, we substitute the sum in (10) with an integral, obtaining:

$$\min \mathcal{C}(\boldsymbol{k}) \approx \zeta k_1^{-\gamma/2} \left(\int_{\mathcal{X}} \lambda(x)^{\frac{2}{\gamma+2}} \, \mathrm{d}x \right)^{\frac{\gamma+2}{2}}. \tag{11}$$

We observe that, when the distance is the norm-1, this approach from [16] can be extended to higher dimensions computing integrals similar to (6).[2] Under other distances, things are not as simple, but in principle one can determine the best partitioning of the domain into k_1 Voronoi cells[3] V_i with center b_i, such that

$$\mathcal{C}(\mathcal{A}) = \sum_i \int_{V_i} C_a(x, b_i) \, \mathrm{d}x \tag{12}$$

is minimum, and store in the cache objects $\{b_i\}_i$. Similarly to [16], we prefer to avoid such geometric complications, and stick for simplicity to the norm-1 case.

5.2. Chain topology

Here we extend the approach recalled in previous section to a chain network of N caches, where requests arrive at the leaf cache 1, and are possibly forwarded

[2] In the d dimensional case we have $c(r) = a_d \lambda r^{\gamma+d}$, for an appropriate constant a_d.

[3] This task is not hard when the domain $\mathcal{X}$ can be exactly partitioned into k_1 Voronoi cells of the same shape. Otherwise, for sufficiently large cache sizes, one can neglect border effects and approximately consider k_1 Voronoi cells of the same shape covering the entire domain.

along the chain up to the node providing the best approximizer. In a chain the cost incurred by request r for object x, served by approximizer $\alpha = (o', j)$ is $C(r, \alpha) = C_a(x, o') + h(1, j)$. As request originates always at the leaf cache 1, we simplify the notation and denote $h(1, j)$ by h_j. We naturally assume $h_i > h_j$ if $i > j$. The N-th cache in the chain is the repository, where the approximation cost is negligible. In the following formulas, we recover this situation considering that the last cache has infinite cache size.

Let $k_{i,j}$ be the number of cache slots devoted by cache j to region i. Each of these slots is used to approximate requests falling in a square of area $1/k_{i,j}$ and radius $r_{i,j} = \sqrt{1/(2k_{i,j})}$. Hence the cost incurred by requests falling in a square of region i and served by cache j is:

$$c_{i,j}(r_{i,j}) = 4\int_0^{r_{i,j}}\int_0^{r_{i,j}-x} [(x+y)^\gamma + h_j]\lambda_i \,\mathrm{d}y\,\mathrm{d}x = 4\lambda_i \frac{r_{i,j}^{\gamma+2}}{\gamma+2} + 2\lambda_i r_{i,j}^2 h_j \quad (13)$$

The cost $C_{i,j}$ incurred by all requests falling in region i and served by cache j, as function of $k_{i,j}$, reads:

$$C_{i,j}(k_{i,j}) = \zeta\lambda_i k_{i,j}^{-\frac{\gamma}{2}} + \lambda_i h_j \quad (14)$$

In general a region i can be served by several caches along the path (every cache for which $k_{i,j} > 0$). However observe that a single request (i.e., a point of the region) will be always served by one specific cache, cache j^* with $j^* = \operatorname{argmin}_j C_{i,j}$ (ties can be neglected). We encode previous property by introducing weights $w_{i,j} \in [0, 1]$, where $w_{i,j}$ represents the fraction of region i served exclusively by cache j. Let $\boldsymbol{w}_j$ be the vector of $\{w_{i,j}\}_i$.

We obtain the optimization problem:

$$
\begin{aligned}
\underset{\boldsymbol{w}_2,\ldots,\boldsymbol{w}_N}{\text{minimize}} \quad & \zeta k_1^{-\gamma/2}\left(\sum_{i=1}^{M}\left(1-\sum_{j=2}^{N} w_{i,j}\right)\lambda_i^{\frac{2}{\gamma+2}}\right)^{\frac{\gamma+2}{2}} + \\
& \sum_{i=1}^{M}\left(1-\sum_{j=2}^{Z} w_{i,j}\right) w_i \lambda_i h_1 + \\
& \sum_{j=2}^{N}\left[\zeta k_j^{-\gamma/2}\left(\sum_{i=1}^{M} w_{i,j}\lambda_i^{\frac{2}{\gamma+2}}\right)^{\frac{\gamma+2}{2}} + \sum_{i=1}^{M} w_{i,j}\lambda_i h_j\right] \qquad (15) \\
\text{subject to} \quad & w_{i,j} \geq 0 \qquad \forall j > 1, \forall i \\
& \sum_{j=2}^{N} w_{i,j} \leq 1 \qquad \forall i
\end{aligned}
$$

where notice that we have separated the contribution of cache 1, and taken as decision variables vectors $\boldsymbol{w}_j$, with $j > 1$, since $\boldsymbol{w}_1 = \mathbf{1} - \sum_{j=2}^{N} \boldsymbol{w}_j$. Moreover, notice that the constraints in (15) are sufficient to guarantee that also the following obvious constraints hold:

$$
\begin{aligned}
w_{i,j} \leq 1 & \qquad \forall j > 1, \forall i \\
0 \leq w_{i,1} \leq 1 & \qquad \forall i
\end{aligned}
$$

In this form, (15) is a convex minimization problem over a convex domain, thus it has a global minimum. Without loss of generality, let the M regions be sorted in increasing values of λ_i. Employing the standard method of Lagrange multipliers, KKT conditions imply that the global optimum is attained when cache 1 handles all most popular regions region $i > i^*$ (i.e., $w_{i,1} = 1$, $i > i^*$), plus possibly a piece of region i^* (if $0 < w_{i^*,1} < 1$). Cache 1 does not allocate any slot to regions $i < i^*$.

Previous result allows us to prove the following interesting property about the structure of the optimal solution:

Proposition 5.2. *In the case of a chain topology, with requests arriving only at the first cache, the best solution of the continuous-domain, finite-M problem (15) is characterized by a set of popularity thresholds* $\lambda_0^* = \min\{\lambda_i\} \leq \lambda_1^* \leq$

$\lambda_2^* \leq \ldots \leq \lambda_{N-1}^* \leq \lambda_N = \max\{\lambda_i\}$, such that cache j approximates all requests falling in regions i with $\lambda_{j-1}^* < \lambda_i < \lambda_j^*$, plus possibly a portion of a region with $\lambda_i = \lambda_{j-1}^*$, and a portion of a region with $\lambda_i = \lambda_j^*$.

Proof. It is sufficient to apply the above property about the regions handled by cache 1, filtering out the requests handled by cache 1, and iteratively applying the same result to the request process forwarded upstream to caches $2, \ldots, N$. □

When the set of popularity values is not finite, it is possible to extend the result in Proposition 5.2, letting M diverge. We partition $\mathcal{X}$ into N sub-domains $\mathcal{X}_j$, $j = 1, \ldots, N$, stacked in vector $\boldsymbol{\mathcal{X}}$, such that cache j handles only requests falling into domain $\mathcal{X}_j$, and we seek to minimize:

$$\mathcal{C}(\boldsymbol{\mathcal{X}}) = \sum_{j=1}^{N} \left[\zeta k_j^{-\gamma/2} \left(\int_{\mathcal{X}_j} \lambda(x)^{\frac{2}{\gamma+2}} \, \mathrm{d}x \right)^{\frac{\gamma+2}{2}} + h_j \int_{\mathcal{X}_j} \lambda(x) \, \mathrm{d}x \right] \tag{16}$$

In principle we would like to find the best partitioning:

$$\boldsymbol{\mathcal{X}}^* = \arg\min_{\boldsymbol{\mathcal{X}}} \mathcal{C}(\boldsymbol{\mathcal{X}})$$

In this asymptotic case we can restate Proposition 5.2 as follows, providing a simpler and more elegant proof.

Proposition 5.3. *In the case of a chain topology with requests arriving only at the first cache, the best partition $\boldsymbol{\mathcal{X}}^*$ is characterized by the following property: for any $i < j$, $\inf_{\mathcal{X}_i^*} \lambda(x) \geq \sup_{\mathcal{X}_j^*} \lambda(x)$.*

Proof. By contradiction, let us assume that we find two non negligible areas $\Delta\mathcal{X}_i \subseteq \mathcal{X}_i^*$ and $\Delta\mathcal{X}_j \subseteq \mathcal{X}_j^*$ such that:

$$\sup_{\Delta\mathcal{X}_j} \lambda(x) > \inf_{\Delta\mathcal{X}_i} \lambda(x)$$

Then we can always find two non-negligible areas $\Delta\mathcal{X}_i' \subseteq \Delta\mathcal{X}_i$ and $\Delta\mathcal{X}_j' \subseteq \Delta\mathcal{X}_j$ such that we jointly have:

$$\int_{\Delta\mathcal{X}_i'} \lambda(x)^{\frac{2}{2+\gamma}} \, \mathrm{d}x = \int_{\Delta\mathcal{X}_j'} \lambda(x)^{\frac{2}{2+\gamma}} \, \mathrm{d}x \tag{17}$$

and

$$\inf_{\Delta\mathcal{X}'_j} \lambda(x) \geq \sup_{\Delta\mathcal{X}'_i} \lambda(x) > 0 \tag{18}$$

Now let us see what happens if we 'swap' $\Delta\mathcal{X}'_i$ with $\Delta\mathcal{X}'_j$, i.e., if we take a new partition $\boldsymbol{\mathcal{X}}'$ where $\mathcal{X}'_i = (\mathcal{X}^*_i \setminus \Delta\mathcal{X}'_i) \cup \Delta\mathcal{X}'_j$ and $\mathcal{X}'_j = (\mathcal{X}^*_j \setminus \Delta\mathcal{X}'_j) \cup \Delta\mathcal{X}'_i$. Note that by construction

$$\mathcal{C}(\boldsymbol{\mathcal{X}}') = \mathcal{C}(\boldsymbol{\mathcal{X}}^*) + (h_j - h_i) \int_{\Delta\mathcal{X}'_i} \lambda(x)\,\mathrm{d}x + (h_i - h_j) \int_{\Delta\mathcal{X}'_j} \lambda(x)\,\mathrm{d}x$$

Therefore, since $h_j > h_i$, we have $\mathcal{C}(\boldsymbol{\mathcal{X}}') \leq \mathcal{C}(\boldsymbol{\mathcal{X}}^*)$ if we can show that

$$\int_{\Delta\mathcal{X}'_j} \lambda(x)\,\mathrm{d}x \geq \int_{\Delta\mathcal{X}'_i} \lambda(x)\,\mathrm{d}x.$$

Denoted with $\beta = 2/(2+\gamma) < 1$ we have:

$$\begin{aligned}
&\int_{\Delta\mathcal{X}'_j} \lambda(x)\,\mathrm{d}x = \int_{\Delta\mathcal{X}'_j} \lambda(x)^\beta \lambda(x)^{1-\beta}\,\mathrm{d}x \\
&\geq (\inf_{\Delta\mathcal{X}'_j} \lambda(x))^{1-\beta} \int_{\Delta\mathcal{X}'_j} \lambda(x)^\beta\,\mathrm{d}x \\
&= (\inf_{\Delta\mathcal{X}'_j} \lambda(x))^{1-\beta} \int_{\Delta\mathcal{X}'_i} \lambda(x)^\beta\,\mathrm{d}x && \text{by (17)} \\
&\geq (\sup_{\Delta\mathcal{X}'_i} \lambda(x))^{1-\beta} \int_{\Delta\mathcal{X}'_i} \lambda(x)^\beta\,\mathrm{d}x && \text{by (18)} \\
&= \int_{\Delta\mathcal{X}'_i} (\sup_{\Delta\mathcal{X}'_i} \lambda(x))^{1-\beta} \lambda(x)^\beta\,\mathrm{d}x \\
&\geq \int_{\Delta\mathcal{X}'_i} \lambda(x)^{1-\beta} \lambda(x)^\beta\,\mathrm{d}x = \int_{\Delta\mathcal{X}'_i} \lambda(x)\,\mathrm{d}x
\end{aligned}$$

□

5.3. Extension to equi-depth trees

Previous results obtained for the chain topology can be easily extended to trees with L leaves at the same depth D, where requests arrive only at the leaves and all caches at the same level have the same size. Let h_{D-j} be the (equal) cost to reach the cache at level j starting from a leaf. We assume the spatial arrival rate at leaf ℓ to be given by $\lambda_\ell(x) = \beta_\ell \lambda(x)$, for some constant $\beta_\ell > 0$, i.e., spatial arrival rates at different caches are identical after rescaling by a constant factor. Moreover arrival processes at different leaves are assumed to be independent. We will call *equi-depth* tree a cache network with the above characteristics. We naturally assume $h_i > h_j$ if $i > j$.

Proposition 5.4. *In an equi-depth tree the optimal cost is achieved by replicating the same allocation at each cache of the same level. The allocation to be replicated is the one obtained in the special case of a chain topology ($L = 1$).*

Proof. Suppose to increase the number of nodes in the topology, creating a system of L parallel chain topologies. Each leaf now has an independent path towards a dedicated copy of the root node. By doing so the total cost in the system of parallel chains is surely not larger than the total cost achievable in the original tree, and, in general, it might be smaller (this because we can independently place objects in every chain so as to minimize the cost induced by the requests arriving at the corresponding leaf). On the other hand, the optimal allocation on each chain is the same, since the objective function in (15) is linear with respect to parameter β_ℓ. Therefore, by adopting such equal allocation on each cache of the same level in the original tree, we obtain exactly the same total cost achieved in the system of parallel chains, hence this allocation is optimal. □

One crucial assumption of chain topologies (and equi-depth trees) is that requests arrive only at the leaf (leaves). In the next section we discuss what happens when this assumption does not hold, considering the simplest possible case with just two caches.

5.4. A tandem network with arrivals at both nodes

In general cache networks that do not belong to the class of equi-depth trees, the simple optimal structure described in Proposition 5.2 is, unfortunately, lost. To see why, it is sufficient to consider the simple case of a tandem network with two identical caches (hereinafter called the leaf and the parent), where the same external arrival process $\lambda(x)$ of requests arrives at both nodes (see scenario 2 in Figure 7). Now, let us suppose that the cost h to reach the parent from the leaf is large (but it does not need to be disproportionally large). Then the leaf will not find particularly convenient to forward its requests to the parent, unless maybe for objects very close to the ones stored in the parent (whichever they are). On the other hand, the parent has to locally approximate all requests, hence it will need to adequately cover the entire domain $\mathcal{X}$ like an isolated cache. As a consequence, we do not expect any clear separation of $\mathcal{X}$ into a sub-domain handled by the leaf, and a sub-domain handled by the parent. In particular, the property that we had before, according to which a single cache has to allocate slots to cover a particular region of the domain, does not hold anymore.

A more formal explanation of what happens in this simple case can be provided by the following model. Again, we divide the domain, both at the leaf and at the parent cache, into M regions of unitary area. The request rate over each region is assumed to be constant and we denote it by λ_i and $\beta\lambda_i$ for the leaf and the parent cache, respectively (hence by setting $\beta = 0$ we can recover previous case in which requests arrive only at the leaf). Let $k_{i,1}$ and $k_{i,2}$ be the number of slots devoted to region i by the leaf and the parent node, respectively. Notice that now both quantities are in general different from zero. The leaf node will forward to the parent the requests falling in a fraction $(1 - w_{i,1})$ of region i, and it is natural to assume that these requests are those falling farther from the locally stored objects, i.e., at a distance larger than $r^*_{1,i} = \sqrt{w_{1,i}}r_{1,i}$, where $r_{1,i} = \sqrt{1/(2k_{i,1})}$. Therefore the approximation cost (14) is changed to:

$$C_{i,1}(k_{i,1}, w_{i,1}) = \zeta\lambda_i {w_{i,1}}^{\frac{\gamma+2}{2}} {k_{i,1}}^{-\frac{\gamma}{2}}. \tag{19}$$

Requests forwarded to the parent cache will experience an additional movement cost h, plus a local approximation cost at the parent, that we model by assuming that the total area of the subregion forwarded to the parent cache $k_{i,1}2r^2_{1,i}(1 - w_{1,i})$ will be served by the $k_{i,2}$ points at the parent, within squares of radius:

$$\sqrt{\frac{k_{i,1}r^2_{i,1}(1 - w_{i,1})}{k_{i,2}}} = \sqrt{\frac{1 - w_{i,1}}{2k_{i,2}}} \tag{20}$$

Moreover, at the parent cache the local requests will generate an approximation cost similar to (14) (with no retrieval cost).

The total approximation cost in the network is then:

$$\begin{aligned}\mathcal{C}(A) = \zeta \sum_{i=1}^{M} \lambda_i {w_{i,1}}^{\frac{\gamma+2}{2}} {k_{i,1}}^{-\frac{\gamma}{2}} \\ + \zeta \sum_{i=1}^{M} \lambda_i (\beta + (1 - w_{i,1})^{\frac{\gamma+2}{2}}) {k_{i,2}}^{-\frac{\gamma}{2}} + h \sum_{i=1}^{M} \lambda_i (1 - w_{i,1}).\end{aligned} \tag{21}$$

This cost should be minimized over $\{w_{i,1}\}_i$, $\{k_{i,1}\}_i$, and $\{k_{i,2}\}_i$. By finding the optimal values for $\{k_{i,1}\}_i$ and $\{k_{i,2}\}_i$ given $\{w_{i,1}\}_i$, we get

$$\begin{aligned}\mathcal{C}(\boldsymbol{w}) = \zeta k_1^{-\frac{\gamma}{2}} \left(\sum_{i=1}^{M} \lambda_i^{\frac{2}{2+\gamma}} w_{i,1} \right)^{\frac{2+\gamma}{2}} \\ + \zeta k_2^{-\frac{\gamma}{2}} \left(\sum_{i=1}^{M} \lambda_i^{\frac{2}{2+\gamma}} (\beta + (1 - w_{i,1})^{\frac{\gamma+2}{2}})^{\frac{2}{2+\gamma}} \right)^{\frac{2+\gamma}{2}} \\ + h \sum_{i=1}^{M} \lambda_i (1 - w_{i,1}).\end{aligned} \tag{22}$$

Note that for $\beta = 0$ we recover the cost resulting from (15) in the case of a tandem network. Computing the derivative of the above cost with respect to $w_{i,1}$ we get:

$$\begin{aligned}\frac{\partial \mathcal{C}(w)}{\partial w_{i,1}} = \zeta k_1^{-\frac{\gamma}{2}} \frac{\gamma+2}{2} \left(\sum_{i=1}^{M} \lambda_i^{\frac{2}{2+\gamma}} w_{1,i} \right)^{\frac{\gamma}{2}} \lambda_j^{\frac{2}{2+\gamma}} \\ - \zeta k_2^{-\frac{\gamma}{2}} \frac{\gamma+2}{2} \left(\sum_{i=1}^{M} \lambda_i^{\frac{2}{2+\gamma}} (\beta + (1 - w_{1,i})^{\frac{\gamma+2}{2}})^{\frac{2}{2+\gamma}} \right)^{\frac{\gamma}{2}} \\ \times \lambda_j^{\frac{2}{2+\gamma}} \frac{(1 - w_{1,j})^{\frac{\gamma}{2}}}{(\beta + (1 - w_{1,j})^{\frac{\gamma+2}{2}})^{\frac{\gamma}{2+\gamma}}} - h\lambda_j.\end{aligned} \tag{23}$$

Imposing the optimality conditions, we find that there may be multiple regions with different popularities λ_i for which $w_{1,i}* \in (0,1)$, i.e., for which the leaf forwards part of the requests to the parent. The structure of the solution in Proposition 5.2 might be lost, leading to optimal allocations where both caches handle portions of the same region.

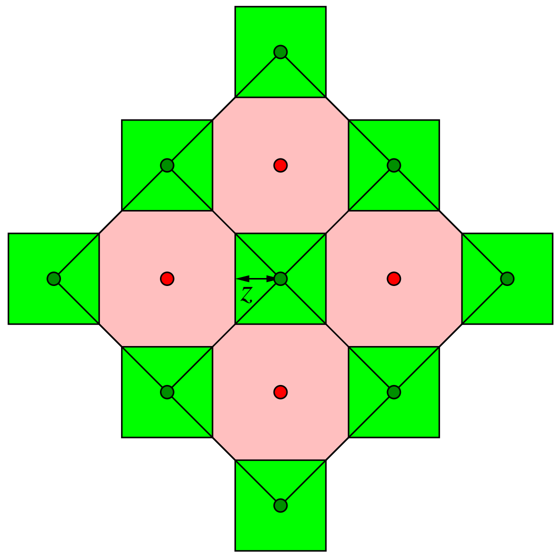

Figure 5: Optimal allocation in the tandem network with uniform arrival process at both nodes. Red areas denote the portion of the domain approximated by red centroids to be stored at the leaf cache. Green areas denote the portion of the domain approximated by green centroids to be stored at the parent cache.

To shed light into this phenomenon, we have further investigated the special case in which λ is uniform over the whole domain. In this case it is convenient to shift over space the two regular tessellations so that the centroids at the leaf and at the parent are as far as possible, as shown in Fig. 5. This allows the leaf to forward the requests farthest from its centroids to the parent, where they are better approximated.

Requests arriving at the leaf are approximated by the leaf in the red portion of the domain, as depicted in Fig. 5, while they are approximated by the parent in the green portion of the domain. Distance z (in Fig. 5) that defines the separation between the two portions can be easily computed (for $\gamma = 1$) as $z = \max\{0, (r-h)/2\}$, where r is the radius of the square of each tessellation (note that if $h > r$ requests are not forwarded from the leaf to the parent). Then one can easily compute the reduction $\Delta c = \frac{8}{3} z^3$ in the approximation cost for requests arriving at the leaf, provided by each slot of the second cache, and compute the resulting overall approximation cost (the approach can be generalized to $\gamma \neq 1$, but we omit the details here).

6. NetDuel: an online dynamic policy

Although in our work we have focused on the static, offline problem of content allocation at similarity caches, we have also devised an online, λ-unaware dynamic policy NetDuel, which is a networked version of policy Duel we have proposed in [16]. At high-level, it is based on the following idea: each (real) content currently is the cache is paired to a (virtual[4]) content competing with it. The cumulative saving in the total cost produced by the real and the virtual objects are observed over a suitable time window, and if the saving of the virtual object exceeds the saving of the real one by a sufficient amount, the virtual replaces the real in the cache. Otherwise, at the end of the observation window, the virtual object is discarded, and afterwards the real object will be paired to a new virtual object taken from the arrival process. In contrast to Greedy and LocalSwap, NetDuel does not require information about object arrival rates $\{\lambda_r\}_r$, and converges more slowly because it needs to estimate such rates from the arrival process itself. On the other hand, it can automatically adapt to dynamic object popularity.

NetDuel achieves an allocation close to the optimal one, suggesting that effective online dynamic policies can be devised for networks of similarity caches, at least under the assumption that each node knows when to forward requests upstream.

7. Numerical experiments

We now verify the theoretical findings in previous sections and the performance of proposed algorithms by running numerical and simulation experiments

[4]The cache stores only metadata of a virtual object, not the object itself. Virtual objects are taken from the arrival process.

on simple scenarios. We will consider both a synthetic arrival process (sec. 7.2) and a more realistic request stream based on an Amazon trace (sec. 7.3).

7.1. Methodology

We implemented the Greedy algorithm and the computation of the continuous approximation (15) using the C language. To run LocalSwap and NetDuel, instead, we developed an ad-hoc event driven simulator, based on the ns-2 network simulator engine [34], using the C++ language. The simulator can take as input either a synthetically generated stream of requests, or an actual trace. An input file provides all parameters and the description of a generic tree-like network topology. Note, however, that all results presented here have been obtained in a simple tandem network of two nodes. The output of the simulator is a file containing the objects stored in each cache of the network after a given number of requests have been processed. To find the best approximizer along the forwarding path, the simulator emulates a simple protocol sending a message all the way up to the repository and back, dynamically discovering the cache providing the best approximation cost. The simulator code is available from the authors upon request.

7.2. Synthetic arrival process

To test our algorithms, we consider 10000 objects falling on the points of a bi-dimensional $L \times L$ grid with $L = 100$, equipped with the norm-1 metric and the local cost $C_a(x, y) = d(x, y)$, i.e., we take (unless otherwise specified) $\gamma = 1$. The request process follows a Gaussian distribution, such that the request rate of object i is proportional to $\exp(-d_i^2/(2\sigma^2))$, where d_i is the hop distance from the grid center. To jointly test our continuous approximations, we assume that each grid point i is the center of a small square of area 1, on which λ is assumed to be constant and equal to λ_i.

We first consider a simple tandem network with arrivals only at the leaf, and fixed cost h to reach the parent (scenario 1 in Fig. 7). In Fig. 6 we compare the total cost produced by Greedy, LocalSwap, the continuous approximation

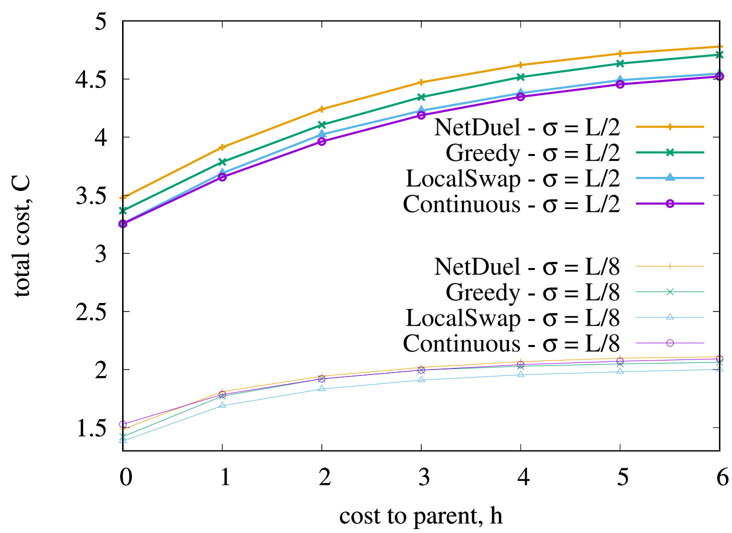

Figure 6: Total cost obtained by Greedy, LocalSwap, continuous approximation and NetDuel in a tandem network with arrivals at the leaf, for $\sigma = L/2$ (thick curves) or $\sigma = L/8$ (thin curves).

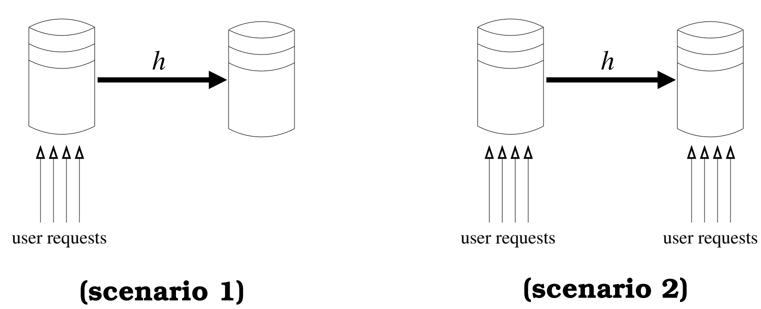

Figure 7: Simple tandem networks with two identical caches, with arrivals only at the leaf (scenario 1 on the left), or with arrivals at both caches (scenario 2 on the right). Requests forwarded from the leaf to the parent cache incur the additional fixed cost h.

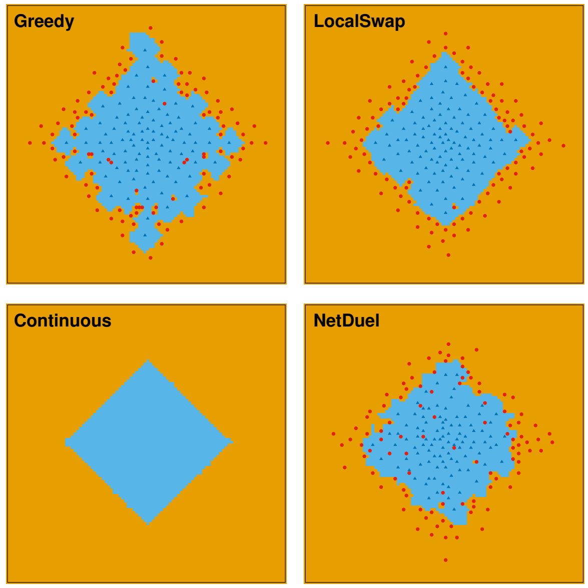

Figure 8: Allocations obtained by GREEDY, LOCALSWAP, continuous approximation and NETDUEL in the tandem network with $\sigma = L/8$, $h = 3$. Circle marks for the parent cache and triangle marks for the leaf cache.

(the solution of (15)) and NETDUEL, as function of h, for a larger Gaussian ($\sigma = L/2$) or a narrow Gaussian ($\sigma = L/8$). We observe that LOCALSWAP performs better than GREEDY, which performs better than NETDUEL. The continuous approximation does not necessarily provide a lower bound to discrete algorithms/policy, since it is a different system where the request space is continuous, rather than constrained on the grid points. However, we do observe that the continuous approximation curve gets closer to the curve produced by LOCALSWAP for $\sigma = L/2$ (thick curves), since in this case λ varies more smoothly over the domain.

In Fig. 8 we show the allocations (circles for the parent, triangles for the leaf) produced by the four approaches above in the case $\sigma = L/8$ and $h = 3$, using two different colors for the sub-domains where requests arriving at the

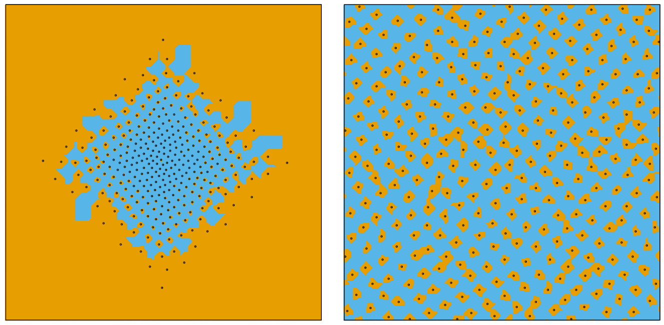

Figure 9: Parent allocation obtained by LocalSwap in a tandem network with arrivals at both nodes. Gaussian traffic (left plot) and Uniform traffic (right plot).

leaf are approximated by the leaf or the parent.[5] [6]

While our algorithms are completely oblivious to the threshold-based solution predicted by the continuous approximation, they achieve, qualitatively, the same cache allocation structure. Differences emerge at the boundary between the area served by the leaf and the area served by the parent, and are more evident for Greedy and NetDuel. Next, we see what happens when the crucial assumption underlying the above structure, namely, the fact that requests arrive only at the leaf, is removed.

In Fig. 9 we report, for a larger system with 100000 contents, the allocation produced at the parent by LocalSwap in a tandem network with requests arriving at both nodes (scenario 2 in Fig. 7), showing also with two different colors the regions where requests arriving at the leaf are approximated by the leaf or the parent. We consider both a Gaussian arrival process with $\sigma = L/8$ (left plot), and a simple Uniform process (right plot), and fixed $h = 3$. Notice that the parent cache covers also the central part of the domain, in contrast to Fig. 8. Results produced by LocalSwap suggest that now, for the requests

[5]For the continuous approximation, we do not show stored contents, and (border) squarelets are considered as handled exclusively by the parent if $w_{i,2} > w_{i,1}$.

[6]On an Intel i7 desktop computer equipped with 8GB of DDR4 RAM, the running time of Greedy, LocalSwap and NetDuel to produce the allocations in Fig. 8 were, respectively, 3 minutes, 5 minutes and 12 minutes.

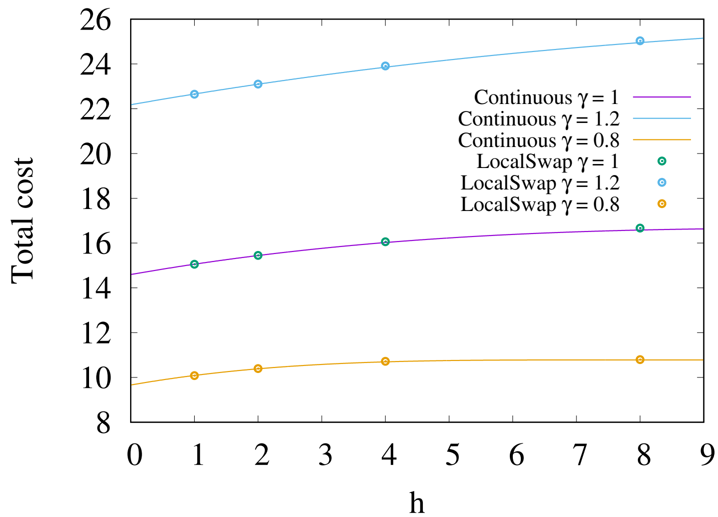

Figure 10: Total cost in the tandem network with arrivals at both nodes, λ uniform, as function of h, for different values of γ, according to LocalSwap (points) and continuous approximation (curves).

arriving at the leaf, the regions served directly by the leaf and the regions approximated by the parent are intertwined in a complex way. For uniform λ, Fig. 10 shows the accuracy of the continuous approximation based on the shifted regular square tessellations shown in Fig. 5.

7.3. Amazon trace

By crawling the Amazon web-store, the authors of [35] built an image-based dataset of users' preferences for millions of items. Using a neural network pre-trained on ImageNet, each item is embedded into a d-dimensional space, on which Euclidean distance is used as item similarity. We consider as request process the timestamped reviews left by users for the 10000 most popular items belonging to the baby category, with $d = 100$. The resulting trace, containing about 10.3M requests, is fed into a cache of size 100, with a parent cache of the same size (a tandem network) reachable by paying an additional fixed cost $h = 150$. The local approximation cost is set equal to the Euclidean distance.

In Fig. 11 we show the allocations produced by LocalSwap in both caches,

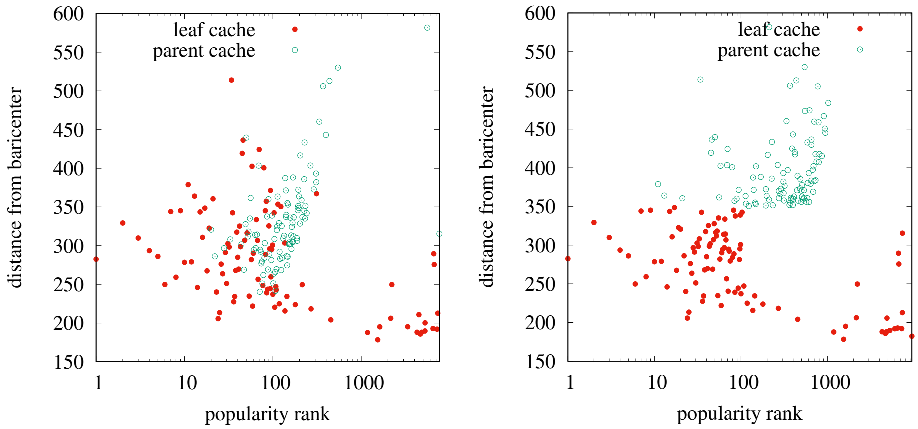

Figure 11: Allocations obtained by LocalSwap in a tandem network with arrivals at the leaf according to Amazon trace. Unconstrained version (left) and constrained version (right).

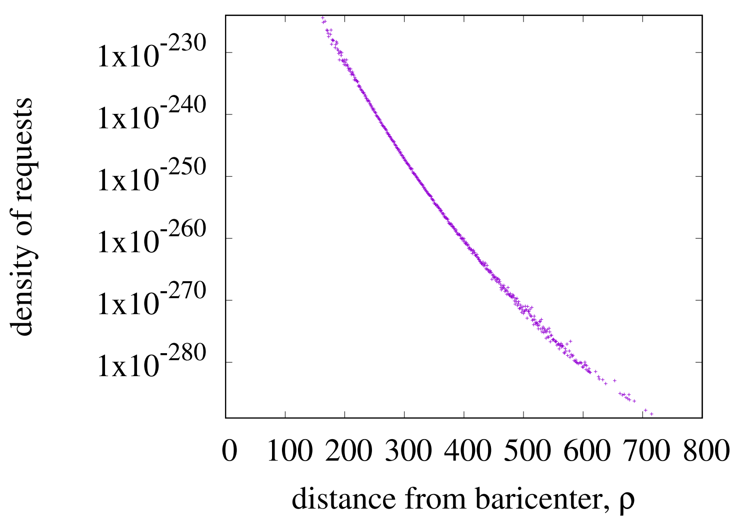

Figure 12: Density of requests of Amazon trace within spherical shells at distance $d \in [\rho_k, \rho_{k+1}]$ from the baricenter.

reporting, for each stored item, the popularity rank (x axes) and the distance from the baricenter (y axes). Across the entire catalog we found no correlation between popularity rank and distance from the baricenter. Nevertheless, we do observe that the leaf cache tends to store items that are either very popular or very close to the baricenter. The resulting total cost is $C = 266$ (left plot in Fig. 11).

Moreover, by computing the request density within spherical shells at distance $d \in [\rho, \rho + 1]$ from the baricenter, we found a decreasing trend in ρ, see Fig. 12, which justifies the attempt of 'enforcing' the structure of the optimal solution that we found in chain topologies fed only from the leaf. We do so by constraining the leaf (parent) cache to store only contents at distance from the baricenter smaller (larger) than a given threshold d^*. The constrained LOCAL-SWAP algorithm obtains, for the best possible $d^* = 350$, a total cost $C = 269$ (only 1% worse than before), right plot in Fig. 11, suggesting that a simple allocation and forwarding rule based on the distance from the baricenter is close to optimal also in a realistic scenario.

8. Conclusions and future work

In this paper we have made a first step into the analysis of networks of similarity caches, focusing on the offline problem of static content allocation. Despite the NP-hardness of the problem, effective greedy algorithms can be devised with guaranteed performance, but their implementation become prohibitive as the system size increases. For very large request space/catalog size, we have relaxed the problem to the continuous, obtaining for equi-depth tree topologies an easily implementable solution with a simple structure, which greatly simplifies the related request forwarding problem. The above simple structure is unfortunately lost in more general networks. We have also proposed a first online dynamic policy, NETDUEL, whose effectiveness is confirmed by preliminary simulation results. More experiments under both synthetic and real traces could be done to confirm the findings of our preliminary assessment of proposed algorithms and policies.

Future work will focus on: i) the design of practical, scalable algorithms which can deal with similarity caching network having general topology, and large catalog size; ii) the investigation of simple request forwarding strategies for the online setting, in the absence of complete information about which items are stored in upstream caches; iii) the extension of algorithms and policies to the case in which multiple (e.g., the m closest) approximating objects are needed, so as to offer several alternatives to the user; iv) additional numerical experiments under synthetic and real traces.

References

[1] A. Gionis, P. Indyk, R. Motwani, Similarity search in high dimensions via hashing, in: Proceedings of the 25th International Conference on Very Large Data Bases, VLDB '99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 518–529.
URL `https://dl.acm.org/doi/10.5555/645925.671516`

[2] F. Falchi, C. Lucchese, S. Orlando, R. Perego, F. Rabitti, A Metric Cache for Similarity Search, in: Proceedings of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval, LSDS-IR '08, ACM, New York, NY, USA, 2008, pp. 43–50. `doi:10.1145/1458469.1458473`.

[3] 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, WWW '09, ACM, New York, NY, USA, 2009, pp. 441–450. `doi:10.1145/1526709.1526769`.

[4] U. Drolia, K. Guo, J. Tan, R. Gandhi, P. Narasimhan, Cachier: Edge-caching for recognition applications, in: Proc. of the IEEE ICDCS, IEEE, 2017, pp. 276–286. `doi:10.1109/ICDCS.2017.94`.

[5] 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. doi:10.1145/3132211.3134456.

[6] P. Guo, B. Hu, R. Li, W. Hu, Foggycache: Cross-device approximate computation reuse, in: Proc. of the MobiCom, 2018, pp. 19–34. doi:10.1145/3241539.3241557.

[7] 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.
URL https://www.usenix.org/system/files/conference/hotedge18/hotedge18-papers-venugopal.pdf

[8] A. Araldo, F. Martignon, D. Rossi, Representation Selection Problem : Optimizing Video Delivery through Caching, in: IFIP Netw., 2016. doi:10.1109/IFIPNetworking.2016.7497212.

[9] M. Choi, J. Kim, J. Moon, Wireless Video Caching and Dynamic Streaming Under Differentiated Quality Requirements, IEEE JSAC 36 (2018).

[10] Z. Qu, B. Ye, B. Tang, S. Guo, S. Lu, W. Zhuang, Cooperative caching for multiple bitrate videos in small cell edges, IEEE Trans.Mob.Comp. 19 (2) (2020) 288–299. doi:10.1109/TMC.2019.2893917.

[11] 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 (6) (2018) 1300–1313. doi:10.1109/JSAC.2018.2844983.

[12] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, I. Stoica, Clipper: A Low-Latency Online Prediction Serving System, in: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), USENIX Association, Boston, MA, 2017, pp. 613–627.

URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[13] Y. Jin, L. Jiao, Z. Qian, S. Zhang, N. Chen, S. Lu, X. Wang, Provisioning Edge Inference as a Service via Online Learning, in: IEEE SECON, 2020. doi:10.1109/SECON48991.2020.9158425.

[14] T. S. Salem, G. Castellano, G. Neglia, F. Pianese, A. Araldo, Towards Inference Delivery Networks: Distributing Machine Learning with Optimality Guarantees, in: Proc. of 19th Mediterranean Communication and Computer Networking Conference (MedComNet 2021), 2021.

[15] Multi-access edge computing (mec), https://www.etsi.org/technologies/multi-access-edge-computing.

[16] M. Garetto, E. Leonardi, G. Neglia, Similarity caching: Theory and algorithms, in: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, IEEE Press, 2020, pp. 526–535. doi:10.1109/INFOCOM41043.2020.9155221.

[17] N. Choungmo Fofack, P. Nain, G. Neglia, D. Towsley, Performance evaluation of hierarchical ttl-based cache networks, Computer Networks 65 (2014) 212 – 231. doi:https://doi.org/10.1016/j.comnet.2014.03.006. URL http://www.sciencedirect.com/science/article/pii/S1389128614001108

[18] D. S. Berger, P. Gland, S. Singla, F. Ciucu, Exact analysis of ttl cache networks, Performance Evaluation 79 (2014) 2–23, special Issue: Performance 2014. doi:https://doi.org/10.1016/j.peva.2014.07.001. URL https://www.sciencedirect.com/science/article/pii/S0166531614000649

[19] S. Ioannidis, E. Yeh, Adaptive caching networks with optimality guarantees, IEEE/ACM Transactions on Networking 26 (2) (2018) 737–750. doi:10.1109/TNET.2018.2793581.

[20] M. Dehghan, L. Massoulie, D. Towsley, D. S. Menasche, Y. C. Tay, A utility optimization approach to network cache design, IEEE/ACM Trans. Netw. 27 (3) (2019) 1013–1027. doi:10.1109/TNET.2019.2913677. URL https://doi.org/10.1109/TNET.2019.2913677

[21] P. Guo, W. Hu, Potluck: Cross-application approximate deduplication for computation-intensive mobile applications, in: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, 2018, pp. 271–284.

[22] F. Chierichetti, R. Kumar, S. Vassilvitskii, Similarity Caching, in: Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '09, ACM, New York, NY, USA, 2009, pp. 127–136. doi:10.1145/1559795.1559815.

[23] 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), 2021.

[24] E. J. Rosensweig, D. S. Menasche, J. Kurose, On the steady-state of cache networks, in: 2013 Proceedings IEEE INFOCOM, 2013, pp. 863–871. doi:10.1109/INFCOM.2013.6566874.

[25] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, G. Caire, Femtocaching: Wireless content delivery through distributed caching helpers, IEEE Transactions on Information Theory 59 (12) (2013) 8402–8413. doi:10.1109/TIT.2013.2281606.

[26] E. Leonardi, G. Neglia, Implicit coordination of caches in small cell networks under unknown popularity profiles, IEEE Journal on Selected Areas in Communications 36 (6) (2018) 1276–1285. doi:10.1109/JSAC.2018.2844982.

[27] Y. Li, S. Ioannidis, Universally stable cache networks, in: IEEE INFOCOM

2020 - IEEE Conference on Computer Communications, 2020, pp. 546–555. doi:10.1109/INFOCOM41043.2020.9155416.

[28] M. Costantini, T. Spyropoulos, Impact of popular content relational structure on joint caching and recommendation policies, in: 2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT), 2020, pp. 1–8.

[29] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, G. Caire, Femtocaching: Wireless video content delivery through distributed caching helpers, in: Proceedings of IEEE INFOCOM, 2012, pp. 1107–1115. doi:10.1109/INFCOM.2012.6195469.

[30] J. Zhou, O. Simeone, X. Zhang, W. Wang, Adaptive offline and online similarity-based caching, IEEE Networking Letters 2 (4) (2020) 175–179. doi:10.1109/LNET.2020.3031961.

[31] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, G. Caire, Femtocaching: Wireless video content delivery through distributed caching helpers, in: 2012 Proceedings IEEE INFOCOM, 2012, pp. 1107–1115. doi:10.1109/INFCOM.2012.6195469.

[32] M. L. Fisher, G. L. Nemhauser, L. A. Wolsey, An analysis of approximations for maximizing submodular set functions—II, Springer Berlin Heidelberg, Berlin, Heidelberg, 1978, pp. 73–87. doi:10.1007/BFb0121195.

[33] G. Calinescu, C. Chekuri, M. Pál, J. Vondrák, Maximizing a monotone submodular function subject to a matroid constraint, SIAM Journal on Computing 40 (6) (2011) 1740–1766. doi:10.1137/080733991.

[34] The network simulator – ns–2, https://www.isi.edu/nsnam/ns/.

[35] 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, SIGIR '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 43–52. `doi:10.1145/2766462.2767755`.

[36] A. Krause, D. Golovin, Submodular Function Maximization, Cambridge University Press, 2014, pp. 71–104. `doi:10.1017/CBO9781139177801.004`.

Appendix A. Proof of Proposition 4.2

Proof. We first show that constraints are matroid ones. The empty set obviously belongs to $\mathcal{I}$, and if $\mathcal{A} \subset \mathcal{B}$ with $\mathcal{B} \in \mathcal{I}$, then $\mathcal{A} \in \mathcal{I}$. Finally, given two allocations with $|\mathcal{A}| < |\mathcal{B}|$, there exists a cache i that stores less elements under $\mathcal{A}$ than under $\mathcal{B}$, i.e., such that $\sum_{o':(o',i)\in\mathcal{A}} 1 < \sum_{o':(o',i)\in\mathcal{B}} 1$. Then, there exists an object o that is stored at i under $\mathcal{B}$, but not under $\mathcal{A}$. As $\sum_{o':(o',i)\in\mathcal{A}} 1 < \sum_{o':(o',i)\in\mathcal{B}} 1 \le k_i$, $\mathcal{A} \cup (o,i)$ is still a feasible allocation.

We now prove that $G(\mathcal{A})$ is a non-negative monotone submodular function.

$$
\begin{aligned}
G(\mathcal{A}) &= \sum_r \lambda_r C(r,\emptyset) - \sum_r \lambda_r C(r,\mathcal{A}) \\
&= \sum_r \lambda_r \left(C(r,\emptyset) - C(r,\mathcal{A})\right) \\
&= \sum_r \lambda_r \left(C(r,\emptyset) - \min_{\alpha\in S\cup\mathcal{A}} C(r,\alpha)\right) \\
&= \sum_r \lambda_r \left(C(r,\emptyset) - \min\left(\min_{\alpha\in\mathcal{A}} C(r,\alpha), C(r,\emptyset)\right)\right) \\
&= \sum_r \lambda_r \left(C(r,\emptyset) - \min_{\alpha\in\mathcal{A}} \min\left(C(r,\alpha), C(r,\emptyset)\right)\right) \\
&= \sum_r \max_{\alpha\in\mathcal{A}} \lambda_r \Big(C(r,\emptyset) - \min\left(C(r,\alpha), C(r,\emptyset)\right)\Big) \\
&= \sum_r \max_{\alpha\in\mathcal{A}} \lambda_r \Big(\max\left(C(r,\emptyset) - C(r,\alpha), 0\right)\Big)
\end{aligned}
$$

Then $G(\mathcal{A}) = \sum_r \max_{\alpha\in\mathcal{A}} M_{r,\alpha}$, where $M_{r,\alpha} \ge 0$ for all r and α. The set function is obviously monotone (i.e., if $\mathcal{A} \subset \mathcal{B}$, then $G(\mathcal{A}) \le G(\mathcal{B})$) and non-negative and corresponds to the utility of a facility location problem that is known to be submodular (e.g., [36], but it is also easy to check directly). $\square$

Appendix B. Pseudocode of Greedy algorithm

After an initial initialization corresponding to the empty allocation (lines 1–3), the algorithm performs K steps (line 4), where K is the total cache capacity in the network. In each step, we consider the addition of any possible approximizer at each cache, i.e., an approximizer not already present in a cache which has not yet been filled up (line 6), searching for the one that minimizes the total cost resulting from the addition of the considered approximizer (line 7). Then we add the found approximizer to the current allocation (lines 14–16), and proceed to the next step.

Algorithm 1 GREEDY

Require: cache sizes $k_i, \forall i \in \mathcal{K}$, arrival rates $\{\lambda_r\}_r, \forall r$, costs $C(r, \alpha), \forall (r, \alpha)$

1: $\mathcal{A} = \emptyset$ ▷ Initialize set of approximizers
2: $\mathcal{O}_i = \emptyset$, $\forall i \in \mathcal{K}$ ▷ Initialize set of allocated objects in each cache
3: $a_i = 0$, $\forall i \in \mathcal{K}$ ▷ Initialize number of allocated objects in each cache
4: **for** $i = 1 \dots K$ **do**
5:     $C = \infty$
6:     **for all** $\alpha = (o, i) : o \notin \mathcal{O}_i$, $a_i < k_i$ **do**
7:         $C^* = \sum_r \lambda_r C(r, \mathcal{A} \cup \{\alpha\})$ ▷ cost resulting from addition of α
8:         **if** $C^* < C$ **then**
9:             $C = C^*$
10:             $o^* = o$
11:             $i^* = i$
12:         **end if**
13:     **end for**
14:     $\mathcal{A} \Leftarrow \mathcal{A} \cup (o^*, i^*)$
15:     $\mathcal{O}_{i^*} \Leftarrow \mathcal{O}_{i^*} \cup o^*$
16:     $a_{i^*} \Leftarrow a_{i^*} + 1$
17: **end for**
18: **return** $\mathcal{A}$ ▷ Final set of approximizers

Appendix C. Pseudocode of LocalSwap algorithm

We start from a random allocation of objects at the caches (line 1), and perform at most `max_iter_noimprov` attempts to improve the current allocation, where `max_iter_noimprov` is a parameter of the algorithm, to be chosen sufficiently large to achieve convergence. In each attempt, we consider a real or emulated request for an object o (line 4), and evaluate what would happen if: i) we evict an object y from a cache i along the forwarding path of o; ii) insert o at cache i in place of y. By considering all possible substitutions as above (line 6), we compute the largest possible negative variation $\Delta\mathcal{C}^*$ in the total network cost (line 8–11) keeping track of the substitution that produces such variation (line 10). If we indeed obtain a negative variation (line 13) (note that $\Delta\mathcal{C}$ is initialized to zero on line 5), we actually perform the corresponding substitution (line 14).

Algorithm 2 LocalSwap

Require: parameter `max_iter_noimprov`, cache sizes $\{k_i\}_i$, arrival rates $\{\lambda_r\}_r, \forall r$, costs $C(r, \alpha), \forall(r, \alpha)$

Allocate k_i distinct random objects in each cache ▷ Initialize $\mathcal{A}$
`iter` $= 0$
while `iter` $<$ `max_iter_noimprov` **do**
  generate request for object o according to $\{\lambda_r\}_r$ ▷ real or emulated
  $\Delta\mathcal{C} = 0$
  for all $\alpha = (y, i) \in \mathcal{A}$ **do**
    $\Delta\mathcal{C}^* = \mathcal{C}(\mathcal{A} \cup \{(o, i)\} \setminus \{(y, i)\}) - \mathcal{C}(\mathcal{A})$
    if $\Delta\mathcal{C}^* < \Delta\mathcal{C}$ **then**
      $\Delta\mathcal{C} = \Delta\mathcal{C}^*$
      $(y_e, i_e) \Leftarrow (y, i)$
    end if
  end for
  if $\Delta\mathcal{C} < 0$ **then**
    $\mathcal{A} \Leftarrow \mathcal{A} \cup \{(o, i_e)\} \setminus \{(y_e, i_e)\}$
    `iter` $= 0$
  else
    `iter` $\Leftarrow$ `iter` $+ 1$
  end if
end while
return $\mathcal{A}$ ▷ Final set of approximizers