GRADES: Gradient Descent for Similarity Caching

Anirudh Sabnis*, Tareq Si Salem†, Giovanni Neglia†,
Michele Garetto‡, Emilio Leonardi§, and Ramesh K. Sitaraman*
*University of Massachusetts, Amherst,{asabnis,ramesh}@cs.umass.edu
†Inria, Université Côte d'Azur{tareq.si-salem, giovanni.neglia}@inria.fr
‡Università degli Studi di Torino, {garetto}@di.unito.it
§Politecnico di Torino, leonardi@polito.it

***Abstract*—A similarity cache can reply to a query for an object with similar objects stored locally. In some applications of similarity caches, queries and objects are naturally represented as points in a continuous space. Examples include 360° videos where user's head orientation—expressed in spherical coordinates—determines what part of the video needs to be retrieved, and recommendation systems where the objects are embedded in a finite-dimensional space with a distance metric to capture content dissimilarity. Existing similarity caching policies are simple modifications of classic policies like LRU, LFU, and qLRU and ignore the continuous nature of the space where objects are embedded. In this paper, we propose GRADES, a new similarity caching policy that uses gradient descent to navigate the continuous space and find the optimal objects to store in the cache. We provide theoretical convergence guarantees and show GRADES increases the similarity of the objects served by the cache in both applications mentioned above.**

I. INTRODUCTION

Similarity searching [1] is a key building block for a large variety of applications including multimedia retrieval [2], [3], recommender systems [4], [5], [6], genome study [7], [8], machine learning training [9], [10], [11], and serving [12], [13]. Given a query for an object, the goal is to retrieve one or more similar objects from a repository.

In the traditional setting, a cache is used to speed up object retrieval: once similarity search has identified the set of similar objects in the global catalog, the system checks if some of these objects are stored in the cache memory. In this setting, the cache performs a local *exact* lookup for the objects. Similarity search over the catalog can itself be a time-consuming operation, equivalent to linearly scanning the whole catalog [14]. Moreover, if users generating the queries are located far from the repository, they may experience long delays.

In order to solve these problems, the seminal papers [4], [3] proposed, almost at the same time, a different use of the cache: clients' requests are directly forwarded to the cache; then the cache performs a similarity search over the set of locally stored objects and possibly serves the requests without the need to forward the query to the (remote) repository. The cache can then reduce the overall serving time at the cost of providing objects *less similar* than those the repository would provide. They named this operation *similarity caching*, in contrast with the traditional *exact caching*. Later, the idea has been proposed under the name of *soft caching* [15] to take advantage of edge caches for content recommendation.

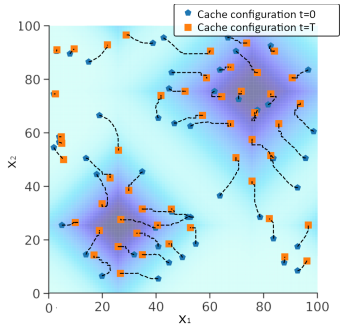


Fig. 1: Cached objects' movements in the representation space ($\mathbb{R}^2$) during $[0, T]$ when the cache is managed by GRADES. The catalog is made by the points in a 100×100 grid, dark shaded areas correspond to more popular objects. Dissimilarity cost $C_d(x, y) = 1/10\|x - y\|_1$, retrieval cost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

In many applications similarity is quantified using supervised machine learning techniques that collectively go under the name of distance metric learning [16]. These techniques learn how to map similar objects to vectors in $\mathbb{R}^d$ (called *embeddings*) that are close according to p-norm distances, cosine similarity, or Mahalanobis distances. Requests and objects live then in a continuous space. This is also the case for other potential applications of similarity caching like 360° videos, where requests for parts of the video are implicitly dictated by the user's head orientation expressed in spherical coordinates.

Existing dynamic policies for similarity caches adapt and extend well-known exact caching policies, like LRU and LFU, to deal with the notion of a *dissimilarity cost*, i.e., how distant is a cached object from the request. As a consequence, they treat requests and objects as discrete entities and ignore the continuous representation space.

This paper proposes GRADES, the first similarity caching policy designed to exploit object embedding in $\mathbb{R}^d$ with a distance that captures dissimilarity costs. While previous policies update the cache state replacing a cached object with (in general) a distant one—corresponding to "jumps" in the representation space—GRADES incrementally updates the (embedding of) each object using a gradient descent step to progressively reduce the dissimilarity cost.

Qualitatively, as shown in Fig. 1, the objects in the cache smoothly move in the representation space to find their optimal

position, i.e., where they can serve a large number of requests with small dissimilarity cost.

We prove that in a stationary setting, with an opportune choice of the gradient step sizes, GRADES converges to a cache configuration that corresponds to a critical point of the service cost (likely a local minimum). Our experiments based on realistic traces (made available online [17]) shows that GRADES outperforms existing similarity caching policies both for 360° videos and recommendation systems applications.

The paper is organized as follows: related work in Sect. II, formal problem definition in Sect. III, GRADES and its theoretical guarantees in Sect. IV, experimental results in Sect. V.

II. Related Work

Most existing policies for similarity caches generalize well-known exact caching policies, like LRU and LFU, to the new context, where besides (exact)-hits and misses, also *approximate-hits* are possible. For example SIM-LRU [4], [12] maintains the content in an ordered queue and serves an object from the cache if it is closer to the request than a given threshold (an approximate hit occurs). The object is then moved to the front of the queue. When no object in the cache is close enough to the request, there is a *miss*. The answer is then retrieved from the server and inserted at the front of the queue, possibly evicting items from the back. RND-LRU [4] is a variant of SIM-LRU where the threshold is replaced by a random variable that is a function of the dissimilarity cost. As SIM-LRU and RND-LRU are adaptions of LRU, qLRU-ΔC [18] modifies qLRU [19] introducing a refresh probability that depends on the similarity. Finally, DUEL [18] is inspired by LFU, and decides which object to evict by tracking the dissimilarity cost (i.e., the cost associated to the distance between a cached object and the request) accumulated over a given time-window. In our experiments we compare GRADES with SIM-LRU, qLRU-ΔC, and DUEL.

Our algorithm was inspired by the work from Jorge Cortés and his coauthors on coordination algorithms for mobile agents (see e.g. [20], [21], [22]). In their setting mobile agents (e.g., drones) place themselves in the space to be able to detect the largest number of events in the environment. Similarly, the objects in the cache need to position themselves to optimally serve the requests popping out over the space. Despite similarities at a high-level, their work focuses on a two-dimensional space and needs to take into account agents' movement and communication constraints that do not hold in our context.

From another point of view, GRADES gradient update can be considered as a generalization of stochastic K-means algorithms, where the function we want to minimize is not necessarily the squared Euclidean distance (as it is the case for K-means). Our proofs rely on techniques for non-convex optimization originally proposed in [23] also to study K-means.

Online caching policies based on gradient methods have been proposed in the stochastic request setting (see, e.g., [24], [25]), and, more recently, in the adversarial setting [26]. In these papers, the gradient step updates a vector of length equal to the catalog size, whose component i (in $[0, 1]$) represents which fraction of object i should be stored in the cache or equivalently the probability to store i in the cache. Differently from this line of work, GRADES uses the gradient step to modify the objects in the cache and maintains a vector of size equal to the cache—then much smaller than the catalog size.

A costly operation in any similarity search system is to find the closest object to the request. A simple solution is to index the collection, e.g., with a tree based data structure, to find the exact closest object. Unfortunately, when the number of dimensions d of the representation space exceeds 10, such an approach has a computational cost comparable to a full scan of the collection [14]. For this reason a number of approximate search techniques have been developed, which trade accuracy for speed and provide one or more points close to the request, but not necessarily the closest. Prominent examples are the solutions based on locality sensitive hashing [27], product quantization [28], [29], pivots [30], or graphs [31]. In the experiments in this paper, we have performed an exact similarity search, but any of these approximated search techniques could be used in GRADES.

III. Problem Definition

We consider a similarity search system where a server answers users' queries with the most similar object from a locally stored catalog. We assume that each request or object in the catalog can be represented as a point in the d-dimensional Euclidean space $\mathbb{R}^d$. In what follows we will refer to such representations as *embeddings* and, for the sake of simplicity, we will identify each object/request with its embedding (e.g., we will say that object x belongs to $\mathbb{R}^d$).

Requests satisfied by the server incur a *retrieval cost* C_r, which quantifies the delay the user experiences to retrieve the object from the remote server, and/or the additional load for the server, and/or the additional load for the network.

In alternative, the request may be satisfied by a *similarity cache* which stores a subset of the catalog. The cache provides, in general, a less similar object than what the server could provide, but incurs a negligible retrieval cost, as, for example, it is located closer to the user, or uses a faster memory storage or can perform faster lookup operations on the smaller set of stored contents. For simplicity we assume all objects have the same size and the cache can store up to k objects.

Our model of the system is similar to the one considered in previous papers on similarity caching in the continuous setting like [4], [18].

Let $\mathcal{Z}$ and $\mathcal{X}$ denote the catalog and the set of possible requests, respectively. Both sets may be finite or infinite, but we require them to be compact (to be able to retrieve a closest object to a given request). The "quality" of a similarity search for x depends on how similar the response object z is to the request. We assume the *dissimilarity cost* is quantified by the function $C_d(x, z) = h(\|x - z\|)$, where $h : \mathbb{R} \to \mathbb{R}^+$ is a non-decreasing non-negative function and $\|\cdot\|$ is a norm in $\mathbb{R}^d$ (e.g., the Euclidean one). For example Faiss (Facebook

AI Similarity Search) library [32] for multimedia retrieval supports all p-norms for $p \in [1, \infty]$.

The state of the cache at time t is given by the set of objects $\mathcal{S}_t$ currently stored in it, $\mathcal{S}_t = \{y_t^1, y_t^2, \dots, y_t^k\}$, with $y_t^i \in \mathcal{X} \subset \mathbb{R}^d$. Requests arrive first at the cache. Given a request for object x_t at time t, let i_t denote the index of the most similar object to the request (if there are many equally similar ones we arbitrarily select one), i.e., $i_t \in \arg\min_{i=1,\dots,k} C_d(x_t, y_t^{(i)})$. If the cache satisfies the request x_t, it will use content $y_t^{(i_t)}$, and the user will incur the dissimilarity cost $C_d(x_t, \mathcal{S}_t) \triangleq C_d(x_t, y_t^{(i_t)}) = \min_{y \in \mathcal{S}_t} C_d(x_t, y)$, but the retrieval cost is negligible. Alternatively, the cache can forward the request to the server, where it will be satisfied by the most similar object in the catalog. The request will generate the retrieval cost C_r, and the user will experience the dissimilarity cost $C_d(x_t, \mathcal{Z}) = \min_{z \in \mathcal{Z}} C_d(x_t, z) \leq C_d(x_t, \mathcal{S}_t)$.

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

In the impossibility to compare $C_d(x_t, \mathcal{S}_t)$ with the request-dependent value $C_d(x_t, \mathcal{Z}) + C_r$, the cache compare $C_d(x_t, \mathcal{S}_t)$ with a constant threshold value C_θ. If $C_d(x_t, \mathcal{S}_t) \leq C_\theta$, the cache serves the request locally, otherwise it forwards it to the server. We assume C_θ is set once and for all offline. Figure 2 illustrates how requests are served.

As $C_d(x_t, \mathcal{S}_t) = C_d(x_t, y_t^{(i_t)})$, the final cost to serve request x (denoted by $C(x_t, \mathcal{S}_t)$) depends only on $y_t^{(i_t)}$:

$$C(x_t, \mathcal{S}_t) = C(x_t, y_t^{(i_t)}) = \begin{cases} C_d(x_t, y_t^{(i_t)}), & \text{if } C_d(x_t, y_t^{(i_t)}) \leq C_\theta, \\ C_r + C_d(x_t, \mathcal{Z}), & \text{otherwise.} \end{cases} \quad (1)$$

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

In our theoretical analysis in Sect. III, we consider the case when requests arrive according to a Poisson process and are i.i.d. distributed. In the finite case ($|\mathcal{X}| < \infty$), we recover the classic independent reference model [33], where object x is requested with rate λ_x. In the continuous case, we need to consider a spatial density of requests and objects in a set $\mathcal{A} \subset \mathcal{X}$ are requested with rate $\int_{\mathcal{A}} \lambda_x \, \mathrm{d}x$.

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

$$\mathcal{C}(\mathcal{S}) \triangleq \begin{cases} \sum_x \lambda_x C(x, \mathcal{S}), & \text{finite case} \\ \int_{\mathcal{X}} \lambda_x C(x, \mathcal{S}) \, \mathrm{d}x, & \text{continuous case.} \end{cases} \quad (2)$$

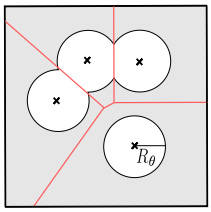


Fig. 2: Coverage of the request space ($\subset \mathbb{R}^2$) by 4 objects in the cache with norm-2 as dissimilarity cost. Crosses represent the objects. Each object is the closest point to requests in the corresponding Voronoi cell delimited by the red lines. Consider a request x falling in the Voronoi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

Finding an optimal set of objects $\mathcal{S}^*$ to store is NP-hard as it is a generalization of the problem considered in [18] (where $C_d(x, \mathcal{Z}) = 0$ for each $x \in \mathcal{X}$). Nevertheless, we propose a dynamic policy, that, under the stationary request process, can achieve a stationary point of $\mathcal{C}(\mathcal{S})$.

IV. A Gradient-based Algorithm

The key idea of our algorithm is to let the objects stored in the cache gradually "move" in the space $\mathbb{R}^d$ to reach a position where they can be used as approximate answers for a large number of requests (see Fig. 1). Upon a request at time t, the most similar object in the cache, $y_t^{(i_t)}$, is moved in the direction opposite to the gradient of the service cost ($\nabla_y C(x_t, y_t^{(i_t)})$) proportionally to a time-varying step-size (or learning rate) η_t:

$$y_{t+1}^{(i_t)} = y_t^{(i_t)} - \eta_t \nabla_y C\left(x_t, y_t^{(i_t)}\right). \quad (3)$$

It is possible to prove that $C(x, y)$ is differentiable everywhere and then the gradients in (3) exist with probability 1 when the request process is continuous (the proof is omitted because of space contraints). When the request process is discrete, the probability that the gradient $\nabla_y C(x_t, y_t)$ does not exist may be non-zero, but we can then perturb the request by a small random vector $\epsilon \in \mathbb{R}^d$ and consider $\nabla_y C(x_t + \epsilon, y_t)$.

The attentive reader may frown upon the simple algorithm (3). First, it potentially updates the cache upon every request, even when $C_d(x_t, y_t^{(i)}) \leq C_\theta$ and the cache would not need to retrieve any object. Second, even if $y_t^{(i)}$ is the embedding of an object in the catalog, $y_{t+1}^{(i)}$ may not correspond to any object in the catalog.

In the following sections we address all issues mentioned above. After having refined the update rule (3) (Sect. IV-A), we prove that this idealized algorithm indeed converges to a critical point of $\mathcal{C}(\mathcal{S})$ (Sect. IV-B). Then, in Sect. IV-C we present a practical algorithm which 1) satisfies all our requirements, 2) keeps the state of the cache "close" to the state of the idealized algorithm, and 3) is more reactive and thus more suitable to non-stationary request processes.

A. Introducing a Projection

As requests are only for objects in the bounded set $\mathcal{X}$, there exists a norm-2 ball with radius R — $\mathcal{B}_2(R) = \{y \in \mathbb{R}^d, \|y\|_2 \leq R\}$) — such that $\mathcal{X} \subset \mathcal{B}_2(R)$ and $C(x, y) = C_r$ for each $y \notin \mathcal{B}_2(R)$ and each $x \in \mathcal{X}$. There is no advantage to store in the cache objects that do not belong to $\mathcal{B}_2(R)$ as they do not contribute to approximate any request. We then modify (3) in order to make closer to $\mathcal{B}_2(R)$ any cached object $y_t^{(i)}$ that the gradient update may have brought out of $\mathcal{B}_2(R)$. We write

$$y_{t+1}^{(i)} = y_t^{(i)} - \eta_t g_t^{(i)}, \tag{4}$$

$$g_t^{(i)} = \begin{cases} \nabla_y C(x_t, y_t^{(i)}), & \text{if } (i = i_t) \wedge (y_t^{(i)} \in \mathcal{B}_2(R)) \\ f(\|y_t^{(i)}\| - R)\frac{y_t^{(i)}}{\|y_t^{(i)}\|_2}, & \text{if } y_t^{(i)} \notin \mathcal{B}_2(R), \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

where $f(u) = \frac{\mathrm{d}}{\mathrm{d}u}\frac{u^4}{1+u^2} = 2u^3\frac{2+u^2}{(1+u^2)^2}$ (for technical reasons explained in the proof of Theorem IV.2).

B. Convergence

In this section we provide convergence results for the basic algorithm described by (4). We assume the cache update rule generates embeddings that always correspond to objects in the catalog. Moreover, we will ignore the cost of updates made *after* the request is served. These two simplifications will be removed in the next section.

It will be useful to denote the cache state as a vector $\mathbf{y}_t = (y_{t,1}, \ldots, y_{t,k}) \in \mathbb{R}^{k \times d}$, obtained concatenating the embeddings of the different objects in the cache. Similarly, we define the different costs as function of $\mathbf{y}_t$ and then write $C_d(\mathbf{y}_t)$, $C(\mathbf{y}_t)$, and $\mathcal{C}(\mathbf{y}_t)$.

We are now going to prove that algorithm (4) converges almost surely to a stationary point of $\mathcal{C}(\mathbf{y})$.

The trajectory of $\mathbf{y}_t$ is bounded almost surely:

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

This lemma, whose proof is sketched in App. A, is used in the proof of the following convergence result.

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

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

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

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

The sequence $(\mathbf{y}_t)$ converges then to a critical point of $\mathcal{C}(\cdot)$, i.e., a point where the gradient is zero. This may be a saddle point, a local maximum or a local minimum of $\mathcal{C}(\cdot)$. The latter is more likely as it is the only one locally stable. The proof of Theorem IV.2 is in App. B.

Algorithm 1 GRADES

Let k be the cache size and x the object requested
if $(|\mathcal{S}_{V,t}| < k) \wedge (x \notin \mathcal{S}_{V,t})$ **then** ▷ still space in cache
    Insert x in VC
    Retrieve and insert $\rho(x)$ in PC
    $\mu(x) = \rho(x)$
$y_V = \arg\min_{y \in \mathcal{S}_{V,t}} C_d(x, y)$
Update $S_{V,t}$ according to (4)
if $C_d(x, y_V) \leq C_\theta$ **then** ▷ virtual hit
    if $\|x - y_V\| < \|\mu(y_V) - y_V\|$ **then** ▷ x approximates y_V better than $\mu(y_V)$
        Evict $\mu(y_V)$
        Retrieve and Insert $\rho(x)$ in PC
        $\mu(y_V) = \rho(x)$
    GRAFT_HIT_UPDATE$(x, S_{V,t})$
$y_P = \arg\min_{y \in \mathcal{S}_{P,t}} C_d(x, y)$
$\xi \sim$ Uniform$(0, 1)$
if $\xi < p$ **then**
    $(\text{update}, \omega) =$GRAFT_MISS_UPDATE$(S_{V,t}, x, \rho(x))$
    if update **then**
        Evict ω and $\mu(\omega)$
        Retrieve and Insert $\rho(x)$ in VC and PC
        $\mu(\rho(x)) = \rho(x)$
        $y_P = \rho(x)$
if $(C_d(x, y_P) \leq C_\theta) \vee (\rho(x)$ inserted in PC$)$ **then**
    Serve y_P
else
    Retrieve and Serve $\rho(x)$

C. Implementation

In this section we present our complete caching policy GRADES, whose pseudo-code is in Algorithm 1. Theorem IV.2 shows that the basic gradient update (4) attains a critical point of the expected cost $\mathcal{C}(\cdot)$. Nevertheless, we have assumed that this update rule always generates embeddings in $\mathbb{R}^d$ that correspond to objects in the catalog. However, if the catalog has a finite number of objects, this is unlikely to happen, as the update (4) can potentially generate any real vector. Moreover, the update (4) may modify an object in the cache upon each request and then generate a high load on the server and the network to retrieve the new modified objects.

In Sect. IV-C1 we describe how our algorithm addresses these issues. We then move on in Sect. IV-C2 to describe some additional features that provide a higher adaptivity of the algorithm to deal with highly non-stationary request processes, allowing for some random insertions with probability p.

1) Dealing with Finite Catalogue and Reducing Server Load: We propose to maintain a virtual cache (VC) and a physical cache (PC). The VC only stores some metadata, but no actual object; its use is common to other policies like 2-LRU [19] or ADAPTSIZE [34]. The VC is sometimes called shadow cache.

In our case the VC stores k vectors in $\mathbb{R}^d$ that are updated upon each request according to the basic algorithm in (4).

These vectors are the embeddings of the objects we would like to store in the cache, but, as discussed above, such objects may not exist, or they may not have been retrieved yet from the server. The PC contains objects from the catalog together with their embeddings.

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

We now describe in details Alg. 1 using the following additional notation:

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

The gray lines correspond to changes to increase algorithm adaptivity and will be discussed in Sect. IV-C2.

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

If the cache is already full, the closest object in VC i.e., y_V will be updated according to (4) (lines 6–7). Upon a virtual hit, if x is closer to y_V than the currently matching object $\mu(y_V)$ in the PC, GRADES takes advantage of this request to replace $\mu(y_V)$ with $\rho(x)$ (lines 9–12). In a stationary setting, the state of VC converges to a critical point of the cost (Theorem IV.2) and the PC should become closer and closer to it. Finally, the most similar object in PC is served if it is close enough to x, or if in any case $\rho(x)$ has been retrieved (line 11).

2) Increasing Adaptivity: According to what we described above, only the closest object in VC is updated upon a request (unless some projection back to $\mathcal{B}(R)$ is needed). A potential problem is that if an object x far from any other object has been accidentally inserted in VC (and the corresponding object $\rho(x)$ in PC), it may never be updated and may uselessly occupy cache space. Moreover, if at some point the request process changes abruptly, some objects in the cache that were initially useful may find themselves too far from the new requests. Again, the gradient algorithm, by itself, would not update such objects.

To overcome this problem, we can graft to GRADES a more dynamic caching policy that occasionally (with probability p) updates the VC, hopefully evicting the least useful object in the VC.

The "grafting" is described by the grey lines in Algorithm 1 and has been designed to support general cache eviction algorithms like LRU, LFU, and their variants. The grafted caching policy internally maintains its own data structure, e.g., an ordered queue for LRU. Upon an approximate hit, the hit update rule of the grafted policy is called (line 13). For example, LRU would move the requested object (if present in the cache) to the front of the queue. Also, with probability p, GRADES invokes the miss update rule of the grafted policy, that may lead to select an element ω to be evicted. GRADES then updates accordingly the VC and the PC (lines 19–22).

TABLE I: Traces description

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

V. EXPERIMENTS

In this section, we empirically validate our algorithm through simulations. First we demonstrate the benefit of the algorithm, by using synthetic traces. Next, to demonstrate real world applicability of our algorithm, we use GRADES in the domain of caching for 360 videos and recommendation systems. We assume that the catalog coincides with the set of possible requests ($\mathcal{Z} = \mathcal{X}$) and then set $C_\theta = C_r$. The retrieval cost C_r is always equal to 1. Table I summarizes the main characteristics of the traces. Further details about the experimental setup and the properties of request traces will be described in the corresponding subsections. To the best of our knowledge, there are no public traces for similarity caching; we made our traces available online [17].

We compare GRADES with the following algorithms.

a) GREEDY is an offline static algorithm that progressively fills the cache inserting the object that provides the largest cost saving given the set of objects already inserted. The algorithm provides a $\frac{1}{2}$ approximation in terms of cost savings [35].

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

c) SIM-LRU [4] maintains the content in an ordered queue as LRU. It moves objects to the front upon an approximate hit, and evicts objects from the back when needed.

d) q**LRU-**ΔC [18] is a variant of qLRU [19] that, upon an approximate hit, moves the object to the front with a probability which is proportional to the service cost reduction the object has guaranteed on the current request.

e) DUEL [18], upon a request for object x not in cache, x is matched with an object y in the cache in a tournament aimed at deciding if x is a better candidate to be stored in the cache as compared to y. The decision is made by comparing the cost savings x and y provide over a fixed interval of time (f). If the new object x provides a larger cost saving, then x replaces y in the cache.

A. Synthetic Traces

We consider a setting similar to [18]. The catalog is made by the points of a $L \times L$ bi-dimensional grid with $L = 313$. For any two objects x and y on the grid we define the approximation cost to be proportional to the norm-1 distance between the two points x and y, in particular $C_a(x, y) = \frac{1}{10}||x - y||_1$. The cache has size $k = 313$.

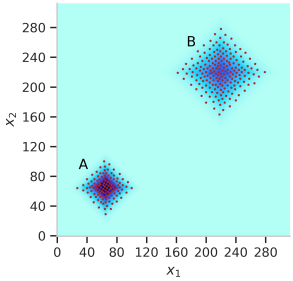


Fig. 3: The heatmap depicts the popularity distribution of objects in the grid. The darker regions A and B contain the most popular content. The circles represent the final configuration produced by the GRADES policy ($\eta = 0.64$) under the trace *Synthetic* in Table I.

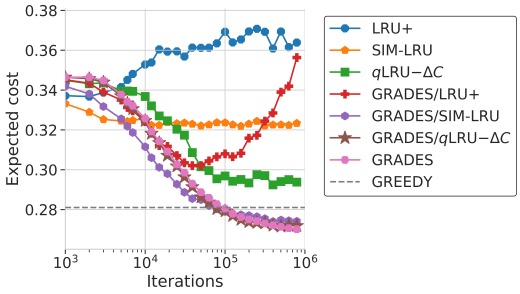


Fig. 4: Expected cost $\mathcal{C}(\cdot)$ incurred by different policies under the trace *Synthetic* in Table I: LRU+, SIM-LRU, qLRU-ΔC ($q = 10^{-2}$), GREEDY, and GRADES ($\eta = 0.64$, plain and grafted with $p = 10^{-1}$). Cache size $k = 313$.

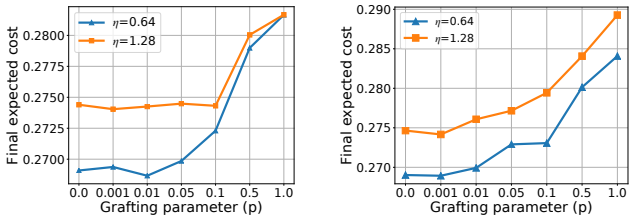
The traffic is generated under the *Independent Reference Model* [33]. There are two popular regions A and B centered around coordinates $(65, 65)$ and $(220, 220)$, respectively; they are produced by a mix of two Gaussian distributions. In particular, an object at (norm-1) distances d_1 from the center of A and d_2 from the center of B is requested with probability

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

The popularity distribution of the objects in the grid is depicted in the heat-map in Fig. 3. Figure 1 corresponds to a rescaled version of the same process.

Figure 4 shows the performance of GRADES without any graft and with different grafts (LRU+, SIM-LRU, qLRU-ΔC) for a quite large value of the grafting parameter ($p = 10^{-2}$). GRADES/X denotes GRADES grafted with policy X. We observe that GRADES achieves the smallest cost, however by grafting SIM-LRU we can make the initial transient faster. Figure 3 also shows the final cache configuration reached by GRADES: as expected, the density of the objects in the cache is higher where the request density is higher.

The effect of the grafting parameter p is shown in Fig. 5 and depends on the specific grafted policy. We see that



(a) qLRU-ΔC (b) SIM-LRU

Fig. 5: Effect of the grafting parameter p on the final expected cost $\mathcal{C}(\cdot)$. GRADES/qLRU-ΔC ($q = 10^{-2}$) and GRADES/SIM-LRU for different learning rates under the trace *Synthetic* in Table I. Cache size $k = 313$.

GRADES/qLRU-ΔC[1] is relatively insensitive to the grafting up to $p = 0.05$, but for larger values of p the cost increases, and approaches the cost of qLRU-ΔC alone (about 0.295 as it can bee seen in Fig. 4). This happens because, for large p, more and more cache updates are due to qLRU-ΔC, which, even upon an approximate hit, may introduce the requested object with probability proportional to q. For GRADES/SIM-LRU, the cost again increases as p increases, but it is always much smaller than the cost of SIM-LRU alone (about 0.32). The explanation is that SIM-LRU never introduces new objects on approximate hits. Hence, as far as the current cache allocation provides approximate answers, the function GRAFT_MISS_UPDATE in Alg. 1 does not modify the current cache allocation.

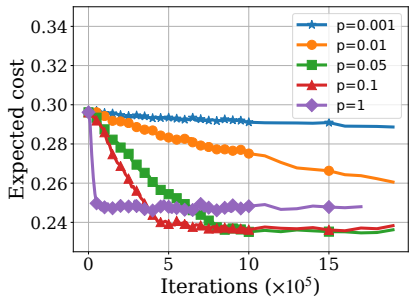


Fig. 6: Effect of grafting parameter p on the expected cost $\mathcal{C}(\cdot)$ in a dynamic setting for GRADES/qLRU-ΔC ($\eta = 0.64$). The cache is initialized to a stationary configuration as in Fig. 3. Now, only requests corresponding to region B from the synthetic trace are made. Cache size $k = 313$.

Until now, we have considered a stationary request scenario, where there is no evident advantage from grafting a more reactive policy to GRADES. In Fig. 6 we consider a highly non-stationary setting. At time 0, the cache is initialized as in Fig. 3, but then, suddenly, the request process changes abruptly and no more requests for objects in region A are generated. The cache should reach a new configuration where all cached objects are located in region B, achieving a lower cost, as

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

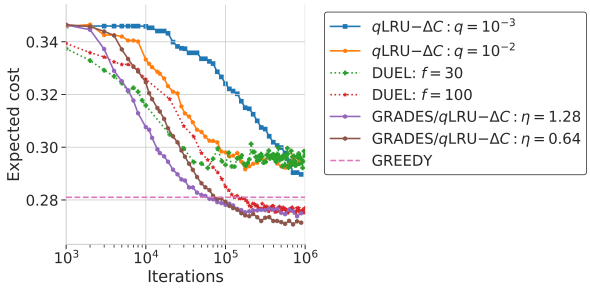


Fig. 7: Expected cost $\mathcal{C}(\cdot)$ incurred by the different policies under the trace *Synthetic* in table I: qLRU-ΔC ($q = 10^{-3}, 10^{-2}$), DUEL , GREEDY, and GRADES ($\eta = 1.28, 0.64$ grafted with $p = 1$ and $q = 10^{-3}$). Cache size $k = 313$.

now the same number of objects should cover a smaller area. Figure 6 shows that a higher value of p enables faster migration of objects from region A to region B in the cache.

Figure 7 shows the synergy between GRADES and the grafted policy. qLRU-ΔC, DUEL, and GRADES, all have parameters (q, η, and f) that can be tuned to find an optimal trade off between convergence speed and final cost: they can converge fast to configurations within a large neighborhood of a critical point (for high q, high η, and low f, respectively), or slowly to configurations within a smaller neighborhood. Experiments in [18], in a setting similar to ours, show that qLRU-ΔC achieves a worse cost-vs-speed tradeoff than DUEL. Figure 7 confirms that this is the case, but when qLRU-ΔC is grafted on GRADES, the resulting policy improves on top of DUEL. In fact GRADES/qLRU-ΔC achieves a better trade-off as it is able to converge to a cost comparable to DUEL in a shorter time, or equivalently to a smaller cost in roughly the same time. Note also that DUEL's expected cost at steady state is noisier than GRADES/qLRU-ΔC's cost, showing the advantage of smoothly updating the state using gradients.

B. 360° *Videos*

We test our algorithms on 360° video traces. A 360° video is an immersive, spherical video [36], [37]. The video is first projected on to a 2D plane to be encoded by classic 2D video encoders. The video is divided into time segments, and each segment is further spatially divided into tiles. The VR headset is optimized to fetch the required tiles based on the head position of the user. A system responsible for the delivery of 360° videos can cache the popular tiles in nearby caches [38]. Moreover, tiles at the periphery of the user's field of view could be approximated by neighbouring tiles that are stored at the cache and can then be served with low latency. Similarity caching may then be useful in this context, specially in the future when the number of tiles will increase and close tiles will become more similar.

We generated a sequence of tiles' requests for 360° videos using the approach proposed in [39]. We took real traces from 8 videos watched by 48 users each, and then built a *navigation graph* for each video, i.e., a Markov Chain that represents the spatial and temporal viewing correlations for the video. The videos we considered have on average 207 segments, each with 25 tiles. From each navigation graph we can generate an arbitrary number of possible views of the video. We generated then a trace with 10000 users as follows. At time $t = 0$, each user selects one of the videos at random and starts watching the video from a random segment of the video. The user then walks through the navigation graph to view the complete video. Once the user reaches the last segment in the video, it selects a new video uniformly at random (with replacement) and starts watching the selected video from the first segment. The process is repeated till 10 million requests are generated.

We assume each tile can approximate at most 4 tiles (the adjacent ones), with a fixed approximation cost $C_a = 0.1$.

Figure 8 compares the performance of GRADES/qLRU-ΔC and qLRU-ΔC. Note that in this setting, the representation space exhibits a very rough granularity, as the tiles of a segment cannot be used to approximate those of another segment and each segment is decomposed in a 5×5 grid of tiles. Nevertheless, GRADES/qLRU-ΔC shows significant improvement with respect to existing similarity caching policies and approaches the cost of GREEDY.

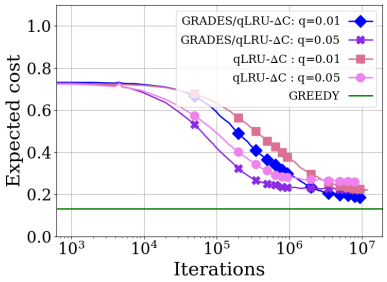


Fig. 8: Expected cost $\mathcal{C}(\cdot)$ incurred by the different caching policies for the trace 360° *videos* in Table I. GRADES/qLRU-ΔC ($q = 0.01, 0.05$) with $\eta = 1.0$ and qLRU-ΔC ($q = 0.01, 0.05$). Cache size $k = 4000$.

C. *Machine Learning Traces*

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

a) Amazon trace: The paper [40] proposes a technique to embed the images of Amazon products in a 100-dimensional space, where the Euclidean distance between two items captures the similarity of the sets of users who purchased or viewed both items. We have restricted ourselves to the products in the category "Baby" and we have assumed that a request for a given item was issued at time t, if a user left a review for the considered item at the same time.

b) CiteULike trace: The CiteULike dataset [41] contains a bipartite network of 22,715 users and 153,277 tags, where each edge represents a timestamped tag creation. The embeddings in a 100-dimensional space are obtained using the collaborative metric learning model proposed in [42]. As for the Amazon trace, the Euclidean distance within this space encodes the similarity between users and items, where the items here are the tags. We generated the trace considering

that an object (tag) is requested when one user adds the corresponding tag.

c) Movielens trace: We have trained the RecVAE collaborative filtering model from [43] on the Movielens dataset [44] to embed users' rating histories in a $d = 200$ dimensional space. Users with similar rating histories are mapped to vectors close according to the Euclidean distance. We have generated the trace by embedding every batch of 38 ratings from the same user (38 is the median number of ratings across all users) and assigning it the timestamp of the latest rating in the batch.

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

The time-average cost of different caching policies is shown in Fig. 9 for the Amazon trace and 10% approximability. Although an object is able to approximate only 10% of the catalog on average, similarity caching policies significantly reduce the cost in comparison to an exact caching policy like LRU, with qLRU-ΔC and GRADES/qLRU-ΔC achieving the lowest service cost. The empirical distribution of pairwise distances in Amazon trace (due to space constraints we cannot show it) shows that objects are quite scattered in this high-dimensional space. The objects in the virtual cache are then in general far from any object in the catalog and we could expect gradient methods to perform poorly. Nevertheless, Fig. 9 shows that GRADES/qLRU-ΔC outperforms existing similarity caching policies. Similar results hold for the other two traces.

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

VI. Conclusions

In this paper we have proposed GRADES, a new caching policy for similarity caching systems that takes advantage from the fact that objects and requests are typically embedded in a continuous metric space. GRADES outperforms traditional caching policies in stationary scenarios, converging to provably optimal configurations under mild assumptions.

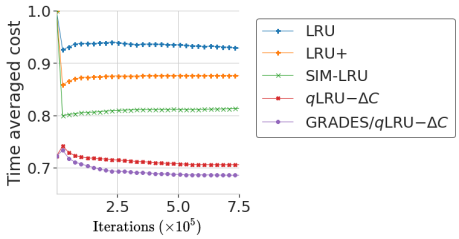
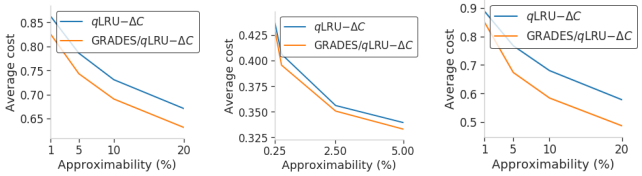


Fig. 9: The time averaged cost incurred by different policies under the trace *Amazon trace*: LRU, LRU+, SIM-LRU, qLRU-ΔC ($q = 10^{-3}$) and GRADES/qLRU-ΔC ($\eta = 6.9 \times 10^2$, grafted with p = 1). The level of approximability is 10%. Cache size $k = 100$.



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

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

Moreover we have shown that GRADES can be grafted to any traditional caching policy, obtaining flexible schemes that achieve arbitrary trade-offs between convergence speed and average costs at steady state. The performance of GRADES and its extensions has been evaluated in several synthetic and realistic scenarios.

VII. Acknowledgement

This work was supported in part by the National Science Foundation under Grant No. CNS-1763617 and No. CNS-1901137 and by Inria under the exploratory action MAMMALS.

Appendix A
Sketch of Lemma IV.1 Proof

Proof. We observe that for $y_t(i) \in \mathcal{B}_2(R)$, $\|g_t^{(i)}\|_2$ is bounded and for $y_t(i) \notin \mathcal{B}_2(R)$, $\|g_t^{(i)}\|_2 \leq 2 + 2\|y_t^{(i)}\|_2$, then there exists positive constants A_l and B_l for $l = 1, 2, 3, 4$, such that for $\|g_t^{(i)}(x, y_t^{(i)})\|_2^l \leq A_l + B_l\|y_t^{(i)}\|_2^l$.

We also observe that $\inf\{y_t^{(i)} \cdot g_t^{(i)}, \forall y_t^{(i)} \notin \mathcal{B}_2(R+1)\} > 0$ for all $i = 1, \ldots, k$.

We will prove boundedness using the function $f_t = \sum_{i=1}^k \phi(\|y_t^{(i)}\|_2^2)$, where $\phi(u) = \mathbb{1}_{u \geq (R+1)^2}(u - (R+1)^2)^2$. There exist positive constants A and B such that $\left(A_0 + B_0\|y_t^{(i)}\|_2^4\right) \leq A/k + B\phi(\|y_t^{(i)}\|_2^2)$.

Taking advantage of the inequality $\phi(v) - \phi(u) \leq (v - u)\phi'(u) + (v - u)^2$ applied for $u = \|y_t^{(i)}\|_2^2$ and $v = \|y_{t+1}^{(i)}\|_2^2$,

and of the inequalities above, we can derive the following inequality

$$f_{t+1} - f_t \leq \sum_{i=1}^{k} \left[-2\eta_t y_t^{(i)} \cdot g_t^{(i)} \phi'(\|y\|_2^2) \right] + \eta_t^2 \left(A + B f_t \right). \tag{5}$$

We take now the conditional expectation given $H_t = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t)$:

$$\mathbb{E}_x[f_{t+1} - f_t | H_t] \leq \sum_{i=1}^{k} \left[-2\eta_t y_t^{(i)} \cdot \mathbb{E}_x \left[g_t^{(i)} \right] \phi'(\|y_t^{(i)}\|_2^2) \right] + \eta_t^2 \left(A + B f_t \right) \leq \eta_t^2 (A + B f_t)., \tag{6}$$

where the last inequality follows from $\phi'(\|y_t^{(i)}\|_2^2) = 0$ for $y_t^{(i)} \in \mathcal{B}_2(R+1)$ and $y_t^{(i)} \cdot g_t^{(i)} > 0$ for $y_t^{(i)} \in \mathcal{B}_2(R+1)$.

From (6) we can reason as in [23, Sect. 5.2] to show that $\{f_t\}_{t \in \mathbb{N}}$ is a quasi-martingale and apply convergence results for quasi-martingales [23, Sect. 4.4] to conclude that f_t converges almost surely to a random variable $f_\infty \geq 0$ with $\mathbb{E}[f_\infty] < \infty$. We want to conclude that $f_\infty = 0$ with probability one, and then $\mathbf{y}_t$ is bounded almost surely.

Moving around the terms in (5) and summing over t, we obtain

$$\sum_{t=1}^{\infty} \sum_{i=1}^{k} 2\eta_t y_t^{(i)} \cdot g_t^{(i)} \phi'(\|y\|_2^2) \leq f_1 - f_\infty + \sum_{t=1}^{\infty} \eta_t^2 \left(A + B f_t \right) < +\infty \text{ a.s.} \tag{7}$$

because $\sum_t \eta_t^2 < \infty$ and f_t converges to $f_\infty < \infty$.

Let $\mathcal{H}$ be the set of sequences such that $f_\infty > 0$. For each sequence in $\mathcal{H}$, $f_t > f_\infty/2$ for large t, and then there exists at least a value i_t and an opportune $\epsilon > 0$ such that $\|y_t^{(i_t)}\|_2 > (R+1) + \epsilon$. Then both $y_t^{(i)} \cdot g_t^{(i)}$ and $\phi'(\|y\|_2^2)$ are bounded below by positive quantities and the series in the LHS of (7) diverges as $\sum_t \eta_t = +\infty$. But we have concluded that this series converges a.s., then $\Pr(\mathcal{H}) = 0$ and $f_\infty = 0$ a.s.. Each sequence $(\mathbf{y}_1, \mathbf{y}_2, \ldots)$ is then bounded a.s. □

APPENDIX B
SKETCH OF THEOREM IV.2 PROOF

Proof. We denote by $y_{t,i}$ the i-th component of the vector $\mathbf{y}_t \in \mathbb{R}^{k \times d}$, and by y_i the i-th component of a generic vector $\mathbf{y} \in \mathbb{R}^{k \times d}$.

We define $F(u) = u^4/(1+u^2)$, $L(\mathbf{y}) = \sum_{i=1}^{k} \mathbb{1}_{y_t^i \notin \mathcal{B}_2(R)} F\left(\|y_t^{(i)}\|_2 - R \right)$, and $\tilde{C}(x, \mathbf{y}) = C(x, \mathbf{y}) + L(\mathbf{y})$. We observe that $\nabla_{\mathbf{y}} \tilde{C}(x, \mathbf{y}_t) = (g_t^{(1)}, \ldots, g_t^{(k)})$, i.e., the dynamic in (4) is evolving according to the gradient of $\tilde{C}(x, \mathbf{y}_t)$. Similary we define $\tilde{\mathcal{C}}(\mathbf{y}) = \mathbb{E}_x\left[\tilde{C}(x, \mathbf{y})\right] = \mathcal{C}(\mathbf{y}) + L(\mathbf{y})$. The function $L(\mathbf{y})$ is continuously differentiable up to the third order, then $\tilde{\mathcal{C}}(\mathbf{y})$ is continuously differentiable up to the second/third order when $\mathcal{C}(\mathbf{y})$ is so. Moreover, we observe that $\|\nabla_{\mathbf{y}} \tilde{\mathcal{C}}(\mathbf{y})\|_2 > 0$ for $y^{(i)} \notin \mathcal{B}_2(R)$ for some i. Then, $\liminf_{t \to \infty} \|\nabla_{\mathbf{y}} \tilde{\mathcal{C}}(\mathbf{y}_t)\|_2 = 0$, implies that $\liminf_{t \to \infty} \|\nabla_{\mathbf{y}} \mathcal{C}(\mathbf{y}_t)\|_2 = 0$.

Lemma IV.1 shows that the sequence $H = (\mathbf{y}_1, \mathbf{y}_2, \ldots)$ is bounded a.s. Consider a bounded sequence H, such that $\mathbf{y}_t \in \mathcal{B}_2(R')$ for some $R' > 0$. $\nabla \tilde{C}(x, \mathbf{y}_t)$ exists a.s. and it is bounded for any $x \in \mathcal{X}$. By the dominated convergence theorem, it follows that we can invert the expectation and the gradient $\mathbb{E}_x\left[\nabla_{\mathbf{y}} \tilde{C}(x, \mathbf{y}_t)\right] = \nabla_{\mathbf{y}} \mathbb{E}_x\left[\tilde{C}(x, \mathbf{y}_t)\right] = \nabla_{\mathbf{y}} \tilde{\mathcal{C}}(\mathbf{y}_t)$.

As $\tilde{\mathcal{C}}()$ is continously differentiable up to the second order upon $\mathcal{B}_2(R')$, the partial derivatives are bounded, in particular, there exist two constants c_1 and c_2 such that $|\partial \tilde{\mathcal{C}}(\mathbf{y})/\partial y_i| \leq c_1$ and $|\partial^2 \tilde{\mathcal{C}}(\mathbf{y})/\partial y_i \partial y_j| \leq c_2$ for each $i, j \in \{1, 2, \ldots, kp\}$.

Using Taylor formula we can arrive to

$$\tilde{\mathcal{C}}(\mathbf{y}_{t+1}) - \tilde{\mathcal{C}}(\mathbf{y}_t) \leq -\eta_t \nabla_{\mathbf{y}} \tilde{\mathcal{C}}(\mathbf{y}_t) \cdot \nabla_{\mathbf{y}} \tilde{C}(x_t, \mathbf{y}_t) + \eta_t^2 b, \tag{8}$$

where $b = c_2 c_1^2 kp/2$.

After summing for $t = 1, \ldots, T$, we take the expected value, and let T diverges to obtain

$$\sum_{t=1}^{\infty} \eta_t \, \mathbb{E}\left[\|\nabla_{\mathbf{y}} \tilde{\mathcal{C}}(\mathbf{y}_t)\|_2^2 \right] \leq \mathbb{E}\left[\tilde{\mathcal{C}}(\mathbf{y}_1) \right] + \sum_{t=1}^{\infty} \eta_t^2 b < +\infty. \tag{9}$$

The series on the LHS is then summable. It follows that

$$\sum_{t=1}^{\infty} \eta_t \|\nabla_{\mathbf{y}} \tilde{\mathcal{C}}(\mathbf{y}_t)\|_2^2 < +\infty \text{ a.s.} \tag{10}$$

and then we can complete the proof of the first thesis:

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

Consider now that $\mathcal{C}()$ is continously differentiable up to the third order and then its third order partial derivatives are bounded over $\mathcal{B}_2(R')$, i.e., $|\partial^3 \tilde{\mathcal{C}}(\mathbf{y}_t)/\partial y_i \partial y_j \partial y_l| \leq c_3$. Let $a(\mathbf{y}) = \|\nabla_{\mathbf{y}} \tilde{\mathcal{C}}(\mathbf{y})\|_2^2$. This is continuously differentiable up to the second order. We can then use again the Taylor formula as above and obtain:

$$a(\mathbf{y}_{t+1}) - a(\mathbf{y}_t) \leq -2\eta_t \sum_{i,j=1}^{kp} \frac{\partial \tilde{\mathcal{C}}(\mathbf{y})}{\partial y_j} \frac{\partial^2 \tilde{\mathcal{C}}(\mathbf{y})}{\partial y_i \partial y_j} \frac{\partial \tilde{C}(x_t, \mathbf{y}_t)}{\partial y_i} + \eta_t^2 \frac{c_2' c_1^2 (kp)^2}{2},$$

where $c_2' = 2kp(c_2^2 + c_1 c_3)$.

If we now compute the conditional expectation given the history up to t, $H_t = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t)$, we obtain

$$\mathbb{E}[a(\mathbf{y}_{t+1}) - a(\mathbf{y}_t) | H_t] \leq 2c_2 kp \eta_t \|\nabla_{\mathbf{y}} \tilde{\mathcal{C}}(\mathbf{y}_t)\|_2^2 + \eta_t^2 \frac{c_2' c_1^2 (kp)^2}{2}.$$

From which we can show that $\{a(\mathbf{y}_t)\}_{t \in \mathbb{N}}$ is a quasi-martingale and converges almost surely to a random variable a_∞ with finite expected value. As $a_\infty > 0$ if and only if $\|\nabla_{\mathbf{y}} \tilde{\mathcal{C}}(\mathbf{y}_t)\|_2^2 > 0$ for large t, it is possible to reason as at the end of the proof of Lemma IV.1 and use (10) to conclude that $a_\infty = 0$ almost surely. It follows that

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

□

REFERENCES

[1] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, September 2001.

[2] S. Berchtold, C. Böhm, B. Braunmüller, D. A Keim, and H. Kriegel. Fast parallel similarity search in multimedia databases. *ACM SIGMOD Record*, 26(2):1–12, 1997.

[3] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti. A metric cache for similarity search. In *Proc. of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval*, LSDS-IR '08, pages 43–50, New York, NY, USA, 2008. ACM.

[4] S. Pandey, A. Broder, F. Chierichetti, V. Josifovski, R. Kumar, and S. Vassilvitskii. Nearest-neighbor caching for content-match applications. In *Proc. of the 18th International Conference on World Wide Web*, WWW '09, pages 441–450, New York, NY, USA, 2009. ACM.

[5] D. Asanov. Algorithms and methods in recommender systems. *Berlin Institute of Technology, Berlin, Germany*, 2011.

[6] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri. Soft cache hits: Improving performance through recommendation and delivery of related content. *IEEE Journal on Selected Areas in Communications*, 36(6):1300–1313, June 2018.

[7] B. Yan, D. R. Lovley, and J. Krushkal. Genome-wide similarity search for transcription factors and their binding sites in a metal-reducing prokaryote geobacter sulfurreducens. *Biosystems*, 90(2):421–441, 2007.

[8] A. F. Auch, H. Klenk, and M. Göker. Standard operating procedure for calculating genome-to-genome distances based on high-scoring segment pairs. *Standards in genomic sciences*, 2(1):142, 2010.

[9] J. Weston, S. Chopra, and A. Bordes. Memory networks. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2015.

[10] A. Graves, G. Wayne, and I. Danihelka. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 2014.

[11] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *Proc. of The 33rd International Conference on Machine Learning*, volume 48, pages 1842–1850, New York, New York, USA, 20–22 Jun 2016. PMLR.

[12] D. Crankshaw, X. Wang, J. E. Gonzalez, and M. J. Franklin. Scalable training and serving of personalized models. In *NIPS 2015 Workshop on Machine Learning Systems (LearningSys)*, 2015.

[13] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, Boston, MA, 2017. USENIX Association.

[14] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, volume 98, pages 194–205, 1998.

[15] T. Spyropoulos and P. Sermpezis. Soft cache hits and the impact of alternative content recommendations on mobile edge caching. In *Proc. of the Eleventh ACM Workshop on Challenged Networks*, CHANTS 16, page 5156, New York, NY, USA, 2016. Association for Computing Machinery.

[16] A. Bellet, A. Habrard, and M. Sebban. *Metric learning*, volume 9. Morgan & Claypool Publishers, 2015.

[17] Similarity caching trace repository. https://sim-cache.gitlabpages.inria.fr/similarity-caching-traces/.

[18] M. Garetto, E. Leonardi, and G. Neglia. Similarity Caching: Theory and Algorithms. In *IEEE Intl. Conference on Computer Communications (INFOCOM)*, Toronto, Canada, July 2020.

[19] M. Garetto, E. Leonardi, and V. Martina. A unified approach to the performance analysis of caching systems. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 1(3):12:1–12:28, May 2016.

[20] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, April 2004.

[21] J. Cortés and F. Bullo. Coordination and geometric optimization via distributed dynamical systems. *SIAM Journal on Control and Optimization*, 44(5):1543–1574, 2005.

[22] J. Cortés, S. Martínez, and F. Bullo. Spatially-distributed coverage optimization and control with limited-range interactions. *ESAIM: COCV*, 11(4):691–719, 2005.

[23] L. Bottou. On-line learning and stochastic approximations. In David Saad, editor, *On-line Learning in Neural Networks*, pages 9–42. Cambridge University Press, New York, NY, USA, 1998. online version updated in May 2018.

[24] S. Ioannidis, L. Massoulié, and A. Chaintreau. Distributed caching over heterogeneous mobile networks. In *Proc. of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 311–322, 2010.

[25] S. Ioannidis and E. Yeh. Adaptive caching networks with optimality guarantees. *SIGMETRICS Perform. Eval. Rev.*, 44(1):113124, 2016.

[26] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis. Learning to cache with no regrets. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 235–243, 2019.

[27] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *IEEE FOCS*, pages 459–468, 2006.

[28] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

[29] A. Babenko and V. Lempitsky. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence*, 37(6):1247–1260, 2014.

[30] B. Naidan, L. Boytsov, and E. Nyberg. Permutation search methods are efficient, yet faster search is possible. *arXiv preprint arXiv:1506.03163*, 2015.

[31] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[32] Faiss: A library for efficient similarity search. https://engineering.fb.com/data-infrastructure/faiss-a-library-for-efficient-similarity-search/.

[33] E. G. Coffman and P. J. Denning. *Operating systems theory*, volume 973. Prentice-Hall Englewood Cliffs, NJ, 1973.

[34] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter. Adaptsize: Orchestrating the hot object memory cache in a content delivery network. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 483–498, Boston, MA, March 2017. USENIX Association.

[35] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. *An analysis of approximations for maximizing submodular set functions—II*, pages 73–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[36] X. Corbillon, G. Simon, A. Devlic, and J. Chakareski. Viewport-adaptive navigable 360-degree video delivery. In *2017 IEEE international conference on communications (ICC)*, pages 1–7. IEEE, 2017.

[37] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proc. of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, pages 1–6, 2016.

[38] M. Zink, R. Sitaraman, and K. Nahrstedt. Scalable 360 video stream delivery: Challenges, solutions, and opportunities. *Proc. of the IEEE*, 107(4):639–650, 2019.

[39] J. Park and K. Nahrstedt. Navigation graph for tiled media streaming. In *Proc. of the 27th ACM International Conference on Multimedia*, pages 447–455, 2019.

[40] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. In *Proc. of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52, New York, NY, USA, 2015. Association for Computing Machinery.

[41] H. Wang, N. Wang, and D. Yeung. Collaborative deep learning for recommender systems. In *Proc. of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD 15, page 12351244, New York, NY, USA, 2015. Association for Computing Machinery.

[42] C. Hsieh, L. Yang, Y. Cui, T. Lin, S. Belongie, and D. Estrin. Collaborative metric learning. In *Proc. of the 26th International Conference on World Wide Web*, WWW 17, page 193201, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[43] I. Shenbin, A. Alekseev, E. Tutubalina, V. Malykh, and S. I. Nikolenko. Recvae: A new variational autoencoder for top-n recommendations with implicit feedback. In *Proc. of the 13th International Conference on Web Search and Data Mining*, WSDM 20, page 528536, New York, NY, USA, 2020. Association for Computing Machinery.

[44] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.