User-centric optimization of caching and recommendations in edge cache networks

Dimitra Tsigkari and Thrasyvoulos Spyropoulos
EURECOM, Biot, France
Email: {dimitra.tsigkari, thrasyvoulos.spyropoulos}@eurecom.fr

***Abstract*—On streaming platforms such as Youtube and Netflix, recommendations influence a large share of content consumption. In this context, user experience depends on both the quality of the recommendations (QoR) and the quality of service (QoS) of the delivered content. However, network decisions (such as caching) affecting QoS are usually made without explicit knowledge of the recommender's actions. Similarly, recommendation decisions are made without considering the potential delivery quality of the recommended content. In this paper, we propose to jointly optimize caching and recommendations in a generic network of caches, towards maximizing the quality of experience (QoE). This coincides with the recent trend for large content providers to also act as Content Delivery Network (CDN) owners. We formulate this joint optimization problem and prove that it can be approximated up to a constant. To the best of our knowledge, this is the first polynomial algorithm to achieve a constant approximation ratio for the joint problem. Our numerical experiments show important performance gains of the proposed algorithm over baseline schemes and existing algorithms.**

I. Introduction

A. Motivation

With the growing use of video on demand services, it can be expected that users await high quality of service (QoS). In particular, it has been shown that, on video streaming platforms, low bitrate can result in an increase in abandonment rate [1]. To this end, placing contents in caches close to the user can ensure high bitrate, short initial delay, etc. for the delivered content. Moreover, it is increasingly understood that the user's overall quality of experience (QoE) largely depends on *both* the QoS of the delivery *and* the interest of the user in the delivered content [2]. Meanwhile, the recommendation systems employed by these platforms significantly shape user requests; in Netflix, for example, 80% of requests come from the recommendations to the user [3]. The traditional role of a recommendation system has been to make personalized recommendations to the user *based solely on content interest*, *i.e.*, propose contents from a large catalogue that best match her interests. However, the QoS, *e.g.*, where the content is cached and whether this allows the recommended content to be delivered at low or high streaming quality, is not usually considered by most recommenders.

At first glance, content caching and recommendation systems appear to be independent, since they are usually governed by two different entities: Content Providers (CP) and network provider, or CP and 3rd party Content Delivery Networks (CDN). However, major CPs like Netflix and Google have recently started partnering with Internet Service Providers (ISPs) to implement their own caches inside the network: Netflix Open Connect and Google Global Cache. This allows the same entity to control and coordinate both content caching and recommendations, towards ensuring high user experience and minimizing delivery costs.

Some recent works study the problem of optimizing caching and/or recommendation policies by taking into account their interplay [4]–[9]. However, many of these works still focus on one side of the problem, *e.g.*, caching-friendly recommendations [5], [9], or recommendation-aware caching policies [6]. Works that do try to control both the caching and recommendation policies are usually based on heuristics [7], [8]. Hence, *a formal joint treatment of the two problems is largely missing.*

B. Our approach and contributions

The main goal of this paper is to formulate and study *the problem of optimizing both sets of control variables jointly*: (i) what content to store at each cache (in a network of caches), and (ii) what content to recommend to each user, based on the user's location in the caching network and the user's predicted preferences for contents. Our main contributions are:

- We introduce a simple yet generic metric of QoE for a recommendation-driven content application that depends on the content placement (a proxy of QoS) and the quality of recommendations that appear to the users. Based on this model, we formulate the problem of optimally choosing both sets of variables towards maximizing the aggregate users' QoE.
- While such joint caching and recommendation problems have been shown before to be NP-hard, to the best of our knowledge, we provide the first polynomial algorithm for the problem that has an approximation guarantee (in fact a constant one) for both equal and variable-sized contents.
- We investigate the performance of our algorithm through both synthetic and real data, and compare it with the state-of-the-art. Our results validate the theoretical guarantees and demonstrate that significant performance gains can be achieved with respect to baseline policies and to the best existing heuristics for the joint problem.

II. Problem Setup

A. Caching Network

We consider a set of C caches with capacity $\mathcal{C}_j$, $j = 1, \dots, C$ and a content catalogue $\mathcal{K}$. We assume that $\mathcal{C}_j \ll |\mathcal{K}|$, as is common in most caching setups. We will consider both equal-sized and variable-sized contents. For the latter, the size of content $i \in \mathcal{K}$ is denoted by σ_i. In our model, the caches are filled or updated during off-peak hours. Therefore, in what follows, all the problem parameters are considered to be known for the time period between two cache updates, as is common in caching-related works, *e.g.*, [10].

Definition 1 (Caching variable)**.** We let x_{ij} be the binary variable, where $x_{ij} = 1$ when the content i is cached in cache j, and $x_{ij} = 0$ otherwise. We denote the corresponding matrix by $X = \{x_{ij}\}_{i,j}$.

We consider a set $\mathcal{U}$ of users, each of which has access to a subset of caches. We denote this set by $\mathcal{C}(u)$ for user $u \in \mathcal{U}$. A request for content i by user u is served by one of the caches belonging to $\mathcal{C}(u)$ where the requested content is stored, *i.e.*, by one of the caches of the set $\{j : j \in \mathcal{C}(u) \text{ and } x_{ij} = 1\}$. The access to a cache could be over multiple links (as in hierarchical caching or in Information-centric networking) or direct (*e.g.*, wireless connectivity to a nearby small cell [10]). For the purposes of our analysis, such networks can be represented as a generic bipartite graph between users and (associated) caches, as shown in Fig. 1. Specifically, we assume that every edge of this graph has a weight s_{uj}, which denotes the *expected* streaming rate that can be supported between user u and cache j. This rate may differ from cache to cache, and may depend on channel quality, number of hops, scheduling policy, congestion level, etc.

Finally, we assume that there is a large cache C_0 that fits all the contents, *i.e.*, $x_{i0} = 1$ for all $i \in \mathcal{K}$, and is accessible by all users, *i.e.*, $C_0 \in \mathcal{C}(u)$, for all $u \in \mathcal{U}$. This could be a large cache deep in the network. For this reason and w.l.o.g., we let $s_{u0} < s_{uj}$, for all j and u, as is commonly assumed (*e.g.*, in [10]). This setup is generic and could capture a variety of caching networks, such as femto-caching framework [10], hierarchical CDN networks [11], etc.

B. Recommendations

A list of N_u recommended contents appears to the user $u \in \mathcal{U}$. This number may vary from user to user depending on the device used, as is the case in Netflix [3], for example. The recommendations are personalized and might depend on various factors such as user ratings (*e.g.*, via collaborative filtering), past user behavior, etc. [12]. State-of-the-art recommenders usually first assign a utility or "score" (or "rank") to each content for each user u, and then select the N_u items with the highest scores [12], [13]. Our model uses these utilities, denoted by $r_{ui} \in [0, 1]$, as input to our problem. For example, this could be the (normalized) predicted score through collaborative filtering of content i for user u.

Motivated by the discussion in Section I, we assume that both caching and recommendation decisions are made by the same entity (*e.g.*, Netflix, Google).

Definition 2 (Recommendation variable)**.** We let $y_{ui} \in \{0, 1\}$ denote the binary variable for content i being recommended to user u ($y_{ui} = 1$) or not ($y_{ui} = 0$). We denote by Y the $|\mathcal{U}| \times |\mathcal{K}|$ matrix of y_{ui}. Then, the equation $\sum_{i\in\mathcal{K}} y_{ui} = N_u$, for all $u \in \mathcal{U}$, captures the fact that N_u contents are recommended.

C. User model

The user makes content requests, affected by the aforementioned recommendations, according to the following model:

- with probability α_u the user requests a recommended content. For simplicity, we assume each of the N_u recommended items will be chosen with equal probability;
- with probability $(1-\alpha_u)$ the user ignores the recommendations and requests a content i of the catalogue with probability p_{ui}.

Essentially, α_u captures the percentage of time a user u tends to follow the recommendations. For example, it is estimated, on average, that $\alpha_u = 0.8$ on Netflix [3], but it can of course differ among users. Assuming prior knowledge of the user's disposition to follow the recommendations is common in related works (*e.g.*, [5], [7]) and also in other works on recommendation systems (*e.g.*, [14]). In practice, α_u might change over longer time intervals both because of intrinsic changes to user behavior or due to decreasing/increasing trust in the recommender. Nevertheless, in this work, we assume that our optimization happens at a smaller time scale, for which we can assume that the parameter α_u is roughly constant (but it can be recalibrated at longer intervals).

Furthermore, the assumption that each recommended content will be clicked with equal probability $1/N_u$ is also common in related works, and might hold in scenarios where the recommended items are "unknown" to the user, and hence she cannot evaluate their utility, before requesting them.

As for the p_{ui}, they capture the probability of user u requesting the content i outside of recommendations (*e.g.*, through the search bar). This could be an arbitrary distribution over the catalogue (*e.g.*, with probability mass only on content the user already "knows"). Alternatively, given the utilities r_{ui}, a reasonable choice could also be their normalized values:

$$p_{ui} = r_{ui} / \sum_{k\in\mathcal{K}} r_{uk}. \quad (1)$$

D. Example

To better elucidate our model, we present a small-scale example and Fig. 1 that illustrates the variables and the parameters defined. We consider a network of $C = 2$ caches of capacity 2, and a large cache C_0 containing a catalogue that consists of $|\mathcal{K}| = 9$ equal-sized contents. As shown, cache 1 contains contents 1 and 4, *i.e.*, $x_{11}, x_{14} = 1$ and $x_{1j} = 0$ for any other content j. There are 3 users present in the network. An edge between a user u and a cache j means that user u can fetch a content from cache j and the edge-weight is the corresponding rate. For example, for user 1, we have that $C(1) = \{0, 1\}$. Note that such an edge might actually correspond to a path of multiple physical links.

In this example, a single recommendation ($N_u = 1$) appears to every user (illustrated by a dashed-line arrow). For example, the content 4 is recommended to user 1 (*i.e.*, $y_{14} = 1$). If user 1 requests it, then it can be streamed from cache 1 at rate s_{11}. However, if user 1 requests, say, the content 2, this will be fetched from cache C_0 at a (lower) rate s_{10}. Lastly, arrows from users to recommendations display the probabilities α_u.

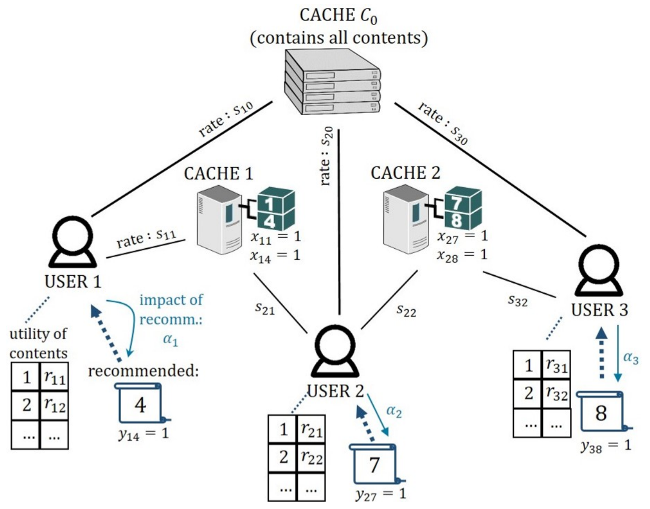


Fig. 1. Illustration of the example in Section II-D.

E. Quality of experience (QoE)

In the context of media streaming platforms, the user's entertainment and contentment with the provided services are affected by the quality of the recommendations she receives, *i.e.*, if they are tailored to her tastes or not. On the other hand, it has been observed that low QoS (*e.g.*, low streaming rates, rebufferings, etc.) greatly affects user experience and, most importantly (for CPs), retention/abandonement rates [1]. In fact, some recent experimental evidence suggests that users might be willing to tradeoff (some) content relevance for (better) QoS [2]. In this direction, *we model the user quality of experience as a twofold quantity*: one part relates to the quality of recommendations; the second part relates to the streaming rate[1] experience.

Definition 3 (Quality of Recommendations - QoR)**.** The quality of recommendations, as perceived by user u, is equal to $\sum_{i\in\mathcal{K}} y_{ui}\varphi(r_{ui})$, where φ is any non-decreasing function.

The function φ represents the impact of a recommended content's utility r_{ui} in the overall experience. It could be a linear function, or a concave function (*e.g.*, $\log(r_{ui})$) to capture diminishing returns beyond a minimum content utility.

Regarding the impact of streaming rate, the actual experienced rate depends on which cache it is streamed from. We assume, as in [10], that a content i requested by user u will be fetched by the "best" *connected* cache that stores it.

Definition 4 (Ordered streaming rates)**.** If $C(u)$ is the set of caches that user u has access to, we let $s_{u(1)} = \max\{s_{uj}, j \in \mathcal{C}(u)\}$ denote the maximum rate for user u. Similarly, $s_{u(2)}$ denotes the second highest rate for u, and so forth[2].

By definition, $s_{u|\mathcal{C}(u)|} = s_{u0}$, for every $u \in \mathcal{U}$, since we assumed that $s_{u0} < s_{uj}$, for all $j = 1, \ldots, C$.

In the following lemma, the expected streaming rate is given as a function of the caching policy (x_{ij}), the recommendations (y_{ui}) and the content popularities (p_{ui}).

Lemma 1 (Quality of Service - QoS)**.** *The rate at which user $u \in \mathcal{U}$ will download content $i \in \mathcal{K}$ upon request (for a given cache allocation X) is equal to:*

$$s_u(X,i) := \sum_{j=1}^{|\mathcal{C}(u)|} \Big[s_{u(j)} x_{i(j)} \prod_{l=1}^{j-1}(1 - x_{i(l)})\Big], \tag{2}$$

where $x_{i(j)}$ are similarly the caching variables assuming a rate-based ordering. Moreover, the expected streaming rate (that measures the QoS) for a user u is equal to:

$$\bar{s}_u = \alpha_u \sum_{i\in\mathcal{K}} \frac{y_{ui}}{N_u} \sum_{j=1}^{|\mathcal{C}(u)|} \Big[s_{u(j)} x_{i(j)} \prod_{l=1}^{j-1}(1 - x_{i(l)})\Big]$$
$$+(1-\alpha_u)\sum_{i\in\mathcal{K}} p_{ui} \sum_{j=1}^{|\mathcal{C}(u)|} \Big[s_{u(j)} x_{i(j)} \prod_{l=1}^{j-1}(1 - x_{i(l)})\Big]. \tag{3}$$

Proof. For a requested content $i \in \mathcal{K}$, the term $\prod_{l=1}^{j-1}(1 - x_{i(l)})x_{i(j)}$ captures the fact that i will be retrieved by the cache (j) (*i.e.*, the cache with the j-th highest rate) for lack of any other cache with higher rate in $\mathcal{C}(u)$ where the content is cached (*i.e.*, $x_{i(l)} = 0, l < j$). Then, this request will be served to the user at rate $s_{u(j)}$. Of course, if i is not cached in any cache, then it will be retrieved by the large cache C_0, which is ranked last, resulting in low streaming rate. Essentially, $s_u(X,i)$ is the highest rate associated to content i for user u among all the locations where i is cached.

Then, the formula of $\bar{s}_u$ easily follows by taking into account the user model explained in Section II-C. □

Remark 1. When estimating the QoS, instead of s_{uj} we can consider $\psi(s_{uj})$ for any non-decreasing function ψ of s_{uj}. Thus, we can calculate $\bar{s}_u$ by replacing $s_{u(j)}$ with $\psi(s_{u(j)})$. W.l.o.g. and for the sake of simplicity, we assumed here that ψ is the identity function, *i.e.*, $\psi(s_{uj}) = s_{uj}$. Note also that if we let $\psi(s_{uj}) = 1$ if $j \in \mathcal{C}(u)\setminus C_0$ and $\psi(s_{uj}) = 0$ otherwise, then (3) will estimate the cache hit rate per user for the (small) caches: upon a request, it counts 1 if the content is cached and therefore, retrieved from a small cache nearby.

Definition 5 (QoE function)**.** The quality of experience for user $u \in \mathcal{U}$ as a function of the caching and recommendation variables is defined as $\bar{s}_u + \beta_u \sum_{i\in\mathcal{K}} y_{ui}\varphi(r_{ui})$, where $\bar{s}_u$ is given by (3) and $\beta_u > 0$ is a tuning parameter. Then the aggregate quality of experience over all users is equal to:

$$f(X,Y) := \sum_{u\in\mathcal{U}} \Big[\bar{s}_u + \beta_u \sum_{i\in\mathcal{K}} y_{ui}\varphi(r_{ui})\Big]. \tag{4}$$

[1]While we call, for simplicity, the value s_{uj} the "streaming rate" this could also correspond to any measure of streaming experience that relates to initial buffering delay, rebufferings and other phenomena [15].

[2]As the rates s_{uj} are sorted for every user, the notation $s_{u(k)_u}, k = 1, \ldots, |\mathcal{C}(u)|$, is more appropriate. For simplicity, we drop the sub-index u.

Modeling QoE in this fashion implies a tradeoff between QoS and QoR, as evidenced in the earlier discussed works. The value of β_u, which might differ from user to user, captures the importance of each factor. High β_u means user u is more sensitive to recommendation quality, while low β_u that she is more sensitive to streaming quality. It is beyond the scope of this paper to investigate good choices for β_u or φ (or ψ in Remark 1). Instead, our focus is to propose efficient algorithms for *any* values and conforming functions.

TABLE I
IMPORTANT NOTATION

Notation	Description
$\mathcal{K}$	catalogue of contents
$\mathcal{U}$	set of users in the network
C_0	large cache containing the entire catalogue
C	number of caches in the network (C_0 is excluded)
$\mathcal{C}_j$	capacity of cache j, $j = 1, \cdots, C$
$\mathcal{C}(u)$	set of caches that user u communicates with
r_{ui}	utility of content i for user u
s_{uj}	streaming rate between user u and cache j
σ_i	size of content i
N_u	number of recommended contents for the user u
α_u	prob. that user u follows the recommendations
p_{ui}	prob. that user u requests content i while not following the recommendations
x_{ij}	caching variable, $x_{ij} = 1$ when content i is cached in cache j, and $x_{ij} = 0$ otherwise
y_{ui}	recommendation variable, $y_{ui} = 1$ when content i is recommended to user u, and $y_{ui} = 0$ otherwise

F. Joint recommendation and caching

We ask the following question: *How can we make caching and recommendation decisions in order to maximize QoE?*

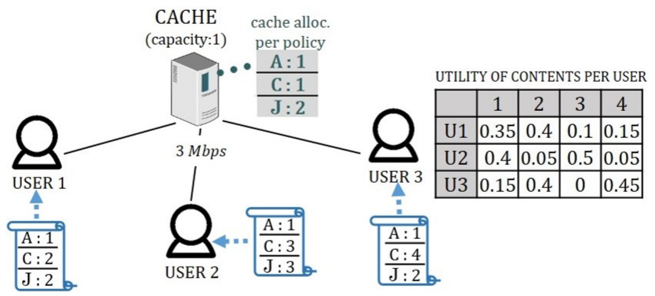
To better understand the tradeoffs involved, we present a toy example depicted in Fig. 2, and two "naive" policies:

Policy C, for "Conservative". This policy caches the $\mathcal{C}_j$ most popular contents (among all users connected to the cache j); it then recommends to each user u the N_u contents with the highest utility for this user, regardless of whether they are cached or not. In fact, this policy captures today's status quo.

Policy A, for "Aggressive". This policy has the same caching policy as policy C, but recommends only cached contents (the most relevant to user u among them). It is closer to cache-friendly recommendation policies like the one proposed in [6].

Note that both policies take the caching and recommendation decisions separately. In this example, we will attempt to show the benefits of a policy that jointly takes these decisions.

Referring to Fig. 2, suppose we have a catalogue of 4 equal-sized contents and 3 users, all connected to the large cache C_0 (not shown in the figure, for simplicity) that contains all files and a smaller cache C_1 of capacity 1. All users can download a content from C_1 with rate 3 Mbps while the rate from C_0 is 2 Mbps. We assume that the number of recommended items is $N_u = 1$ and $\alpha_u = 1$ for all users, *i.e.*, user requests are based exclusively on recommendations. We depict the utilities r_{ui} for each content i and user u on the right side.



UTILITY OF CONTENTS PER USER

	1	2	3	4
U1	0.35	0.4	0.1	0.15
U2	0.4	0.05	0.5	0.05
U3	0.15	0.4	0	0.45

Fig. 2. Toy example presented in Section II-F. On the left: illustration of the network together with the caching and recommendation decisions made by the A, C, and J policies. On the right: the matrix of content utilities per user.

Both policies A and C will cache the item with the highest aggregate utility, *i.e.*, content 1. Policy A would recommend this cached item to *all* users. Policy C would instead recommend the item with highest utility per user, namely contents 2, 3, and 4 respectively. It is easy to see that policy C would lead to better QoR, while policy A to better streaming rate. Nevertheless, neither policy is optimal with respect to maximizing the QoE (as defined in (4)).

A better option would be to cache content 2, observing that this would then facilitate the recommender. More precisely, it allows one to recommend content 2 to both users 1 and 3, achieving cache hits for them with maximum or close to maximum QoR (for user 1 and 3 respectively). Instead, for user 2, the content 3 is recommended (with utility $r_{23} = 0.5$), since content 2 would seriously degrade the user's QoR (r_{22} = 0.05 only). This policy, which we refer to as "J" for Joint in Fig. 2, outperforms policies A and C, in this example, in terms of QoE (for any beta and conforming ϕ functions).

In this example, it is easy to see how to outperform the simple policies A and C (even find the optimal one). However, this task becomes significantly harder for larger scenarios.

III. PROBLEM FORMULATION AND ANALYSIS

The optimization problem we are targeting is the following:

QoE problem.

$$\begin{aligned} \underset{X,Y}{\text{maximize}} \quad & f(X,Y) \\ \text{subject to} \quad & \sum_{i\in\mathcal{K}} \sigma_i x_{ij} \le \mathcal{C}_j \text{ for every } j = 1,\dots,C; && (5) \\ & \sum_{i\in\mathcal{K}} y_{ui} = N_u \text{ for every } u \in \mathcal{U}; && (6) \\ & x_{ij}, y_{ui} \in \{0,1\}, && (7) \end{aligned}$$

where, according to Eq. (2), (3), and (4), $f(X,Y)$ is equal to

$$\sum_{u\in\mathcal{U}} \sum_{i\in\mathcal{K}} \left[\alpha_u \frac{y_{ui}}{N_u} s_u(X,i) + (1-\alpha_u)\, p_{ui}\, s_u(X,i) + \beta_u y_{ui} \varphi(r_{ui})\right]$$

and $s_u(X,i) := \sum_{j=1}^{|\mathcal{C}(u)|} \left[s_{u(j)}\, x_{i(j)} \prod_{l=1}^{j-1}(1 - x_{i(l)})\right]$. The constraints in (5) are the capacity constraints for every cache. In the case of equal-sized contents, (5) suggests that no more than $\mathcal{C}_j$ items can fit in cache j, and the constraints in (6) suggest that each user receives N_u recommendations. Finally, as expressed in (7), x_{ij} and y_{ui} are binary/decision variables.

Lemma 2. *The QoE problem is NP-hard.*

Proof. A simple instance of the QoE problem is the femtocaching problem in [10] which is NP-hard. □

A. Intuition on joint optimization

As we saw in Lemma 2, even just the caching part (*i.e.*, maximizing in variable X) of the QoE problem is hard to solve. For this simpler problem, the authors in [10] propose algorithms with approximation guarantees by exploiting submodularity properties (for definition, see [16]) of the objective. However, these algorithms do not account for the recommendation part of the QoE problem (variable Y) and therefore, the approximation guarantees do not extend to the joint problem.

One could be tempted to extend the methodology in [10] by using both sets of variables X and Y as the ground set. However, the authors of [7] prove that a subcase of the QoE problem (when $\beta_u = 0$) is not submodular in X and Y.

Furthermore, the authors of [6] consider problem variants where the caching decision is "recommendation-aware". They manage to retrieve submodularity properties and use the methodology of [10] to derive efficient algorithms with approximation guarantees. However, their objective and problem setup does not contain recommendation variables. It is thus significantly different than the QoE problem. Finally, a brief qualitative comparison of these works can be found in Table II.

TABLE II
STATE-OF-THE-ART WORKS ON CACHING AND/OR RECOMMENDATIONS

Related Works	Variables		How many caches?	Approx. guarantees
	Caching	Recomm.		
[10]	✓	✗	Network	✓
[6]	✓	✗*	Network	✓
[7]	✓	✓	Single cache	✗
This work	✓	✓	Network	✓

*In [6], although the problem formulation does not contain any recommendation variable, the caching variable is "recommendation-aware".

This discussion raises the question of whether the QoE problem can be efficiently approximated and how. In the next section, we prove that this is indeed the case. By first considering something akin to a primal decomposition [17] of the original problem: rather than handling variables X and Y at the same time as the ground set, we show that:

(i) for the problem on variables Y, *i.e.*, fixing variables X ("inner" problem), the global maximizer can be found efficiently;
(ii) the problem on variables X ("outer" problem), given the global maximizer of Y, is in fact submodular.

This property will allow us to devise a combined algorithm for the joint problem that is polynomial in the problem size and, somewhat surprisingly, retains the approximation guarantees of the much simpler "caching-only" problems in [10] and [6].

B. Towards efficient algorithms

The key to our methodology is the following lemma.

Lemma 3. *The QoE problem is equivalent to the problem:*

Outer problem.

$$\begin{aligned} \underset{X}{\text{maximize}} \quad & f^*(X) := f(X, \underset{Y}{\text{argmax}} f(X,Y)) \qquad (8) \\ \text{subject to} \quad & (5), (6), \text{ and } (7). \end{aligned}$$

The equivalence of the two problems follows straightforwardly from the well known identity [18]:

$$\max_{X,Y} f(X,Y) = \max_X(\max_Y f(X,Y)). \qquad (9)$$

1) Inner problem and algorithm: The first step is to find a closed-form expression for f^* for any cache allocation, *i.e.*, matrix X. Hence, given X, the problem of choosing the recommendation policy, *i.e.,* matrix Y, is the problem of finding $f^*(X)$, as defined in (8). We formulate this problem:

Inner problem.

$$\begin{aligned} \underset{Y}{\text{maximize}} \quad & f(X,Y) \\ \text{subject to} \quad & (6) \text{ and } y_{ui} \in \{0,1\}. \end{aligned}$$

The following lemma will help us tackle the inner problem.

Lemma 4. *If $F^*_u(X) := \underset{Y}{max} \left(\overline{s}_u + \beta_u \sum_{i\in\mathcal{K}} y_{ui}\varphi(r_{ui})\right)$, for any u and any placement X, then $f^*(X) = \sum_{u\in\mathcal{U}} F^*_u(X)$.*

Proof. The inner problem can be decoupled into $|\mathcal{U}|$ problems since, given a cache allocation X, the recommendation decisions (variable Y) for a user do not interfere with the decisions for the other users. Note also that the constraint in (6) is decoupled for every user. □

By (2) in Lemma 1, we can write $F^*_u(X)$ as follows.

$$\begin{aligned} F^*_u(X) \quad &= \quad \max_Y \Bigg(\sum_{i\in\mathcal{K}} y_{ui} \big(\frac{\alpha_u}{N_u} s_u(X,i) + \beta_u \varphi(r_{ui}) \big) \Bigg) \\ &+ \quad (1-\alpha_u) \sum_{i\in\mathcal{K}} s_u(X,i) p_{ui}. \qquad (10) \end{aligned}$$

Next, we introduce the notion of V-value of a content, which is its value in terms of the inner problem.

Definition 6 (V-value and ordered V-values)**.** We define, as V-value of a content $i \in \mathcal{K}$ for user $u \in \mathcal{U}$ and for a given cache allocation X, the quantity

$$V_{ui}(X) := \frac{\alpha_u}{N_u} s_u(X,i) + \beta_u \varphi(r_{ui}). \qquad (11)$$

Similar to Def. 4, we define the ordered V_{ui} (sorted in decreasing order) as the ordered sequence $\{V_{u[k]}\}_{k\in\mathcal{K}}$[3].

The next lemma states that the optimal solution for the inner problem is to recommend to every user u the N_u contents with the highest V-value associated to the cache placement.

[3]We do not use the same notation as in Def. 4 because the ordering here is done with respect to the V-value and not the streaming rate. In general, $V_{u(k)}(X) \neq V_{u[k]}(X)$, for all $u \in \mathcal{U}$ and $k = 1, \dots |K|$.

Lemma 5. *For a given cache allocation X, we consider the matrix Y' such that $y'_{u[k]} = 1$ for $k = 1, \dots, N_u$, and $y'_{u[k]} = 0$ otherwise, where $[k]$ is the content index associated to the k-th highest V-value for any user $u \in \mathcal{U}$. Then*

$$F^*_u(X) = \sum_{k=1}^{N_u} V_{u[k]}(X) + (1 - \alpha_u) \sum_{i \in \mathcal{K}} \big(s_u(X, i) p_{ui} \big)$$
$$\text{and } f^*(X) = f(X, Y') = \sum_{u \in \mathcal{U}} F^*_u(X). \tag{12}$$

Proof. It is straightforward to prove the result above through contradiction, *i.e.*, assuming some content m with lower V-value than the $V_{u[N_u]}$ should have been included instead. □

Based on the previous lemma, here is a summary of the algorithm that finds the solution for the Inner Problem:

Inner algorithm (subroutine)

```
Input: $\mathcal{U}$, $\mathcal{K}$, $N_u$, $X$, $\{\beta_u\}$, $\varphi$, $\{\alpha_u\}$, $\{r_{ui}\}$, $\{s_{uj}\}$
1 Start with empty matrix $Y$
2 for every user $u \in \mathcal{U}$ do
3     for every content $i \in \mathcal{K}$ do
4         Calculate $V_{ui}$;
5         Sort $V_{ui}$ in decreasing order : $\{V_{u[k]}\}_{k=1}^{|\mathcal{K}|}$ ;
6     end
7     Set $y_{u[k]} = 1$ for $k = 1, \cdots, N_u$;
8 end
9 Return $Y$
```

2) Complexity of the inner algorithm: The internal for loop (lines $3 - 5$) consists of $|\mathcal{K}|$ calculations. Next, the complexity for the sorting step is $O(\log |\mathcal{K}|)$ in a pre-ordered list and the complexity of the assignment step (line 7) is $O(N_u)$, where $N_u \ll |\mathcal{K}|$. Since these steps are repeated for every user, the total complexity of the inner algorithm is at most $O(|\mathcal{U}| \cdot |\mathcal{K}|)$.

3) Outer problem and submodularity: We proved that the optimal Y can be found efficiently for the inner problem, given any cache allocation X. We will now prove some interesting properties of the outer problem (defined in Lemma 3) that will lead us to an algorithm for the QoE problem.

First, we need to extend f^* as a set function. To do so, we define the ground set that corresponds to cache allocations. More precisely, for any matrix X, we define the corresponding placement P_X of cached items in the network by

$$P_X := \{(i, j) : x_{ij} = 1, i \in \mathcal{K}, j = 1, \dots, C\}.$$

Essentially, P_X consists of the pairs (content, cache) of all the contents cached in the (small) caches in the network. Note that, since, by definition, the large cache C_0 contains the entire catalogue ($x_{i0} = 1$, for all $i \in \mathcal{K}$), X is considered as a $|\mathcal{K}| \times C$ matrix. In other words, P_X belongs to the set $\mathcal{P} := P(\mathcal{K} \times \{1, \dots, C\})$, where $P(\mathcal{K} \times \{1, \dots, C\})$ is the powerset of $\mathcal{K} \times \{1, \dots, C\}$. Inversely, given a placement P, we can define the corresponding matrix X_P such that x_{ij} is equal to 1, for every pair (i, j) in P, and 0 otherwise. Hence, from now on, X and P will be used interchangeably to denote the content allocation across the network of caches. We also define the subset of P representing the storage of the cache m: $P^{(m)} = \{(i, m) \in \mathcal{P} : x_{im} = 1\}$. We can thus extend the definitions of F^*_u, f^*, s_u and V_{ui} to the ground set $\mathcal{P}$.

Lemma 6. *The function F^*_u is monotone increasing for all u.*

Next, we define the marginal gain of F^*_u and we state an immediate consequence of Lemma 6.

Corollary 1 (Marginal gain)**.** *For a cache placement P, and a pair (i, j) such that $(i, j) \notin P$, we denote by*

$$\Delta F^*_u(P, (i, j)) := F^*_u(P') - F^*_u(P), \tag{13}$$

where $P' := P \cup \{(i, j)\}$, the marginal gain of F^*_u at P with respect to (i, j). Then, $\Delta F^*_u\,(P, (i, j)) \geq 0$.

Lemma 7. *The set function F^*_u is submodular for all $u \in \mathcal{U}$. Moreover, the set function f^*, as defined in* (8)*, is monotone increasing and submodular.*

The lemma above implies that f^* has the diminishing returns property. In other words, as the cache placement set becomes larger, the benefit of adding a new element (content, cache) to the set (*i.e.,* the marginal gain) decreases.

We omit the proofs of the Lemmas above due to space constraints. The detailed proofs can be found in [19].

C. QoE algorithms and guarantees

In the previous section, we managed to prove through the decomposition in (9) that $f^*(X)$ is submodular for any cache allocation X. The theory on submodularity optimization suggests that different greedy algorithm variants give constant approximations for the outer problem, and thus for the QoE problem (by Lemma 3). In fact, the factor of approximation depends on the type of constraints. In particular, in the QoE problem formulation, the constraints in (5) lead to different algorithms and approximation guarantees when the contents are equal-sized, *i.e.*, $\sigma_i = 1$ for all $i \in \mathcal{K}$.

1) The case of equal-sized contents: We define a greedy algorithm that we call the *QoE algorithm*. This algorithm starts with a placement P consisting of empty caches (except for the large cache that contains the entire catalogue) and greedily fills one by one all the available shots. In every round of selection, it calculates the marginal gain of f^* at P with respect to at most $C \cdot |\mathcal{K}|$ elements, *i.e.*, pairs (content, cache). For every such element (i, j), the Inner Algorithm is called (as subroutine) and its solution determines the recommendation decisions corresponding to the cache allocation $P \cup (i, j)$. This allows us to calculate the marginal gain of f^* at P with respect to (i, j) by (13) and (12). Then, the element that maximizes the marginal gain of f^* at P is selected and added to P (ties broken arbitrarily), before the next selection round begins. This procedure is repeated until all caches are full. The algorithm is summarized below.

Since the constraints in (5) are matroid constraints, as in [10], the theory on submodular maximization [16] suggests

QoE algorithm (for equal-sized contents)

```
Input: $C$, $\{\mathcal{C}_j\}$, $\mathcal{U}$, $\mathcal{K}$, $\{N_u\}$, $\{s_{uj}\}$, $\{r_{ui}\}$, $\{\beta_u\}$, $\{\alpha_u\}$
1  Start with empty caches, i.e., $P = \cup_{j=1}^{C} P^{(j)}$, where $P^{(j)} = \emptyset$, for all $j = 1, \dots, C$
2  Outer algorithm:
3  while caches are not full, i.e., $|P^{(j)}| < \mathcal{C}_j$ for all $j$, do
4      for every (not full) cache $j = 1, \dots, C$, do
5          for every content $i \in \mathcal{K}$ s.t. $(i,j) \notin P^{(j)}$, do
6              Estimate $\Delta f^*(P,(i,j))$ by calling Inner Algorithm(X); Store $\Delta f^*(P,(i,j))$ in a sorted list.
7          end
8      end
9      $(\eta,\theta) :=$ argmax$_{(i,j)} \Delta f^*(P,(i,j))$.
10     Add $(\eta,\theta)$ to $P$, i.e., $P^{(\theta)} \leftarrow P^{(\theta)} \cup (\eta,\theta)$.
11 end
12 Return $X^* \leftrightarrow P, Y^* = f^*(X^*)$
```

that a $1/2$-approximation is achievable by the above greedy algorithm. In particular, if we let OPT denote the optimal objective function value of the QoE problem with equal-sized contents, and (X^*, Y^*) denote the feasible solution given by the QoE algorithm, then

$$f(X^*, Y^*) \geq \frac{1}{2} OPT.$$

2) The general case of variable-sized contents: The difference between the two cases is the capacity constraints. In the general case, the constraints in (5) are knapsack constraints. However, the QoE algorithm as defined above is oblivious of the content's size. We adapt the QoE algorithm so that, in every round of selection, it adds to the cache the element (content, cache) that maximizes the ratio of marginal gain to the content's size, *i.e.*, $\frac{\Delta f^*(P,(i,j))}{\sigma_i}$, while satisfying the constraints in (5). We call this algorithm *s-QoE algorithm.*

However, in the case of variable-sized contents, both QoE and s-QoE algorithms can perform arbitrarily badly [20]. According to [20], it suffices to choose the maximum objective function value achieved by the two algorithms in order to achieve a $\frac{1-1/e}{2}$-approximation. Specifically, if we let OPT_s denote the optimal objective function value of the QoE problem in the case of variable-sized contents, and (X^*, Y^*), (X_s, Y_s) denote the feasible solutions given by the QoE and s-QoE algorithms respectively, then

$$\max\{f(X^*, Y^*), f(X_s, Y_s)\} \geq \frac{1-1/e}{2} OPT_s.$$

3) Complexity and implementation speed-ups: The complexity of the QoE and the s-QoE algorithms are the same.

The algorithms need to run at most $\sum_{j \in C} |\mathcal{C}_j|$ times in order to fill all caches. At each iteration, they evaluate the marginal gain of $C \cdot |\mathcal{K}|$ pairs (cache, content). For every evaluation, they call the *Inner Algorithm* of complexity $O(|\mathcal{U}| \cdot |\mathcal{K}|)$. Then, the complexity of the sorting step is $O(\log(C \cdot |\mathcal{K}|))$ in a pre-ordered list. Therefore, the total complexity of the QoE and s-QoE algorithms is $O(|\mathcal{U}| \cdot |\mathcal{K}|^2 \cdot C \cdot \sum_{j \in C} |\mathcal{C}_j|)$.

Implementation-wise, the method of lazy evaluations [20] avoids unnecessary calculations in the selection process of the caching placement. Furthermore, distributed techniques can be applied for submodular maximization [21]. These techniques achieve the same approximation guarantees.

D. The case C = 1

In this section, we mention briefly a result for the case where $C = 1$, *i.e.*, apart from the large cache C_0, there is only one cache. In this case, we can prove that the QoE problem can be transformed into an Integer Linear Program (ILP) problem and, therefore, common optimization methods can be applied to find the optimal solution for small problem's instances. This will be useful in the next section since it allows us to compare the solution of our algorithm with the optimal joint policy. For the interested reader, the proof can be found in [19].

IV. Performance Evaluation

In this section we compare the proposed policy (*QoE algorithm*) with other policies and validate its theoretical approximation guarantees. We consider a variety of scenarios.

A. Scenario 1

Firstly, we compare the objective function value achieved by the QoE algorithm with the optimal one (oracle). For this, we consider a scenario with a single cache and the large cache C_0. As mentioned in Section III-D, the QoE problem for $C = 1$ can be transformed into an ILP problem. We use the standard MATLAB solver to obtain the optimal objective value.

We consider 20 users connected to the cache and a catalogue of 200 unit-sized contents. We assume that the cache can fit 15 contents and every user receives $N = 2$ recommendations. The small size of the scenario is necessary to be able to calculate the optimal objective value. We will consider much larger scenarios subsequently. Moreover, the impact of the recommendations is determined by α_u, whose values follow a uniform distribution between 0.7 and 0.9 (in line with the statistics gathered on Netflix [3]). In this scenario, we consider a synthetic dataset for the utilities r_{ui} and the popularities p_{ui}. We chose p_{ui} such that the aggregate content popularities over all users, *i.e.*, $\sum_u p_{ui}$, follow a Zipf distribution (with parameter 0.6). Then, r_{ui} are chosen randomly in $[0, 1]$ such that their normalized value, *i.e.*, $r_{ui}/\sum_k r_{uk}$, are equal to p_{ui}, for every $i \in \mathcal{K}$, as in (1).

In this scenario, we measure the QoS as cache hits, as explained in Remark 1, and the QoR (Def. 3) by considering $\varphi(r_{ui}) = \log(r_{ui})$. For a variety of values of $\beta_u = \beta > 0$, we queried the oracle and we calculated the QoE given by the proposed algorithm. Table III shows the approximation ratio that our policy achieves for some values of β.

As we saw in Section III-C, the ratio $f(X^*, Y^*)/OPT$ cannot be lower than $1/2$. We observe that, in practice, the achieved ratio is much higher than $1/2$, close to 1. In fact, among all the different values we considered (30 in total), the lowest observed approximation ratio was equal to 0.9757.

Observation 1. Our numerical results validate the theoretical approximation guarantees of our policy and also suggest a much better approximation ratio in practice.

TABLE III
APPROXIMATION RATIO $(f(X^*, Y^*)/OPT)$

Parameter β	0.01	0.95	1.7	2.5	lower bound:
Approx. ratio	1	0.9757	0.9979	1	0.5

Next, we investigate if this close-to-optimal performance is reflected in the QoS-QoR tradeoffs. At the same time, we will compare these tradeoffs with the ones achieved by a proposed heuristic in the literature for a similar problem [7].

Cache-aware recommendations (CAwR). CAwR [7] makes caching and recommendation decisions at every cache independently. It decomposes the problem into the caching and recommendation steps. First, given the content preference distribution for every user (equivalent to the content popularity distribution p_{ui} or content utilities r_{ui} of our model) and the weight every user gives to recommendations (the α_u of our model), the aggregate request probability of every content is calculated. Then, the $\mathcal{C}_j$ items with the highest probability are cached. Note that, in the case of variable-sized contents, the cache allocation decisions are made by solving a $0-1$ knapsack problem, where the "value" of every content is the aforementioned probability and the "weight" is its size. Then, in the recommendation step, the recommendations are made partially by cached contents and by non-cached contents that are of high utility for the particular user. The balance between cached and non-cached contents is determined by a so-called distortion parameter $r_d \in [0, 1)$, which is similar to the parameter β of our model.

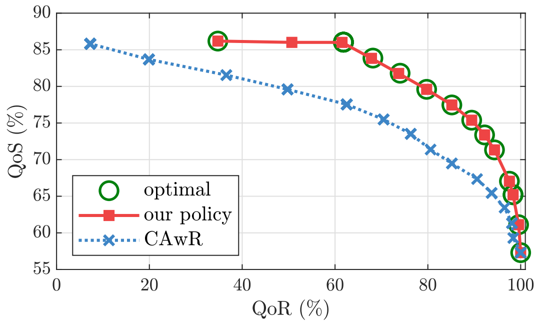


Fig. 3. Scenario 1, QoR-QoS tradeoff points for some values of β and r_d.

Figure 3 depicts the QoS-QoR tradeoffs given by the oracle, our policy, and CAwR as points in the plane. We obtained these tradeoffs for a variety of values of β and the distortion parameter r_d. We remind the reader that each of these points corresponds to a different objective tradeoff, between QoS and QoR, that a CP might have, *i.e.*, these curves could also be interpreted as Pareto curves. The QoR values (x-axis) are normalized with respect to the two "extreme" policies A and C (defined in Section II-F). For example, QoR $= 50\%$ implies that the QoR value lies in the middle of the interval $[R_A, R_C]$, where R_A and R_C are the QoR values achieved by policies A and C respectively. Moreover, the normalized QoS values (y-axis) give the cache hit rate.

Observation 2. Our policy's tradeoff curve almost coincides with the optimal, while it dominates that of CAwR, *i.e.,* our policy outperforms CAwR in terms of at least QoS or QoR (or both).

For example, for a desired value of QoS of around 84%, CaWR achieves 20% QoR and our policy 68%. More importantly, most of the tradeoffs of our policy (*e.g.*, around $80-95\%$ QoR and $70-80\%$ QoS) are not achievable by any tuning of the CAwR algorithm. Points in the extreme right lead to the maximum QoR and both policies recommend the same items, *i.e.*, the ones with *the highest* utility per user.

B. Scenario 2

We proceed with simulating larger scenarios. For this, we consider a single cache with 100 connected users. The catalogue consists of 6000 equal-sized contents[4]. The probabilities α_u are chosen randomly in $[0.7, 0.9]$, for all $u \in \mathcal{U}$. For this scenario, we use a real dataset for the matrix of utilities r_{ui}:

MovieLens dataset:. The MovieLens dataset [23] is a collection of 5-star movie ratings collected on MovieLens, an online movie recommendation service. Here, we used a variety of subsets of the total $20,000,263$ ratings available in the original dataset. It is commonly assumed that the utility of a content for a user is the *predicted* rating of this user for the content [13]. Therefore, we interpret the rating as the content utility. Since the range of ratings is $0.5-5$ with 0.5 increments, we map every rating r to a random number in the interval $(r/5-0.1, r/5]$. As is common, this matrix is quite sparse. To obtain the missing r_{ui} ratings we perform matrix completion through the TFOCS software [24].

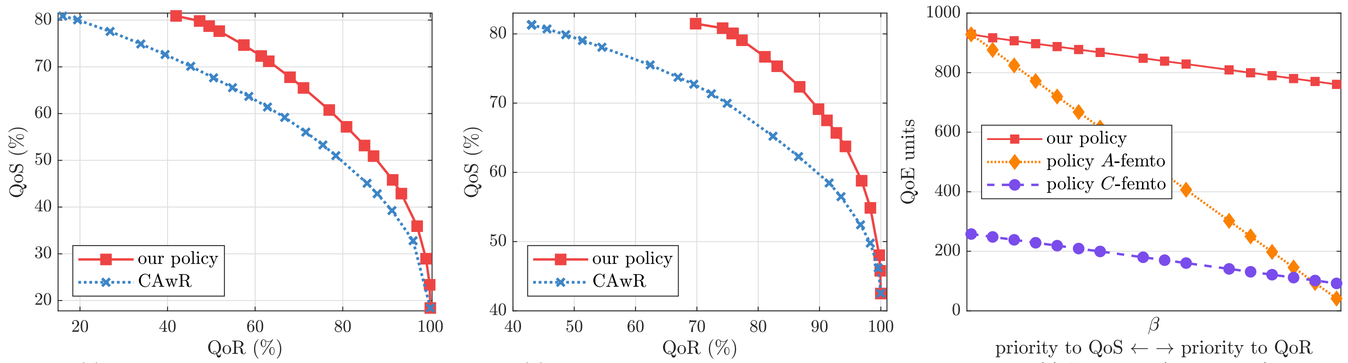
1) Equal-sized contents: We assume that the contents are of unit size and that the cache size is equal to 1% of the catalogue size. Moreover, $N = 5$ and p_{ui} are the normalized r_{ui}. As before, we measure the QoS as cache hits and the QoR as $\sum_i \log(r_{ui})$. In Figure 4(a), we plot the tradeoffs achieved by CAwR and our policy for different values of r_d and β.

Observation 3. The QoS-QoR tradeoff curve of our policy dominates that of CAwR even in larger, more realistic scenarios, driven by real datasets.

We notice, for example, that for QoS of value 81%, CAwR algorithm achieves 16% QoR while our algorithm 42%. Similar gains are noticeable in terms of QoS for different values of QoR. This is an encouraging finding that suggests that the theoretical gains could also be experienced in practice.

2) Variable-sized contents: So far, we have considered scenarios with equal sized content (*e.g.*, chunks). Here, we focus on a scenario with variable-sized contents. The contents'

[4]Note that according to [22], the total number of titles (movies and TV shows) available on Netflix in the USA is equal to 5848.



(a) Scenario 2 with equal-sized contents (b) Scenario 2 with variable-sized contents (c) Scenario 3 (multi-cache)

Fig. 4. (a)-(b) QoS-QoR tradeoff points, (c) QoE versus parameter β.

sizes vary from 1 to 15 size units and we adjust the cache capacity to fit up to 2.3% of the total size of the catalogue $\sum_{i\in\mathcal{K}} s_i$. For $N = 4$ and in the same setup as before, we plot in Fig. 4(b) the tradeoffs achieved by the two policies. As explained in Section III-C, our policy runs both QoE and s-QoE algorithms and selects the maximum between the two objective function values. As expected, the difference between the tradeoff curves is similar to the one in Fig. 4(a). In particular, we observe a relative gain of up to 63% in QoR and up to 15% in QoS of our policy with respect to CAwR.

Observation 4. The gains of our proposed algorithm are consistent for all the different scenarios and parameters considered, including the scenario of variable content sizes.

C. Scenario 3

So far, we studied scenarios with a single cache in order to be able to compare the performance of the proposed policy with the related work. We remind the reader that the approximation guarantees of our policy hold for arbitrary networks of caches where users might have access to more than one cache. The algorithm proposed in [10] makes caching decisions taking into account such coverage overlaps. However, this problem setup does not contain recommendations. In this scenario, we evaluate the performance, in terms of QoE, of our policy and some non-joint policies whose caching decisions are made according to [10].

A-femto and C-femto policies. They generalize the policies A and C described in Section II-F in a network of caches. They both make the caching decisions based on the femto-caching policy proposed in [10] that takes into account the fact that users have access to multiple caches in the network. Then, the recommendations part of the policies A and C is applied.

We consider a cellular network in a square area of $500\ m^2$ with 9 small-cell BS (helpers) and a macro-cell base station (the large cache of our scenario). A total of 100 users are placed in the area according to a homogeneous Poisson point process (in line with the related works [10], [6]), while helpers are placed in a grid. Helpers' communication ranges are set to 200 m, which results in an average of 3.5 helpers per user. Without loss of generality, we assume that the streaming rate from the large cache (or macro-cell cache) C_0 is 0.5 Mbps, while the s_{uj} rates for edge caches are chosen randomly between 2 and 15 Mbps[5]. In fact, the required Internet connection speed on YouTube [25] is 0.5 Mbps, and the recommended speed to watch a video in $4K$ is 20 Mbps.

We consider a subset of 6000 unit-sized contents of the Movielens dataset, α_u, φ as in Scenario 2, and ψ being the identity function. We set the helper's capacity to 1.5% of the catalogue size and $N = 5$.

For different values of $\beta > 0$, the achieved QoE of our policy, the A-femto, and the C-femto policies are shown in Fig. 4(c). We observe that, for β close to 0, *i.e.*, priority is given to the QoS, the performance of the A-femto policy and our policy coincide. This is because both policies make the same caching and recommendation decisions, *i.e.*, cache and recommend the most popular items. The QoE achieved by the C-femto policy is lower since, although it provides the best QoR, the recommended items are not necessarily among the cached ones and thus, they need to be retrieved from the large cache at the cost of lower QoS. In fact, this is illustrated in small scale in the toy example in Sec. II-F. As β increases, the priority moves towards QoR, and hence, the performance of the A-femto policy starts to worsen until it is dominated by the one of the C-femto policy. Our policy continues to perform better than both of them as a result of caching and recommendation orchestration.

Observation 5. The performance gains of the proposed policy over baseline non-joint policies are prominent in generic networks of caches as well.

V. Related Work

Caching and recommendations interplay. In the early work [26], the authors propose heuristic algorithms for recommendations in P2P networks that take into account both service cost and user preferences. In [5], the authors propose a recommendation algorithm that tries to bias requests towards

[5]As we are interested in capturing both wired (CDN) and wireless (femto-caching), the physical layer details are beyond the scope of this analysis.

contents in the cache. In a similar spirit, [4] proposes a reordering of the videos appearing in the YouTube related videos section by "pushing" the cached items on top of the list. However, the caching policy in these works is fixed.

Considering now different setups, [6] introduces the concept of "soft cache hits" that allows the user to choose an alternative cached content if the initially requested is not locally cached. The authors of [6] propose caching policies that are recommendation-aware while the recommender comes *after* the caching decisions. It thus can be seen as a simple decomposition algorithm. A decomposition of the joint problem is also proposed in [7] for a problem setup closer to our work. Targeting cache hit rate maximization, their policy first decides on caching, accounting for the impact of recommendations, and then adjusts the recommendations in order to favor cached items. However, no performance guarantees are given. Finally, the authors in [8] formulate a joint problem in the different context of prefetching content over a time-varying channel.

Joint optimization theory. Submodularity-based proofs for various caching-related problems have flourished since the seminal paper of [10], where the focus is on one set of variables (caching). While largely different problems, the decomposition method and submodularity of the outer problem followed in our work, is similar in spirit to the combinatorial methods in [27] and [28].

VI. Conclusion

In this paper, we studied the problem of maximization of users' QoE through joint caching and recommendation decisions in a network of caches. This is a problem of great interest as entities like Netflix can now manage both caching and recommendations in their network. We expressed user's QoE as a balanced sum of QoS (affected by the caching allocation) and QoR (determined by the recommendations the user receives). The model we considered is generic since QoS can be replaced by any caching gain/profit. We proposed the *QoE algorithm* that has $\frac{1}{2}$-approximation guarantees (or $\frac{1-1/e}{2}$ in the case of contents of heterogeneous sizes). Our numerical results showed that our algorithm outperforms baseline policies and other proposed schemes in the literature.

Acknowledgements

This work has been supported by the French National Research Agency (ANR) under the "5C-for-5G" JCJC project with reference number ANR-17-CE25-0001.

References

[1] H. Nam, K.-H. Kim, and H. Schulzrinne, "QoE matters more than QoS: Why people stop watching cat videos," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[2] P. Sermpezis, S. Kastanakis, J. I. Pinheiro, F. Assis, D. Menasché, and T. Spyropoulos, "Towards QoS-aware recommendations," 2019.

[3] C. A. Gomez-Uribe and N. Hunt, "The Netflix recommender system: Algorithms, business value, and innovation," *ACM Transactions on Management Information Systems (TMIS)*, vol. 6, no. 4, p. 13, 2016.

[4] D. K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen, "Cache-centric video recommendation: an approach to improve the efficiency of YouTube caches," *ACM Trans. on Multimedia Comput., Comm., and Applications (TOMM)*, vol. 11, no. 4, p. 48, 2015.

[5] T. Giannakas, P. Sermpezis, and T. Spyropoulos, "Show me the cache: Optimizing cache-friendly recommendations for sequential content access," in *IEEE WoWMoM*, 2018, pp. 14–22.

[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*, vol. 36, no. 6, pp. 1300–1313, 2018.

[7] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos, "Jointly optimizing content caching and recommendations in small cell networks," *IEEE Trans. Mob. Comput.*, vol. 18, no. 1, pp. 125–138, 2019.

[8] Z. Lin and W. Chen, "Joint pushing and recommendation for susceptible users with time-varying connectivity," in *Proc. IEEE GLOBECOM*, 2018, pp. 1–6.

[9] L. Song and C. Fragouli, "Making recommendations bandwidth aware," *IEEE Trans. Information Theory*, vol. 64, no. 11, pp. 7031–7050, 2018.

[10] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[11] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[12] G. Adomavicius and Y. Kwon, "Improving aggregate recommendation diversity using ranking-based techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 896–911, 2011.

[13] X. Amatriain, "Building industrial-scale real-world recommender systems," in *Proc. ACM RecSys*, 2012, pp. 7–8.

[14] M. Bressan, S. Leucci, A. Panconesi, P. Raghavan, and E. Terolli, "The limits of popularity-based recommendations, and the role of social ties," in *Proc. ACM SIGKDD*, 2016, p. 745–754.

[15] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia, "A survey on quality of experience of http adaptive streaming," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 469–492, 2014.

[16] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functions—II," in *Polyhedral combinatorics*. Springer, 1978, pp. 73–87.

[17] S. Boyd, L. Xiao, A. Mutapcic, and J. Mattingley. (2007) Notes on decomposition methods. [Online]. Available: https://see.stanford.edu/materials/lsocoee364b/08-decomposition_notes.pdf

[18] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[19] D. Tsigkari and T. Spyropoulos, "An approximation algorithm for joint caching and recommendations in cache networks," 2020.

[20] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *Proc. ACM SIGKDD*, 2007, pp. 420–429.

[21] A. Clark, B. Alomair, L. Bushnell, and R. Poovendran, "Scalable and distributed submodular maximization with matroid constraints," in *Proc. IEEE WiOpt*, 2015, pp. 435–442.

[22] Flixable. (2019) Netflix Museum. [Online]. Available: https://flixable.com/netflix-museum/

[23] F. M. Harper and J. A. Konstan, "The Movielens datasets: History and context," *ACM Trans. on Interactive Intelligent Sys. (TiiS)*, vol. 5, no. 4, p. 19, 2016.

[24] S. R. Becker, E. J. Candès, and M. C. Grant, "Templates for convex cone problems with applications to sparse signal recovery," *Mathematical Programming Computation*, vol. 3, no. 3, p. 165, Jul 2011.

[25] YouTube Help. (2019) System Requirements. [Online]. Available: https://support.google.com/youtube/answer/78358?hl=en

[26] D. Munaro, C. Delgado, and D. S. Menasché, "Content recommendation and service costs in swarming systems," in *Proc. IEEE ICC*, 2015.

[27] E. M. Craparo, J. P. How, and E. Modiano, "Throughput optimization in mobile backbone networks," *IEEE Transactions on Mobile Computing*, vol. 10, no. 4, pp. 560–572, Apr. 2011.

[28] T. Lukovszki, M. Rost, and S. Schmid, "Approximate and incremental network function placement," *Journal of Parallel and Distributed Computing*, vol. 120, pp. 159–169, 2018.