Abstract—Streaming platforms, like Netflix and YouTube, strive to offer high streaming quality (SQ), in terms of bitrate, delays, etc., to their users. Meanwhile, a significant share of content consumption of these platforms is heavily influenced by recommendations. In this setting, the user’s overall experience is a product of both the user’s interest in a recommended content, i.e., the recommendation quality (RQ), and the SQ of this content. However, network decisions (like caching) that affect the SQ are usually made without considering the recommender’s actions. Likewise, recommendations are chosen independently of the potential delivery quality. In this paper, we define a metric of streaming experience (MoSE) that captures the fundamental tradeoff between the SQ and RQ. We aim to jointly optimize caching and recommendations in a generic network of caches, with the objective of maximizing this metric. This is in line with the recent trend for content providers to simultaneously act as Content Delivery Network owners, implying that the same entity may handle both caching and recommendation decisions. We formulate this joint optimization problem and prove that it can be approximated up to a constant factor. To the best of our knowledge, this is the first polynomial algorithm to achieve a constant approximation ratio for the joint problem. Moreover, our numerical experiments show important performance gains of our algorithm over baseline schemes and existing algorithms in the literature.

Index Terms—Caching, recommendation systems, wireless networks, multimedia streaming services.

I. INTRODUCTION

A. Motivation

ON PLATFORMS of streaming services such as YouTube, Netflix, and Spotify, state-of-the-art recommendation systems are employed in order to help the users to navigate through their catalogues. Traditionally, the goal of the recommender is to offer personalized recommendations based on the user’s interests, i.e., present contents of an ever-evolving catalogue that are relevant to her tastes. Therefore, the user’s engagement with the service is closely related to the recommendation quality (RQ) [2]. Moreover, these recommendations are responsible for a large share of the generated user requests: on Netflix, 80% of requests come from the recommendations that appeared to the user [2].

At the same time, the streaming quality (SQ) of the delivered content plays a significant role in the overall user’s experience on the service. This quality can be characterized, for example, by metrics of quality of service (QoS) such as bitrate, initial delays, etc., or/and by metrics of quality of experience (QoE), i.e., QoS as perceived by the user. Furthermore, the SQ is closely related to the user engagement to the service. It has been shown that, on platforms of video streaming services, low bitrate can lead to an increase in the abandonment rate [3]. When a user abandons a viewing session, or the service all together, there is an associated revenue loss for the Content Provider (CP). In fact, Akamai estimates this loss for platforms where ads are displayed (like YouTube) in [4]. Caching mechanisms within the Content Delivery Networks (CDN) that store contents in caches close to the user can ensure a better delivery in terms of SQ [5] while alleviating the backhaul link traffic. In future architectures, envisioned for wireless networks beyond 5G, it has been widely argued that there will exist a very large number of small caches placed at the edge of the network and the caching problem in this “small cache” setting has been the focus of entire lines of research (e.g., [6] or see [7] for a survey on this subject). Therefore, making the right caching decisions is crucial in today’s and, more importantly, in future architectures. This is exactly where the influence of the recommendations on users’ requests could come into the picture. For this reason, some recent works propose caching and recommendation policies that take into account their interplay [8]–[14]. In fact, it has been argued that such an approach could be beneficial not only for the network’s performance but also for the user [15].

At first glance, content caching and recommendation systems seem to be independent, since they are usually handled by two different entities: the CP and a 3rd party CDN (like Akamai). However, major CPs like Netflix and Google started partnering with Internet Service Providers (ISPs) to implement their own CDN solutions inside the network: Netflix Open Connect and Google Global Cache. In particular, Netflix partners with ISPs and provides them with Open Connect caches that are implemented within the ISP network in order to localize the traffic as close to the user as possible [16]. Netflix is responsible for filling the caches during off-peak hours [17] and, of course, for choosing user recommendations [2]. Therefore, caching and recommendation decisions can already be handled jointly in today’s architectures. The
trend towards end-to-end network slicing in future wireless networks further supports such an approach. In this setting, CPs like the aforementioned ones will own their own virtual network (slice) including communication, storage, and CPU resources at the base stations of the Radio Access Network (RAN). Moreover, handling caching and recommendations together seems appropriate in future architectures like the “femtocaching” scenario [6]. In this setting, each cache will cover fewer users (than today’s CDN caches) and receive a limited amount of requests. This would render predicting content popularities and thus caching decisions particularly challenging [18].

Motivated by the above, in this paper, we define a metric of streaming experience (MoSE) that captures the user’s interest in a recommended content and in the quality that this will be delivered. The MoSE is modeled as a function of caching and recommendation decisions. Next, we address the problem of maximizing users’ MoSE in a setup where a CP jointly controls these decisions. Furthermore, a strong motivation behind our work is the fact that Netflix steers user recommendations towards popular (and possibly cached) contents. This is done, for example, through the “Trending Now” section of every user’s main page. In particular, “Trending Now” recommendations include short-term popular contents “combined with the right dose of personalization” [2]. Meanwhile, at Netflix, caching decisions are based on (predicted) content popularities [19]. Besides, we will show later that caching without taking into account the user recommendations and biasing recommendations later (to favor cached contents) is suboptimal. This further corroborates our approach of jointly optimizing caching and recommendations. Moreover, a recent development that supports such an approach is that Netflix mobile app introduced the feature “Downloads for You” [20]. When a user enables this feature, she allows Netflix to choose and proactively prefetch recommended contents on the user’s device that the user can then watch while she is offline or over unstable Internet connections.

Despite the arguments given above supporting a joint approach, many of the related works on caching and recommendations still focus on one side of the problem, e.g., network-friendly recommendations [8], [9], or recommendation-aware caching policies [10]. Some works that do try to modify both caching and recommendations are usually based on heuristics [11]–[14]. In addition, most of the aforementioned works study the problem from the CP’s point of view, i.e., with the goal of maximizing cache hit rate, without taking into account the user’s experience.

B. Our Approach and Contributions

In this paper, we formulate and analytically study the problem of jointly optimizing both variables: (i) what content to store at each cache, and (ii) what content to recommend to each user, based on their location in the caching network and their predicted preferences. In this direction, our main contributions are the following:

- We introduce the metric of streaming experience (MoSE) for a recommendation-driven content application that is expressed as a balanced sum of SQ (as a function of the caching variables) and RQ (as a function of the recommendation variables). Based on this model, we formulate the problem of optimally choosing both sets of variables towards maximizing users’ MoSE.
- The joint caching and recommendation problem has been shown before to be NP-hard and, thus, it cannot be solved optimally in polynomial time (unless P = NP). However, we provide a polynomial-time algorithm that approximates the optimum objective function value within a constant factor.
- Through numerical evaluations, we show a near-optimal performance and significant gains of our proposed policy over a variety of baseline schemes and existing heuristics for the joint problem. We also implemented distributed versions of our policy and we show that important speedups can be achieved. Our evaluations were conducted on both real and synthetic datasets, and using realistic values for the problem parameters.

This paper extends our earlier work [1] by providing the proofs of the theoretical results and a comprehensive evaluation of our proposed algorithm in a variety of scenarios and for different input parameters. Finally, the current work discusses in depth the applications of the presented problem and the possible extensions.

II. PROBLEM SETUP

A. Caching Network

We consider a set of $C$ caches with capacity $C_j$, $j = 1, \ldots, C$ and a content catalogue $K$. Moreover, $C_j \ll |K|$, $j = 1, \ldots, C$, as is an important restriction in most caching setups and especially in future wireless networks. We will consider both equal and variable-sized contents. In the second case, we denote by $\sigma_i$ the size of content $i$, where $i = 1, \ldots, |K|$.

**Definition 1 (Caching Variables):** 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 $U$ of users, each of which has access to a subset of caches. We denote this set by $C(u)$ for user $u \in U$. A request for content $i$ by user $u$ is served by one of the caches belonging to $C(u)$ where the requested content is stored, i.e., by one of the caches of the set $\{j : j \in C(u) \text{ and } x_{ij} = 1\}$. The access to a cache could be over multiple links (as in hierarchical caching or ICNs) or direct (e.g., wireless connectivity to a nearby small cell [6]). 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, every edge of this graph has a weight $s_{uj}$, which denotes the cache-specific streaming quality that can be supported between user $u$ and cache $j$. This quality

$1$In fact, according to estimations of the size of Netflix catalogue [21] and the size of Open Connect appliances [22], the cache capacity of the appliances varies from 0.1% to 2.3% of the entire catalogue.

$2$This assumption is realistic. At Netflix, for example, information on routes, network proximity to the users, etc. are gathered by the Open Connect caches and are sent regularly to the Netflix cloud [16].
can be related to estimations on the QoS or QoE, i.e., delays, rebufferings, rate switching, etc. Moreover, it may differ from cache to cache, or may depend on channel quality, number of hops, scheduling policy, congestion level, etc. Finally, there is cache to cache, or may depend on channel quality, number of rebufferings, rate switching, etc. Moreover, it may differ from can be related to estimations on the QoS or QoE, i.e., delays, 

**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 [2], for example. The recommendations are personalized and might depend on various factors such as user ratings (e.g., via collaborative filtering), past user behavior, viewing times, etc. [25]. State-of-the-art recommenders usually assign a utility (or “score” or “rank”) to each content for each user $u$ [25], [26]. We denote by $r_{ui} \in [0, 1]$ these utilities. Typically, the CP would select the $N_u$ items with the highest $r_{ui}$ to feature the recommendations list of user $u$. In this work, the recommendation decisions (i.e., deciding which contents will appear to the user’s recommendations list) are made not only based on the utilities $r_{ui}$ but also on the caching decisions. Therefore, our model uses the utilities $r_{ui}$ (that are derived from a typical recommender) as input for our problem.

**Definition 2 (Recommendation Variables):** 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 matrix of $y_{ui}$. Then, the equations $\sum_{i \in K} y_{ui} = N_u$ for all $u \in \mathcal{U}$, capture the fact that $N_u$ contents are recommended.

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

**C. User Model**

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

- with probability $\alpha_u$ the user requests a recommended content. Each of the $N_u$ recommended items will be chosen with equal probability by the user;
- with probability $(1 - \alpha_u)$ the user ignores the recommendations and request a content $i$ of the catalogue with probability $p_{ui}$.

Essentially, $\alpha_u$ 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 [2], 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., [9], [11]) and also in other works on recommendation systems (e.g., [27]). In practice, $\alpha_u$ might change over longer time intervals both because of intrinsic changes to user behavior or due to decreasing/increasing trust in the recommender. Such changes could be addressed by dynamic or stochastic models, that are out of the scope of this work. In this work, we assume that our optimization happens at a smaller time scale, for which the parameter $\alpha_u$ 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
TABLE I

| Notation | Description |
|----------|-------------|
| $K$      | catalogue of contents |
| $U$      | 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) |
| $C_j$    | capacity of cache $j$, $j = 0, \ldots, C$ |
| $C(u)$   | set of caches that user $u$ communicates with |
| $r_{ui}$ | utility of content $i$ for user $u$ |
| $s_{ui}$ | cache-specific streaming quality between user $u$ and cache $j$, $s_{ui}(j)$ for ordered qualities (see Def. 4) |
| $\sigma_i$ | size of content $i$ |
| $N_u$    | number of recommended contents for user $u$ |
| $\alpha_u$ | probability that user $u$ follows the recommendations |
| $p_{ui}$ | probability that user $u$ requests content $i$ while not following the recommendations |
| $\varphi$ | function of $r_{ui}$ that captures the impact of $r_{ui}$ in the perceived recommendations quality |
| $x_{ij}$ | 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 |

A reasonable choice could also be the normalized values:

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

**D. Example**

To better elucidate our model thus far, we present a small-scale example and Fig. 1 that illustrates the variables and the parameters defined above. We consider a network of $C = 2$ caches of capacity 2, and a large cache $C_0$ containing the entire catalogue that consists of $|K| = 9$ equal-sized contents. As shown, cache 1 contains contents 1 and 4 (i.e., $x_{11}, x_{14} = 1$), while $x_{12} = 0$ for any other $j$. 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 $j$. 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. The corresponding cache-specific stream quality is indicated as an edge weight. 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 streaming quality $s_{11}$. However, if user 1 requests, say, the content 2, this will be fetched from cache $C_0$ at a (lower) quality $s_{10}$. Lastly, arrows from users to recommendations display the probabilities $\alpha_u$.

**E. Metric of Streaming Experience (MoSE)**

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 SQ (e.g., low bitrates, rebufflers, etc.) greatly affects user experience and, most importantly (for CPs), retention/abandonment rates [3]. In fact, experiments on video streaming services have shown that the user’s overall experience depends on both the streaming quality and the user’s interest in a content [28]. Moreover, some recent experimental evidence suggests that users might be willing to tradeoff (some) content relevance for (better) QoS [15]. In this direction, we define the metric of streaming experience as a twofold quantity: one part relates to the recommendation quality; the second part relates to the streaming quality.

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

The function $\varphi$ represents the impact of a recommended content’s utility $r_{ui}$ in the user’s perceived RQ. It could be a linear function, or, more commonly, a concave function (e.g., $\log(r_{ui})$) to capture diminishing returns beyond a minimum content utility. Moreover, we can demand a minimum quality $r_{\text{min}}$ for any recommendation if we define $\varphi$ as follows:

$$\varphi(r_{ui}) = \begin{cases} \log(r_{ui}) & \text{if } r_{ui} \geq r_{\text{min}}, \\ -\infty & \text{otherwise}. \end{cases} \quad (2)$$

Regarding the SQ, this depends on which cache the requested content is streamed from. A content $i$ requested by user $u$ will be fetched by the “best” connected cache that stores it, as in [6].

**Definition 4** (Ordered Cache-Specific Qualities): If $C(u)$ is the set of caches that user $u$ has access to, we let $s_{u(1)} = \max\{s_{uj}, j \in C(u)\}$ denote the maximum (cache-specific) quality for user $u$. Similarly, $s_{u(2)}$ denotes the second highest quality for $u$, and so forth.4

By definition, $s_{u|C(u)} = s_{u0}$, for every $u \in U$, since we assumed that $s_{u0} < s_{uj}$, for all $j = 1, \ldots, C$. In the following lemma, the expected streaming quality (SQ) is given as a function of the caching policy ($x_{ij}$), the values of $s_{uj}$, the recommendations ($y_{ui}$), and the users preferences ($r_{ui}$).

**Lemma 1** (Expected Streaming Quality-SQ): For a given cache allocation $X$, a content $i \in K$ will be streamed to user $u$ (upon request) in the quality:

$$s_u(X, i) := \sum_{j=1}^{[C(u)]} \sum_{l=1}^{j-1} \prod_{l=1}^{j-1} (1 - x_{il(l)}) \left[ s_{u(j)} x_{ij} \right], \quad (3)$$

where $x_{i(j)}$ are similarly the caching variables assuming $s_{uj}$-based ordering.5 The expected streaming quality (SQ) for a user $u$ is equal to:

$$\pi_u = \alpha_u \sum_{i \in K} \frac{y_{ui}}{N_u} s_u(X, i) + (1 - \alpha_u) \sum_{i \in K} p_{ui} s_u(X, i). \quad (4)$$

**Proof:** For a requested content $i \in K$, $\prod_{l=1}^{j-1} (1 - x_{il(l)}) x_{ij}$ captures the fact that $i$ will be retrieved by the cache $(j)$ (i.e., the cache with the $j$-th highest quality) for lack of any other cache with higher quality in $C(u)$ where the content is cached (i.e., $x_{i(l)} = 0$, $l < j$). Then, this request will be served

---

4This quantity can serve as an additional safeguard to support the earlier assumptions: e.g., if any recommended content’s utility is above $r_{\text{min}}$, then, $\alpha_u$, the user’s trust in the recommendations, will not be compromised.

5As the qualities $s_{uj}$ are sorted for every user, the notation $s_{u(k)}$ would be more appropriate. For simplicity, we drop the sub-index $u$.

6Given the $s_{uj}$-based ordering, $x_{ij}$ indicates if the content $i$ is cached in the cache that offers the $j$-th highest quality for user $u$. 

---

3This quantity can serve as an additional safeguard to support the earlier assumptions: e.g., if any recommended content’s utility is above $r_{\text{min}}$, then, $\alpha_u$, the user’s trust in the recommendations, will not be compromised.
in the cache-specific quality $s_{uj}$. Of course, if $i$ is not cached in any cache, it will be retrieved from $C_0$ which is ranked last, resulting in low streaming quality. Essentially, $s_u(X, i)$ is the highest cache-specific quality associated to content $i$ for user $u$ among all the locations where $i$ is cached. Finally, given that, upon request, the content $i$ will be streamed in the quality $s_u(X, i)$, and given the probabilities of such a request to happen (through recommendations or not), the formula of the expected streaming quality, $\pi_{ui}^{\text{est}}$, easily follows.

Remark 1: When estimating the SQ, $s_{uj}$, the cache-specific streaming quality, can be chosen to be a function of QoS-related or QoE-related estimations. For example, $s_{uj}$ could be the estimated bitrate between user $u$ and cache $j$, or it could also include factors related to rebuffering probabilities, jitter, delays, etc., as is commonly considered in works related to streaming experience [29]. Our framework is defined in such a way that is flexible enough to optimize whatever such values(s) the CP deems appropriate or is able to estimate. Alternatively, $s_{uj}$ could be equal to:

$$s_{uj} = \begin{cases} 1 & \text{if } j \in C(u) \setminus C_0, \\ 0 & \text{otherwise}. \end{cases} \quad (5)$$

In that case, $\sum_u \pi_{ui}^{\text{est}}$ estimates the expected cache hits for the (small) caches: upon a request, it counts 1 if the content is cached. In Table II, we provide a variety of examples functions/values for $s_{uj}$ that exist already in the literature on multimedia streaming services.

Definition 5 (MoSE Function): The metric of streaming experience for user $u \in U$ as a function of the caching and recommendation variables is defined as $\pi_u^{\text{est}} + \beta_u \sum_{i \in K} y_{ui} \varphi(r_{ui})$, where $\pi_u$ is given by (4) and $\beta_u > 0$ is a tuning parameter. Then the aggregate MoSE over all users is equal to:

$$f(X, Y) := \sum_{u \in U} \left[ \pi_u + \beta_u \sum_{i \in K} y_{ui} \varphi(r_{ui}) \right]. \quad (6)$$

Modeling the streaming experience in this fashion implies a tradeoff between SQ and RQ, as discussed at the beginning of this section. The value of $\beta_u$ is the weight we attach to the RQ and quantifies the importance of the RQ compared to the SQ. Moreover, the value of $\beta_u$ might differ from user to user: large $\beta_u$ means the user $u$ is more sensitive to the RQ, while small $\beta_u$ that she is more sensitive to the SQ. The choice of $\beta_u$ can depend, for example, on user behavior: we might want a small $\beta_u$ (i.e., priority to the SQ) for a user who often abandons the viewing session when the streaming quality is low. Similarly, one could imagine more complex models (e.g., based on machine learning). However, it is beyond the scope of this paper to investigate such models or good choices for $\beta_u$, $\varphi$, and $s_{uj}$. Instead, our focus is to propose efficient algorithms for any values and conforming functions.

We stress here that the MoSE was defined in such a generic way that can be adjusted according to the needs of the CP. Particularly, the CP can choose the quantities $s_{uj}$ and the function $\varphi$ based on available measurements in network, user behavior, etc. We provide a detailed example set of choices (derived mostly from related work) for the different MoSE components, as shown in Table II.

F. Joint Recommendation and Caching

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

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 $C_j$ most popular contents (for the 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. This policy captures today’s status quo.

Policy A, for “Aggressive”. This policy has the same caching policy as policy C, but it recommends only cached contents (the most relevant to the user among them). It is closer to cache-friendly recommendation policies like the one in [10].

Note that both policies make the caching and recommendation decisions separately. In this example, we will attempt to show the benefits of a policy that makes jointly 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. We measure the SQ through the estimated bitrate. All users can download a content from $C_1$ or $C_0$ with bitrate 3 Mbps or 2 Mbps respectively. We further assume that $N_u = 1$ and $\alpha_u = 1$ for all users. We depict the utilities $r_{ui}$ on the right side.

Both policies would cache the item with the highest aggregate utility, i.e., content 1. Policy A would then recommend this item to all users. Policy C would instead recommend the

| Table II: Example Functions/Values for MoSE Components |
|-----------------------------------|---------------------------------|---------------------------------|
| Component                          | Example functions/values            | Comments                          |
| $s_{uj}$ in SQ                     | • QoS metrics                      | e.g., related to bitrate, delays, packet loss, etc. |
|                                   | • estimated QoS as a function $\psi$ of QoS | as in [30] (exponential QoS-QoE relation, IQX hypot.) |
| (as in Lemma 1)                    | • QoS metrics                      | or as in [31] (logarithmic QoS-QoE relation) |
|                                   | • estimated multidimensional QoE    | as in [32] as in [29]            |
|                                   | • cache hits (as defined in (5))   | requires no network-related measurements/estimations |
| $\beta_u > 0$ (weight factor)      | • small $\beta_u$                  | $\beta_u$ represents the sensitivity of user $u$ to the SQ and RQ. |
|                                   | • large $\beta_u$                  | small/large $\beta_u$ priority to the SQ/RQ |
| function $\varphi$ in RQ           | • logarithmic function             | it captures the diminishing returns property |
| (as in $\sum_{i \in K} y_{ui} \varphi(r_{ui})$) | • as defined in (2)                | (as often observed in human perception [33]) |
|                                   |                                   | it captures minimal QR thresholds per user |

6Since our focus is proactive caching, we consider the estimated or average SQ. An interesting direction for future work would be to consider dynamic policies where the SQ is time-varying or even unknown.
suggest that each user receives both A and C in this example in terms of MoSE (foricy which we refer to as “J” for Joint in Fig. 2, outperforms recommended (with utility to maximum RQ. Instead, for user 2, the content 3 is rec-
arcing something akin to a primal decomposition [34] of the problem), the global maximizer can be found efficiently; (7) suggests that no more than \( C_j \) items can fit in cache \( j \), and the constraints in (8) suggest that each user receives \( N_u \) recommendations. Finally,
as expressed in (9), the problem’s variables, \( x_{ij} \) and \( y_{ui} \), are binary.

**Lemma 2:** The MoSE problem is NP-hard.

*Proof:* An instance of the MoSE problem is the femto-
caching problem in [6] which is NP-hard. 

### A. Intuition on Joint Optimization

As we saw in Lemma 2, even just the caching part (i.e., max-
imizing in variable \( X \)) of the MoSE problem is hard to solve. For this simpler problem, the authors in [6] propose algorithms with approximation guarantees by exploiting submodularity properties of the objective. However, these algorithms do not account for the recommendation part of the MoSE problem (variable \( Y \)) and, therefore, the approximation guarantees do not extend to the joint problem.

One could be tempted to extend the methodology in [6] by using both sets of variables \( X \) and \( Y \) as the ground set. However, the authors of [11] prove that a subcase of the MoSE problem (when \( \beta_u = 0 \)) is not submodular in \( X \) and \( Y \).

Furthermore, the authors of [10] consider problem vari-
ants where the caching decision is “recommendation-aware”. They show that this problem is hard even for one cache, but manage to retrieve submodularity properties and use the methodology of [6] to derive algorithms with approximation guarantees. However, their objective and problem setup do not contain recommendation variables (among other things, the recommender’s actions are fixed, and the caching policy simply knows what the recommender would do). It is thus significantly different than the MoSE problem. Finally, a brief qualitative comparison of these works is shown in Table III.

This discussion raises the question of whether the MoSE 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 [34] 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 \( X \) (“inner” problem), the global maximizer can be found efficiently;
(ii) the problem on variables \( X \) (“outer” problem), given the global maximizer of \( Y \) (for any \( X \)), is in fact submodular. This property will allow us to devise an 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 considered in [6] and [10].

### B. Towards Efficient Algorithms

The key to our methodology is the following lemma.
Lemma 3: The MoSE problem is equivalent to the problem:

**Outer problem**

\[
\begin{aligned}
\text{maximize} & \quad f^*(X) := f\left(X, \arg \max_{Y} f(X, Y)\right) \\
\text{subject to} & \quad (7), (8), \text{and} (9).
\end{aligned}
\]  

(10)

The equivalence of the two problems follows straightforwardly from the well known identity [35]:

\[
\max_{X,Y} f(X, Y) = \max_{X} \left( \max_{Y} f(X, Y) \right).
\]  

(11)

1) **Inner Problem and Algorithm:** The first step would be 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 (10). We formulate this problem:

**Inner problem**

\[
\begin{aligned}
\text{maximize} & \quad f(X, Y) \\
\text{subject to} & \quad (8) \text{ and } y_{ui} \in \{0, 1\}.
\end{aligned}
\]

The following lemma states that the inner problem can be decoupled into \(|U|\) problems.

**Lemma 4:** If \(F^*_u(X) := \max_Y \bar{f}_u + \beta_u \sum_{i \in K} y_{ui} \varphi(r_{ui})\), for any \(u\) and any placement \(X\), then \(f^*(X) = \sum_{u \in U} F^*_u(X)\).

**Proof:** Given a cache placement \(X\), it is easy to see that the recommendation decisions (variable \(Y\)) for a user do not interfere with the decisions for the other users. Moreover, the constraints in (8) are decoupled for every user.

By (3) in Lemma 1, we can write \(F^*_u(X)\) as follows.

\[
F^*_u(X) = \max_Y \left( \sum_{i \in K} y_{ui} \left( \frac{\alpha_u}{N_u} s_u(X, i) + \beta_u \varphi(r_{ui}) \right) \right) + (1 - \alpha_u) \sum_{i \in K} s_u(X, i) p_{ui}.
\]

(12)

Next, we introduce the notion of V-value, which is the coefficient of \(y_{ui}\) in (12).

**Definition 6 (V-Value and Ordered V-Values):** We define, as V-value of a content \(i \in K\) for user \(u \in U\) and for a given cache allocation \(X\), the quantity

\[
V_{ui}(X) := \frac{\alpha_u}{N_u} s_u(X, i) + \beta_u \varphi(r_{ui}),
\]

(13)

where \(s_u(X, i)\) is defined in (3). Similar to Def. 4, we define the ordered \(V_{ui}\) (sorted in decreasing order) as the ordered sequence \(\{V_{ui}[k]\}_{k \in \mathbb{K}}\).

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

\[
F^*_u(X) = \sum_{k=1}^{N_u} V_{ui}[k](X) + (1 - \alpha_u) \sum_{i \in K} (s_u(X, i) p_{ui}),
\]

(14)

\[
\text{and } f^*(X) = f(X, Y') = \sum_{u \in U} F^*_u(X).
\]

(15)

In words, 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 \(X\). Note that this solution depends on the solution of the outer (caching) problem and, hence, the inner problem needs to be solved as part of solving the outer problem, as shown in (11).

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

Based on Lemma 5, the algorithm that finds the solution for the Inner Problem is summarized in Algorithm 1.

2) **Complexity of the Inner Algorithm:** The internal for loop (lines 3-5) consists of \(|K|\) calculations. Next, the complexity for the step of storing the \(N_u\) highest V-values is \(O(\log N_u)\), however \(N_u\) is considered to be a constant. Since these steps are repeated for every user, the total complexity of the inner algorithm is at most \(O(|U| \cdot |K|)\).

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 want now to solve the outer problem (defined in Lemma 3, (14), (15)), i.e., with respect to the caching variables \(X\). While often caching problems can fit into the category of knapsack/general assignment problems, this is not the case for the outer problem. We note that the “profit” or gain of storing a content into a cache is not a constant and depends on the solution of the inner problem. Nevertheless, we will now prove some interesting properties of the outer problem that will lead us to an algorithm for the MoSE problem.

First, we extend \(f^*\) as a set function. For any matrix \(X\) we define the corresponding placement \(P_X\) of cached items:

\[
P_X := \{(i, j) : x_{ij} = 1, i \in K, j = 1, \ldots, C + 1\}.
\]

Essentially, \(P_X\) consists of the pairs (content, cache) of all the cached contents. Since, by definition, the large cache \(C_0\) contains the entire catalogue (i.e., \(x_{i0} = 1\), for all \(i \in K\)), \(X\) is a \(|K| \times (C + 1)\) matrix. In other words, \(P_X\) belongs to the set \(P := P(K \times \{1, \ldots, C + 1\})\), where \(P(K \times \{1, \ldots, C + 1\})\) is the powerset of \(K \times \{1, \ldots, C + 1\}\). Inversely, given a placement \(P\), we can define the corresponding matrix \(P_X\) such that \(x_{ij}\) is equal to 1, for every pair \((i, j)\) in \(P\), and 0 otherwise.

**Algorithm 1:** Inner Algorithm (Subroutine)

| Input: \(U, K, N_u, X, \{\beta_u\}, \varphi, \{\alpha_u\}, \{r_{ui}\}, \{s_{ui}\}\) |
| --- |
| 1 Start with empty matrix \(Y\) |
| for every user \(u \in U\) do |
| for every content \(i \in K\) do |
| Calculate \(V_{ui}\); |
| Store \(\{V_{ui}[k]\}_{k=1}^{N_u}\) (in decreasing order) ; |
| end |
| Set \(y_{ui}[k] = 1\) for \(k = 1, \ldots, N_u\); |
| end |
| Return \(Y\) |

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 quality \(s_{ui}\). In general, \(V_{ui}(X) \neq V_{ui}[k](X)\), for all \(u \in U\) and \(k = 1, \ldots, |K|\).
Hence, from now on, $X$ and $P$ will be used interchangeably to denote the subset of the allocation across the network of caches. We also denote the subsets of a placement $P$ representing the storage of the cache $c$: $P^{(m)} := \{(i, m) \in P\}$. We can thus extend $F_u^*, F^*$, $s_u$ and $V_{ui}$ to the ground set $P$.

**Lemma 6**: The set function $F_u^*$ is monotone increasing for all $u \in \mathcal{U}$.

*Proof*: We consider two cache placements $P$ and $Q$ such that $P \subseteq Q \subseteq \mathcal{P}$ and we will prove that $F_u^*(P) \leq F_u^*(Q)$. Since $P \subseteq Q$, the contents cached in $P$ are also available in $Q$ with the same or better streaming quality, i.e.,

$$s_u(P, i) \leq s_u(Q, i), \text{ for all } i \in \mathcal{K}. \quad (16)$$

This is easily proven by contradiction, assuming that there exist a content $\eta$ such that $s_u(P, \eta) > s_u(Q, \eta)$.

Next, by Definition 6, the following inequalities are true

$$V_{ui}(P) \leq V_{ui}(Q), \quad (17)$$

$$V_{uk}(P) \leq V_{uk}(Q), \text{ for all } i, k \in \mathcal{K}. \quad (18)$$

Finally, it follows by (12) that $F^*_u(P) \leq F^*_u(Q)$.

**Lemma 7**: The set function $F_u^*$ is submodular for all $u \in \mathcal{U}$.

We consider two cache placements $A$ and $B$ such that $A \subseteq B \subseteq \mathcal{P}$ and $(i, j) \in \mathcal{P} \setminus B$. We need to prove that

$$\Delta F_u^*(A, (i, j)) \geq \Delta F_u^*(B, (i, j)). \quad (19)$$

In other words, the marginal benefit of adding content $i$ to the cache $j$ in $A$ is greater than or equal to the marginal benefit in $B$. This means that the function $F_u^*$ has the diminishing returns property.

In order to prove Lemma 7, we need a few intermediate results. All the results and proofs given here are true for any $u \in \mathcal{U}$, so for simplicity, the index $u$ will be omitted throughout the two following lemmas and the proof of Lemma 7.

**Lemma 8**: Given $P$, a placement of cached items, and a pair $(i, j) \in (K \times \{1, \ldots, C\})$ such that $(i, j) \notin P$, we write $P' := P \cup \{(i, j)\}$. Then, $s(P', i) = \max\{s_j, s(P, i)\}$.

*Proof*: By (3), after adding $(i, j)$ to the placement, if $s_j$ is greater than $s(P, i)$, then $i$ will be retrieved from cache $j$, i.e., $s(P', i) = s_j$. On the other hand, if cache $j$ does not offer a better quality for the user than before, the quality associated to $i$ will stay the same, i.e., $s(P', i) = s(P, i)$.

**Lemma 9**: Given $P$, a placement of cached items, and a pair $(i, j) \in (K \times \{1, \ldots, C\})$ such that $(i, j) \notin P$, the following statements are true:

a. $\Delta F^*(P, (i, j)) = 0$ if and only if $s_j \leq s(P, i)$;

b. $\Delta F^*(P, (i, j)) > 0$ if and only if $s_j > s(P, i)$.

In the second case, the marginal gain is equal to

$$\Delta F^*(P, (i, j)) = (1 - \alpha)p_i(s_j - s(P, i))$$

$$+ \begin{cases} 0, & \text{if } V_i(P') \leq V_{[N]}(P), \\ V_i(P') - M, & \text{if } V_i(P') > V_{[N]}(P), \end{cases}$$

where $P' = P \cup \{(i, j)\}$ and $M = \max\{V_i(P), V_{[N]}(P)\}$.

Essentially, Lemma 9 states that adding $(i, j)$ to $P$ will lead to a positive marginal gain of $F^*$ if and only if the cache $j$ can provide a higher (cache-specific) quality for the user than any other cache where $i$ was already cached (in $P$).

*Proof*: By Lemma 5, $\Delta F^*(P, (i, j)) > 0$ if and only if

$$\sum_{k=1}^{N} \left(V_k(P') - V_k(P)\right) > 0 \quad (20)$$

or

$$\sum_{k \in \mathcal{K}} (s(P', k) - s(P, k)) > 0. \quad (21)$$

Given that the only difference between $P$ and $P'$ is the content $i$ in cache $j$, the qualities of the contents other than $i$ remain the same as the addition of $(i, j)$ does not affect them. As a result, the inequality in (21) is true if and only if $s(P', i) > s(P, i)$. By Lemma 8, $s(P', i) = s_j$ and, therefore, (21) is equivalent to $s_j > s(P, i)$. Next, when the inequality (20) holds, $V_i(P') > V_i(P)$ because, otherwise, $V_k(P') = V_k(P)$, for every $k \in \mathcal{K}$. By (13), the inequality $V_i(P') > V_i(P)$ is equivalent to $s(P', i) = s_j > s(P, i)$. Hence, we proved that the inequality in (20) implies $s_j > s(P, i)$, which is equivalent to (21). Therefore, statement (b) holds. Since $\Delta F^*(P, (i, j)) \geq 0$ (by Corollary 1), it follows that $\Delta F^*(P, (i, j)) = 0$ if and only if $s_j \leq s(P, i)$.

Next, we calculate $\Delta F^*(P, (i, j))$ when $s_j > s(P, i)$. First, note that, by (14), the expression $F^*(P') - F^*(P)$ consists of two summands. The summand with coefficient $(1 - \alpha)p_i$ is equal to the summand with coefficient $(1 - \alpha)p_i$ for all $k \in \mathcal{K}$.

In order to calculate the other summand, we compare $V_i(P')$ with the $V$-values of the recommended items in $P$ before adding $(i, j)$, i.e., the values $V_k(P)$ for $k = 1, \ldots, N$.

If $V_i(P') < V_{[N]}(P)$, content $i$ will not feature in the recommendations list after caching it in $j$. If $V_i(P') = V_{[N]}(P)$, then content $i$ may make it to the recommendations list by replacing the $[N]$-th item in the list, assuming that ties are broken arbitrarily in the selection process. In both cases, nothing changes in terms of the $[N]$ highest $V$-values in $P'$, which implies that $\sum_{k=1}^{N} (V_k(P') - V_k(P)) = 0$.

On the other hand, if $V_i(P') > V_{[N]}(P)$, content $i$ will definitely feature in the recommendations list after adding it in $j$, which implies (20). We consider two subcases:

- $i$ was already among the recommendations in $P$ even before caching it in $j$, i.e., $V_i(P) \geq V_{[N]}(P)$. In this case, since the streaming quality is better at $j$, a part of the marginal gain will come from the difference in $V$-value of $i$ before and after adding $(i, j)$. This means that $\sum_{k=1}^{N} (V_k(P') - V_k(P)) = V_i(P') - V_i(P)$.
- $i$ was not recommended before caching it in $j$, i.e., $V_i(P) < V_{[N]}(P)$. Since $V_i(P') > V_{[N]}(P) > V_i(P)$, content $i$ gets into the recommendations list by replacing...
the $N$-th recommendation. Hence, a part of the marginal gain will come from the difference of the new V-value of $i$ and the V-value of the $[N]$-th item in $P$, i.e.,

$$\sum_{k=1}^{N} (V[k]_P - V[k](P)) = V(P') - V[N](P).$$

Then, the result follows by replacing the findings above in the expression $F^*(P') - F^*(P)$.

We can now prove Lemma 7.

**Proof of Lemma 7:** For two placements $A$ and $B$ such that $A \subseteq B \subseteq P$ and a pair $(i,j) \in P \setminus B$, we need to prove (19).

As before, $A'$ and $B'$ are the sets $A \cup (i,j)$ and $B \cup (i,j)$ respectively. Since $A \subseteq B$, eq. (16) (Lemma 6) implies that

$$s(A,i) \leq s(B,i). \quad (22)$$

In line with Lemma 9, we examine the following cases:

- $\Delta F^*(A(i,j)) = 0$; $\Delta F^*(A(i,j)) > 0$. The first case is equivalent to $s_j \leq s(A,i)$, by Lemma 9. Then, by (22), $s_j \leq s(B,i)$. We invoke once again Lemma 9 and we get that $\Delta F^*(B(i,j)) = \Delta F^*(A(i,j)) = 0$.

Concerning the second case, it is equivalent to $s_j > s(A,i)$ and $\Delta F^*(A(i,j))$ is given by the formula in Lemma 9. We consider three subcases:

1. $s_j \leq s(B,i)$;
2. $s_j > s(B,i)$ and $V_i(B') \leq V[N](B)$;
3. $s_j > s(B,i)$ and $V_i(B') > V[N](B)$.

In the first subcase, $\Delta F^*(B(i,j)) = 0$, by Lemma 9 and, therefore, $\Delta F^*(A(i,j)) = \Delta F^*(B(i,j)) = 0$.

Next, $s_j > s(B,i)$ is equivalent to $\Delta F^*(B(i,j)) > 0$. Since $s_j > s(A,i)$ as well, it holds that

$$s_j = s(A',i) \leq s(B',i). \quad (23)$$

If $V_i(B') \leq V[N](B)$, Lemma 9, (22) and (23) imply that

$$\Delta F^*(B(i,j)) = (1 - \alpha) p_i (s_j - s(B,i)) \leq \Delta F^*(A(i,j)).$$

If $V_i(B') > V[N](B)$, by (22), (23) and (18), it follows that

$$V_i(A') = V_i(B') > V[N](B) \geq V[N](A). \quad (24)$$

Combining this with (22) and Lemma 9, in order to prove $\Delta F^*(A(i,j)) \geq \Delta F^*(B(i,j))$, we only need to prove that

$$\max\{V_i(A), V[N](A)\} \leq \max\{V_i(B), V[N](B)\}. \quad (25)$$

It follows by (22) that $V_i(A) \leq V_i(B)$, and therefore $V_i(A) \leq \max\{V_i(B), V[N](B)\}$. Moreover, $V[N](A) \leq \max\{V_i(B), V[N](B)\}$, by (24). We then obtain (25) and this concludes the proof.

**Lemma 10:** The set function $f^*$, as defined in (10), is monotone increasing and submodular.

**Proof:** By Lemma 4, $f^*(X) = \sum_{u \in U} F_u^*(X)$. It is easy to prove that monotonicity and submodularity are preserved under non-negative linear combinations. Therefore, the result is an immediate consequence of Lemmas 6 and 7.

**C. MoSE Algorithms and Guarantees**

We managed to prove through the decomposition in (11) that $f^*(X)$ is submodular. The theory on submodularity optimization suggests that different greedy algorithm variants give constant approximations for the outer problem, and thus for the MoSE problem. In fact, the factor of approximation depends on the constraints in (7).

**Algorithm 2: MoSE Algorithm (Equal-Sized Contents)**

**Input:** $C$, $\{C_j\}$, $U$, $K$, $\{X_u\}$, $\{s_u\}$, $\{r_u\}$, $\{\beta_u\}$, $\{\alpha_u\}$

1. Start with empty caches, i.e., $P = \bigcup_{j=1}^{C} P[j]$, where $P(j) = \emptyset$, for all $j = 1, \ldots, C$.

2. **Outer algorithm:**

3. **while** caches are not full, i.e., $|P[j]| < C_j$ for all $j$, **do**
4. **for** each (not full) cache $j = 1, \ldots, C$, **do**
5. **for** every content $i \in K$ s.t. $(i,j) \notin P[j]$, **do**
6. Estimate $\Delta f^*(P(i,j))$ by calling Inner Algorithm($X$):
7. **Store max $\Delta f^*(P(i,j))$.**
8. **end**
9. **end**
10. $(\eta, \theta) = \arg \max_{(i,j)} \Delta f^*(P(i,j)).$
11. Add $(\eta, \theta)$ to $P$, i.e., $P(\theta) \leftarrow P(\theta) \cup (\eta, \theta)$.
12. **end**
13. Return $X^* \leftrightarrow P$, $Y^* = f^*(X^*)$.

1) **The Case of Equal-Sized Contents:** We define a greedy algorithm that we call the MoSE 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 $C \cdot |K|$ elements, i.e., pairs (content, cache), by solving the Inner Algorithm (as subroutine). It then selects and adds to $P$ the element that maximizes the marginal gain (ties broken arbitrarily), before the next selection round begins. The algorithm is summarized in Algorithm 2.

**Theorem 1 (Homogeneous Sizes):** If we let $OPT$ denote the optimal objective function value of the MoSE problem with equal-sized contents, and $(X^*, Y^*)$ denote the feasible solution given by the MoSE algorithm, then

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

**Proof:** Since the constraints in (7) are matroid constraints, as in [6], the theory on submodular maximization [36] suggests that a $1/2$-approximation is achievable by the above greedy algorithm.

2) **The General Case of Contents of Heterogeneous Sizes:** The fundamental difference between the two cases is the capacity constraints. The constraints in (7) in the general case are knapsack constraints. However, the MoSE algorithm is oblivious of the content’s size. The following algorithm is an adaptation of the MoSE algorithm that takes size into account. More precisely, 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, while satisfying the constraints in (7). It is summarized in Algorithm 3.

**Theorem 2 (Heterogeneous Sizes):** If we let $OPT$ denote the optimal objective function value of the MoSE problem in the general case (contents of heterogeneous sizes), and $(X^*, Y^*)$, $(X_s, Y_s)$ denote the feasible solutions given by the
MoSE and s-MoSE algorithms respectively, then
\[
\max\{f(X^*, Y^*), f(X_1, Y_1)\} \geq 1 - \frac{1}{e} OPT_s.
\]

Proof: In the case of variable-sized contents, both MoSE and s-MoSE algorithms can perform arbitrarily badly [37]. According to the result in [37], it suffices to choose the maximum objective function value achieved by the two algorithms in order to achieve a \(1 - \frac{1}{e}\)-approximation. \qed

3) Complexity, Implementation Speed-Ups and Distributed Techniques: It is easy to see that the complexity of both the MoSE and the s-MoSE algorithms is the same. The algorithms need to run at most \(\sum_{j=1}^{\beta} |C_j|\) times in order to fill all caches. At each iteration, they evaluate the marginal gain of at most \(C \cdot |K|\) pairs (content, cache). For every evaluation, they call the Inner Algorithm of complexity \(O(|U| \cdot |K|)\) and complete \(|U|\) calculations that concern the non-recommendation part of the objective function. Therefore, the total complexity of the MoSE and s-MoSE algorithms is \(O(|U| \cdot |K|^2 \cdot C \cdot \sum_{j=1}^{\beta} |C_j|)\).

Implementation-wise, there is a way to speed up both algorithms by using the so-called lazy evaluations method [37]. This method takes advantage of the monotonicity and submodularity of the objective function in order to avoid unnecessary calculations in the selection process of the caching placement. Recent works propose methods for further acceleration, e.g., randomized greedy algorithm in [38].

Finally, we note that there is a technique suggested in the literature for distributed/multi-processor implementations of greedy algorithms of submodular maximization [39], such as the MoSE algorithm. More precisely, for a set of \(m\) processors/nodes, this technique starts by partitioning the ground set, i.e., the catalogue of contents, into \(m\) subsets. Then each processor solves in parallel the MoSE problem only on one of the subsets by applying our proposed policy. This leads to \(m\) solutions (caching allocation and users' recommendations) \(DS_1, \ldots, DS_m\). Next, we define a new subset of the catalogue by merging the contents for which the caching variable was equal to 1 in at least one of the previous solutions. We then run on this subset the MoSE algorithm which gives the solution \(MS\).

D. The Single-Cache Case (\(C = 1\))

We study now the case where \(C = 1\), i.e., apart from the large cache \(C_0\), there is only one cache. We prove that, in this case, the MoSE problem can be transformed into an Integer Linear Program (ILP) problem and, thus, 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 will allow us to compare the performance of our algorithm with the optimal joint policy.

We introduce the variable \(\{z_{ui}\}_{i,u}\) such that \(z_{ui} = x_i y_{ui}\). The objective of the MoSE problem in (7) becomes
\[
g(X, Y, Z) = \sum_{u \in U} \sum_{i \in K} \left[ \frac{\alpha_i}{N} ((s_{u1} - s_u) z_{ui} + s_u y_{ui}) + (1 - \alpha_i) p_{ui}((s_{u1} - s_u) x_i + s_u) + \beta_u y_{ui} r_{ui} \right].
\]

Therefore, the MoSE problem for \(C = 1\) is equivalent to:

**Z Problem**

\[
\begin{align*}
\text{maximize} & \quad g(X, Y, Z) \\
\text{subject to} & \quad (7), (8), \quad z_{ui} = x_i y_{ui}; \quad (28) \\
& \quad x_{ij}, y_{ui}, z_{ui} \in \{0, 1\}. \quad (29)
\end{align*}
\]

The equivalence comes from the fact that a pair \((\tilde{X}, \tilde{Y})\), where \(\tilde{X} = \{\tilde{x}_i\}\) and \(\tilde{Y} = \{\tilde{y}_{ui}\}\), is optimal for the MoSE problem if and only if \((\tilde{X}, \tilde{Y}, Z)\) where \(Z = \{\tilde{z}_{ui}\}\) such that \(\tilde{z}_{ui} = \tilde{x}_i \tilde{y}_{ui}\) is optimal for the Z problem.

Notice that, although \(g(X, Y, Z)\) is linear in the variables \(X, Y\) and \(Z\), the constraints (28) are nonlinear. However, we will prove that these constraints can be replaced by the following inequalities:
\[
\begin{align*}
z_{ui} &\leq x_i, \quad (30) \\
z_{ui} &\leq y_{ui}, \quad \text{for all } u \in U, i \in K. \quad (31)
\end{align*}
\]

**Lemma 11**: The MoSE problem for \(C = 1\) is equivalent to the following ILP problem:

**MoSE ILP problem**

\[
\begin{align*}
\text{maximize} & \quad g(X, Y, Z) \\
\text{subject to} & \quad (7), (8), (29) - (31).
\end{align*}
\]

Proof: It suffices to prove that a solution for the Z problem is also a solution for the MoSE ILP problem and the inverse. Let us assume that \((\bar{X}, \bar{Y}, \bar{Z})\) is a solution for the Z problem. Since \(\bar{X}\) and \(\bar{Y}\) are binary variables, the expression \(\bar{z}_{ui} = \bar{x}_i \bar{y}_{ui}\) for all
\( \bar{x}_i \bar{y}_{ui} \) implies the inequalities (30) and (31). Hence, \((\bar{X}, \bar{Y}, \bar{Z})\) is also a solution for the MoSE ILP problem.

Inversely, let us assume that \((\bar{X}, \bar{Y}, \bar{Z})\) is a solution for the MoSE ILP problem. It suffices to prove that \( \tilde{z}_{ui} = \bar{x}_i \tilde{y}_{ui} \) for every \( u \in U \) and \( i \in K \). For the \( u \) and \( i \) such that \( \bar{x}_i = 0 \) or \( \tilde{y}_{ui} = 0 \), the inequality constraints imply that \( \tilde{z}_{ui} = 0 \). For the \( u \) and \( i \) such that \( \bar{x}_i = 1 \) and \( \tilde{y}_{ui} = 1 \), we will necessarily have that \( \tilde{z}_{ui} = 1 \) since the coefficient of \( \bar{z}_{ui} \) in the objective function \( g \) is strictly positive in a maximization problem. Hence, considering that all variables are binary, it follows that \( \tilde{z}_{ui} = \bar{x}_i \tilde{y}_{ui} \), and this concludes the proof.

**IV. PERFORMANCE EVALUATION**

In this section, we validate the theoretical approximation guarantees of the proposed policy (MoSE algorithm) and we compare it with other policies in a variety of scenarios.

**A. Scenario 1**

As a first step, we compare the performance of the MoSE algorithm with its distributed implementations 2-DMoSE and 4-DMoSE (i.e., in 2 and 4 processors, see Section III-C3) and with the optimal policy (oracle). We consider a scenario with a single cache and the large cache \( C_0 \) that contains the entire catalogue. As shown in Section III-D, the MoSE problem for \( C = 1 \) can be transformed into an ILP problem. Therefore, in order to find the optimal policy (oracle), we use the standard MATLAB solver which employs methods such as branch-and-bound, cutting-plane method or exhaustive search.

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 \( \alpha_u \), whose values follow a uniform distribution between 0.7 and 0.9 (in line with the statistics gathered on Netflix [2]). 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 \( i \in K \), as in (1).

In this scenario, we measure the SQ as cache hits with the values \( s_{ui} \) as in (5), and the RQ (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 MoSE given by the proposed algorithm and its distributed implementations. For some of the values \( \beta \), Table IV shows the approximation ratio achieved and Table V shows the average execution time per instance of the problem.

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, as is also observed for other submodular problems, e.g., in [40]. In fact, among all the different values of \( \beta \) we considered (30 in total), the lowest observed approximation ratio was equal to 0.9757. Moreover, the approximation ratios achieved by the distributed algorithms 2-DMoSE and 4-DMoSE are also close to 1. However, as expected by the discussion in Section III-C3, they do not perform as well as the (centralized) MoSE algorithm for some instances of the problem (e.g., for \( \beta = 3.2 \)).

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

Regarding Table V, we see that the average execution time of the MoSE algorithm is of much lesser magnitude than the oracle’s one. We note that we implemented MoSE with the lazy evaluations technique which avoids unnecessary calculations (see Section III-C3). When MoSE is implemented in 2 and 4 processors, the execution time decreases significantly, and we observe a 2x speedup.

**Observation 2:** Implementing the MoSE algorithm leads to significant savings in execution time when compared with the oracle. These savings can be further pronounced in the case of a distributed/multi-processor implementation.

Next, we investigate if this close-to-optimal performance is reflected in the SQ-RQ 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 [11].

**Cache-Aware Recommendations (CAwR):** CAwR [11] 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 \( \alpha_u \) of our model), the aggregate request probability of every content is calculated. Then, the \( N \) 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 \( \beta \) of our model.

Figure 3 depicts the SQ-RQ tradeoffs given by the oracle, our policy, and CAwR as points in the plane. We obtained these tradeoffs for 30 different values of \( \beta_u = \beta \) in the range...
Fig. 3. Scenario 1, SQ-RQ tradeoff points for some values of the parameters \( \beta \) and \( r_d \).

[0.01, 70] and the distortion parameter \( r_d \). The RQ values (x-axis) are normalized with respect to the two “extreme” policies A and C (defined in Section II-F). For example, \( RQ = 50\% \) implies that the RQ value lies in the middle of the interval \([R_A, R_C]\), where \( R_A \) and \( R_C \) are the RQ values achieved by policies A and C respectively. Moreover, since \( RQ \) are as in (5), the normalized SQ values (y-axis) give the cache hit rate.

We remind the reader that each of these points corresponds to a different objective tradeoff, between SQ and RQ, that a CP might have, i.e., these curves could also be interpreted as Pareto curves. As we discussed in Section II-E, \( \beta \) captures the weight we attach to the RQ compared to the SQ. For a small value of \( \beta \) (on the left), caching decisions are based on the aggregate (over all users) interest in contents and recommendations concern mostly cached items. This leads to high SQ/cache hits but compromised RQ. As \( \beta \) increases, we trade off a better RQ for a worse SQ. In fact, a better RQ would imply recommendations to each user that are close to her tastes and it is \( \beta \) that determines how close. As these tastes can differ from one user to the other and the cache capacity is limited, caches cannot store all the different recommended contents and this leads to decreased SQ/cache hits.

**Observation 3:** Our policy’s tradeoff curve almost coincides with the optimal. Furthermore, it dominates the tradeoff curve of CAwR, i.e., our policy outperforms CAwR in terms of at least SQ or RQ (or both).

For example, for a desired value of SQ of around 84%, CAWR achieves 20% RQ and our policy 68%. More importantly, most of the tradeoffs of our policy (e.g., around 80 – 95% RQ and 70 – 80% SQ) are not achievable by any tuning of the CAwR algorithm. Finally, we observe that for large \( \beta \) and small \( r_d \) (points in the extreme right) the recommendation and caching decisions of the two policies coincide. In fact, both policies recommend to each user the contents with the highest utility for the user and both policies store the contents with the highest aggregate probability to be requested (given the aforementioned recommendations).

B. Scenario 2

We proceed with simulating larger scenarios. For this, we consider a single cache with 100 or 200 connected users and a catalogue consisting of 6000 or 10000 contents. We consider realistic values (according to footnote 1, p. 1827) for cache capacity varying from 1% to 2.3% of the entire catalogue. The probabilities \( \alpha_u \) are chosen randomly in \([0.7, 0.9]\), in line with the statistics gathered on Netflix [2], and \( N \) varying from 2 to 10. For these experiments, we use a real dataset for the matrix of utilities \( r_{ui} \).

**MovieLens Dataset:** The MovieLens dataset [42] is a collection of 5-star movie ratings collected on MovieLens, an online movie recommendation service. This dataset has also been used in related works on caching and recommendations, e.g., in [10]. Here, we used a variety of subsets of the total 20000263 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 [26]. 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 \((r/5 \cdot 0.1, r/5)\). As is common, this matrix is quite sparse. To obtain the missing ratings, we perform matrix completion through the TFOCS software [43]. TFOCS performs nuclear norm minimization in order to find the missing entries of a low-rank matrix.

1) **Equal-Sized Contents:** We assume that the contents are of unit size. We will show the performance improvement achieved by our policy over a baseline scheme, policy \( \gamma \), and the earlier introduced CAwR. To begin, we define policy \( \gamma \).

**Baseline Policy \( \gamma \):** It is a generalization of policies A and C. Policy \( \gamma \) caches the most popular contents and then recommends a combination of cached contents and contents with high utility per user depending on the parameter \( \gamma \). More specifically, it recommends \([\gamma \cdot N]\) cached contents, where \( \lceil \cdot \rceil \) denotes the ceiling function, while the rest of the recommendations are the contents with the highest utility per user. For \( \gamma = 0 \), policy \( \gamma \) coincides with policy C, and, for \( \gamma = 1 \), it coincides with policy A.

As before, we measure the SQ as cache hits and the RQ as \( \sum_i \log(r_{ui}) \). In Figure 4(a), we plot the tradeoffs achieved by policy \( \gamma \), CAwR, and our policy for different values of the parameters \( \gamma \), \( r_d \), and \( \beta \) respectively. In this instance, \( N = 2 \), which results in 3 possible objective values for policy \( \gamma \). In fact, one point corresponds to recommending 2 cached items, the next one to recommending one among the cached items and the one with the highest utility, and the last point to recommending the 2 first contents ranked in terms of utility.

**Observation 4:** The SQ-RQ tradeoff curve of our policy dominates that of CAwR and that of the baseline policy \( \gamma \) in large, realistic scenarios, driven by real datasets.

We notice, for example, that, in terms of SQ, there is a relative improvement of up to 10% with respect to CAwR and of up to 54% with respect to policy \( \gamma \), while the improvement is much larger in terms of RQ. This is an encouraging finding that suggests that the theoretical gains could also be experienced in practice. Finally, note that the performance gain of our policy over policy \( \gamma \) is mainly due to the joint decisions on caching and recommendations that our policy makes.

\(^9\)Note that according to [41], the total number of titles (movies and TV shows) available on Netflix in the USA is equal to 5848.
2) Contents With Heterogeneous Sizes: So far, we have considered scenarios with equal-sized content (e.g., chunks), as is often assumed in related work [6]. Here, we turn our attention to a scenario with contents of heterogeneous size, as analyzed in Section III-C2. The sizes of the contents were chosen in \{1, 15\} and, according to the findings of [44] on YouTube videos, 90% of the contents have a size of at most 2 size units, while only 0.1% have a size over 10 size units. We adjust the cache capacity to 2.3% of the total size of the catalogue \(\sum_{i \in K} s_i\). Figure 4(b) depicts the tradeoffs achieved by the two policies. In this context, our policy runs both the MoSE and s-MoSE algorithms and selects the maximum achieved objective function value between the two, as explained in Section III-C. As expected, the difference between the tradeoff curves is similar to the one in Fig. 4(a). More specifically, we observe a relative gain of up to 63% in RQ and up to 15% in SQ of our policy with respect to CAwR.

Observation 5: Heterogeneous content sizes do not have an impact on the performance gains of our policy which, in this context, still outperforms existing schemes.

3) RQ-Related Constraints: While the previous results are promising, one might argue that the proposed policy could still recommend some rather unrelated contents, i.e., contents of utility \(r_{uj}\) close to 0, in favor of a higher objective value, or worse, that some users might receive much better recommendations, i.e., tailored to their tastes, than others. For this reason, we will evaluate the performance of our policy and that of existing schemes when additional constraints on RQ are added to the problem. In particular, we measure RQ by considering \(\varphi\) as in (2). This leads to recommendations of contents whose utility per user is at least \(r_{min}\). Since the recommendation decisions are made by solving the “inner problem” (as explained in Section III-B), the caching decisions also take into account this constraint. Subsequently, we adjust CAwR such that, at the recommendation step, the contents with \(r_{ui} < r_{min}\) cannot be recommended to the user.

Figure 4(c) demonstrates that, for values of \(\beta\) close to 0 and values of \(r_d\) close to 1, the performance in SQ for both policies naturally drops when \(r_{min} = 0.6\) in comparison to the performance when \(r_{min} = 0\). This is because fewer contents can be recommended per user and these can largely differ from one user to the next. Therefore, only a few of them can be cached due to the limited cache capacity, and less cache hits will occur. In fact, in the dataset used for this experiment, on average, for every user, only 3% of the catalogue is of utility greater than or equal to 0.6.

Observation 6: Our policy does not choose to radically compromise RQ, leading to similar performance trade-offs even when additional strict constraints on RQ are imposed.

We notice that the tradeoff points of our policy for \(r_{min} = 0\) and \(r_{min} = 0.6\) coincide for most of the values of \(\beta\), while the maximum observed gain of the latter over the former in terms of RQ is 25%. Finally, we observe that even the constraint version of our (close-to-optimal) policy is still able to outperform CAwR with looser constraints on RQ.

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 [6] 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 MoSE, of our policy and some non-joint policies whose caching decisions are made according to [6].

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 [6], [10]), 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. In this scenario, we will measure the SQ as a function of the estimated bitrate. More precisely, we assume that \(s_{uj} = \psi(b_{uj})\), where \(b_{uj}\) are the estimated bitrate that can be supported between user \(u\) and cache \(j\) and \(\psi\) is an increasing function of \(b_{uj}\). Without loss of generality, we assume that the rate from the large cache (or macro-cell cache) \(C_0\) is 0.5 Mbps, while the \(b_{uj}\) values for edge caches are chosen randomly between...
2 and 15 Mbps. In fact, the required Internet connection speed on YouTube [45] 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 and α_u and ϕ, as in Scenario 2. We set the helper’s capacity to 1.5% of the catalogue size and N = 5. We will compare the performance of our policy, in terms of the MoSE, with two policies that are based on the algorithm proposed in [6]:

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 [6] 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.

For ψ being the identity function, i.e., s_{ui} = b_{ui} and for different values of β > 0, the achieved MoSE of our policy, the A-femto, and the C-femto policies are shown in Fig. 5(a). We observe that, for β close to 0, i.e., priority is given to the SQ, 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 MoSE achieved by the C-femto policy is lower since, although it provides the best RQ, 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 SQ. In fact, this is illustrated in small scale in the toy example in Section II-F. As β increases, the priority moves towards RQ, 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. Furthermore, the performance gap between our policy and the C-femto policy remains constant.

Observation 7: The performance gains of the proposed policy over non-joint policies are prominent in generic networks of caches as well.

In the cases studied above, we have considered the SQ and RQ functions (i.e., ψ and ϕ respectively) being the identity or the function in (5) and the logarithmic function respectively. One might wonder how these choices affect the performance of our policy. For this reason, we explore their impact here. We ran on the same dataset as above the experiment for: i) both ψ and ϕ being linear functions (Fig. 5(b)), and ii) both ψ and ϕ being logarithmic functions (Fig. 5(c)). We remind the reader that the rationale for the logarithmic function has been elaborated in Section II-E. We notice that the relative improvement in performance of the proposed policy in comparison to the A-femto and C-femto policies are similar for the different choices of functions considered (Fig. 5(a)–(c)). We note that a variety of coefficients of these functions have been considered in every case. Furthermore, we observe that even though the range of the y-axis varies in Fig. 5(a)–(c), the relative gains are similar and the preceding analysis on the relative performance of the three policies holds in every case.

Observation 8: The performance improvements are consistent for different choices of SQ and RQ functions.

V. RELATED WORK

Hierarchical Caching: Optimization of hierarchical caching (e.g., CDNs or ICNs) has been widely explored both in the context of wired [24] and wireless networks [6]. Various aspects of this problem have been explored such as caching for different video streaming qualities [23] etc. See, for example, a recent survey on caching in [7]. Nevertheless, these works are oblivious to the impact of the recommendations, beyond the simple (usually IRM) popularity model used as input.

Caching-Recommendation Interplay: In an early work in this direction [46], the authors propose heuristic algorithms for recommendations in P2P networks that take into account both service cost and user preferences. In [9], the authors propose a recommendation algorithm that tries to bias requests towards cached contents. In a similar spirit, [8] proposes a reordering of the videos appearing in YouTube’s related videos section by “pushing” on top of the list the cached items. However, the caching policy in these works is fixed. Considering now different setups, [10] 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. Although the caching policies in [10] are recommendation-aware, the recommender comes after the caching decisions. A decomposition algorithm for the joint problem is proposed in [11] 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. Similarly, [12] proposes a decomposition heuristic for the joint caching and recommendation problem. In [13], the authors adopt a different approach and employ machine learning techniques to devise caching and recommendation policies. Finally, the authors in [14] formulate a joint problem in the somewhat different context of prefetching content over a time-varying channel.

**Joint Optimization Theory:** Submodularity-based proofs for caching-related problems have flourished since the seminal paper of [6], where the focus is on one set of variables (caching). The decomposition and submodularity method we use is similar in spirit to the methods in [47] and [40]. While the former studies quite a different problem than ours, the latter proposes an approximation algorithm for the joint caching and routing problem in cache networks.

**VI. CONCLUSION**

In this paper, we studied the problem of jointly making caching and recommendation decisions in a generic caching network. This is a problem of great interest as entities like Netflix can now manage both caching and recommendations in their network. To this end, we introduced a metric of user’s streaming experience (MoSE) as a balanced sum of SQ (affected by the caching allocation) and RQ (determined by the recommendations the user receives) and we formulated the problem of maximizing users’ MoSE. This formulation captures the user’s expectations for SQ and RQ from a recommendation-driven application, while, at the same time, allows us to explore the underlying SQ-RQ tradeoffs of the problem. Moreover, the model we considered is generic since SQ can be replaced by any caching gain/profit. We proposed a polynomial-time algorithm that has $\frac{1}{2}$ approximation guarantees (or $1 - \frac{1}{e}$ in the case of contents of heterogeneous size). Our numerical results in realistic scenarios show important performance gains of our algorithm with respect to baseline schemes and existing heuristics. An interesting direction for future work is to consider transmission capacities for the caches and introduce request routing as a variable of the problem.

**REFERENCES**

[1] D. Tsigkari and T. Spyropoulos, “User-centric optimization of caching and recommendations in edge cache networks,” in Proc. IEEE WoWMoM, Cork, Ireland, 2020, pp. 244–253.
[2] C. A. Gomez-Uribe and N. Hunt, “The Netflix recommender system: Algorithms, business value, and innovation,” ACM Trans. Manag. Inf. Syst., vol. 6, no. 4, p. 13, 2016.
[3] H. Nam, K.-H. Kim, and H. Schulzrinne, “QoS matters more than QoE: Why people stop watching cat videos,” in Proc. IEEE INFOCOM, San Francisco, CA, USA, 2016, pp. 1–9.
[4] “Understanding the value of consistency in OTT video delivery (white paper),” Akamai, Cambridge, MA, USA, Rep., 2019. [Online]. Available: https://www.akamai.com/resources/white-paper-understanding-the-value-of-consistency-in-ott
[5] T. V. Doan, L. Pajevic, V. Bajpai, and J. Ott, “Tracing the path to YouTube: A quantification of path lengths and latencies toward content caches,” IEEE Commun. Mag., vol. 57, no. 1, pp. 80–86, Jan. 2019.
[6] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, “FemtoCaching: Wireless content delivery through distributed caching helpers,” IEEE Trans. Inf. Theory, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.
[7] G. S. Paschos, G. Iosifidis, M. Tao, D. Towsley, and G. Caire, “The role of caching in future communication systems and networks,” IEEE J. Sel. Areas Commun., vol. 36, no. 6, pp. 1111–1125, Jun. 2018.
[8] D. K. Krishnapa, M. Zink, C. Gwizdol, and P. Halvorsen, “Cache-centric video recommendation: An approach to improve the efficiency of YouTube caches,” ACM Trans. Multimedia Comput. Commun. Appl. (TOMM), vol. 11, no. 4, p. 48, 2015.
[9] T. Giannakas, P. Sermpezis, and T. Spyropoulos, “Show me the cache: Optimizing cache-friendly recommendations for sequential content access,” in Proc. IEEE WoWMoM, Chania, Greece, 2018, pp. 14–22.
[10] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, “Soft cache hits: Improving performance through recommendation and delivery of related content,” IEEE J. Sel. Areas Commun., vol. 36, no. 6, pp. 1300–1313, Jun. 2018.
[11] L. E. Chatzielethierou, M. Karaliopoulos, and I. Koutsopoulos, “Jointly optimizing content caching and recommendations in small cell networks,” IEEE Trans. Mobile Comput., vol. 18, no. 1, pp. 125–138, Jan. 2019.
[12] K. Qi, B. Chen, C. Yang, and S. Han, “Optimizing caching and recommendation towards user satisfaction,” in Proc. IEEE WCSP, Hangzhou, China, 2018, pp. 1–7.
[13] D. Liu and C. Yang, “A deep reinforcement learning approach to proactive content pushing and recommendation for mobile users,” IEEE Access, vol. 7, pp. 83120–83136, 2019.
[14] Z. Lin and W. Chen, “Joint pushing and recommendation for susceptible users with time-varying connectivity,” in Proc. IEEE GLOBECOM, Abu Dhabi, UAE, 2018, pp. 1–6.
[15] S. Kastanakis, P. Sermpezis, V. Kotronis, D. S. Menasche, and T. Spyropoulos, “Network-aware recommendations in the wild: Methodology, realistic evaluations, experiments,” IEEE Trans. Mobile Comput., early access, Dec. 4, 2020, doi: 10.1109/TMC.2020.3042606.
[16] “Open Connect Overview.” Netflix. [Online]. Available: https://openconnect.netflix.com/Open-Connect-Overview.pdf (accessed Mar. 10, 2021).
[17] “Open Connect Fill Patterns.” Netflix. 2020. [Online]. Available: https://openconnect.zendesk.com/hc/en-us/articles/360035618011-Fill-patterns
[18] M. Leconte, G. Paschos, L. Gkatziakis, M. Draief, S. Vassilaras, and S. Chouvardas, “Placing dynamic content in caches with small population,” in Proc. IEEE INFOCOM, San Francisco, CA, USA, 2016, pp. 1–9.
[19] “Netflix and Fill.” Netflix Technology Blog. 2016. [Online]. Available: https://netflixtechblog.com/netflix-and-fill-c43a32b490c0
[20] “Downloads for You Takes On-the-Go to the Next Level.” Netflix. 2020. [Online]. Available: https://about.netflix.com/en/news/downloads-for-you-takes-on-the-go-to-the-next-level
[21] G. S. Paschos, E. Bastug, I. Land, G. Caire, and M. Debbah, “Wireless caching: Technical misconceptions and business barriers,” IEEE Commun. Mag., vol. 54, no. 8, pp. 16–22, Aug. 2016.
[22] “Open Connect Appliances.” Netflix. 2020. [Online]. Available: https://openconnect.netflix.com/en/appliances/
[23] K. Poularakis, G. Iosifidis, A. Argyriou, and L. Tassiulas, “Video delivery over heterogeneous cellular networks: Optimizing cost and performance,” in Proc. IEEE INFOCOM, Toronto, ON, Canada, 2014, pp. 1078–1086.
[24] S. Borst, V. Gupta, and A. Walid, “Distributed caching algorithms for content distribution networks,” in Proc. IEEE INFOCOM, San Diego, CA, USA, 2010, pp. 1–9.
[25] G. Adomavicius and Y. Kwon, “Improving aggregate recommendation diversity using ranking-based techniques,” IEEE Trans. Knowl. Data Eng., vol. 24, no. 5, pp. 896–911, May 2012.
[26] X. Amatriain, “Building industrial-scale real-world recommender systems,” in Proc. ACM RecSys, 2012, pp. 7–8.
[27] M. Bressan, S. Leucci, A. Panconesi, P. Raghavan, and E. Torllo, “The limits of popularity-based recommendations, and the role of social ties,” in Proc. ACM SIGKDD, 2016, pp. 745–754.
[28] W. Li, P. Spachos, M. Chignell, A. Leon-Garcia, L. Zucherman, and J. Jiang, “Impact of technical and content quality on overall experience of OTT video,” in *Proc. IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, 2016, pp. 930–935.

[29] H. Batteram, G. Damm, A. Mukhopadhyay, L. Philippart, R. Odysses, and C. Urrutia-Valdés, “Delivering quality of experience in multimedia networks,” *Bell Labs Techn. J.*, vol. 15, no. 1, pp. 175–193, Jun. 2010.

[30] M. Fiedler, T. Hossfeld, and P. Tran-Gia, “A generic quantitative relationship between quality of experience and quality of service,” *IEEE Netw.*, vol. 24, no. 2, pp. 36–41, Mar./Apr. 2010.

[31] P. Reichl, S. Egger, R. Schatz, and A. D’Alconzo, “The logarithmic nature of QoE and the role of the Weber-Fechner law in QoE assessment,” in *Proc. IEEE ICC*, Cape Town, South Africa, 2010, pp. 1–5.

[32] L. Skorin-Kapov and M. Varela, “A multi-dimensional view of QoE: The ARCU model,” in *Proc. MIPRO*, Opatija, Croatia, 2012, pp. 662–666.

[33] S. Dehaene, “The neural basis of the Weber–Fechner law: A logarithmic mental number line,” *Trends Cogn. Sci.*, vol. 7, no. 4, pp. 145–147, 2003.

[34] S. Boyd, L. Xiao, A. Mutapcic, and J. Mattingley, “Notes on Decomposition Methods,” 2007. [Online]. Available: https://see.stanford.edu/materials/lsocoee364b/08-decomposition_notes.pdf

[35] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[36] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, “An analysis of approximations for maximizing submodular set functions—III,” in *Polyhedral Combinatorics*. Heidelberg, Germany: Springer, 1978, pp. 73–87. 

[37] 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.

[38] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrak, and A. Krause, “Lazier than lazy greedy,” in *Proc. AAAI Conf. Artif. Intell.*, vol. 29, 2015, pp. 1812–1818.

[39] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause, “Distributed submodular maximization,” *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 8330–8373, 2016.

[40] M. Delghan et al., “On the complexity of optimal request routing and content caching in heterogeneous cache networks,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1635–1648, Jun. 2017.

[41] “Netflix Museum.” Flixable. 2019. [Online]. Available: https://flixable.com/netflix-museum/

[42] F. M. Harper and J. A. Konstan, “The MovieLens datasets: History and context,” *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, p. 19, 2016.

[43] S. R. Becker, E. J. Candès, and M. C. Grant, “Templates for convex cone problems with applications to sparse signal recovery,” *Math. Program. Comput.*, vol. 3, no. 3, pp. 165–218, Jul. 2011.

[44] A. Abhari and M. Soraya, “Workload generation for YouTube,” *Multimedia Tools Appl.*, vol. 46, no. 1, pp. 91–118, 2010.

[45] “System Requirements.” YouTube Help. 2019. [Online]. Available: https://support.google.com/youtube/answer/78358?hl=en

[46] D. Munaro, C. Delgado, and D. S. Menasché, “Content recommendation and service costs in swarming systems,” in *Proc. IEEE ICC*, 2015, pp. 5878–5883.

[47] E. M. Craparo, J. P. How, and E. Modiano, “Throughput optimization in mobile backbone networks,” *IEEE Trans. Mobile Comput.*, vol. 10, no. 4, pp. 560–572, Apr. 2011.

Dimitra Tsigkari (Student Member, IEEE) received the degree in mathematics from the Aristotle University of Thessaloniki, Greece and the master’s degree in mathematics and applications from Pierre-et-Marie-Curie University (Paris 6), France. She is currently pursuing the Ph.D. degree with EURECOM, France. Her research interests include caching, recommendation systems, and network optimization.

Thrasyvoulos Spyropoulos received the Diploma degree in electrical and computer engineering from the National Technical University of Athens, Greece and the Ph.D. degree in electrical engineering from the University of Southern California. He was a Postdoctoral Researcher with INRIA and then a Senior Researcher with the Swiss Federal Institute of Technology Zurich. He is currently a Professor with EURECOM, Biot. He was a recipient of the best paper Award in IEEE SECON 2008 and IEEE WoWMoM 2012.