Caching and Recommendation Decisions at Transcoding-Enabled Base Stations

Dimitra Tsigkari* and Thrasyvoulos Spyropoulos†*
* EURECOM, Biot, France.
† Technical University of Crete, Chania, Greece.
Email: dimitra.tsigkari@eurecom.fr, spyropoulos@isc.tuc.gr

***Abstract*—In the context of on-demand video streaming services, both the caching and the recommendation decisions have an impact on the user satisfaction, and thus, financial implications for the Content Provider (CP). The idea of co-designing these decisions has been recently proposed in the literature as a way to minimize delivery costs and traffic at the backbone Internet. However, related work does not take into account that every content exists in multiple versions/streaming qualities, or at best treats each version as a separate content, when it comes to caching. In this paper, we explore how transcoding a content at the edge could avoid placing multiple related versions of this content in the same cache, thus better utilizing capacity (leading to an increase of the CP's profit). To this end, we formulate the problem of jointly deciding on caching, recommendations, and user-transcoder assignments with the goal of increasing the profit (revenue minus the incurred costs). We propose an iterative algorithm that is based on a decomposition of the formulated problem into two subproblems. We show that both subproblems, although NP-hard, are equivalent to problems in the literature for which algorithms with approximation guarantees exist. Our numerical evaluations in realistic scenarios show that the proposed policy leads to important financial gains of up to 29% when compared to the scenario where edge transcoding is not exploited.**

I. Introduction

A. Motivation and Related Work

With on-demand video streaming services dominating today's Internet traffic, wireless caching, *i.e.,* caching at the base station (BS), would play a significant role in future wireless architectures beyond 5G. Therefore, caching decisions are crucial when one takes into account the vast catalogs of such services paired with limited cache capacity at the edge. At the same time, the recommendations that appear at the user's interface have a strong impact on content requests, shaping 80% of the content requests [1]. Co-designing these decisions has been recently proposed in the literature (*e.g.,* [2], [3], [4]) as a way to steer requests towards cached contents while ensuring that users still receive recommendations that are relevant to their tastes. The joint decisions are possible when the Content Provider (CP) leases a virtual network slice, and thus, can design both caching and recommendation policies while overcoming potential obstacles of today's Content Delivery Networks (CDNs) such as https requests. This offers an at-scale content distribution for the CP.

This work has been supported by the French National Research Agency under the "5C-for-5G" JCJC project with ref. number ANR-17-CE25-0001 and by the H2020 MonB5G project (grant agreement number 871780).

An important limitation of almost all the related work on jointly optimizing caching and recommendations, is the assumption that every content, *e.g.,* a movie, is available in only one version[1]/bitrate with a certain size and popularity. However, in practice, each content might be available in a large number of versions. Different versions/qualities are delivered to different users according to the device used and to the user's Internet connection. Highly sophisticated adaptive bitrate (ABR) streaming schemes are also in place during playback to dynamically select the quality per chunk served. Therefore, a single content consists usually of a bundle of several files, sometimes more than 1,000 [5], that correspond not only to various bitrates, but also encoding profiles, audio files, subtitles, etc. When it comes to deciding what to cache, a first approach would be to cache the entire bundle per (popular) content, without taking into account which version every user would prefer or support. However, given the limited cache capacity, another approach would be to treat each version as a separate file with its own popularity, *i.e.,* assign popularities per pair [content, quality] rather than per content. This latter approach is adopted by today's large CDNs, like the one built by Netflix [5].

While this latter policy somewhat alleviates the problem, the increasing availability of edge computing capabilities (*e.g.,* at MEC servers) suggests a great opportunity of further improvement. In particular, a cached version could be *transcoded*, on the spot, in order to serve a request for a (lower) version not currently cached. Several works focus on joint caching and transcoding policies, and sometimes in conjunction with the ABR streaming scheme that is in place, *e.g.,* [6], [7], [8], [9]. These works already suggest that such an approach can improve users' quality of experience (QoE) and alleviate the traffic at the backbone Internet. However, none of the above works exploit the impact of recommendations on content requests as a way of managing the limited cache and processing capacity at the edge.

B. Our approach and Contributions

We propose a scheme of jointly optimizing caching, recommendation, and transcoding decisions, see Fig. 1. We focus on proactive decisions that can be made at the CP's core cloud at a remote location, without the need of storing users' data

[1]Throughout this paper, the terms version, bitrate, and (streaming) quality are used interchangeably.

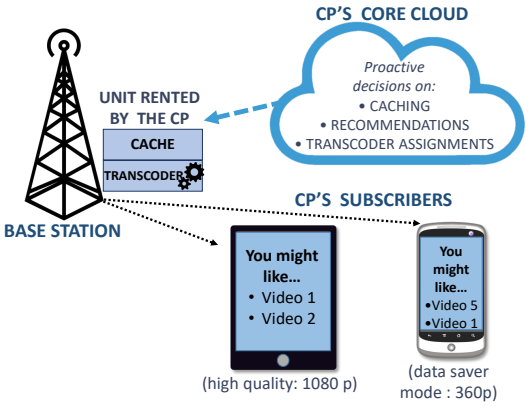


Fig. 1. The problem studied in this work: proactive decisions on caching, recommendations, and transcoding assignments for a transcoder-enabled base station. Our approach takes into account the users' preferences in contents and preferred streaming quality, as well as the limited cache and processing capacities at the edge.

at the base station. To the best of our knowledge, this is the first work that proposes and studies such a problem[2].
Our contributions are the following:

• We formulate the problem of deciding jointly on caching, recommendations, and transcoding assignments at the base station. We adopt a revenue-driven approach that attempts to maximize the CP's profit (revenue minus costs), and is able to generalize simpler metrics like cache hit rate [3] or user experience [4].

• As the problem in hand is NP-hard, we decompose it into two subproblems that each have favorable theoretical properties. On one hand, the caching and recommendations subproblem (with transcoding decisions assumed fixed and given) while still NP-hard, can be approximated to a constant. On the other hand, the transcoding subproblem, also NP-hard, is $1/2$-approximable. We thus propose an algorithm that is based on the alternating optimization of the two subproblems and provably converges to a stationary point in a small finite number of iterations.

• We conduct numerical evaluations in realistic scenarios using a real dataset. We show that transcoding at the edge can increase the CP's profit by up to 29% when compared to a joint caching and recommendations scheme. We also perform a sensitivity analysis with respect to key problem parameters, such as cache and processing capacities.

II. Problem Setup

Content Requests: The CP owns a content catalog $\mathcal{K}$ that is accessible to a set $\mathcal{U}$ of users through the CP's streaming service. A list of N_u recommended contents appears to every user $u \in \mathcal{U}$. As is common in related works, the user makes a content request according to the following model:

- with probability α_u, the user requests a recommended content. Each of the N_u recommended items could be chosen with equal probability;
- with probability $(1-\alpha_u)$, the user ignores the recommendations and requests (*e.g.,* through the search bar) a content $i \in \mathcal{K}$ of the catalog with probability p_{ui}.

The quantity α_u represents the percentage of time a user u follows the recommendations, and it could be estimated by the CP based on the user's past behavior.

Recommendations: Typically, the CP would select the N_u most relevant contents to feature in the recommendations list of every user. We denote by $r_{ui} \in [0,1]$ the (predicted) relevance or utility of a content i to the user u. These values are the result of today's state-of-the-art recommender systems [1] and are inputs for our problem. 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 relevances r_{ui}, but also on caching and transcoding assignments. 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}.

CP's Revenues: When a user u requests a content i, this content is associated with an expected revenue R_{ui} described by: $R_{ui} = f_{ui}(r_{ui})$, where f_{ui} is an increasing function of r_{ui} that describes the impact of the user's (predicted) interest in a content on the CP's revenues. This function could, for example, be estimated based on the CP's revenue model (ad-based, subscription-based, etc.) and on data on user behavior.

Caching Variables: Every user is associated to a single BS, and therefore, we focus on caching decisions of one BS. Each content of the catalog is available in L different versions/qualities, where 1 is the lowest quality and L is the highest. Every user u has a preferred streaming quality or her device/connection can support a certain quality denoted by q_u. We denote by $X=(x_{ij})$ the (binary) caching variables: $x_{ij}=1$ if content i is cached in quality j, and $x_{ij}=0$ otherwise. The cache capacity at the BS is equal to $\mathcal{C}$, and every pair [content i, quality j] is of size σ_{ij}. Typically, the higher the quality, the larger the size of the corresponding file.

Finally, caching and recommendation decisions are proactive and are made at the CP's core cloud. Proactive caching decisions (during off peak hours) have been proposed in the literature as a way of avoiding peak traffic requests [10], and they have become a trend even in today's CDN architectures [11].

Transcoding and Fetching Costs: Upon request of content i by user u, if the requested pair [content i, quality q_u] is cached at the BS, it will be delivered to the user with no cost. If the content is cached in a quality $\ell > q_u$, then this can be transcoded at the BS and delivered to the user. This action will cost (to the CP) $T_{ui\ell}$ currency units (see also Fig. 2). This can be a function of:

- fees that the CP might need to pay upon usage of computational capabilities at the BS. This can be paid either by usage or on a monthly basis, as is also the case for today's cloud services [12];
- the additional delays and low QoE that the transcoding action might cause, and therefore, this can lead to user abandonment where a part of R_{ui} will be lost[3].

[2]Close to this idea, in [9], the recommendations play the role of suggesting similar content when the ones requested are not cached. In contrast, we model the problem where recommendations *precede* (and therefore, steer) the requests, as is the case in today's streaming services.

[3]According to estimates [13], a 1% abandonment rate can already lead to a loss in the CP's revenue of $85,000$ US dollars ($) per year from ad impressions.

However, since transcoding is a computationally heavy task, only a few contents can be transcoded. We assume that only $\mathcal{B}$ threads/processing units/cores are available at a time. For this reason, in our framework, we decide on user-transcoder assignments. We denote by z_u the (binary) transcoding-assignment variables, where $z_u = 1$ if requests made by user u can be transcoded if necessary, and $z_u = 0$ otherwise. We also denote by Z the vector of z_u, $u \in \mathcal{U}$. These decisions are made proactively and at the same time as caching and recommendations. This way, recommendations for contents that cannot be transcoded will be reduced.

Remark 1. It is important to note here that our approach is complimentary to *online* ABR policies, such as DASH, that operate at a different time scale, *i.e.,* during playback. Our scheme operates *offline*, attempting to proactively identify which version of the content to cache at the edge based on estimated statistics about the quality each user can support, while taking into account the potential for edge transcoding. ABR policies can later decide, per chunk, which chunk version the user can actually support at the moment, based on the caching decisions and the processing capacity.

If the requested content is not cached in quality ℓ for all $\ell \geq q_u$, then the content will be fetched via a backhaul link (from the macro-cell or the backbone Internet) at the cost of F_{uiq_u} currency units (we make the natural assumption that $F_{uiq_u} > T_{ui\ell}$). This will be also the case if a higher quality of the requested content is cached, but the user is not assigned to the transcoder, *i.e.,* $z_u = 0$. The price F_{uiq_u} can be a function of:

- fees that the CP pays to a last-mile Internet Service Provider (ISP) or CDN for the delivery of the content;
- loss in revenue caused by additional delays and increase in user abandonment (in terms of lost ad impressions or user churn).

Therefore, the incurred cost associated to every request made by user u for content i (in quality q_u) will be:

$$\begin{aligned} Q_{ui}(X,Z) := & (1-x_{iq_u}) \sum_{\ell > q_u} x_{i\ell} z_u T_{ui\ell} \prod_{j=q_u}^{\ell-1} (1-x_{ij}) \\ & + (1-x_{iq_u}) \sum_{\ell > q_u} x_{i\ell}(1-z_u) F_{ui\ell} \prod_{j=q_u}^{\ell-1} (1-x_{ij}) \\ & + \prod_{\ell \geq q_u} (1-x_{i\ell}) F_{ui\ell}, \end{aligned} \tag{1}$$

where the first summand is the expected transcoding cost that the CP pays when the requested pair $[i, q_u]$ is not cached, but the same content is cached at a higher quality, and the user has been assigned to the transcoder ($z_u = 1$). From all the different qualities, the one closest to q_u will be transcoded. The second summand in (1) is the expected fetching cost that the CP pays when the requested $[i, q_u]$ is not cached, the content is cached at a higher quality, but the user has not been assigned to the transcoder ($z_u = 0$). Finally, the last summand represents the fetching cost the CP pays to a last-mile ISP when the requested content is not cached at the BS at any quality higher or equal to the preferred one, and the content will be retrieved through backhaul links.

III. Problem Formulation and Analysis

A. The CRTr problem

The problem of maximizing the CP's profit (revenue minus costs) as a function of caching, recommendation, and transcoding-assignment decisions is formulated as follows:

CRTr Problem.

$$\max_{X,Y,Z} \quad \sum_{u,i} \left(\frac{\alpha_u}{N_u} y_{ui} + (1-\alpha_u) p_{ui} \right) [R_{ui} - Q_{ui}(X,Z)] \tag{2}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{K}} y_{ui} = N_u, \ \forall u \in \mathcal{U}; \tag{3}$$

$$\sum_{i \in \mathcal{K}} \sum_{j \in \{1,L\}} x_{ij} \sigma_{ij} \leq \mathcal{C}; \tag{4}$$

$$\sum_{u \in \mathcal{U}} w_u z_u \leq \mathcal{B}; \tag{5}$$

$$x_{ij}, y_{ui}, z_u \in \{0,1\}, \ \forall u, i, j, \tag{6}$$

where the constrains in (3) suggest that each user receives N_u recommendations and the constraint in (4) is the cache capacity constraint for the BS's cache. Moreover, (5) is the processing capacity constraint that will be satisfied in expectation. In particular, w_u is the average number of processing units/cores that a user will utilize (based on the considered request model and caching and recommendation variables):

$$\begin{aligned} w_u := & \sum_{i \in \mathcal{K}} \left[\frac{\alpha_u}{N_u} y_{ui} + (1-\alpha_u) p_{ui} \right] \big((1-x_{iq_u}) \cdot \\ & \sum_{\ell > q_u} x_{i\ell} c_{\ell q_u} \prod_{j=q_u}^{\ell-1} (1-x_{ij}) \big), \end{aligned}$$

where $c_{\ell q_u}$ denotes the number of processing units/cores required for transcoding from source quality ℓ to destination quality $q_u < \ell$. We assume that $c_{\ell q_u} \leq c_{hq_u}$, for $q_u < \ell < h$, which is in line with related works that assume that the larger the source file is, the larger the processing needs are, *e.g.,* [6]. According to the expression above, the source file will be the one utilizing the least amount of cores among all the versions that are cached at the BS.

Lemma 1. The CRTr Problem is NP-hard.

Proof. The caching subproblem alone is a knapsack problem, which is NP-hard [14]. □

B. Our Heuristic

Based on the alternating optimization method, we decompose the problem into two subproblems: the caching and recommendations (CR) subproblem and the transcoding (Tr) subproblem. In particular, if F denotes the objective function in the CRTr problem in (2), and for a given Z' transcoding assignment vector, we define:

$$\textbf{CR-subproblem:} \quad \max_{\substack{X,Y \\ \text{s.t. (3),(4),(6)}}} \quad F(X,Y,Z'). \tag{7}$$

The CR-subproblem is NP-hard since it contains the caching decisions subject to cache capacity constraints. Nevertheless,

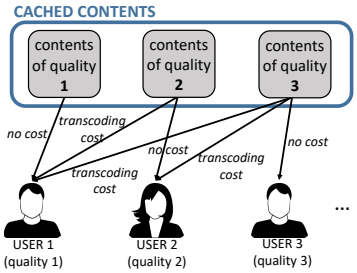


Fig. 2. An example of a weighted bipartite graph illustrating the incurred transcoding costs (if any) between cached contents of different qualities and users of different quality preferences. The contents and/or qualities that are not cached will be fetched via the backhaul link with a fetching cost (not depicted here).

the following lemma shows that its solution can be approximated up to a constant factor.

Lemma 2. The CR-subproblem is approximable by a factor of $\frac{1-1/e}{2}$.

Proof. The key observation is that cached contents of different qualities will be delivered at different costs, *e.g.,* we can picture this as a bipartite graph (see Fig. 2). On the other hand, in the femtocaching problem in [10] (that contains only caching variables) and in the problem defined in [4] (that contains both caching and recommendation variables), a similar bipartite graph describes the connectivity between users and different caches. We prove that the CR-subproblem can be mapped to the problem in [4], and therefore, it is submodular and monotone increasing.

Under the same request model as ours, the objective function in [4] is the balanced sum $stream.\ quality + \beta_u \cdot recomm.\ quality$ capturing a tradeoff between the streaming quality in terms of expected Quality of Service (QoS) that a user can enjoy as a result of cache-friendly recommendations, and the distortion of the recommendations she will experience. We will show that the CR-subproblem's objective function in (7) can be mapped to the term of streaming quality in [4]. We note that our framework does not measure recommendation quality[4], and thus, $\beta_u = 0$ for all $u \in \mathcal{U}$. Therefore, the objective function in [4], for $\beta_u = 0$, is equal to:

$$\sum_{u,i} \left(\frac{\alpha_u}{N_u} y_{ui} + (1-\alpha_u) p_{ui} \right) s_u(X,i), \tag{8}$$

where $s_u(X,i)$ is the expected QoS for user u and (requested) content i, for a given cache allocation X. Similar in spirit to femtocaching [10], the term $s_u(X,i)$ is expressed in a way that every requested content will be delivered from the cache that ensures the best QoS for the user, given the cache allocation. This cache might be the original server (that contains the entire catalog) in case the particular content is not cached at any cache that the user has access to. The subset of caches a user u has access to is denoted by $\mathcal{C}(u)$, while the cache-specific QoS for cache j is denoted by s_{uj}. According to [4], $s_u(X,i)$ is calculated by the following expression:

$$s_u(X,i) := \sum_{j=1}^{|\mathcal{C}(u)|} \left[s_{u(j)} x_{i(j)} \prod_{l=1}^{j-1} (1 - x_{i(l)}) \right], \tag{9}$$

where the $s_{u(j)}$ are the ordered cache-specific QoS, *i.e.*, $s_{u(1)}$ denotes the maximum (cache-specific) QoS for user u, $s_{u(2)}$ is the second highest QoS for user u, and so on.

Upon request of a content, instead of choosing the cache with the "best" QoS among a set of caches, our objective picks the "best" available quality (or version) of the requested content among the ones cached (given the cache allocation and the transcoding assignments). If the content is not cached in any quality or if transcoding is not possible for the particular user, the content will be delivered in the requested quality by the backhaul link. In this case, we conventionally use the symbol 0 to describe the quality. Therefore, in our problem setup $\mathcal{C}(u,i)$ for user u is defined as the set of available qualities and is a subset of the set $\{0,\dots,L\}$. Formally, we can write: $\mathcal{C}(u,i) := \{0\} \cup \{q_u\} \cup \{q_u+1,\dots,L \text{ if } z_u = 1\}$. The CP's profit of our framework can be mapped to the quantity $s_u(X,i)$ if we notice that every quality (or version) j brings $\mathcal{S}_{uj}$ profit as described below:

$$\mathcal{S}_{uj} := \begin{cases} R_{ui} - F_{uiq_u}, & \text{for } j = 0, \\ R_{ui}, & \text{for } j = q_u, \\ R_{ui} - T_{uij}, & \text{for } j > q_u. \end{cases}$$

Therefore, by defining a similar ordering of $\mathcal{S}_{uj}$, these ordered values can replace the quantities $s_{u(j)}$ in (9) which then implies that our objective can be written as the objective in [4] expressed in (8). This mapping of our problem's quantities to the quantities used in [4] allows us to prove that the objective function of the CR-subproblem is monotone increasing and submodular. The theory on submodular maximization suggests that a $\frac{1-1/e}{2}$-approximation is achievable by the policy described in [4]. □

According to the policy proposed in [4], the policy for the CR-subproblem starts with the cache being empty and greedily fills it as long as the capacity constraint is not violated. In every round of selection, it estimates the benefit (or marginal gain) in terms of the objective function in (7) for all pairs [content, quality] while deciding on the recommendations (as a subroutine). It then selects the pair that leads to the highest benefit before the next selection round begins. This is a size-oblivious algorithm. However, since every pair [content, quality] is of different size, the policy also runs a size-aware algorithm that repeats the same process as above but this time selecting the pair with the highest ratio of benefit to the pair's size. Finally, it suffices to choose the solution that achieves the highest objective function value that resulted by the size-oblivious and size-aware algorithms.

Next, given a pair of caching and recommendations decisions (X', Y'), we define the second subproblem that aims to decide on the user-transcoder assignments (variables z_u):

$$\textbf{Tr-subproblem:} \quad \max_{\substack{Z \\ \text{s.t. (5),(6)}}} F(X', Y', Z). \tag{10}$$

[4]Even thought the objective in (7) does not explicitly maximize the recommendation quality, the dependency of R_{ui} on the relevances r_{ui} already "pushes" the CP towards a limited distortion of the recommendations.

Lemma 3. The Tr-subproblem is NP-hard, but approximable by a factor of $1/2$.

Proof. The TR-subproblem is similar to user-association problems in small-cell networks (studied, for example, in [15]). In particular, the Tr-subproblem evaluates the benefit (in terms of profit) of assigning a user to the transcoder, given the expected number of cores the user will utilize. This number, denoted by w_u in (5), corresponds to the weight of an object in the definition of the 0-1 knapsack problem. Therefore, the subproblem is equivalent to the 0-1 knapsack problem, and thus, NP-hard. However, the well-known modified greedy algorithm for the knapsack problem can achieve a $1/2$-approximation[5] [14]. □

In particular, according to the greedy algorithm for the knapsack problem, the algorithm for the TR-subproblem consists of choosing the users to grant a transcoding assignment based on the ratio of benefit each user brings (in terms of the objective function in (10)) to the weight w_u (which is the expected number of processing units that each user will utilize). The modification that ensures a $\frac{1}{2}$-approximation guarantee (see [14]) consists of running the greedy algorithm first, and then comparing its solution with the highest benefit of any single user. The larger of the two is chosen as the final solution (transcoding assignment vector).

Based on the alternating optimization of the two subproblems, we now define our heuristic:

CRTr algorithm: An initial value for the transcoding assignment vector $Z^{(0)}$ is set to $Z^{(0)} = [1, 1, \ldots, 1]$. Then, at every iteration $k = 1, 2, \ldots$:

1) Solve the CR-subproblem for $Z^{(k-1)}$ (transcoding assignment vector) with the approximation algorithm in [4].
2) Given the solution of CR-subproblem $(X^{(k)}, Y^{(k)})$ solve the Tr-subproblem with a greedy algorithm for the knapsack problem [14].

Lemma 4. The CRTr algorithm converges to a stationary point within 2 iterations.

Proof. In particular, the CRTr algorithm converges after solving the CR-subproblem of the second iteration, making it only a 3-step algorithm. We will prove that a fourth step of solving the Tr-subproblem for $k = 2$, *i.e.,* deriving $Z^{(2)}$, will result in $Z^{(2)} = Z^{(1)}$. Let us assume, on the contrary, that $Z^{(2)} \neq Z^{(1)}$. Without loss of generality, this means that after the second iteration, User A who was not assigned to the transcoder will be prioritized over User B who will lose her assignment to the transcoder, *i.e.,* $z_A^{(2)} = 1$ and $z_B^{(2)} = 0$, while $z_A^{(1)} = 0$ and $z_B^{(1)} = 1$. First, the fact that $z_A^{(1)} = 0$ and $z_B^{(1)} = 1$ means that, within the first iteration, the relative gain of assigning User B was larger than the one concerning User A. Let us denote by $G_u(X', Y')$ the relative gain in terms of the greedy algorithm for Tr-subproblem (*i.e.,* gain of objective function in (10) divided by w_u) of assigning user u to the transcoder, given (X', Y'). Therefore, the previous observation implies that $G_B(X^{(1)}, Y^{(1)}) \geq G_A(X^{(1)}, Y^{(1)})$. We remind the reader that $(X^{(1)}, Y^{(1)})$ was derived given $Z^{(0)}$, *i.e.,* all users are assigned to the transcoder. Since, by the definition of the problem, $F_{uiq} > T_{ui\ell}$, *i.e.,* the fetching costs are higher than the transcoding costs, if at least one user is not assigned (given the limited processing capacity) in $Z^{(k)}$, for $k > 1$, then the relative gain G_u will decrease for this user. Formally, this means that: $G_u(X^{(1)}, Y^{(1)}) \geq G_u(X^{(k)}, Y^{(k)})$, for all $k = 2, 3, \ldots$, for this user in question. On the other hand, for every user who is assigned to the transcoder, at iteration $k > 1$, her relative gain is greater than or equal to the relative gain at iteration $k = 1$. In fact, given that only a subset of users will be assigned to the transcoder, the caching (and recommendation) decisions will naturally concern the most relevant contents for this group of users. Formally, this means that $G_u(X^{(1)}, Y^{(1)}) \leq G_u(X^{(k)}, Y^{(k)})$, for every user in this group. Therefore, $G_B(X^{(2)}, Y^{(2)}) \geq G_B(X^{(1)}, Y^{(1)}) \geq G_A(X^{(1)}, Y^{(1)}) \geq G_A(X^{(2)}, Y^{(2)})$. However, the inequality $G_B(X^{(2)}, Y^{(2)}) \geq G_A(X^{(2)}, Y^{(2)})$ leads to a contradiction: User B should be prioritized over User A, and therefore, it is not possible that $z_A^{(2)} = 1$ while $z_B^{(2)} = 0$. □

The running time of the CRTr algorithm is the sum of the running times of the two subproblems' algorithms, *i.e.,* $O(|\mathcal{U}| \cdot |\mathcal{K}|^2 \cdot L \cdot \mathcal{C} \cdot \log(|\mathcal{U}|))$. Implementation-wise, the complexity of solving the CR-subproblem can be alleviated either through methods that avoid unnecessary calculations or through distributed implementations (see, for example, [16]).

IV. Performance Evaluation

In this section, we numerically evaluate the gains that can be achieved through the proposed CRTr policy, compared to the case where edge transcoding is not exploited (*i.e.,* the state-of-the-art policies for joint caching and recommendations). Moreover, we perform a sensitivity analysis with respect to key parameters, such as cache and processing capacity. First, we present the default input parameters.

Catalog, Recommendations, and Revenue: Our scenario consists of 100 users connected to the BS that have access to a catalog of 4000 movie titles (which corresponds to a typical catalog of movies in today's streaming services [17]). Every content is available in 5 different qualities [18]. The size of different contents and qualities is calculated through statistics on movie lengths and data usage for various qualities. In particular, given an average length of 2.17 hours [19], the length of every movie is selected randomly in the interval $[1.34, 3]$ (in hours). Based on Netflix's data usage [20], we assume data usage of 0.3, 0.7, 1, 3, and 4.5 Gb per hour for qualities 1 to 5 respectively. Hence, file sizes σ_{ij} vary from 0.4Gb to 13.5Gb. The users prefer or their devices/connections can support a specific quality as follows: 10% for quality 1, 20% for quality 2, 40% for quality 3, 20% for quality 4, and 10% for quality 5. These percentages were derived by adapting findings on (wired) Internet connectivity in France [21]. Every user receives N_u recommendations, where N_u is randomly chosen between 20 and 35. The probability the user follows the recommendations, α_u, varies in $[0.6, 1)$ (as in Netflix, where the average is equal to 0.8 [1]). For the matrix of content utilities r_{ui}, a subset of the Movielens dataset [22] containing 5-star ratings of movies was used, where the

[5]We note that a better approximation can be achieved through the FPTAS scheme at the cost of higher complexity, see [14].

ratings were mapped in the interval $[0, 1]$ (as in [4]). The popularity of contents (when requested outside of the recommendations) is equal to the normalized content relevances r_{ui} aggregated over the users, *i.e.*, $r_{ui}/\sum_u r_{ui}$. Finally, the values R_{ui} (revenue per content) were derived through the equation $0.1 + R_0 * r_{ui}$, and therefore, vary from \$0.1 and \$1.1. This equation could, for example, capture an ad-based revenue model (*e.g.,* YouTube), where r_{ui} can be interpreted as the user retention rate, and thus, the quantity $R_0 * r_{ui}$ is the portion of ad-based revenue.

Caching, Transcoding, and Costs: We consider a BS with caching and processing capabilities (for which the exact numbers will be specified in what follows). The processing capacity is counted in number of cores (processing units). We assume that transcoding contents of qualities 2 and 3 to lower qualities requires a single core, while transcoding qualities 4 and 5 requires two cores (see Sec. III-A). The transcoding cost (that the CP pays to the mobile network provider) in the first case is \$0.02 per hour, and in the second case \$0.04 per hour [12]. If transcoding is not possible (because of the cache allocation or the transcoding assignments), the requested content will be fetched from a macro-cellular station that contains the entire catalog, as it has been envisioned for future wireless networks, *e.g.,* [10]. In this case, the CP pays to a last-mile ISP fetching fees equal to \$0.15 per Gb [23].

First, we fix the processing capacity to 20 cores, which, as we explained above, means that requests of only 10 to 20 users can be satisfied through transcoding. The number of cores that each user might use (varying from 1 to 2) has been calculated in expectation as a result of the content request model we described in Sec. II. We are particularly interested in evaluating the gains of the transcoding-enabled scenario versus the gains of a scenario where only caching and recommendations are optimized, without the possibility of transcoding. For this reason, we will compare with the *NoTransc policy* that is defined as follows:

NoTransc Policy: This policy solves the joint caching and recommendations problem for a BS with no processing capabilities (*i.e.,* $z_u = 0$ for all u). The solution is derived by applying only the first step of our heuristic (single iteration), which is essentially the policy proposed in [4]. This policy treats every pair [content, quality] as a separate file.

In Fig. 3, we plot the relative increase in net income (profit) achieved by our policy with respect to the scenario where no transcoding is possible, *i.e.,* the y-axis is normalized with respect to the profit achieved by the NoTransc policy. The points correspond to different relative cache sizes, *i.e.,* as a fraction of the total size of the catalog $\sum_{i,j} \sigma_{ij}$. We focus on cache capacity varying between 0.1% and 2%. This is in line with caches that are used today by Netflix in its in-house CDN (Open Connect), see [24] and [25]. We observe that for small cache sizes between 0.1% and 0.6%, the relative gain is larger than or equal to 5%, and can reach up to 29%. It is important to note that even gains of 5% imply large monetary profits in real-world systems. Especially when referring to large CPs, like Netflix, that have more than 200 million subscribers and report annual profits of more than 1 billion US dollars [26]. As the cache size increases, more files can be cached and transcoding costs can be avoided. Hence, the gains of enabling transcoding at the edge decreases. However, the size of these gains not only depend on the revenue and cost models, but also on key parameters that we will explore in what follows.

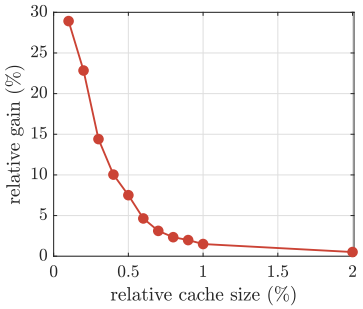
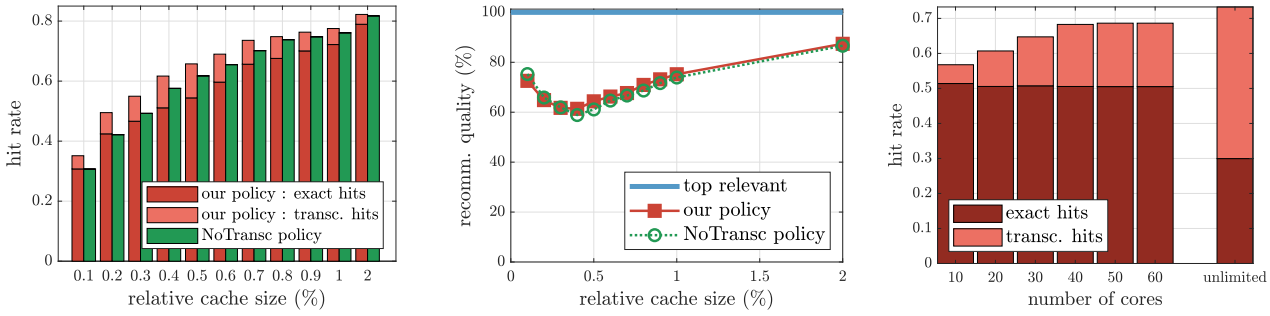


Fig. 3. Relative gain in net income achieved by our policy when compared to the gains achieved by a scheme without transcoding (NoTransc policy).

In order to better understand Fig. 3, we study the distribution of the cache hit rate between exact cache hits (*i.e.*, the requested quality is cached) and the transcoding cache hits (*i.e.*, a transcoding action takes place in order to deliver the requested content in the requested quality). In Fig 4(a), we plot the hit rate achieved by our policy (dark red for exact hits and light red/pink for transcoding hits) and the hit rate of NoTransc policy (only exact hits, by design). We see that, for cache capacity of 0.4%, the exact hit rate is equal to 0.5, while the transcoding rate is 0.11. As the cache capacity increases, the hit rate of NoTransc is close to the total hit rate of our policy, which renders the gain of transcoding smaller (as observed in Fig. 3). We note that the gain of transcoding is restricted by both the transcoding cost itself and the processing capacity.

We stress here that jointly optimizing caching and recommendations (even without transcoding) can lead to high hit rates (up to 0.82), as it has been shown already in related works, *e.g.,* [4], [3]), and can been observed in Fig. 4(a). This is because the recommendations are "nudged", *i.e.,* instead of recommending to each user the most relevant contents (which are the contents with the highest r_{ui} for every u), the recommendations favor cached items. Therefore, we evaluate how our scheme can potentially affect the users and their perception of the recommendations. For that, we measure the recommendations quality (RQ) as defined in [4]: for each user u, RQ measures the sum of relevance of the received recommendations, *i.e.,* $\sum_i r_{ui} y_{ui}$. Fig. 4(b) shows the aggregate RQ (summed over the users) achieved by our policy and by the NoTransc policy. The y-axis is regularized with respect to the top relevant contents (baseline recommendations). We see that the quality achieved by both policies is between 60% and 87%. The RQ of the two policies are very close to each other (where in most cases our policy slightly outperforms the NoTransc policy) while our policy achieves a higher profit.

Finally, we investigate how the processing capacity affects the hit rate and the distribution of exact and transcoding hits. Fig. 4(c) depicts the distribution of the hits for different processing capacities and for fixed cache size of 0.4%. We see that, in principle, as the number of processing units increases,



(a) (b) (c)

Fig. 4. (a) Hit rate achieved by our policy and by the NoTransc policy for different cache capacities; (b) Recommendations quality (RQ) for different cache capacities for our policy and the NoTransc policy; (c) Distribution of exact and transcoding hits achieved by our policy for different processing capacities.

the percentage of the transcoding hits increases. However, the cache capacity still constrains this percentage since only a few contents can be cached in high qualities (note that the higher the quality, the larger the file is), and thus, can be transcoded on-the-fly. For reference, we also depict the hits in the case of unlimited processing capacity. We see that, in this case, the transcoding hits outnumber the exact hits. This would lead to higher gains in net income and higher RQ. However, this balance between transcoding and exact hits reflects the difference between fetching and transcoding costs.

V. Conclusions and Future Work

We presented a generic framework that captures the economic gains of a CP that employs a BS with caching and transcoding capabilities. We modeled the problem of jointly optimizing (proactive) caching, recommendation, and transcoding decisions with the goal of maximizing the net income. To this end, we proposed a policy based on the method of alternating optimization. Our numerical evaluations showed that, in realistic scenarios, enabling transcoding at the edge (in conjunction with a joint caching and recommendation scheme) can be significantly beneficial, especially in case of very limited cache capacity. An interesting direction for future work is to design an online decision mechanism that adjusts the transcoding assignments given the flow of requests and possibly in conjunction with the ABR scheme that is in place. Another direction would be to tackle the problem in the setting of a network of BSs with overlapping coverage.

References

[1] 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.

[2] K. Qi *et al.*, "Optimizing caching and recommendation towards user satisfaction," in *Proc. IEEE WCSP*, 2018, pp. 1–7.

[3] 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.

[4] D. Tsigkari and T. Spyropoulos, "An approximation algorithm for joint caching and recommendations in cache networks," *IEEE Trans. on Network and Service Management*, vol. 19, no. 2, pp. 1826–1841, 2022.

[5] Netflix Tech Blog. (2017) Content popularity for Open Connect. [Online]. Available: https://netflixtechblog.com/content-popularity-for-open-connect-b86d56f613b

[6] T. X. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Trans. on Mobile Computing*, vol. 18, no. 9, pp. 1965–1978, 2018.

[7] H. A. Pedersen and S. Dey, "Enhancing mobile video capacity and quality using rate adaptation, RAN caching and processing," *IEEE/ACM Trans. on Networking*, vol. 24, no. 2, pp. 996–1010, 2015.

[8] C. Wang *et al.*, "Video caching and transcoding in wireless cellular networks with mobile edge computing: a robust approach," *IEEE Trans. on Vehicular Tech.*, vol. 69, no. 8, pp. 9234–9238, 2020.

[9] C. Li *et al.*, "Joint transcoding- and recommending-based video caching at network edges," *IEEE Systems Journal*, pp. 1–10, 2021.

[10] K. Shanmugam *et al.*, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[11] Netflix. (2020) Open Connect fill patterns. [Online]. Available: https://openconnect.zendesk.com/hc/en-us/articles/360035618071-Fill-patterns

[12] AWS. (2022) Amazon EC2 on-demand pricing. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/

[13] Akamai. (2020) Understanding the Value of Consistency in OTT Video Delivery (White Paper). [Online]. Available: https://www.akamai.com/resources/white-paper/understanding-the-value-of-consistency-in-ott

[14] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[15] F. Pantisano *et al.*, "Cache-aware user association in backhaul-constrained small cell networks," in *Proc. IEEE WiOpt*, 2014, pp. 37–42.

[16] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause, "Distributed submodular maximization," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 8330–8373, 2016.

[17] Insider. (2022) How Netflix, Disney+, HBO Max, and more major streamers compare on content and cost. [Online]. Available: https://www.businessinsider.com

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

[19] Statista. (2022) Average length of the top 10 highest-grossing movies in the U.S. and Canada from 1980 to 2021. [Online]. Available: https://www.statista.com/statistics/1292523/lenght-top-movies-us/

[20] Netflix Help Center. (2022) How to control how much data Netflix uses. [Online]. Available: https://help.netflix.com/en/node/87

[21] D. Belson, "State of the Internet Q4 2016 report," *Akamai Technologies*, vol. 9, no. 4, 2017.

[22] F. M. Harper and J. A. Konstan, "The Movielens datasets: History and context," *ACM Trans. on Inter. Intell. Sys.*, vol. 5, no. 4, p. 19, 2016.

[23] Microsoft Azure. (2022) Content Delivery Network Pricing. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/cdn/

[24] G. S. Paschos *et al.*, "Wireless caching: Technical misconceptions and business barriers," *IEEE Commun. Mag.*, vol. 54, no. 8, pp. 16–22, 2016.

[25] Netflix. (2020) Open Connect appliances. [Online]. Available: https://openconnect.netflix.com/en/appliances/

[26] Netflix Investors. (2022) Company profile. [Online]. Available: https://ir.netflix.net/ir-overview/profile/default.aspx