SOBA: Session optimal MDP-based network friendly recommendations

Theodoros Giannakas[1], Anastasios Giovanidis[2], and Thrasyvoulos Spyropoulos[1]
[1] EURECOM, Sophia-Antipolis France, first.last@eurecom.fr
[2] Sorbonne University CNRS-LIP6, Paris, France, anastasios.giovanidis@lip6.fr

***Abstract*—Caching content over CDNs or at the network edge has been solidified as a means to improve network cost and offer better streaming experience to users. Furthermore, nudging the users towards low-cost content has recently gained momentum as a strategy to boost network performance. We focus on the problem of optimal policy design for Network Friendly Recommendations (NFR). We depart from recent modeling attempts, and propose a Markov Decision Process (MDP) formulation. MDPs offer a unified framework that can model a user with random session length. As it turns out, many state-of-the-art approaches can be cast as subcases of our MDP formulation. Moreover, the approach offers flexibility to model users who are reactive to the quality of the received recommendations. In terms of performance, for users consuming an arbitrary number of contents in sequence, we show theoretically and using extensive validation over real traces that the MDP approach outperforms myopic algorithms both in session cost as well as in offered recommendation quality. Finally, even compared to optimal state-of-art algorithms targeting specific subcases, our MDP framework is significantly more efficient, speeding the execution time by a factor of 10, and enjoying better scaling with the content catalog and recommendation batch sizes.**

I. Introduction

A. Motivation

With multimedia traffic from Netflix, YouTube, Amazon, Spotify, etc. comprising the lion's share of Internet traffic [1], reducing the "cost" of serving such content to users is of major interest to both content providers (CP) and network operators (NO) alike. This cost includes the actual monetary cost for the CP to lease or invest in network and cloud resources, but also network-related costs, related to resource congestion, slowing down other types of traffic, stalling multimedia streams etc.

Caching popular content near users has been a key step in this direction in wired networks through the use of CDNs [2], and more recently in wireless networks through femtocaching [3]. In addition to cost reduction for CPs and NOs, caching also allows for higher streaming rate, shorter latency, etc. [4], which results in an improved viewing/listening experience for the user. When platforms of video streaming services can not offer high bitrate, user abandonment rates rise [5]. Hence, reducing the cost of bringing interesting content to users will benefit everyone: the users, the content providers, and the network operators.

Recommendation systems (RSs) in popular content platforms play an important role for this task: they suggest interesting content to users. For example, 80% of requests in Netflix, and more than 50% on YouTube, stem from the platform recommendations [6], [7]. The traditional role of an RS has been to make personalized recommendations to the user, suggesting items from a vast catalog that best match her interests using techniques like collaborative filtering [8], deep neural networks [9], matrix factorization [10], etc. The vast majority of popular RS systems focus on content relevance and similarity, but they do not account for the network cost of delivery. Such operation, which ignores network costs in content recommendation algorithms inevitably leads to largely sub-optimal network performance for all parties involved.

B. Related Work

A handful of recent works have spotted the interplay between recommendation-network vs QoS-cost, and have proposed to modify the recommendation algorithms towards a more *network-friendly* operation [11], [12], [13], [14], [15], [16], [17]. The main objective of almost all these works is to recommend content that is highly interesting to the user while at the same time involves low delivery cost. A simple solution to achieved this is *to favor cached content* [18]. While various implementation barriers are sometimes cited [19], the increasing convergence of CPs and NOs [20], especially in the context of network slicing and virtualization suggests, that in the very near future content providers will be the owners of their own network (slice), and will be able to directly infer the potential network cost of recommending and delivering some content versus another.

To date, a number of these early network-friendly RS proposals are basically (sometimes efficient) heuristics [14], [16]. A large number of these works focuses on *myopic* algorithms, where the RS aims to minimize delivery cost only for the next content request [18]. In practice, however, when visiting popular applications like YouTube, Vimeo, Spotify, etc., a user [21], [22] consumes several contents one after the other, guided and impacted by the RS system at each of these steps. As a result, what the RS recommends while the user watches some content in a viewing session, will not impact the selection and delivery cost of just the next request, but also all subsequent requests until the end of that session.

Myopic schemes are thus sub-optimal. Instead, one should aim to find the optimal action now, that will *minimize the expected cost over the entire session, taking into account both what the RS could suggest in future steps, as well*

as how the user might react to them. A couple of recent works have attempted to tackle this exact problem using convex optimization [12], [23]. While the authors manage to formulate the problem as a biconvex [12] and linear program [23], the latter yielding an optimal solution, these works are characterized by the following two key *shortcomings*: (i) the problem formulation requires the user session to be of infinite length in order to derive closed form expressions for the objective; (ii) although the problem is an LP in [23], we will see that the runtime of their algorithm is quite slow.

C. Contributions and Structure

In this work, we approach the above Network Friendly Recommendations (NFR) problem in a novel way. Our main contributions can be summarized as follows:

(C.1) We propose a unified MDP framework to minimize the expected caching cost over a *random session of arbitrary length*, while suggesting the user high quality content. Our approach is parameterized in such a way that many state-of-the-art methods be adapted to our framework.

(C.2) The MDP is formed using as problem unknowns the continuous *item recommendation frequencies* per viewed content. In doing so, we do not need to search for the optimal N-sized recommendation batch per viewed content, thus avoiding the curse of dimensionality of MDPs. Our formulation uses the least number of variables (K^2 specifically, where K is the size of the catalog) to describe the MDP without losing in optimality, compared to the fully detailed description. Noteworthy is the fact that the complexity of the algorithmic solution becomes insensitive to the size N of recommended batch per viewed content.

(C.3) We express the content transition probabilities in a general way, which enables us to incorporate a variety of user behaviors. Furthermore, the policy iteration steps in the solution of the Bellman equations can be naturally decomposed into simpler continuous subproblems, which can be solved (i) by low complexity linear, or convex programming techniques and (ii) in parallel, offering an additional potential speedup of $K\times$.

(C.4) For sessions with big horizon, the MDP has significant gains in terms of caching cost over myopic policies. When compared to recently published works that consider the infinite horizon NFR problem, our framework (due to the reasons mentioned in (C.2) and (C.3)) is able speed-up the execution time by a factor of 10, while achieving optimal cost performance.

The paper is structured as follows. Section II sets up the problem, presents the user-RS interaction and introduces the RS input and objectives. Feasible, optimal and sub-optimal recommendation policies are discussed. The problem is formed as an MDP in its general form in Section III and an algorithmic solution is described based on Bellman equations. In Section IV we present our user model and explain its MDP solution. Section V contains the evaluation of our policy in terms of cost and user satisfaction performance against heuristics and state of the art solutions. We conclude the paper in Section VI.

II. Problem Setup

A. User session and recommendations

We consider a user who enters some multimedia application, e.g. YouTube, and requests sequentially a random number of items from its catalog $\mathcal{K}$ ($|\mathcal{K}| = K$). Such applications are equipped with a RS, responsible for helping the users discover new content. Our user has some prior underlying probability to request content i from $\mathcal{K}$, which we denote as $p_0(i)$; with vector p_0 denoting the probability mass function (pmf) for all $i \in \mathcal{K}$. The length of each session is random, and we assume that it follows a geometric distribution with mean $1/(1-\lambda)$. It is further assumed here that the session length is *independent* of the RS suggestions. The user session has the following structure:

- The user starts the session from some random content i drawn from the distribution $\mathbf{p}_0$.
- The RS, at every request, recommends N new contents; we denote this N-sized batch as w.
- The user may follow the recommendations related to content i, by clicking on a content among the N in the batch w,
- Or the user ignores the recommendations and chooses some other item based on initial preferences $p_0(i)$.
- The user exits the session with probability $1-\lambda$ after any request.

B. System input about user preferences

Entertainment oriented applications massively collect data related to user interaction and content ranking, allowing them to become increasingly effective in their recommendations. According to the RS literature [24], [8], [25] user ratings are used to infer the level of similarity [26] between contents. In our paper, we formalise the notion of *related content* to viewed content i as follows: For every content i, there exists a similarity value with all other items in the catalog $\mathcal{K}$. The similarity with content j is quantified by the value $u_{ij} \in [0,1]$, forming the K-length row vector $\mathbf{u}_i$. This information is summarized in the square non-symmetric $K \times K$ matrix U.

We further denote by $\mathcal{U}_i(N)$, the set with the $N < K$ highest u_{ij} values related to content i. Note that the values of $\mathbf{u}_i$ are not normalized per content, i.e. the matrix U is *not* stochastic. The matrix U, which represents the content relations, is considered as *input* for the RS.

The RS assumes that the user feels satisfied with some recommendation batch, if this includes items j with high u_{ij} values. *User satisfaction* is denoted by $Q_i(w)$ and is quantified by the ratio

$$Q_i(w) := \sum_{j\in w} u_{ij} \Big/ \sum_{m\in\mathcal{U}_i(N)} u_{im}. \tag{1}$$

It is measured per viewed content i and recommendation batch w; it depends on the entries of U, the size N of the

batch and the policy. The denominator in (1) is the maximum batch quality Q_i^{max} so that $Q_i(w) \in [0,1]$. The expression states that the higher the sum u_{ij} of the recommendation batch, the happier the user is. Both the content popularity vector $\mathbf{p}_0$ and the similarity matrix U is information that the RS has at its disposal, from measurements over time.

C. Network-related costs

From the network's perspective, each content $i \in \mathcal{K}$ has a non-negative network cost $c_i,\ \mathbf{c} = [c_1, \ldots, c_K]^T$, associated to its delivery to the user. The content delivery cost might depend on several factors such as its size (in MB), its routing expenses, its location on the network etc. A session of $L > 1$ requests incurs a cumulative cost on the network. Due to the impact of RS on user requests, the sequence of costs $\{c(S_t)\}_{t=0}^{L}$ will depend on the RS policy, where S_t is the state visited at t and $c(S_t)$ is the cost of the state S_t. Thus, our primary objective is to come up with policies R (to be defined more formally next), which promote low-cost contents and ultimately minimize the session's average cost, while at the same time satisfying the user's natural preference for higher content relevance

$$\underset{R}{\text{minimize}} \left\{ \frac{1}{L} \sum_{t=1}^{L} c(S_t) \right\}. \tag{2}$$

Letting $c_i \geq 0$ to be real positive, gives the flexibility to capture various network-related scenarios such as

1) Maximize cache-hit rate: set $c_i \in \{0,1\}$ for $\{cached,\ uncached\}$ contents respectively.
2) Minimize content delivery cost: set $c_i \in \mathbb{R}$ to include delay and bandwidth in the CDN case.

D. Policies

Our focus in this work is to find policies for arbitrary user sessions in terms of average length. The policies should aim at minimizing the expected session network-cost, while guaranteeing a good (and controllable) level of user satisfaction. Before formally defining the optimization problem in the next section, we present here in detail what is a policy and how it is modeled in our framework, and also three reasonable heuristics. As mentioned previously, when the user visits file i, the RS proposes any N-sized recommendation batch of unique contents (excluding self-recommendation i). The set of all N-sized batches w forms the set of actions when the user views content i, which we denote as $\mathcal{A}_i$. To formally define a *policy*, we need to associate each recommendation batch $w \in \mathcal{A}_i$ with a frequency of use $\mu_i(w)$. The frequencies of all the batches related to i should sum up to 1. This gives rise to two classes of policies.

- *Deterministic:* A unique batch w can appear per viewed object. For every i there is a single action w for which $\mu_i(w) = 1$.
- *Randomized:* At least two actions have $\mu_i(w) > 0$. This means that at every appearance of i, the user might see a different N-tuple of contents, chosen randomly.

The cardinality of the action set $\mathcal{A}_i$ is exploding, leading to $\binom{K-1}{N}$ variables *per item* over which we must optimize. As an example, for catalog $K = 1000$ and $N = 3$ recommendations the RS needs to introduce 165 Billion unknown μ's.

1) Item-wise recommendation frequencies: To overcome this serious modeling issue, we use a different approach. Related to viewed content i, we introduce the item-wise recommendation frequencies $\mathbf{r}_i = \{r_{ij}\}$ as the new set of unknown variables. In fact, these quantities can be expressed through the per-batch frequencies, and they actually summarize their information as follows,

$$r_{ij} = \sum_{w \in \mathcal{A}_i} \mu_i(w) \mathbf{1}_{\{j \in w\}}\ ,\ \forall j \in \mathcal{K}. \tag{3}$$

Therefore, $r_{ij} \in [0,1]$ represents the overall probability of object j to appear in any recommendation batch related to i, without specifying the other $N-1$ elements of the batch. For the vector $\mathbf{r}_i$ we can verify that it satisfies the size N of the recommendation batch, with equality

$$\sum_{j=1}^{K} r_{ij} = \sum_{j=1}^{K} \sum_{w \in \mathcal{A}_i} \mu_i(w) \mathbf{1}_{\{j \in w\}} = N\ \forall i \in \mathcal{K}. \tag{4}$$

If the policy is deterministic, then for every content i there are exactly N entries $r_{ij} = 1$, and the rest are equal to zero. On the other hand, if the policy is randomised, then at least two entries $r_{ij} < 1$. To see this in a small example, consider the randomised policy with feasible batches $\mathcal{A}_i = \{\{1,2\},\{1,3\}\}$ associated with batch-frequencies $\{0.5,\ 0.5\}$. This translates to item-wise frequencies $r_{i1} = 1.0,\ r_{i2} = 0.5$ and $r_{i3} = 0.5$, while the remaining r_{ij}'s are zero. For each content i, we relate a frequency vector $\mathbf{r}_i$ of size K. By concatenating these vectors as $R = [\mathbf{r}_1^T, ..., \mathbf{r}_K^T] \in \mathbb{R}^{K \times K}$ we form the *policy*. We have thus reduced the unknowns to just K^2, a considerable improvement!

Remark: The definition of a policy R through the r_{ij} frequencies, can allow to generate recommendation batches with the appropriate $\mu_i(w)$ batch-frequencies. For a deterministic policy, the N non-zero r_{ij} entries per i define the unique N-sized batch $w \in \mathcal{A}_i$. Now, in the case of a randomised policy, for some j's it holds $r_{ij} < 1$, so there are more than one potential batches. We can use the random vector generation technique found in [27, Fact 1, Probabilistic Placement Policy], where different batches of size N are randomly sampled, while guaranteeing that each content j appears with probability r_{ij}. In the case of our previous simple example given $r_{i1} = 1.0, r_{i2} = 0.5$ and $r_{i3} = 0.5$, we can reproduce the batches and their frequencies as follows. Given $N = 2$ recommendation slots, each slot will be time-shared by contents whose frequencies sum-up to 1. So the first slot will always be occupied by item "1" because $r_{i1} = 1.0$. The second slot will be time-shared by "2" and "3", 50% of the time each, so that $\mu(\{1,2\}) = 0.5$ and $\mu(\{1,3\}) = 0.5$, thus reproducing the more detailed policy. This technique can be generalised to $N > 2$.

2) Simple Myopic Policies: Here we list some practical intuitive policies, which either favor low network-cost or user satisfaction or both, but are myopic in the sense that they consider only the impact of the immediate next request.

TABLE I: Main Notation

$\mathcal{K}$	Content catalog of size K
λ	Prob. that the user stays in the session
N	Recommendation batch size
$\mathbf{p}_0$	Baseline popularity of contents
u_{ij}	Similarity of item j to i
U	Adjacency matrix, $U = [\mathbf{u}_1^T, \dots, \mathbf{u}_K^T]$
$\mathcal{U}_i(N)$	Set of N highest u_{ij} values, related to i
α_{ij}	Prob. to click on j when in i from recommendations
w	Recommendation batch, the RS action
r_{ij}	Prob. that j appears in the recommendation batch w
$Q_i(w)$	User satisfaction by the recommendation batch w
q	Lower level of Q_i enforced by RS
c_i	Network cost of content i
S_t	State/Content visited at time t

- **Top-N policy (R_Q):** Suggest the N files that are most similar to i, i.e. the ones that correspond to the similarities in $\mathcal{U}_i(N)$ (ties broken uniformly). This choice maximizes user satisfaction.
- **Low Cost policy (R_C):** Suggest the N contents with lowest cost. In the case of ties for the cost c_j, recommend contents arbitrarily.
- **q-Mixed policy (R_{MIX}):** Assign $q \cdot 100$ of the budget for user satisfaction. If items are tied in u_{ij}, choose the lowest cost to favor the network. Then assign the remaining budget to the lowest cost items. For $q \to 1$, $R_{MIX} \to R_Q$, and for $q \to 0$, $R_{MIX} \to R_C$

In [18], cached and related items are placed on the top of the recommendation list, while the rest of the list remains intact. Moreover, [28] targets the problem of joint caching and recommendation. For some given cache allocation, the RS's objective is to promote items that minimize the caching cost of the next request *only*, ignoring the possibly many subsequent ones. In both works, the authors allow some *window* of recommendations for the user satisfaction, and the rest is dedicated to the network gains, which is why these policies could be effectively mapped to q-Mixed.

III. Problem Formulation and Solution

We will now cast the problem of optimal sequential recommendations as a Markov Decision Process (MDP) with the objective to minimize the expected cumulative cost in user sessions of arbitrary average length. The user behaviour related to the quality of recommendations will be implicitly taken into account.

A. Defining the MDP

The MDP is defined by the quadruple $(\mathcal{K}, \mathcal{A}, P, \mathbf{c})$ whose entries refer to the following: as state we consider the currently viewed content, hence the state-space $\mathcal{K}$ is the content catalog. Following the discussion in the previous section about per-item frequencies, the action set $\mathcal{A}$ is the set of all $K \times K$ real matrices R, whose entries $r_{ij} \in [0,1]$ determine the frequency of suggesting item j when viewing content i. Based on the assumptions, the user is Markovian, as her next visited state is fully determined by the current one and not the full history. Moreover, P is the probability transition matrix $K \times K$, where P_{ij} is the probability to jump next to content j if the user currently views content i and essentially serves as the *environment* of the MDP will specify the P_{ij} in the following section.

We assume that the RS *knows* the user behavior (the P_{ij} dynamics) and optimizes the actions accordingly. Learning the user while optimizing (e.g. through reinforcement learning) is deferred to future work. Note that we *do not* take into account the time spent on each content by the user nor partial content viewing, but both variations can be easily integrated in our framework. Finally, a random item sequence $\{S_0, S_1, S_2, \dots\}$, with $S_t \in \mathcal{K}$, corresponds to a random sequence of content costs $\{c(S_0), c(S_1), c(S_2), \dots\}$; hence for some $S_t = i$ (the i-th content ID), the cost induced to the network is exactly c_i. The following expression gives the transition probability of state evolution in a general way, letting room for further assumptions to be integrated later on in the model

$$P_{ij} = P\{i \to j\} = \alpha_{ij} \cdot r_{ij} + (1 - \sum_{l=1}^{K} \alpha_{il} \cdot r_{il}) \cdot p_0(j). \quad (5)$$

The above expression is somewhat reminiscent of the random web surfer transitions for PageRank [29], [30], and has the following interpretation. The user can transit to j in two ways. If content j is in the recommendation batch, the user clicks on j with probability $\alpha_{ij} \in [0,1]$. In the event that the user ignores all of the N items in the batch, she chooses j with probability $p_0(j)$ from personal preferences. To see why (5) describes exactly this process, we substitute r_{ij} from (3) to get,

$$P_{ij} = \alpha_{ij} \sum_{w \in \mathcal{A}_i} \mu_i(w) \mathbf{1}_{\{j \in w\}} + (\underbrace{\frac{1}{N} \sum_{j=1}^{K} \sum_{w \in \mathcal{A}_i} \mu_i(w) \mathbf{1}_{\{j \in w\}}}_{\overset{(4)}{=} 1}$$

$$- \sum_{l=1}^{K} \alpha_{il} \sum_{w \in \mathcal{A}_i} \mu_i(w) \mathbf{1}_{\{l \in w\}}) p_0(j) =$$

$$\sum_{w \in \mathcal{A}_i} \mu_i(w) \Big(\alpha_{ij} \mathbf{1}_{\{j \in w\}} + (1 - \sum_{l=1}^{K} \alpha_{il} \mathbf{1}_{\{l \in w\}}) p_0(j) \Big)$$

Observe that for deterministic policies, there is a single w for which $\mu_i(w) = 1$, the P_{ij} is unique, whereas in the case of randomized policies we view P_{ij} as the average transition probability from $i \to j$, over all batches.

Lemma 1. *If $\alpha_{ij} < 1$ and $p_0(i) > 0 \ \forall \ i, j \in \mathcal{K}$, then the MDP $(\mathcal{K}, \mathcal{A}, P, \mathbf{c})$ is unichain, i.e., it has only one class of states for any policy.*

To prove this, one needs to show that the state-space of the MDP forms an irreducible Markov chain, which is true if all state-pairs are communicating, i.e. $P_{ij} > 0$ in (5). It suffices to consider $p_0(j) > 0 \ \forall \ j$ and $\sum_{l=1}^{K} \alpha_{il} \cdot r_{il} < 1$.

B. Optimization Objective

As explained earlier, we consider a user who consumes sequentially a random number of contents before exiting the

session. The induced cost is cumulative over the steps, and since transition probabilities and session-length are random, so is the total cost. The user is considered to start from a given arbitrary state S_0 - whose cost is not accounted for as it is outside the recommender's control, and then her session generates a sequence of costs $c(S_t)$, with $t = 1, \dots, L$, which depends on the policy R. Note that the costs in consecutive states are *not* I.I.D., given the Markovian structure of the problem. The total cost induced by the requests of the user is $\sum_{t=1}^{L} c(S_t)$ and our objective is to minimize is the *average* total cost.

Lemma 2. *The average total cost $v(s)$ starting from initial state $S_0 = s$ can be written as an infinite horizon cost with discounts*

$$v(s) = \mathbb{E}_s \left[\sum_{t=1}^{L} c(S_t) \right] = \mathbb{E}_s \left[\sum_{t=1}^{\infty} \lambda^{t-1} \cdot c(S_t) \right], \quad (6)$$

where the $\mathbb{E}_s$ stands for conditioning on the starting state being $S_0 = s \in \mathcal{K}$, [31, eq. 4.13].

Equality in (6) holds because the random session length L is assumed to be distributed as a $Geom(\lambda)$. The expectation of the total cost is found by applying the law of total expectation

$$\mathbb{E}_s \Big(\sum_{t=1}^{L} c(S_t) \Big) = \sum_{l=1}^{\infty} \mathbb{P}(L = l) \cdot \mathbb{E}_s \Big(\sum_{t=1}^{l} c(S_t) \Big). \quad (7)$$

We refer the reader to [31, Prop. 5.3.1] for more details. The parameter λ is called the discount factor in the sense that the cost incurred in the immediate future is more important than the cost in the far future. The relative importance of future costs depends on the value of $\lambda \in [0, 1]$. Starting from state $i \in \mathcal{K}$ we want to minimize

(Main OP).

$$v^*(i) = \min_{R} \Big\{ \mathbb{E} \Big(\sum_{t=1}^{\infty} \lambda^{t-1} c(S_t, R) \mid S_0 = i \Big) \Big\} \ \forall\ i \ \in \ \mathcal{K} \quad (8)$$

$$\text{subject to} \quad \sum_{j=1}^{K} r_{ij} = N, \ \forall i \in \mathcal{K} \quad (9)$$

$$0 \le r_{ij} \le 1, \ \forall i, j \in \mathcal{K} \quad (10)$$

$$r_{ii} = 0, \ \forall i \in \mathcal{K} \quad (11)$$

$$\sum_{j=1}^{K} r_{ij} \cdot u_{ij} \ge q \cdot Q_i^{max}, \ \forall i \in \mathcal{K}. \quad (12)$$

where $q \in [0, 1]$ is the tuning quality parameter.

S_t is the random variable of the state at step t, and i (or s) is its realisation taking values in $\mathcal{K}$. The optimization variables are the $\{r_{ij}\}$ per-item recommendation frequencies. The feasible space is shaped by the set of constraints imposed on the RS policy, which has to obey four specifications:

- (9): Recommend exactly N items per content view.
- (10): $r_{ij} \in [0, 1]$ is a time-sharing proportion.
- (11): No self-recommendation is allowed.
- (12): Maintain an *average* user satisfaction per viewed content above a pre-defined q, i.e. $\mathbb{E}[Q_i(w)] \ge q$ (see (1)).

Constraint (9) incorporates the number of recommendation slots N in the constraints, following (4). Using the per-item frequencies the solution complexity becomes insensitive to the value of N, something not possible with the initial batch formulation, where the number of batch combinations increases with N.

A hard constraint on the average user satisfaction from the recommendation batch is introduced in (12). If active, the RS is restricted to maintain an average user satisfaction $\ge q$ for every item $i \in \mathcal{K}$, regardless of how the user reacts to good/bad recommendations. We denote the feasible set of policies for viewed content i by $\mathcal{R}_i = \{r_{ij} : (9), (10), (11), (12)\}$. The feasible set is denoted by $\mathcal{R}$ and is convex as the intersection of linear inequalities, equalities and box constraints. It is described by $K^2 + 3K$ linear constraints in total.

C. Optimality

The optimal solution to **Main OP**, i.e., the optimal vector $\mathbf{v}^* = [v^*(1), \dots, v^*(K)]$ for any initial state s is unique and satisfies the Bellman optimality equations (see [Puterman, Theorem 6.2.3] and apply Lemma 1).

Bellman Optimality Equations. The optimal value vector must satisfy the following K (Bellman) equations, one per state,

$$v^*(i) = c_i + \lambda \min_{\mathbf{r}_i \in \mathcal{R}_i} \Big\{ \sum_{j=1}^{K} P_{ij}(\mathbf{r}_i) \cdot v^*(j) \Big\} \ \forall i \in \mathcal{K}. \quad (13)$$

where $P_{ij}(\mathbf{r}_i)$ is defined in (5) and c_i indicates the *immediate cost* of visiting state i. We can apply well established iterative algorithms to solve these equations [31], [32]; we choose here the well-known policy iteration (PI). A key contribution of our work is that unlike "vanilla" MDPs with *discrete actions*, in each iteration we are required to solve a minimization problem in the K-sized variable $\mathbf{r}_i$ for each state i, see (13) which radically reduces the interior minimization step of PI. Importantly, as $\mathbf{v}^*$ remains the same during the policy improvement step (the for loop over the K states), the K minimizers can be straightforwardly parallelized.

D. Versatility of look-ahead horizon via the choice of λ

The MDP has the upside of being flexible on the range of problems it can tackle; part of this flexibility is the application of an arbitrary average session length that can be controlled by λ. Existing works in the literature analyze *infinitely long* sessions, like in [12], [23], which is of course unrealistic. The MDP framework introduced in the current work, uses λ as a tuning parameter that controls the average session length to be equal to $(1 - \lambda)^{-1}$. Let us observe some special cases.

Case: $\lambda \to 0$. The objective function in (8), becomes $v(s) = \mathbb{E}_s[0^0 c(S_1) + 0^1 c(S_2) + \dots] = \mathbb{E}_s[c(S_1)]$ (with the convention $0^0 := 1$) i.e., the user starting state is $S_0 = s$ and does *exactly* one more request which generates loss $c(S_1)$. For $\lambda \to 0$, the only future cost is the immediate cost c_j that is incurred by visiting state $S_1 = j$ at $t = 1$. Thus we can explicitly compute $v(s) = \mathbb{E}_s[c(S_1)] = \sum_{j=1}^{K} P_{s,j} c_j$ and find the optimal policy

by solving K (one for each starting state $s \in \mathcal{K}$) minimization problems

$$\min_{\mathbf{r}_s \in \mathcal{R}_s} \left\{ \sum_{j=1}^{K} P_{s,j}(\mathbf{r}_i) \cdot c_j \right\} \forall\ s\ \in\ \mathcal{K}. \tag{14}$$

Setting $\lambda = 0$ in **Main OP**, our MDP returns the q-Mixed policy.

Case: $\lambda \to 1$. For the infinite horizon, the value $v(s)$ diverges for $\lambda = 1$ (no discount) as it allows infinitely many steps to add-up in the cost. However, we can instead find the *time-average* long-term cost (see [31, Cor.8.2.5]), which is equal to $\lim_{\lambda \to 1}(1-\lambda)v_\lambda(s)$. This is the limit of the ratio of sum cost over average session-length and it is finite for unichain MDPs (Lemma 1).

Short and long length λ. Since $v(s)$ indicates the expected *cumulative cost-to-go*, for $\lambda \to 0$ the state from which the user starts her session matters, and the values of the vector $v(s)$ will differ. On the contrary for $\lambda \to 1$, as the $v(s)$ tend to infinity ($v(s)$ is cumulative and larger as λ grows), the relative difference between the states becomes negligible. That is all the states have approximately the same value and the starting state s does not matter.

In reality, the session length is somewhere in the middle. To determine the value of λ in practice, we use the fact that the mean session-length is equal to $1/(1-\lambda)$. Then, the RS could measure empirical averages of the user session lengths to determine an estimate $\hat{\lambda}$ and substitute this value in the MDP to derive appropriate recommendations.

IV. A User Model

The transition probabilities P_{ij} in (5) allow to model various types of user response to the recommendation policy. The user behaviour is summarised by her click-through probabilities $\{a_{i,j}\}$. These can take specific expressions and be functions of u_{ij} or even r_{ij} to represent some "reactivity" of the user to the policy, and each choice represents a different type of behaviour. Note that as long as P_{ij} is a convex or linear function of r_{ij}, our framework can solve it optimally and efficiently. We study here a specific user, who remains equally curious about any recommended item, provided that the long-term quality of recommendations remains reasonably good. Then, the response of this **"curious" user** is:

$$\alpha_{ij} = \alpha/N \quad \text{for} \quad \mathbb{E}[Q_i] \geq q\ \forall i. \tag{15}$$

The above reads that the user wants to be satisfied by at least q from the RS recommendations (see (12)). Her transition probabilities (i.e., (5)) now become

$$P\{i \to j\} \ = \ \frac{\alpha}{N} \cdot r_{ij} + (1-\alpha) \cdot p_0(j), \tag{16}$$

where notice that $\sum_{l=1}^{K} \frac{\alpha}{N} \cdot r_{ij} = \alpha$. If the user is satisfied, her *expected* click-through rate α remains fixed throughout the session, and she may click any of the N recommended items uniformly at random. Another, perhaps more pragmatic interpretation, is that the RS can just measure some data regarding the user's clickthrough rate and how this relates to the average user satisfaction, and uses these estimates to calibrates the MDP. The tuple (α, q) comprises a wide range of user attitudes ranging from highly curious (high α, low q) to rather conservative (medium/high α, very high q).

MDP Solution. Using (16), the Bellman equations take the form, $\forall i \in \mathcal{K}$,

$$v^*(i) = c_i + \bar{v} + \lambda \frac{\alpha}{N} \min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} r_{ij} \cdot v^*(j) \right\}, \tag{17}$$

where $\bar{v} := \lambda(1-\alpha)\sum_{j=1}^{K} p_0(j) v^*(j)$ is independent of i. Therefore, in each greedy improvement step, the optimizer will have to solve the following optimization problem.

(Inner OP).

$$\min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} r_{ij} \cdot v^*(j) \right\}$$

$$\text{subject to} \quad \mathcal{R}_i = \{r_{ij} : (9), (10), (11), (12)\}$$

where $q \in [0,1]$ is the tuning quality parameter of (12).

Lemma 3. *The greedy improvement step of Policy Iteration for the curious user, reduces to solving the **Inner OP** which is a Linear Program (LP) of size K; the objective and all the constraints are linear on the variables r_{ij}.*

The K LPs in the inner loop of PI can be solved using standard software (e.g. CPLEX). Note here, that solving the MDP returns a randomised policy in general, due to the constraint (12). Moreover, the Bellman equations reveal structural properties of the policy, showing optimality for myopic heuristics as special cases.

Property 1. *For $q = 1$, the optimal policy is the Top-N.*

Proof. For $q = 1$, the rhs of (12) becomes $1 \cdot \sum_{l \in \mathcal{U}_i(N)} u_{il} = Q_i^{\max}$. Assume that the optimal policy for content i is to assign $r_{ij} = 1$ to contents that correspond to $\mathcal{U}_i(N-1)$, and $r_{ij} = x > 0$ to some content with $u_{ij} \notin \mathcal{U}_i(N)$ and $r_{im} = 1 - x$ to the least related item with $u_{im} \in \mathcal{U}_i(N)$. Then the constraint (12) reads

$$\sum_{l \in \mathcal{U}_i(N-1)} u_{il} \ + (1-x)u_{im} + x u_{ij} \geq \sum_{l \in \mathcal{U}_i(N-1)} u_{il} \ + u_{im}$$

$$(u_{ij} - u_{im}) \cdot x \geq 0 \tag{18}$$

By definition, $u_{im} > u_{ij}$ and thus the inequality cannot hold if we assign a positive budget to any item j with $u_{ij} \notin \mathcal{U}_i(N)$. □

Property 2. *For $q = 0$ the optimal policy is the Low Cost.*

Proof. Assume that we can order the optimal values $v^*(i)$ in increasing order $v^*(1) < \cdots < v^*(K)$. To find $v^*(i)$ we need to solve $\min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} r_{ij} \cdot v^*(j) \right\}$. For the case $q = 0$, we can analytically compute $v^*(i)$, because the optimal decision is to assign $r_{ij} = 1$ to the lowest $v^*(j)$ (excluding $v^*(i)$). We can identify two cases for the expression of $v^*(i)$. Case (a): If $1 \leq i \leq N$ then

$$v^*(i) = c_i + \bar{v} + \lambda \left(\sum_{j=1: j \neq i}^{N} v^*(j) + v^*(N+1) \right) \tag{19}$$

where in the above expression we need to make sure we exclude the self recommendation from the evaluation. Else for Case (b): $i > N$, the expression becomes

$$v^*(i) = c_i + \bar{v} + \lambda \sum_{j=1}^{N} v^*(j). \quad (20)$$

We need to compare the values of the states in pairs. There are three possibilities for the pairs. Pair-case (I): $1 \leq i, j \leq N$ and $i < j$, we get the difference (using (19))

$$v^*(i) - v^*(j) < 0 \Rightarrow (c_i - c_j) + \lambda(v^*(j) - v^*(i)) < 0,$$

where for the second term above $N-1$ terms have cancelled out. Notice that due to the ordering, $\lambda(v^*(j) - v^*(i)) > 0$, so it must hold that $c_i - c_j < 0$ for the above expression to have a negative sign. Pair-case (II): $1 \leq i \leq N$ and $j > N$, we use (19) and (20) and we result in the exact same expression for their difference as above. Finally, for Pair-case (III): $N < i < j$, we use (20)

$$v^*(i) - v^*(j) < 0 \Rightarrow c_i - c_j < 0. \quad (21)$$

Therefore, the optimal costs-to-go $v^*(i)$ are ordered exactly as the immediate costs c_i, which concludes that for content i, recommending the N lowest costs excluding i is optimal. □

V. Validation

We evaluate the performance of the proposed MDP recommendation policy in terms of cost and user satisfaction (against other myopic ones) and of computational efficiency.

A. Simulation Setup

Caching Policy and Baseline Cost. In our simulations, we assume that for each dataset, the number of cached items is $M = 0.01 \cdot K$, where K is the size of the corresponding catalog. This number is similar to other works [3] or [14]. We cache the first M IDs of the catalog, i.e., $\mathcal{C} = \{1, \ldots, M\}$. We consider a uniform personal preference distribution $\mathbf{p}_0$ i.e., $\mathbf{p}_0 \sim Uniform(1, \ldots, K)$. Thus, the caching policy is essentially random and the performance of all policies will be unaffected by $\mathbf{p}_0$. We proceed like that as our goal is to understand the *true gains* that come from the RS's *network friendliness*, and *not* the ones hailing from the potential skewness of $\mathbf{p}_0$. Furthermore, the cost of the *non*-cached items is set to an arbitrary price, say 10.0 units, and of the cached ones to an arbitrary smaller price, say 0.0. If we assume that *there is no RS in place*, or equivalently the user never follows recommendations and all requests are generated according to $\mathbf{p}_0$ (i.e., the standard Independent Reference Model), the hit probability $p_{hit} = 0.01$, which easily translates to $\overline{C} = 10.0 \times (1 - p_{hit}) + 0.0 \times p_{hit} = 9.9$ units of cost per request. The numbers above hold for all the plots of the section since uniform $\mathbf{p}_0$ and $M/K = 0.01$ hold everywhere.

Simulation and Metrics. The RS *knows* exactly the user model and the evaluated policy is the optimal one computed by the MDP. The first metric is the average cost, denoted as $\overline{C}$ (which is network-oriented) and the second one is the average user satisfaction denoted as $\overline{Q}$; both are measured *per content request*. We perform a Monte Carlo simulation where we generate 1000 sessions of random size L, where $L \sim Geom(\lambda)$ (λ is a parameter of the simulation). Therefore, we measure C_L (see (2)) and Q_L (see (1)) For some fixed λ, the quantities $\overline{C}$ and $\overline{Q}$ are produced by further averaging C_L and Q_L over 1000 runs. In some experiments we will also report the cache hit relative gain of MDP against other policies which will be defined as $\frac{p_{hit}^{MDP} - p_{hit}^{ref}}{p_{hit}^{ref}} \cdot 100\%$.

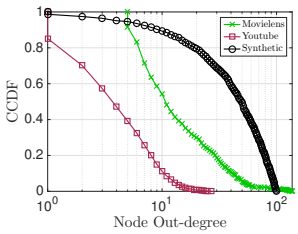


Fig. 1: Traces Out-Degree CCDF

Execution of the PI Algorithm. All experiments were carried out using a PC with RAM: 8 GB 1600 MHz DDR3 and Processor: 1,6 GHz Dual-Core Intel Core i5. The minimizers of **Inner OP** that arise were solved through CPLEX.

B. Traces

We use three datasets to construct three content relation graphs U (Section II-B), two real ones and one synthetic.

MovieLens. We consider the Movielens movie rating dataset [33], containing 69162 ratings (0 to 5 stars) of 671 users for 9066 movies. We apply an item-to-item collaborative filtering to extract the missing user ratings, and then use the cosine similarity with range $[-1, 1]$ of each pair of contents. We floor all values < 0.5 to zero and leave the remaining ones intact. Finally, we remove from the library contents with less than 25 related items to end up with a relatively dense U.

YouTube. We consider the YouTube dataset found in [34]. From this, we choose the largest component of the library and build a graph of 2098 nodes (contents) if there is a link from $i \to j$. As the values of the dataset were $u_{ij} \in \{0, 1\}$, whenever an edge was found, we assigned it a random weight $u_{ij} \sim Uniform(0.5, 1)$.

Synthetic. We consider a synthetic content graph U; this way we can see how the algorithm behaves in a more uniform U. We decide the size of the library K, and for every item in the library we draw a number out of $Uniform(1, 100)$ which serves as the number of neighbors of i. We then assign on the edges a random weight $u_{ij} \sim Uniform(0.5, 1)$.

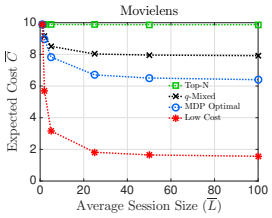
For these datasets, we present the statistics related to U, and its relation to the cached contents. To this end, based on U, we consider there is an edge from $i \to j$ if $u_{ij} > 0$ and we are interested on the out-degree of the nodes. The graph in general is directed.

- $\deg_i^-$: out-degree of node i.
- $\deg_i^-(\mathcal{C})$: out-degree of node i directed *only* to nodes in the set $\mathcal{C}$ (the set of cached items).

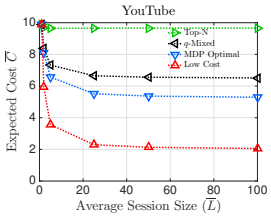
In Fig.1, on the x-axis we see the $\deg_i^-$ in logarithmic scale, and on the y-axis its ccdf. We can conclude for the

TABLE II: Statistics - Datasets

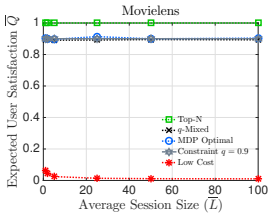
	MovieLens	YouTube	Synthetic
Nodes	1060	2098	2000
Total Edges	20162	11288	99367
mean $\deg^-$	19.02	5.38	49.68
mean $\deg^-(\mathcal{C})$	0.17	0.06	0.51



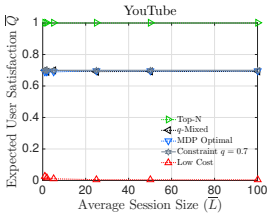
(a) MovieLens - Cost



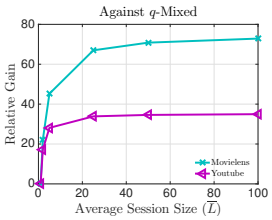
(b) YouTube - Cost



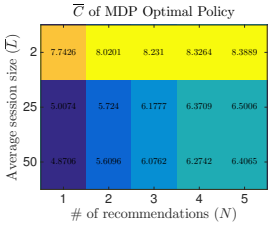
(c) Movielens - User Satisfaction



(d) YouTube - User Satisfaction



(e) Relative Gain



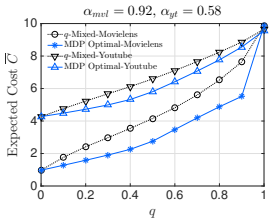
(f) Cost vs num. of recommendations N and mean session size $\overline{L}$ ($\alpha = 0.75$, $q = 0.75$) - Synthetic

Fig. 2: Subfigs. (a) - (e): Metrics vs mean session size $\overline{L}$, mvlns: $\{\alpha = 0.85, q = 0.90, N = 2\}$, yt: $\{\alpha = 0.8, q = 0.7$, N = 2$\}$

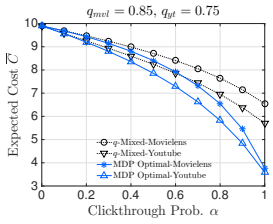
two real traces, that the $\deg_i^-$ tends to be quite high only for a small fraction of the nodes.

C. Results: Sensitivity Analysis

Effect of mean session size ($\overline{L}$). We first compare the performance benefits of MDP which has look-ahead capabilities against myopic ones, when the size of the user session increases. To this end, in Fig. 2, we vary the parameter λ (Section II) to simulate random sessions with mean size $\overline{L} = \{1, 2, 5, 25, 50, 100\}$; we remind the reader that $\overline{L} = (1 - \lambda)^{-1}$. We compare the performance of MDP against the three myopic ones discussed in Section II-D. For the Movielens we set $q = 90\%$, and for the Youtube $q = 70\%$ in order to ensure high user satisfaction $\overline{Q}$ for the cost-oriented policies. This hard constraint of $\overline{Q}$ is depicted in Figs. 2(c), 2(d) with a dashed grey line. For the q-Mixed policy we set it accordingly, hence it becomes a 0.9-Mixed and a 0.7-Mixed policy. The extreme policy Top-N achieves $\overline{Q} = 1$, which is the upper bound for any policy, and the worst $\overline{C}$. In total



(a) vs increasing q



(b) vs increasing α

Fig. 3: MDP-Optimal and q-Mixed evaluated on $\overline{L} = 25$ requests ($N = 2$)

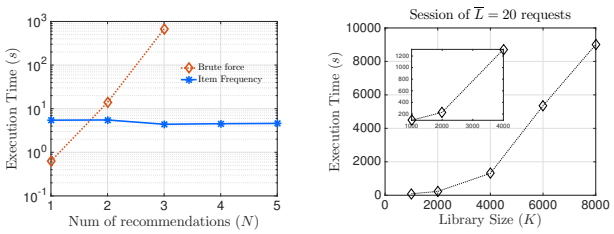
contrast, the Low Cost returns the best possible cost but is *infeasible*. The policy q-Mixed offers user satisfaction at or above the feasibility boundary.

Obs. #1: The MDP-optimal policy keeps the user satisfaction feasible while achieving the minimum cost $\overline{C}$, in Figs. 2(a), 2(b) from the feasible myopic policies. Moreover, in Fig. 2(e), we show the relative gain of the MDP with respect to q-Mixed as reference policy. Note that the respective gains against Top-N, which is omitted from the plot as it has no bias towards $\mathcal{C}$, are more than 1000%. Reasonably, the longer the horizon, the larger the gains of MDP which is equipped with look-ahead capabilities.

A Note on Caching. Our caching is essentially random. We could instead cache the items that have the most neighbors (in terms of U) or cache the top-M items from the stationary distribution as created by the recommendation policy Top-N. Thus, we can loosely state that the cost performances of the MDP we see here, serve as a lower bound.

Effect of q and α. For each dataset, in Fig. 3(a), we pick some α and tighten the quality constraint by increasing q. In the same fashion, in Fig. 3(b), we pick some q for every dataset and increase the value of α. We present here only the average cost per request, since the RS quality achieved is equal to the q value selected. Furthermore, we omit the two extreme policies, Top-N and Low Cost, and only compare to q-Mixed, who also seeks a (suboptimal) tradeoff between cost and user satisfaction. The first thing to notice is that, for $q = 1$ and $q = 0$, the two policies coincide, which is an immediate result of Properties 1 and 2 as the optimal policies are Top-N and Low-Cost respectively. For all intermediate q values, it is evident that the MDP-optimal policy improves performance compared to the q-Mixed. We can also observe that the MDP-optimal policy is able to better exploit the increase of α as the gap between the policies becomes wider. This should not come as a surprise since the myopic policies *do not* take into consideration the dynamics of user transitions. Note that an average session of $\overline{L} = 25$, could loosely correspond to a 45min session of watching YouTube short clips [22].

Effect of recommendation batch size (N). In Fig. 2(f), we focus on the effect of N on the expected cost. We provide a heatmap with increasing $\overline{L}$ on the y-axis and increasing N on the x-axis. We observe that irrespective of $\overline{L}$, the cost is becoming worse with the increase of N even for the Synthetic graph which has a much larger $\deg^-(\mathcal{C}) = 0.51$.



(a) Proposed vs Brute Force, $\overline{L} = 50$

(b) Execution Times vs Library Size K, $\overline{L} = 20$

Fig. 4: MDP: Execution Times

Obs. #3: Network-friendly recommendations *is not* an easy task, but *many* network-friendly recommendations is even harder. To better grasp this, consider a myopic RS. To satisfy both parties (network and user), when the user is at content i, the RS must have many cached *and* related contents to recommend, which by definition are less than cached *or* related.

D. Execution Time Savings (and not only)

Up to this point, we investigated tradeoffs between different policies. Yet an important contribution of this work is its computationally efficient framework, which we discuss next.

Item-frequency vs Batch-frequency formulation. One of the main contributions of this work is that it formulates a continuous problem of item-frequencies, rather than batch-frequencies. This is profitable computationally both in the number of variables and in execution time. In Fig. 4(a), we choose a catalog of $K = 150$, and solve the MDP using our approach and compare it against the brute force solution of a batch-MDP, which enumerates all the feasible tuples (the ones that satisfy the quality constraints), and picks the best one. As claimed in Section II-D, we see that the increase of N is devastating for the batch-frequency MDP, whereas our item-frequency approach is insensitive to it.

Catalog Increase. In this part, we investigate the execution times of our item-frequency MDP algorithm, by varying the catalog size. We select $\overline{L} = 20$, increase the content library size, run the algorithm, and report the time it took until completion. These results show that our MDP can tune recommendations of practical size and not only toy scenarios of some hundred items. As stated in [3], even a library of $K = 1000$ can be considered practical since it could refer to the 1000 most popular files of Netflix for example. The authors in [14] perform simulation with sizes $K = 1000$ and $K = 10000$, which is the same order as our experiments; however, they do not report any execution time results. The MDP needs about 9000 seconds ($\approx$ 2.5h) for a library of 8000, using the 8Gb RAM PC, and under-exploited parallelisation. These run-times will be significantly decreased in a powerful server with multiple cores, as the Policy Iteration algorithm we have implemented runs on as many cores as it finds available.

Here we compare our MDP with the policy of [23] where the objective is to minimize the per request average cost over an *infinite size* session. Their framework easily reduces to ours by setting $\lambda \to 1$ (in practice we set $\lambda = 0.9$, that is 10 steps look-ahead) and assuming $\alpha_{ij} = \frac{\alpha}{N}$, i.e., uniform click towards recommended content. In [23], the authors formulate the average cost minimization as an LP of size K^2 and the optimal solution is found using CPLEX. Their solution is constrained to obey *stationarity* which builds a very demanding set of constraints and is unrealistic as the size of a user session *is finite* in practice. In Table III, for the two datasets, we report the execution time of the algorithm and the achieved mean cost $\overline{C}$ under the stationary regime, i.e., we plug our policy into the objective of [23]. In that table, we refer to our policy as MDP(0.9) (due to the selected λ) and to the one of [23] as *OPT*.

TABLE III: MDP(0.99) vs *OPT* (under [23])

	Cost (units)		Exec. Time (s)	
	MDP(0.9)	*OPT*	MDP(0.9)	*OPT*
Movielens	5.5625	5.5432	105	560
Youtube	6.1005	6.1001	253	1997

The results of this experiment are summarized in Table III. Impressively, there is an execution time speed up by a factor of 5 and 10 for the two datasets, while sacrificing almost nothing in terms of cost performance.

We now do the exact opposite; for smaller sessions, $\overline{L} = \{1, 2, 3, 4, 5\}$, we present the relative gain of MDP over the policy of [23] and the q-Mixed. Our approach finds the optimal cost for all $\overline{L}$. The smaller the horizon, the bigger the gain of MDP with respect to [23]. Reasonably, as the horizon increases, the relative gain fades as [23] is exactly tailored for *very* long sessions. Note that for such small sessions the MDP rutime is obviously even lower than the one shown in Table III, because smaller $\overline{L}$ translates to smaller λ, which implies faster convergence of the policy iteration. Finally, as seen in previous plots, the gain over the q-Mixed is growing with the horizon $\overline{L}$.

TABLE IV: Evaluating on smaller $\overline{L}$, Movielens

Gain over	$\overline{L}$=1	$\overline{L}$=2	$\overline{L}$=3	$\overline{L}$=4	$\overline{L}$=5
[23] (%)	21.29	16.27	10.4528	4.48	4.87
q-Mix (%)	0.01	9.67	21.46	22.39	27.69

VI. Conclusions

We have developed a very promising MDP framework for optimal look-ahead NFR that is able to exploit the structure of the content graph and discover non-obvious recommendations. More importantly, by using item-frequency recommendations in the Bellman equations we have proposed an algorithm that scales well both with the size of content library, as well as with the batch-size. The complexity remains low as the inner optimization problems have less unknowns, they are linear or at worse convex, and allow for parallelisation. Finally, as MDP sets the stage for learning-based approaches, we firmly believe that our reduced variable representation MDP (via the item-frequency) can significantly speed up the training phase of such algorithms.

REFERENCES

[1] Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update," 2015-2020.

[2] D. A. Farber, R. E. Greer, A. D. Swart, and J. A. Balter, "Internet content delivery network," Nov. 25 2003. US Patent 6,654,807.

[3] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless video content delivery through distributed caching helpers," in *Proc. IEEE INFOCOM*, 2012.

[4] 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 Communications Magazine*, vol. 57, no. 1, pp. 80–86, 2018.

[5] H. Nam, K.-H. Kim, and H. Schulzrinne, "Qoe matters more than qos: Why people stop watching cat videos," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, IEEE, 2016.

[6] 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.

[7] R. Zhou, S. Khemmarat, and L. Gao, "The impact of youtube recommendation system on video views," in *In Proc. of ACM IMC 2010*.

[8] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. WWW*, 2001.

[9] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proc. ACM RecSys*, pp. 191–198, 2016.

[10] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, 2009.

[11] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, "Soft cache hits: Improving performance through recommendation and delivery of related content," *IEEE JSAC*, 2018.

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

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

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

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

[16] S. Kastanakis, P. Sermpezis, V. Kotronis, and X. Dimitropoulos, "CABaRet: Leveraging recommendation systems for mobile edge caching," in *Proc. ACM SIGCOMM Workshops*, 2018.

[17] D. Liu and C. Yang, "A learning-based approach to joint content caching and recommendation at base stations," *arXiv preprint arXiv:1802.01414*, 2018.

[18] D. K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen, "Cache-centric video recommendation: an approach to improve the efficiency of youtube caches," *ACM TOMM*, vol. 11, no. 4, p. 48, 2015.

[19] A. Al-Dailami, C. Ruan, Z. Bao, and T. Zhang, "Qos3: Secure caching in https based on fine-grained trust delegation," *Security and Communication Networks*, vol. 2019, 2019.

[20] J. Krolikowski, A. Giovanidis, and M. Di Renzo, "Optimal cache leasing from a mobile network operator to a content provider," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 2744–2752, IEEE, 2018.

[21] "The average mobile YouTube session is now 40 minutes, Google says." https://www.cio.com/article/2949473/the-average-mobile-youtube-session-is-now-40-minutes-google-says.html.

[22] "Google spells out how YouTube is coming after TV." http://www.businessinsider.fr/us/google-q2-earnings-call-youtube-vs-tv-2015-7/.

[23] T. Giannakas, T. Spyropoulos, and P. Sermpezis, "The order of things: Position-aware network-friendly recommendations in long viewing sessions," in *WiOpt*, 2019.

[24] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999*, pp. 230–237, Association for Computing Machinery, Inc, 1999.

[25] L. Spinelli and M. Crovella, "Closed-loop opinion formation," ACM WebSci '17, pp. 73–82, 2017.

[26] Y. Lv, T. Moon, P. Kolari, Z. Zheng, X. Wang, and Y. Chang, "Learning to model relatedness for news recommendation," ACM WWW '11, pp. 57–66, 2011.

[27] B. Blaszczyszyn and A. Giovanidis, "Optimal geographic caching in cellular networks," in *2015 IEEE international conference on communications (ICC)*, pp. 3358–3363, IEEE, 2015.

[28] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos, "Caching-aware recommendations: Nudging user preferences towards better caching performance," in *Proc. IEEE INFOCOM*, 2017.

[29] S. Brin and L. Page., "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[30] O. Fercoq, M. Akian, M. Bouhtou, and S. Gaubert, "Ergodic control and polyhedral approaches to pagerank optimization," *IEEE Trans. on Automatic Control*, vol. 58, pp. 134–148, Jan 2013.

[31] M. L. Puterman, *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[32] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*, vol. 5. Athena Scientific Belmont, MA, 1996.

[33] "https://grouplens.org/datasets/movielens."

[34] http://netsg.cs.sfu.ca/youtubedata/, 2007.